Reproducible Open Benchmarks for Data Analysis Platform

Heiko Mueller

May 25, 2022

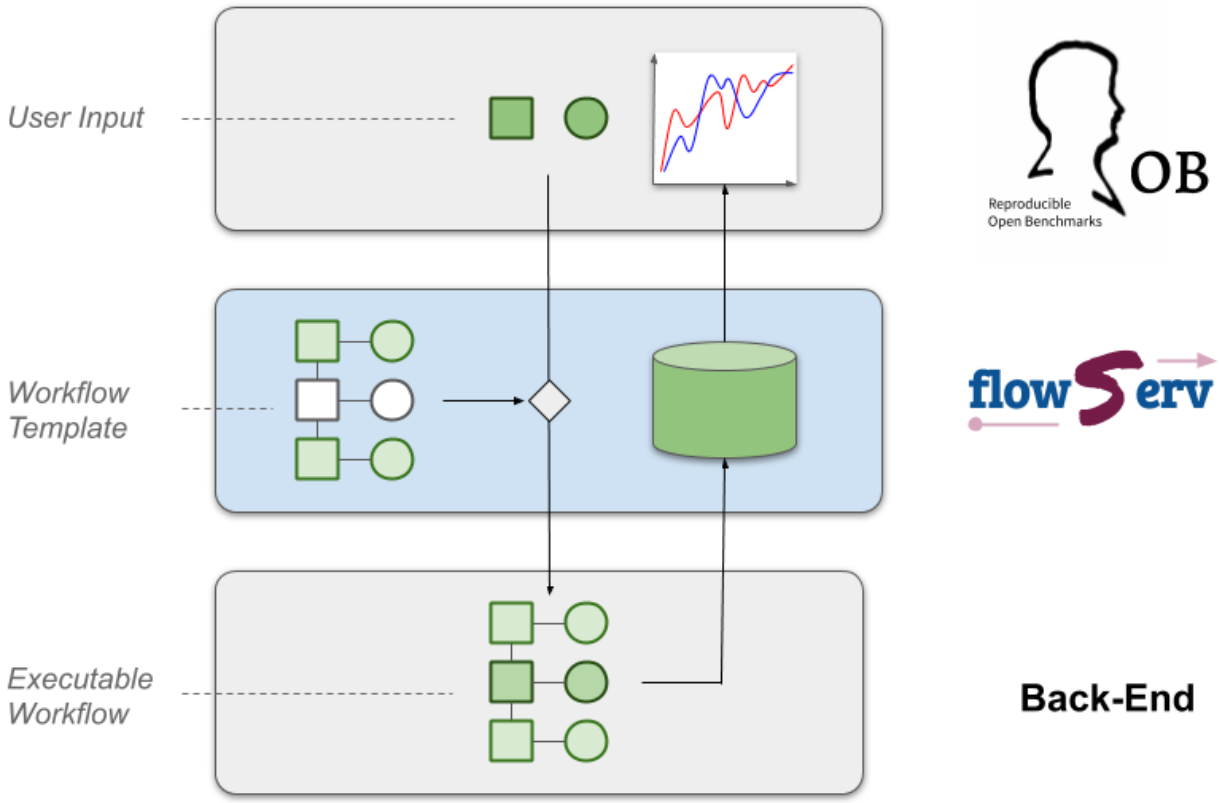
CONTENTS:

Reproducible and Reusable Workflows

This repository contains the implementation of the core infrastructure for the *Reproducible and Reusable Data Analysis Workflow Server* (*flowServ*). This is an experimental prototype to support reuse and evaluation of published data analysis pipelines as well as community benchmarks of data analysis algorithms. *flowServ* is not yet-another workflow engine. The aim instead is to provide a layer between a client (e.g. a Web user interface) and a workflow engine to facilitate the execution of a defined workflow templates (as shown in the figure below). *flowServ* is designed to be independent of the underlying workflow engine.

Workflow templates contain placeholders for workflow steps and/or input data and parameters that are provided by the user (e.g., by providing Docker containers that satisfy the workflow steps or uploading input data files). *flowServ* triggers and monitors the execution of the workflow for the given input values and maintains the workflow results. The API provides the functionality to submit new workflow runs and to retrieve the evaluation results of completed workflow runs.

flowServ was motivated by the Reproducible Open Benchmarks for Data Analysis Platform (ROB). The goal of ROB is to allow user communities to evaluate the performance of their different data analysis algorithms in a controlled competition-style format. In ROB, the benchmark coordinator defines the workflow template along with input data. Benchmark participants provide their own implementation of the variable workfow steps. The workflow engine processes workflows on submission. Execution results are maintained by *flowServ* in an internal database. The goal of *flowServ* is to be a more generic platform that can not only be used for benchmarks but also for other types of data analysis workflows.


User Input
Workflow Template
Executable Workflow
Reproducible Open Benchmarks
flowServ
Back-End

CHAPTER

ONE

WORKFLOW TEMPLATES

Workflow Templates are parameterized workflow specifications for the *Reproducible and Reusable Data Analysis Workflow Server* (**flowServ**). Workflow templates are motivated by the goal to allow users to run pre-defined data analytics workflows while providing their own input data, parameters, as well as their own code modules. Workflow templates are inspired by, but not limited to, workflow specifications for the Reproducible Research Data Analysis Platform (REANA).

1.1 Motivation for Parameterized Workflow Templates

Consider the REANA Hello World Demo. The demo workflow takes as input a file `data/names.txt` containing a list of person names and a timeout parameter `sleeptime`. For each line in `data/names.txt` the workflow writes a line "Hello *name*!" to an output file `results/greetings.txt`. For each line that is written to the output file, program execution is delayed by a number of seconds defined by the *sleeptime* parameter.

Workflow specifications in REANA are serialized in YAML or JSON format. The names of the input and output files as well as the value for the sleep period are currently hard-coded in the workflow specification file (e.g. reana.yaml).

```
inputs:
  files:
    - code/helloworld.py
    - data/names.txt
  parameters:
    helloworld: code/helloworld.py
    inputfile: data/names.txt
    outputfile: results/greetings.txt
    sleeptime: 0
workflow:
  type: serial
  specification:
    steps:
      - environment: 'python:2.7'
        commands:
          - python "${helloworld}"
              --inputfile "${inputfile}"
              --outputfile "${outputfile}"
              --sleeptime ${sleeptime}
outputs:
  files:
   - results/greetings.txt
```

Assume we want to build a system that allows users to run the Hello world demo via a (web-based) interface where they provide a text file with person names and a sleep period value. There are three main parts to such a system. First, we need to display a form where the user can select (upload) a text file and enter a sleep time value. Second, after the user submits their input data, we need to create an updated version of the workflow specification as shown above where we replace the value of `inputfile` and `sleeptime` with the user-provided values. We then pass the modified workflow specification to a REANA instance for execution. There are several way for implementing such a system. Parameterized workflow templates are part of the solution that is implemented for **flowServ**.

1.2 What are Parameterized Workflow Templates?

Similar to REANA workflow specifications, parameterized workflow templates are serialized in YAML or JSON format. Each template has up to six top-level elements: `workflow`, `parameters`, `parameterGroups`, `outputs`, `results`, and `postproc`. Only the `workflow` element is mandatory in a workflow template.

The `workflow` element contains the workflow specification. The structure and syntax of this specification is dependent on the backend (engine) that is used to execute the final workflow. If the REANA Workflow Engine is being used, the workflow specification is expected to follow the the common syntax for REANA workflow specifications.

1.2.1 Template Parameters

The `parameters` section defines those parts of the workflow that are variable with respect to user inputs. We refer to these as *template parameters*. Template parameters can for example be used to define input and output values for workflow steps or identify Docker container images that contain the code for individual workflow steps. The detailed parameter declarations are intended to be used by front-end tools to render forms that collect user input.

An example template for the **Hello World example** is shown below.

```
workflow:
    inputs:
      files:
        - code/helloworld.py
        - $[[names]]
      parameters:
        helloworld: code/helloworld.py
        inputfile: $[[names]]
        outputfile: results/greetings.txt
        sleeptime: $[[sleeptime]]
    workflow:
      type: serial
      specification:
        steps:
          - environment: 'python:2.7'
            commands:
              - python "${helloworld}"
                  --inputfile "${inputfile}"
                  --outputfile "${outputfile}"
                  --sleeptime ${sleeptime}
    outputs:
      files:
       - results/greetings.txt
parameters:
    - name: names
```

(continues on next page)

(continued from previous page)

```
    label: Person names
    description: Text file containing person names
    dtype: file
  - name: sleeptime
    label: Sleep period
    description: Sleep period in seconds
    dtype: int
```

In this example, the workflow section is a REANA workflow specification. The main modification to the workflow specification is a simple addition to the syntax in order to allow references to template parameters. Such references are always enclosed in `$[[...]]`. The parameters section is a list of template parameter declarations. Each parameter declaration has a unique identifier. The identifier is used to reference the parameter from within the workflow specification (e.g., `$[[sleeptime]]` to reference the user-provided value for the sleep period). Other elements of the parameter declaration are a human readable short name, a parameter description, and a specification of the data type. Refer to the Template Parameter Specification for a full description of the template parameter syntax.

Note that for **serial workflows** parameter references are only allowed in the `input` part of the workflow specification or as full workflow steps.

Parameter declarations are intended to be used by front-end tools to render forms that collect user input. Given a set of user-provided values for the template parameters, the references to parameters are replaced withing the workflow specification with the given values to generate a valid workflow that can be executed by the respective workflow engine.

1.2.2 Grouping of Template Parameters

Template parameters can be grouped for display purposes. In a front-end application, each parameter group should be rendered within a separate visual components. The details are dependent on the application.

The structure for the `parameterGroups` element in a workflow template is as follows:

```
parameterGroups:
    - name: 'Unique module name'
      title: 'Module title for display purposes'
      index: 'Index position of the parameter block for ordering during visualization'
```

The group that a parameter belongs to is reference by the unique group name in the `parameterGroups` element of the parameter declaration.

1.2.3 Workflow Outputs

Workflow specifications like those defined by REANA include a list of output files and directories that are generated by each workflow run. The workflow template allows a user to further specify properties for all/some of these output files that are used by front-end applications for display purposes.

If an `outputs` element is present in a workflow template only those files that are listed in the section will be available for individual download via the API. Otherwise, if no `outputs` element is present all files that are returned by the workflow are accessible via the API. Note that the granularity depends on the (implementation-specific) listing of result files for the workflow specification.

The structure for the `outputs` element in a workflow template is as follows:

```
outputs:
    - source: 'Relative path to the file in the run result folder'
```

(continues on next page)

(continued from previous page)

```
      key: 'Unique user-defined key for the resource that can be used for
        accessing the resource in a dictionary (e.g., in the flowapp
        result object)'
      title: 'Header when displaying the file contents (optional)'
      caption: 'Caption when displaying the file contents (optional)'
      format: 'Object containing information about file format (optional)'
      widget: 'Object containing information that specifies the widget to be
        used for displaying the file content (optional)'
```

The structure of the `format` and `widget` element is not further specified. These elements are interpreted by the front-end applications only. See the definition of default file formats and widgets page for details on the supported values for these elements.

1.3 Benchmark Templates

The definition of workflow templates is intended to be generic to allow usage in a variety of applications. With respect to *Reproducible Open Benchmarks* we define extensions of workflow templates that are used to generate the benchmark leader board and compute benchmark metrics.

Benchmark Templates extend the base templates with information about the schema of the benchmark results. The idea is that benchmark workflows contain steps towards the end that evaluate the results of a benchmark run. These evaluation results are stored in a simple JSON or YAML file. Result files are usedto create the benchmark leader board.

1.3.1 Benchmark Results

Benchmark templates add a `results` section to a parameterized workflow template.

```
workflow:
    version: 0.3.0
    inputs:
      files:
        - code/analyze.py
        - code/helloworld.py
        - $[[names]]
      parameters:
        inputfile: $[[names]]
        outputfile: results/greetings.txt
        sleeptime: $[[sleeptime]]
        greeting: $[[greeting]]
    workflow:
      type: serial
      specification:
        steps:
          - environment: 'python:3.7'
            commands:
              - python code/helloworld.py
                  --inputfile "${inputfile}"
                  --outputfile "${outputfile}"
                  --sleeptime ${sleeptime}
                  --greeting ${greeting}
```

(continues on next page)

(continued from previous page)

```
            - python code/analyze.py
                --inputfile "${outputfile}"
                --outputfile results/analytics.json
    outputs:
      files:
       - results/greetings.txt
       - results/analytics.json
parameters:
    - name: names
      label: 'Input file'
      datatype: file
      as: data/names.txt
    - name: sleeptime
      datatype: int
      defaultValue: 10
    - name: greeting
      datatype: string
      defaultValue: 'Hello'
results:
    file: results/analytics.json
    schema:
        - name: avg_count
          label: 'Avg. Chars per Line'
          type: decimal
        - name: max_len
          label: 'Max. Output Line'
          type: decimal
        - name: max_line
          label: 'Longest Output'
          type: string
          required: False
    orderBy:
        - name: avg_count
          sortDesc: true
        - name: max_len
          sortDesc: false
```

The `results` section has three parts: (1) a reference to the result `file` that contains the benchmark run results, (2) the specification of the elements (columns) in the benchmark result `schema`, and (3) the default sort order (`orderBy`) when generating a leader board. The schema is used to extract information from the result file and store the results in a database. In the given example, the benchmark results contain the average number of characters per line that is written by `helloworld.py`, and the length and text of the longest line in the output. When generating the leader board results are sorted by the average number of characters (in descending order) and the length of the longest line (in ascending order).

The benchmark results are generated by the second command in the workflow step by the `analyze.py` script that is part of the benchmark template.

```
"""Analytics code for the adopted hello world Demo. Reads a text file (as
produced by the helloworld.py code) and outputs the average number of characters
per line and the number of characters in the line with the most characters.
"""
```

(continues on next page)

(continued from previous page)

```
import argparse
import errno
import os
import json
import sys


def main(inputfile, outputfile):
    """Write greeting for every name in a given input file to the output file.
    The optional waiting period delays the output between each input name.
    """
    # Count number of lines, characters, and keep track of the longest line
    max_line = ''
    total_char_count = 0
    line_count = 0
    with open(inputfile, 'r') as f:
        for line in f:
            line = line.strip()
            line_length = len(line)
            total_char_count += line_length
            line_count += 1
            if line_length > len(max_line):
                max_line = line
    # Create results object
    results = {
        'avg_count': total_char_count / line_count,
        'max_len': len(max_line),
        'max_line': max_line
    }
    # Write analytics results. Ensure that output directory exists:
    # influenced by http://stackoverflow.com/a/12517490
    dir_name = os.path.dirname(outputfile)
    if dir_name != '':
        if not os.path.exists(dir_name):
            try:
                os.makedirs(dir_name)
            except OSError as exc:  # guard against race condition
                if exc.errno != errno.EEXIST:
                    raise
    with open(outputfile, "w") as f:
        json.dump(results, f)


if __name__ == '__main__':
    args = sys.argv[1:]

    parser = argparse.ArgumentParser()
    parser.add_argument("-i", "--inputfile", required=True)
    parser.add_argument("-o", "--outputfile", required=True)

    parsed_args = parser.parse_args(args)
```

(continues on next page)

(continued from previous page)

```
    main(inputfile=parsed_args.inputfile, outputfile=parsed_args.outputfile)
```

1.3.2 Result Schema Specification

The result schema specification defines a list of columns that correspond to columns in a table that is created in an underlying relational database to store benchmark results. For each column specification the following elements are allowed:

- **name**: Unique column identifier. The value is used as the column name in the created database table.
- **label**: Human-readable name that is used when displaying leader boards in a front-end.
- **type**: Data type of the result values. The supported types are `decimal`, `int`, and `string`. These type are translated into the relational database types `DOUBLE`, `INTEGER`, and `TEXT`, respectively.
- **required**: Boolean value that corresponds to a `NOT NULL` constraint. If the value is `true` it is expected that the generated benchmark result contains a value for this column. The default value is `true`.

The first three elements (`name`, `label`, and `type`) are mandatory.

1.3.3 Generating Leader Board

Leader boards are generated from benchmark results in the database table. The default sort order for results determines the ranking of entries in the leader board. It is defined in the `orderBy` section of the benchmark result specification. The `orderBy` section is a list of columns together with the sort order for column values. This list corresponds to an ORDER BY clause in the SQL query that is used to retrieve benchmark results.

Each entry in the `orderBy` list has the following elements:

- **name**: Unique column identifier
- **sortDesc**: Boolean value to determine the sort order (true: DESCENDING or false: ASCENDING).

Only the `name` element is mandatory. The value has to match one of the column identifiers in the `schema` section. By default all columns are sorted in descending order. If no `orderBy` element is given the first column in the `schema` is used as the sort column.

CHAPTER

TWO

TEMPLATE PARAMETERS

Template parameters are used to define the variable parts of workflow templates. One of the primary use cases for parameter declarations is to render input forms to collect user input data.

The mandatory elements of parameter declarations are:

- **name**: Unique identifier for the parameter. This identifier is used in the template workflow specification to reference the parameter value.
- **label**: Descriptive short-name for the parameter (to be displayed in a front-end input form).
- **dtype**: Type of the expected value. Valid data types are `bool`, `select`, `file`, `float`, `int`, and `string`.
- **index**: The index defines the order in which parameters are presented in the front-end input form.
- **isRequired**: Boolean flag indicating whether the user is required to provide a value for the parameter or not.

In addition, several optional elements can be given for a parameter declaration:

- **description**: Additional descriptive information about the parameter (to be displayed in a front-end input form).
- **defaultValue**: Default value for the parameter.
- **module**: Identifier of the group (module) that the parameter belongs to.

Depending on the data type of a parameter, additional element can be present in the declaration.

2.1 Enumeration Parameters

Parameters of type `select` have a mandatory element `values` that specifies the valid parameter values. Each entry in the values list contains (up-to) three elements:

- **name**: Display name for the value.
- **value**: Actual value if this item is selected by the user.
- **isDefault**: Optional element to declare a list item as the default value (for display purposes).

An example declaration for an enumeration parameter is shown below:

```
- name: 'imageType'
  label: 'Image Type'
  description: 'The type of micrscopy used to generate images'
  type: 'select'
  defaultValue: 'brightfield'
  values:
      - name: 'Brightfield'
```

(continues on next page)

(continued from previous page)

```
      value: 'brightfield'
      isDefault: true
    - name: 'Phasecontrast'
      value: 'phasecontrast'
      isDefault: false
  isRequired: true
```

2.2 Input File Parameters

Parameters of type `file` have one additional optional element `target`. The file target specifies the (relative) target path for an uploaded input file in the run folder. If the target is not specified in the parameter declaration it can be provided by the user as part of the arguments for a workflow run. Note that the default value for a file parameter points to an existing file in the workflow template's file structure but is also used as the default target path.

An example declaration for a file parameter is shown below:

```
- name: 'names'
  label: 'Names File'
  dtype: 'file'
  target: 'data/names.txt'
  index: 0
  isRequired: true
```

2.3 Numeric Parameters

Parameters of type `float` or `int` have an optional `range` element to specify constraints for valid input values. Range constraints are intervals that are represented as strings. Round brackets are used to define open intervals and square brackets define closed intervals. A missing interval boundary is interpreted as (positive or negative) infinity. An open interval does not include the endpoints and a closed interval does.

An example declaration for a integer parameter is shown below:

```
- name: 'maxProportion'
  label: 'Max. Proportion'
  dtype: 'float'
  index: 3
  defaultValue: 0.75
  range: '[0,1]'
```

CHAPTER

THREE

POST-PROCESSING WORKFLOWS

Post-processing workflows are workflows in a template that run after a workflow run finished and the ranking of workflow evaluation results has changed. Post-processing workflows are defined sjust as the main workflow in the template. Post-processing workflows are executed by the same engine as the workflow.

The main difference is that post-porcessing workflows do not have access to any iser arguments. The only argument that can be references `$[[runResultsFile]]` is a JSON file that contains a list of all runs in the current ranking result together with the path to their result files. The format of the file is as follows:

```
postproc:
    workflow:
        version: '0.3.0'
        inputs:
          files:
            - 'postproc_code/'
            - 'data/evaluate/labels.pkl'
        workflow:
          type: 'serial'
          specification:
            steps:
              - environment: 'heikomueller/toptaggerdemo:1.0'
                commands:
              - 'python postproc_code/plot-roc-auc.py \
                    $[[runs]] data/evaluate/labels.pkl results'
              - 'python postproc_code/plot-roc-bg-reject.py \
                    $[[runs]] data/evaluate/labels.pkl results'
        outputs:
          files:
           - 'results/ROC-AUC.png'
           - 'results/ROC-BGR.png'
    inputs:
        files:
            - 'results/yProbBest.pkl'
        runs: '.runs'
```

CHAPTER

FOUR

PACKAGING WORKFLOW TEMPLATES

This file describes how to structure your project and the necessary files that make it easy to add your workflow to the flowServ repository.

Here is a sample layout based on the Hello World Demo as an example.

```
template/
    code/
        helloworld.py
    data/
        names.txt
flowserv.yaml
instructions.txt
benchmark.yaml
```

The *template* folder contains all the code and data files that are provided to the user to run the workflow. The *workflow.yaml* file contains the template specification. User instructions for running the workflow are in the markdown file *instructions.md*. The *flowserv.yaml* file contains all the necessary metadata that is required when adding the workflow template to a flowServ repository.

4.1 Project Manifest File

The project manifest file contains the necessary information when adding a workflow template to a flowServ repository. A template is added to the flowServ repository by specifying the base folder or git repository for the project. The template loader will look for a manifest with name *flowserv.json*, *flowserv.yaml*, or *flowserv.yml* (in that order) in the project folder. The structure of the manifest file is shown below:

```yaml
name: 'Hello World Demo'
description: 'Hello World Demo for ROB'
instructions: 'instructions.txt'
files:
    - source: 'template/code'
      target: 'code'
specfile: 'benchmark.yaml'
```

The *name* and *description* define the project title and a description for display, e.g., in the ROB UI. The *instructions* element refers to the user instructions file. Note that all file references in the description file are relative to the project base folder (e.g., the folder that contains the cloned git repository).

The list of *files* defines the source files (and folders) and their target paths that are copied from the project folder to the template repository. If the *target* element is omitted the empty string is used as the default value. The files that are copied to the template repository define the environment that is created each time the user runs the template workflow.

The *specfile* element points to the workflow specification file.

All elements (except for *files*) in the description file can be overridden by command line arguments when adding the template to the repository. If no *files* element (or no description file is given) all files and folders in the project base folder are copied recursively to the template repository.

CHAPTER

FIVE

CONFIGURATION

All components of the *Reproducible and Reusable Data Analysis Workflow Server* (**flowServ**) are configured using environment variables in an attempt to follow The Twelve-Factor App methodology for application development.

5.1 Web Service API

The **flowServ** Web service API base configuration is controlled by six environment variables: *FLOWSERV_API_DIR*, *FLOWSERV_API_HOST*, *FLOWSERV_API_NAME*, *FLOWSERV_API_PATH*, *FLOWSERV_API_PORT*, and *FLOWSERV_API_PROTOCOL*. Note that RESTful Web services that provide access to the API via HTTP requests may define additional configuration parameters.

The API maintains all files within (sub-folders) of a base directory on the file system. The base directory is specified using the environment variable *FLOWSERV_API_DIR*. The default value is `.flowserv`.

The API name, contained in the API service descriptor, is specified using the environment variable *FLOWSERV_API_NAME*. The default value is `Reproducible and Reusable Data Analysis Workflow Server (API)`.

The base URL for all API resources is composed from the values in the environment variables *FLOWSERV_API_PROTOCOL*, *FLOWSERV_API_HOST*, *FLOWSERV_API_PORT*, and *FLOWSERV_API_PATH* (following the pattern protocol://host:port/path). If the port number is `80` it is omitted from the url pattern. The default values for the environment variables are `http`, `localhost`, `5000`, and `/flowserv/api/v1`, respectively.

5.2 Authentication

flowServ currently supports two modes of authentication. The selected mode is defined by the environment variable *FLOWSERV_AUTH*:

- `open`: Defines an open-access policy to the API that does not require an authenticated user for API calls.
- `default`: The default authentication policy requires a valid user identifier to be provided for API calls.

The environment variable *FLOWSERV_AUTH_TTL* is used to specify the time period (in milliseconds) for which an issued API key (used to authenticate users) is valid after a user login.

5.3 Workflow Engine

The **flowServ** API uses a :class:WorkflowController to handle execution of workflow templates. Different workflow engines will implement their own version of the controller. An instance of **flowServ** will currently use a single controller for the execution of all workflows. This controller is specified using the following two environment variables:

- *FLOWSERV_BACKEND_CLASS*: The name of the Python class that implements the workflow controller interface
- *FLOWSERV_BACKEND_MODULE*: The full name of the module that contains the implementation of the workflow controller interface

The specified controller module is imported dynamically. Each implementation of the workflow controller may define additional environment variables that are required for configuration.

By default, a simple multi-process engine is used that executes every workflow in a separate process. The environment settings for the default engine are as follows:

```
export FLOWSERV_BACKEND_MODULE=flowserv.controller.serial.engine.base
export FLOWSERV_BACKEND_CLASS=SerialWorkflowEngine
```

5.3.1 Serial Engine Workers

When using the :class:SerialWorkflowEngine individual workflow steps can be executed by different workers (execution backends). **flowServ** currently supports execution using the Python `subprocess` package or the use of a Docker engine.

Engine workers are configured using a configuraton file (in Json or Yaml format) that specifies for Docker image identifier the execution backend. The format of the file is a list of entries as shown below:

```
- image: 'heikomueller/openclean-metanome:0.1.0'
  worker: 'docker'
  args:
      variables:
          jar: 'lib/Metanome.jar'
- image: 'heikomueller/toptaggerdemo:0.2.0'
  worker: 'subprocess'
```

In the shown example workflow steps that use the Docker image *heikomueller/openclean-metanome:0.1.0* are executed using the :class:DockerWorker. The class receives the additional mapping of variables that is defined in the configuration as arguments when it is instantiated. Workflow steps that use the image *heikomueller/toptaggerdemo:0.2.0* will be executed as Python sub-processes.

Use the environment variable *FLOWSERV_SERIAL_WORKERS* to reference the configuration file for the engine workers. By default, all workflow steps will be executed as Python sub-processes if no configuration file is given.

5.4 Database

Database connections are established using the environment variable *FLOWSERV_DATABASE*. **flowServ** uses SQLAlchemy for the Object-Relational-Mapping and to access the underlying database. The value of *FLOWSERV_DATABASE* is passed to the SQLAlchemy engine at initialization. The value is expected to be a database connection URL. Consult the SQLAlchemy Database Urls documentation for more information about the format of the URLs.

5.4.1 Connect to SQLite

When using SQLite as the underlying database system, an example value for *FLOWSERV_DATABASE* is:

```
export FLOWSERV_DATABASE=sqlite:////absolute/path/to/foo.db
```

5.4.2 Connect to PostgreSQL

```
export FLOWSERV_DATABASE=postgresql://scott:tiger@localhost/mydatabase
```

The following steps are an example for creating an initial empty database for **flowServ** in PostgreSQL:

```
# Login as user postgres and connect to
# the (local) database server
sudo su - postgres
psql -U postgres
```

```
-- Create user flowserv with password flowserv
CREATE USER flowserv;
ALTER USER flowserv WITH PASSWORD 'flowserv';
-- Create an empty database with owner flowserv
CREATE DATABASE flowserv WITH OWNER flowserv;
```

If the environment variable *FLOWSERV_WEBAPP* is set to *True* scoped database sessions are used for web applications.

5.5 File Store

flowServ needs to store and access files for a variety of components and tasks. The files that are maintained by the system include:

- static files that are associated with a workflow template,
- files that are uploaded by users as input to workflow runs, and
- result files of successful workflow runs.

By default, files are stored on the local file system in the directory that is specified by the *FLOWSERV_API_DIR* variable. Alternative storage backends can be configured using the environment variable *FLOWSERV_FILESTORE* that contains the configuration dictionary for the storage volume factory. The configuration object has to contain the mandatory element `type` that specifies the class of the storage volume that is used and the optional element `name` and `args`. The `name` is used to identify the storage volume and the `args` element contains additional configuration parameters that are passed to the storage volume class constructor. **flowServ** currently supports four types of storage volumes.

5.5.1 File System Store

The default file store maintains all files in subfolders under the directory that is specified by the environment variable *FLOWSERV_API_DIR*. To configure this option, used the following template:

```
"type": "fs"
"args":
    "basedir": "path to the base directory on the file system"
```

5.5.2 Google Cloud File Store

The **Google Cloud Bucket** allows storage of all files using Google Cloud File Store. The type identifier for this volume is `gc`. The storage volume class has one additional configuration parameter to identify the storage bucket.

```
"type": "gc"
"args":
    "bucket": "identifier of the storage bucket"
```

When using the Google Cloud Storage the Google Cloud credentials have to be configured. Set up authentication by creating a service account and setting the environment variable *GOOGLE_APPLICATION_CREDENTIALS*. See the Cloud Storage Client Libraries documentation for more details.

```
export GOOGLE_APPLICATION_CREDENTIALS=[path-to-service-account-key-file]
```

5.5.3 S3 Bucket Store

The **S3 Bucket Store** allows storage of all files using AWS Simple Cloud Storage (S3). The type identifier for this volume is `s3`. The storage volume class has one additional configuration parameter to identify the storage bucket.

```
"type": "s3"
"args":
    "bucket": "identifier of the storage bucket"
```

When using the S3 storage volume the AWS credentials have to be configured. See the AWS S3 CLI configuration documentation for more details.

5.5.4 SFTP File System Store

flowServ also provides the option to store files on a remote file system and access them via `sftp`. This storage volume is not recommended for storing workflow files. It's main purpose is to serve as a storage manager for copying files when executing workflow steps that run on remote maches (e.g., a HPC cluster). To configure the remote storage volume use the following configuration template.

```
"type": "sftp"
"args":
    "hostname": "Name of the remote host"
    "port": post-number
    "sep": "path separator used by the remote file system [default: '/']"
    "look_for_keys": Boolean flag to enable searching for private key files␣
↪[default=False]
```

CHAPTER

SIX

API DOCUMENTATION

6.1 Authentication & Authorization

Authorization is handled by the API. At this point, API calls that access or manipulate submissions and submission runs require the user to be a member of the respective submission.

6.2 Submitting Arguments for Benchmark Runs

The API method to start an new run for a submission takes as one of its arguments a list of objects that represent the user-provided values for template parameters. Each object is expected to be a simple (flat) dictionary. Each dictionary is expected to contain an `name` and `value` element. The `name` references the parameter identifier. The `value` is dependent on the type of the parameter. At this point only strings and numbers are supported. For example,

```
{
    "name": "parameter-id",
    "value": 1
}
```

6.2.1 File Arguments

For parameters of type `file` the argument value is the identifier of a file that was uploaded by a previous API request. File arguments may have an additional element `as` to provide the target path for the file in the workflow execution environment. For example,

```
{
    "name": "parameter-id",
    "value": "file-id",
    "as": "data/myfile.txt"
}
```

CHAPTER

SEVEN

FLOWSERV

7.1 flowserv package

7.1.1 Subpackages

flowserv.client package

Subpackages

flowserv.client.app package

Submodules

flowserv.client.app.base module

Helper methods for test runs of workflow templates.

class flowserv.client.app.base.**Flowserv**(*env: Optional[Dict] = None*, *basedir: Optional[str] = None*, *database: Optional[str] = None*, *open_access: Optional[bool] = None*, *run_async: Optional[bool] = None*, *clear: Optional[bool] = False*, *user_id: Optional[str] = None*)

Bases: `object`

Client environment for interacting with a flowserv instance. This class provides additional functionality for installing flowserv applications. It is primarily intended running flowserv in programming environments, e.g., Jupyter Notebooks.

create_submission(*workflow_id: str*, *name: str*, *members: Optional[List[str]] = None*, *parameters: Optional[List[*flowserv.model.parameter.base.Parameter*]] = None*, *engine_config: Optional[Dict] = None*) → str

Create a new user group for a given workflow. Each group has a a unique name for the workflow, a list of additional group members, and a specification of additional parameters. The parameters allow to define variants of the original workflow template.

Parameters

- **workflow_id** (*string*) – Unique workflow identifier
- **name** (*string*) – Unique team name
- **members** (*list(string), default=None*) – List of user identifier for group members

- **parameters** (*list of flowserv.model.parameter.base.Parameter, default=None*) – Optional list of parameter declarations that are used to modify the template parameters for submissions of the created group.
- **engine_config** (*dict, default=None*) – Optional configuration settings that will be used as the default when running a workflow.

Return type string

`erase()`

Delete the base folder for the test environment that contains all workflow files.

`install`(*source: str*, *identifier: Optional[str] = None*, *name: Optional[str] = None*, *description: Optional[str] = None*, *instructions: Optional[str] = None*, *specfile: Optional[str] = None*, *manifestfile: Optional[str] = None*, *engine_config: Optional[Dict] = None*, *ignore_postproc: Optional[bool] = False*, *multi_user: Optional[bool] = False*, *verbose: Optional[bool] = False*) → str

Create a new workflow in the environment that is defined by the template referenced by the source parameter. Returns the identifier of the created workflow.

If the multi user flag is False this method will also create a group with the same identifier as the workflow.

Parameters

- **source** (*string*) – Path to local template, name or URL of the template in the repository.
- **identifier** (*string, default=None*) – Unique user-provided workflow identifier. If no identifier is given a unique identifier will be generated for the application.
- **name** (*string, default=None*) – Unique workflow name
- **description** (*string, default=None*) – Optional short description for display in workflow listings
- **instructions** (*string, default=None*) – File containing instructions for workflow users.
- **specfile** (*string, default=None*) – Path to the workflow template specification file (absolute or relative to the workflow directory)
- **manifestfile** (*string, default=None*) – Path to manifest file. If not given an attempt is made to read one of the default manifest file names in the base directory.
- **engine_config** (*dict, default=None*) – Optional configuration settings that will be used as the default when running a workflow.
- **ignore_postproc** (*bool, default=False*) – Ignore post-processing workflow specification if True.
- **multi_user** (*bool, default=False*) – If the multi user flag is False a group will be created for the workflow with the same identifier as the workflow.
- **verbose** (*bool, default=False*) – Print information about source and target volume and the files that are being copied.

Return type string

`login`(*username: str*, *password: str*)

Authenticate the user using the given credentials.

Parameters

- **username** (*string*) – Unique user name
- **password** (*string*) – User password

logout()

Logout the currently authenticated user.

open(*identifier: str*) → *flowserv.client.app.workflow.Workflow*

Get an instance of the floserv app for the workflow with the given identifier.

Parameters **identifier** (*string*) – Unique workflow identifier.

Return type *flowserv.client.app.workflow.Workflow*

register(*username: str*, *password: str*) → str

Register a new user with the given credentials.

Parameters

- **username** (*string*) – Unique user name.
- **password** (*string*) – User password.

Return type string

submission(*workflow_id: str*, *group_id: str*) → *flowserv.client.app.workflow.Workflow*

Get the handle for a workflow with a given identifier and for a given user group.

Parameters

- **workflow_id** (*string*) – Unique workflow identifier.
- **group_id** (*string*) – Unique user group identifier.

Return type *flowserv.client.app.workflow.Workflow*

uninstall(*identifier: str*)

Remove the workflow with the given identifier. This will also remove all run files for that workflow.

Parameters **identifier** (*string*) – Unique workflow identifier.

flowserv.client.app.data module

Objects for files that are created as the result of successful workflow runs.

class flowserv.client.app.data.**DataFile**(*run_id: str*, *doc: Dict*, *service:* flowserv.service.api.APIFactory)

Bases: object

Basic object that represents a run result file. Provides access to the file content via the file handle and format-specific load methods.

data() → Tuple[List[str], List[List[str]]]

Load CSV data. Returns a list of column names and a list or rows.

Return type tuple of list and list

json() → Dict

Load file content as Json object.

Return type dict

load() → *flowserv.model.files.FileHandle*

Get handle for the file.

Return type *flowserv.model.files.FileHandle*

text() → str

Read file content as string.

Return type string

flowserv.client.app.run module

Classes that wrap serialized run result dictionaries. The aim is to make it easier for a developer that uses the flowserv application object to access the results and resources of workflow runs.

class flowserv.client.app.run.**Run**(*doc: Dict*, *service:* flowserv.service.api.APIFactory)

Bases: object

Wrapper around a serialized run result. Provides access to file objects for the result files that are generated by successful workflow runs.

files() → List[*flowserv.client.app.data.DataFile*]

Get list of file objects for run result files.

Return type list of flowserv.client.app.data.File

get_file(*key: str*) → *flowserv.client.app.data.DataFile*

Get result file object for the file with the given identifier.

Parameters **key** (*string*) – User-provided file identifier or the file source name.

Return type flowserv.client.app.data.File

is_active() → bool

Check if the run state is in an active state (either PENDING or RUNNING).

Return type bool

is_canceled() → bool

Check if the run state is CANCELED.

Return type bool

is_error() → bool

Check if the run state is ERROR.

Return type bool

is_pending() → bool

Check if the run state is PENDING.

Return type bool

is_running() → bool

Check if the run state is RUNNING.

Return type bool

is_success() → bool

Check if the run state is SUCCESS.

Return type bool

messages() → List[str]

Get list of error messages for runs that are in error state or that have been canceled. If the run is not in an error state the result is an empty list.

Return type list

open_file(*key: str*) → IO

Get result file object and return an opened file handles. This is a shortcut for opening result files.

Parameters **key** (*string*) – User-provided file identifier or the file source name.

Return type flowserv.client.app.data.File

flowserv.client.app.workflow module

Wrapper that provides access to a workflow via the service API.

class flowserv.client.app.workflow.**Workflow**(*workflow_id: str*, *group_id: str*, *service:* flowserv.service.api.APIFactory)

Bases: object

Wrapper object for a single workflow. Maintains workflow metadata and provides functionality to execute and monitor workflow runs via the service API.

cancel_run(*run_id: str*, *reason: Optional[List[str]] = None*)

Cancel the run with the given identifier.

Raises an unauthorized access error if the user does not have the necessary access rights to cancel the run.

Parameters

- **run_id** (*string*) – Unique run identifier
- **reason** (*string, optional*) – Optional text describing the reason for cancelling the run

delete_run(*run_id: str*)

Delete the run with the given identifier.

Raises an unauthorized access error if the user does not have the necessary access rights to delete the run.

Parameters **run_id** (*string*) – Unique run identifier

description() → str

Get descriptive header for the application.

Return type string

get_file(*run_id: str*, *file_id: Optional[str] = None*) → *flowserv.model.files.FileHandle*

Get handle for run result file. If the file identifier is not given the handle for the run result archive is returned.

Parameters

- **run_id** (*string*) – Unique run identifier.
- **file_id** (*string, default=None*) – Unique file identifier.

Return type *flowserv.model.files.FileHandle*

get_postproc_results() → *flowserv.client.app.run.Run*

Get results of a post-processing run. The result is None if no entry for a post-porcessing run is found in the workflow handle.

Return type *flowserv.client.app.run.Run*

property identifier: str

Get the identifier of the associated workflow.

Return type string

instructions() → str

Get instructions text for the application.

Return type string

name() → str

Get application title.

Return type string

parameters() → *flowserv.model.template.parameter.ParameterIndex*

Get parameter declaration for application runs.

Return type flowserv.model.template.parameters.ParameterIndex

poll_run(*run_id*) → *flowserv.client.app.run.Run*

Get run result handle for a given run.

Raises an unauthorized access error if the user does not have read access to the run.

Parameters **run_id** (*string*) – Unique run identifier

Return type *flowserv.client.app.run.Run*

start_run(*arguments: Dict*, *config: Optional[Dict] = None*, *poll_interval: Optional[int] = None*) → *flowserv.client.app.run.Run*

Run the associated workflow for the given set of arguments.

Parameters

- **arguments** (*dict*) – Dictionary of user-provided arguments.
- **config** (*dict, default=None*) – Optional implementation-specific configuration settings that can be used to overwrite settings that were initialized at object creation.
- **poll_interval** (*int, default=None*) – Optional poll interval that is used to check the state of a run until it is no longer in active state.

Return type *flowserv.client.app.run.Run*

flowserv.client.cli package

Submodules

flowserv.client.cli.admin module

Command line interface for the administrative tasks to configure the environment, and the intialize the underlying database. These commands operate locally.

flowserv.client.cli.app module

Command line interface for the flowServ app.

flowserv.client.cli.base module

Main module for the flowserv command line interface. Defines the top-level command group and the API context for all commands.

class flowserv.client.cli.base.**EnvContext**(*vars: Dict*)

Bases: object

Helper class to get default parameter values for from the environment. The different applications `flowserv` and `rob` historically use different environment variables to maintain the workflow and group identifier. This class is used to hide these details from the CLI commands.

access_token() → str

Get the value for the user access token from the environment.

The environment variable that holds the token (*FLOWSERV_ACCESS_TOKEN*) is the same accross both applications.

Raises a missing configuration error if the value is not set.

Return type string

get_group(*params: Dict*) → str

Get the user group (submission) identifier from the parameters or the environment.

Attempts to get the group identifier from the list of parsed parameters. Assumes the the identifier is given by the parameter `group`. If no group identifier is present in the given parameter dictionary an attempt is made to get the identifier from the environment variable that is defined for user groups in this context. If that environment variable is not set an attempt is made to get the value from the environment variable for the workflow identifier. The latter is done since flowserv applications are installed with identical workflow and group identifier.

Raises a missing configuration error if the value is not set.

Parameters **params** (*dict*) – Dictionary of parsed command-line parameters

Return type string

get_workflow(*params: Dict*) → str

Get the workflow identifier from the parameters or the environment.

Attempts to get the workflow identifier from the list of parsed parameters. Assumes the the identifier is given by the parameter `workflow`. If no workflow identifier is present in the given parameter dictionary an attempt is made to get the identifier from the environment variable that is defined for workflows in this context.

Raises a missing configuration error if the value is not set.

Parameters **params** (*dict*) – Dictionary of parsed command-line parameters

Return type string

flowserv.client.cli.cleanup module

Command line interface to list and delete old workflow runs.

flowserv.client.cli.group module

Command line interface to interact with workflow user groups.

flowserv.client.cli.group.**print_group**(*doc*)

Print group handle information to console.

Parameters **doc** (*dict*) – Serialization of a workflow group handle.

flowserv.client.cli.gui module

Command line interface to run a workflow template using a web GUI based on streamlit.

flowserv.client.cli.parameter module

Helper methods for reading workflow template parameters.

flowserv.client.cli.parameter.**read**(*parameters: List[flowserv.model.parameter.base.Parameter]*, *scanner: Optional[flowserv.scanner.Scanner] = None*, *files: Optional[Tuple[str, str, str]] = None*) → Dict

Read values for each of the template parameters using a given input scanner. If no scanner is given, values are read from standard input.

The optional list of file handles is used for convenience when the user is asked to input the identifier of an uploaded file. It allows to display the identifier of available files for easy copy and paste.

Returns a dictionary of argument values that can be passed to the workflow execution engine to run a parameterized workflow.

Parameters

- **parameters** (*list(flowserv.model.parameter.base.Parameter)*) – List of workflow template parameter declarations
- **scanner** (*flowserv.scanner.Scanner*) – Input scanner to read parameter values
- **files** (*list, default=None*) – List of tuples representing uploaded files. Each tuple has three elements: file_id, name, timestamp.

Return type dict

flowserv.client.cli.parameter.**read_file**(*para: flowserv.model.parameter.base.Parameter*, *scanner: flowserv.scanner.Scanner*, *files: Optional[Tuple[str, str, str]] = None*)

Read value for a file parameter.

Parameters

- **para** (*flowserv.model.parameter.base.Parameter*) – Workflow template parameter declaration
- **scanner** (*flowserv.scanner.Scanner*) – Input scanner.

- **files** (*list, default=None*) – List of tuples representing uploaded files. Each tuple has three elements: file_id, name, timestamp.

flowserv.client.cli.parameter.**read_parameter**(*para:* flowserv.model.parameter.base.Parameter, *scanner:* flowserv.scanner.Scanner, *files: Optional[Tuple[str, str, str]] = None*) → Any

Read value for a given template parameter declaration. Prompts the user to enter a value for the given parameter and returns the converted value that was entered by the user.

Parameters

- **para** (*flowserv.model.parameter.base.Parameter*) – Workflow template parameter declaration
- **scanner** (*flowserv.scanner.Scanner*) – Input scanner.
- **files** (*list, default=None*) – List of tuples representing uploaded files. Each tuple has three elements: file_id, name, timestamp.

Return type bool or float or int or string or tuple(string, string)

flowserv.client.cli.repository module

Command line interface to list contents of the global workflow template repository.

flowserv.client.cli.run module

Command line interface to manage workflow runs.

flowserv.client.cli.table module

Helper methods and classes for the command line interface.

class flowserv.client.cli.table.**ResultTable**(*headline*, *types*)

Bases: object

Result table for database queries. Maintains a list or result rows. Provides functionality to format rows for printing.

add(*row*)

Add a row to the table. The length of the row is expected to be the same as the length of the table headline. That is, the row contains one per column in the table.

Parameters **row** (*list(string)*) – List of column values

format()

Format the table rows in tabular format. Each rows is a list of string values. All rows are expected to have the same length. The first row is the header that contains the column names.

Return type list(string)

flowserv.client.cli.table.**align**(*type_id*)

Get align identifier depending on the data type. Numeric types are right aligned. All other types are left aligned.

type_id: string Type identifier (from set of valid type identifier in parameter declarations)

Return type string

flowserv.client.cli.table.**format_row**(*row*, *column_size*, *types*)

Format the given row. Row values are padded using the given list of column widths.

Parameters

- **row** (*list(string)*) – List of cell values in a table row
- **column_size** (*list(int)*) – List of column widths
- **types** (*list(string)*) – List of column type identifier

Return type string

flowserv.client.cli.uploads module

Command line interface to interact with group upload files.

flowserv.client.cli.user module

Command line interface to register a new user.

flowserv.client.cli.workflow module

Administrator command line interface to create, delete and maintain workflow templates in the repository.

flowserv.client.gui package

Submodules

flowserv.client.gui.app module

flowserv.client.gui.forms module

flowserv.client.gui.widget module

Submodules

flowserv.client.api module

Helper method to create a API generator based on the current configuration in the environment valriables.

flowserv.client.api.**ClientAPI**(*env: Optional[Dict] = None*, *basedir: Optional[str] = None*, *database: Optional[str] = None*, *open_access: Optional[bool] = None*, *run_async: Optional[bool] = None*, *user_id: Optional[str] = None*) → *flowserv.service.api.APIFactory*

Create an instance of the API factory that is responsible for generating API instances for a flowserv client.

The main distinction here is whether a connection is made to a local instance of the service or to a remote instance. This distinction is made based on the value of the FLOWSERV_CLIENT environment variable that takes the values ‘local’ or ‘remote’. The default is ‘local’.

Provides the option to alter the default settings of environment variables.

Parameters

- **env** (*dict, default=None*) – Dictionary with configuration parameter values.
- **basedir** (*string, default=None*) – Base directory for all workflow files. If no directory is given or specified in the environment a temporary directory will be created.
- **database** (*string, default=None*) – Optional database connect url.
- **open_access** (*bool, default=None*) – Use an open access policy if set to True.
- **run_async** (*bool, default=False*) – Run workflows in asynchronous mode.
- **user_id** (*string, default=None*) – Optional identifier for the authenticated API user.

Return type *flowserv.service.api.APIFactory*

`flowserv.client.api.service()` → *flowserv.service.api.API*

Context manager that returns a service API that was instantiated from the current configuration settings in the environment.

Return type *flowserv.service.api.API*

flowserv.controller package

Subpackages

flowserv.controller.remote package

Submodules

flowserv.controller.remote.client module

The abstract remote client class is used by the remote workflow controller to interact with a workflow engine. Different workflow engines will implement their own version of the remote client. The client provides the functionality that is required by the workflow controller to execute workflows, cancel workflow execution, get workflow status, and to download workflow result files.

class `flowserv.controller.remote.client.RemoteClient`

Bases: `object`

The remote client class is an abstract interface that defines the methods that are required by the remote workflow controller to execute and monitor remote workflows. Different workflow engies will implement their own version of the interface.

abstract `create_workflow`(*run:* flowserv.model.base.RunObject, *template:* flowserv.model.template.base.WorkflowTemplate, *arguments: Dict*, *staticfs:* flowserv.volume.base.StorageVolume) → *flowserv.controller.remote.client.RemoteWorkflowHandle*

Create a new instance of a workflow from the given workflow template and user-provided arguments.

The static storage volume provides access to static workflow template files that were created when the workflow template was installed.

Implementations of this method will also upload any files to the remomote engine that are required to execute the workflow.

A created workflow may not be running immediately but at minimum scheduled for execution. There is no separate signal to trigger execution start.

The result is a handle to access the remote workflow object.

Parameters

- **run** (*flowserv.model.base.RunObject*) – Handle for the run that is being executed.
- **template** (*flowserv.model.template.base.WorkflowTemplate*) – Workflow template containing the parameterized specification and the parameter declarations.
- **arguments** (*dict*) – Dictionary of argument values for parameters in the template.
- **staticfs** (*flowserv.volume.base.StorageVolume*) – Storage volume that contains the static files from the workflow template.

Return type *flowserv.controller.remote.client.RemoteWorkflowHandle*

abstract get_workflow_state(*workflow_id: str*, *current_state:* flowserv.model.workflow.state.WorkflowState) → *flowserv.model.workflow.state.WorkflowState*

Get information about the current state of a given workflow.

Parameters

- **workflow_id** (*string*) – Unique workflow identifier
- **current_state** (*flowserv.model.workflw.state.WorkflowState*) – Last known state of the workflow by the workflow controller

Return type flowserv.model.workflw.state.WorkflowState

abstract stop_workflow(*workflow_id: str*)

Stop the execution of the workflow with the given identifier.

Parameters **workflow_id** (*string*) – Unique workflow identifier

class flowserv.controller.remote.client.**RemoteWorkflowHandle**(*run_id: str*, *workflow_id: str*, *state:* flowserv.model.workflow.state.WorkflowState, *output_files: List[str]*, *runstore:* flowserv.volume.base.StorageVolume, *client:* flowserv.controller.remote.client.RemoteClient)

Bases: object

Base class for remote workflow handles. Remote workflows may have an identifier that differs from the run identifier that is managed by flowServ. The remote identifier is maintained by the handle, together with the last reported workflow state, and the list of output files that are generated by successful workflow runs. The names of output files are path that are relative to the root directory of the remote environment that executes the workflow.

client: *flowserv.controller.remote.client.RemoteClient*

is_active() → bool

Test if the last state of the workflow is an active state.

Return type bool

output_files: List[str]

poll_state() → *flowserv.model.workflow.state.WorkflowState*

Get the current state of the workflow from the responsible backend.

Uses the client to poll the workflow state. Updates the internal state if the workflow state has changed.

If the state of the workflow has not changed the result of this method is None. Otherwise, the returned value is the new (current) workflow state.

Return type *flowserv.model.workflow.state.WorkflowState*

run_id: str

runstore: *flowserv.volume.base.StorageVolume*

state: *flowserv.model.workflow.state.WorkflowState*

workflow_id: str

flowserv.controller.remote.engine module

Implementation for a workflow controller backend that uses an external (remote) workflow engine (e.g., an existing REANA cluster) for workflow execution. The controller provides functionality for workflow creation, start, stop, and monitpring using an (abstract) client class. For different types of workflow engines only the RemoteClient class needs to be implements.

class flowserv.controller.remote.engine.**RemoteWorkflowController**(*client:* flowserv.controller.remote.client.RemoteClient, *poll_interval: float*, *is_async: bool*, *service: Optional[*flowserv.service.api.APIFactory*] = None*)

Bases: *flowserv.controller.base.WorkflowController*

Workflow controller that executes workflow templates for a given set of arguments using an external workflow engine. Each workflow is monitored by a separate process that continuously polls the workflow state.

cancel_run(*run_id: str*)

Request to cancel execution of the given run. This method is usually called by the workflow engine that uses this controller for workflow execution. It is threfore assumed that the state of the workflow run is updated accordingly by the caller.

Parameters **run_id** (*string*) – Unique run identifier.

exec_workflow(*run:* flowserv.model.base.RunObject, *template:* flowserv.model.template.base.WorkflowTemplate, *arguments: Dict*, *staticfs:* flowserv.volume.base.StorageVolume, *config: Optional[Dict] = None*) → Tuple[*flowserv.model.workflow.state.WorkflowState*, *flowserv.volume.base.StorageVolume*]

Initiate the execution of a given workflow template for a set of argument values. This will start a new process that executes a serial workflow asynchronously. Returns the state of the workflow after the process is stated (the state will therefore be RUNNING).

The set of arguments is not further validated. It is assumed that the validation has been performed by the calling code (e.g., the run service manager).

If the state of the run handle is not pending, an error is raised.

Parameters

- **run** (*flowserv.model.base.RunObject*) – Handle for the run that is being executed.
- **template** (*flowserv.model.template.base.WorkflowTemplate*) – Workflow template containing the parameterized specification and the parameter declarations.
- **arguments** (*dict*) – Dictionary of argument values for parameters in the template.

- **staticfs** (*flowserv.volume.base.StorageVolume*) – Storage volume that contains the static files from the workflow template.
- **config** (*dict, default=None*) – Optional configuration settings are currently ignored. Included for API completeness.

Return type *flowserv.model.workflow.state.WorkflowState*, *flowserv.volume.base.StorageVolume*

flowserv.controller.remote.monitor module

Monitor for remote workflow executions. The monitor is a separate thread that continously polls the remote workflow engine to update the workflow state in the local database.

`class flowserv.controller.remote.monitor.WorkflowMonitor(`*workflow:* flowserv.controller.remote.client.RemoteWorkflowHandle, *poll_interval: float*, *service:* flowserv.service.api.APIFactory, *tasks: Dict*`)`

Bases: `threading.Thread`

Thread that monitors execution of an external workflow. Polls the state of the workflow in regular intervals. Updates the local workflow state as the remote state changes.

`run()`

Poll the remote server continuously until execution is finished.

`flowserv.controller.remote.monitor.monitor_workflow(`*workflow:* flowserv.controller.remote.client.RemoteWorkflowHandle, *poll_interval: float*, *service: Optional[*flowserv.service.api.APIFactory*] = None*`) →` *flowserv.model.workflow.state.WorkflowState*

Monitor a remote workflow run by continuous polling at a given interval. Updates the local workflow state as the remote state changes.

Returns the state of the inactive workflow and the temporary directory that contains the downloaded run result files. The run directory may be None for unsuccessful runs.

Parameters

- **workflow** (*flowserv.controller.remote.client.RemoteWorkflowHandle*) – Handle for the monitored workflow.
- **poll_interval** (*float*) – Frequency (in sec.) at which the remote workflow engine is polled.
- **service** (*contextlib,contextmanager, default=None*) – Context manager to create an instance of the service API.

Return type *flowserv.model.workflow.state.WorkflowState*

flowserv.controller.serial package

Subpackages

flowserv.controller.serial.engine package

Submodules

flowserv.controller.serial.engine.base module

Implementation for a workflow controller backend that is capable of running serial workflow specifications.

This controller allows execution in workflow steps within separate sub-processes.

All workflow run files will be maintained in a (temporary) directory on the storage volume that is associated with the workflow engine. The base folder for these run files can be configured by setting the environment variable *FLOWSERV_SERIAL_RUNSDIR*.

class flowserv.controller.serial.engine.base.**SerialWorkflowEngine**(*service:* flowserv.service.api.APIFactory, *fs: Optional[*flowserv.volume.base.StorageVolume*] = None*, *config: Optional[Dict] = None*)

Bases: *flowserv.controller.base.WorkflowController*

The workflow engine is used to execute workflow templates for a given set of arguments. Each workflow is executed as a serial workflow. The individual workflow steps can be executed in aVolume(env separate process on request.

cancel_run(*run_id: str*)

Request to cancel execution of the given run. This method is usually called by the workflow engine that uses this controller for workflow execution. It is therefore assumed that the state of the workflow run is updated accordingly by the caller.

Parameters **run_id** (*string*) – Unique run identifier

exec_workflow(*run:* flowserv.model.base.RunObject, *template:* flowserv.model.template.base.WorkflowTemplate, *arguments: Dict*, *staticfs:* flowserv.volume.base.StorageVolume, *config: Optional[Dict] = None*) → Tuple[*flowserv.model.workflow.state.WorkflowState*, *flowserv.volume.base.StorageVolume*]

Initiate the execution of a given workflow template for a set of argument values. This will start a new process that executes a serial workflow asynchronously.

The serial workflow engine executes workflows on the local machine and therefore uses the file system to store temporary run files. The path to the run folder is returned as the second value in the result tuple. The first value in the result tuple is the state of the workflow after the process is stated. If the workflow is executed asynchronously the state will be RUNNING. Otherwise, the run state should be an inactive state.

The set of arguments is not further validated. It is assumed that the validation has been performed by the calling code (e.g., the run service manager).

The optional configuration object can be used to override the worker configuration that was provided at object instantiation. Expects a dictionary with an element *workers* that contains a mapping of container identifier to a container worker configuration object.

If the state of the run handle is not pending, an error is raised.

Parameters

- **run** (*flowserv.model.base.RunObject*) – Handle for the run that is being executed.
- **template** (*flowserv.model.template.base.WorkflowTemplate*) – Workflow template containing the parameterized specification and the parameter declarations.
- **arguments** (*dict*) – Dictionary of argument values for parameters in the template.
- **staticfs** (*flowserv.volume.base.StorageVolume*) – Storage volume that contains the static files from the workflow template.
- **config** (*dict, default=None*) – Optional object to overwrite the worker configuration settings.

Return type *flowserv.model.workflow.state.WorkflowState*, *flowserv.volume.base.StorageVolume*

`flowserv.controller.serial.engine.base.`**`callback_function`**(*result*, *lock*, *tasks*, *service*)

Callback function for executed tasks.Removes the task from the task index and updates the run state in the underlying database.

Parameters

- **result** (*(string, dict)*) – Tuple of task identifier and serialized state of the workflow run
- **lock** (*multiprocessing.Lock*) – Lock for concurrency control
- **tasks** (*dict*) – Task index of the backend
- **service** (*contextlib.contextmanager*) – Context manager to create an instance of the service API.

`flowserv.controller.serial.engine.base.`**`run_workflow`**(*run_id: str*, *state:* flowserv.model.workflow.state.WorkflowState, *output_files: List[str]*, *steps:* *List[*flowserv.model.workflow.step.ContainerStep*]*, *arguments: Dict*, *volumes:* flowserv.volume.manager.VolumeManager, *workers:* flowserv.controller.worker.manager.WorkerPool) → Tuple[str, str, Dict]

Execute a list of workflow steps synchronously.

This is the worker function for asynchronous workflow executions. Returns a tuple containing the run identifier, the folder with the run files, and a serialization of the workflow state.

Parameters

- **run_id** (*string*) – Unique run identifier
- **state** (*flowserv.model.workflow.state.WorkflowState*) – Current workflow state (to access the timestamps)
- **output_files** (*list(string)*) – Relative path of output files that are generated by the workflow run
- **steps** (*list of flowserv.model.workflow.step.WorkflowStep*) – Steps in the serial workflow that are executed in the given context.
- **arguments** (*dict*) – Dictionary of argument values for parameters in the template.
- **volumes** (*flowserv.volume.manager.VolumeManager*) – Factory for storage volumes.

- **workers** (*flowserv.controller.worker.manager.WorkerPool*) – Factory for `flowserv.model.workflow.step.ContainerStep` steps.

Return type (string, string, dict)

`flowserv.controller.serial.engine.base.`**`volume_manager`**(*specs: List[Dict]*, *runstore:* flowserv.volume.base.StorageVolume, *runfiles: List[str]*) → *flowserv.volume.manager.VolumeManager*

Create an instance of the storage volume manager for a workflow run.

Combines the volume store specifications in the workflow run confguration with the storage volume for the workflow run files.

Parameters

- **specs** (*list of dict*) – List of specifications (dictionary serializations) for storage volumes.
- **runstore** (*flowserv.volume.base.StorageVolume*) – Storage volume for run files.
- **runfiles** (*list of string*) – List of files that have been copied to the run store.

Return type *flowserv.volume.manager.VolumeManager*

flowserv.controller.serial.engine.config module

Environment variables for configuring the serial workflow engine.

`flowserv.controller.serial.engine.config.`**`ENGINECONFIG`**(*env: Dict*, *validate: Optional[bool] = False*) → Dict

Read engine configuration information from the file that is specified by the environment variable *FLOWSERV_SERIAL_ENGINECONFIG*.

Returns an empty dictionary if the environment variable is not set. If the validate flag is True the read document will be validated against the configuration document schema that is defined in `config.json`.

Parameters

- **env** (*dict*) – Configuration object that provides access to configuration parameters in the environment.
- **validate** (*bool, default=False*) – Validate the read configuration object if True.

Return type dict

`flowserv.controller.serial.engine.config.`**`RUNSDIR`**(*env: Dict*) → str

The default base directory for workflow run files.

Parameters **env** (*dict*) – Configuration object that provides access to configuration parameters in the environment.

Return type string

flowserv.controller.serial.engine.runner module

Execute a serial workflow for given user input arguments.

flowserv.controller.serial.engine.runner.**exec_workflow**(*steps:* *List[*flowserv.model.workflow.step.WorkflowStep*]*, *workers:* flowserv.controller.worker.manager.WorkerPool, *volumes:* flowserv.volume.manager.VolumeManager, *result:* flowserv.controller.serial.workflow.result.RunResult) → *flowserv.controller.serial.workflow.result.RunResult*

Execute steps in a serial workflow.

The workflow arguments are part of the execution context that is contained in the *flowserv.controller.serial.workflow.result.RunResult*. The result object is used to maintain the results for executed workflow steps.

Executes workflow steps in sequence. Terminates early if the execution of a workflow step returns a non-zero value. Uses the given worker factory to create workers for steps that are of class *flowserv.model.workflow.step.ContainerStep*.

Parameters

- **steps** (*list of flowserv.model.workflow.step.WorkflowStep*) – Steps in the serial workflow that are executed in the given context.
- **workers** (*flowserv.controller.worker.manager.WorkerPool, default=None*) – Factory for *flowserv.model.workflow.step.ContainerStep* steps.
- **volumes** (*flowserv.volume.manager.VolumeManager*) – Manager for storage volumes that are used by the different workers.
- **result** (*flowserv.controller.serial.workflow.result.RunResult*) – Collector for results from executed workflow steps. Contains the context within which the workflow is executed.

Return type flowserv.controller.worker.result.RunResult

flowserv.controller.serial.engine.validate module

Validator for engine configuration documents.

flowserv.controller.serial.workflow package

Submodules

flowserv.controller.serial.workflow.base module

Serial workflow that executes a sequence of workflow steps.

Serial Workflows are either created from workflow templates that follow the syntax of the REANA serial workflow specifications or explicitly within Python scripts.

class flowserv.controller.serial.workflow.base.**SerialWorkflow**(*steps: Optional[List[*flowserv.model.workflow.step.WorkflowSte
= None*, *parameters: Optional[List[*flowserv.model.parameter.base.Parameter*]*
= None*, *workers: Optional[*flowserv.controller.worker.manager.WorkerPool
= None*)

Bases: `object`

A serial workflow represents a sequence of *flowserv.model.workflow.step.WorkflowStep*) steps that are executed in order for a given set of input parameters.

At this point we distinguish two types of workflow steps: *flowserv.model.workflow.step.CodeStep* and *flowserv.model.workflow.step.ContainerStep*.

A *flowserv.model.workflow.step.CodeStep* is executed within the same thread and environment as the flowserv engine. A *flowserv.model.workflow.step.ContainerStep* is executed in a separate container-like environment. The execution environment is represented by a *flowserv.controller.worker.base.ContainerWorker* that is associated in the *flowserv.controller.worker.manager.WorkerPool* with the environment identifier of the container step.

add_code_step(*identifier: str*, *func: Callable*, *arg: Optional[str] = None*, *varnames: Optional[Dict] = None*, *inputs: Optional[List[str]] = None*, *outputs: Optional[List[str]] = None*) → *flowserv.controller.serial.workflow.base.SerialWorkflow*

Append a code step to the serial workflow.

Parameters

- **identifier** (*str*) – Unique workflow step identifier.
- **func** (*callable*) – Python function that is executed by the workflow step.
- **arg** (*string, default=None*) – Name of the variable under which the function result is stored in the workflow arguments. If None, the function result is discarded.
- **varnames** (*dict, default=None*) – Mapping of function argument names to names of workflow arguments. This mapping is used when generating the arguments for the executed function. By default it is assumed that the names of arguments for the given function correspond to the names in the argument dictionary for the workflow. This mapping provides the option to map names in the function signature that do not occur in the arguments dictionary to argument names that are in the dictionary.
- **inputs** (*list of string, default=None*) – List of files that are required by the workflow step as inputs.
- **outputs** (*list of string, default=None*) – List of files that are generated by the workflow step as outputs.

Return type *flowserv.controller.serial.workflow.base.SerialWorkflow*

add_container_step(*identifier: str*, *image: str*, *commands: Optional[List[str]] = None*, *env: Optional[Dict] = None*, *inputs: Optional[List[str]] = None*, *outputs: Optional[List[str]] = None*) → *flowserv.controller.serial.workflow.base.SerialWorkflow*

Append a container step to the serial workflow.

Parameters

- **identifier** (*str*) – Unique workflow step identifier.
- **image** (*string, default=None*) – Execution environment identifier.

- **commands** (*list(string), default=None*) – List of command line statements.
- **env** (*dict, default=None*) – Environment variables for workflow step execution.
- **inputs** (*list of string, default=None*) – List of files that are required by the workflow step as inputs.
- **outputs** (*list of string, default=None*) – List of files that are generated by the workflow step as outputs.

Return type *flowserv.controller.serial.workflow.base.SerialWorkflow*

`add_parameter`(*parameter:* flowserv.model.parameter.base.Parameter) → *flowserv.controller.serial.workflow.base.SerialWorkflow*

Add a parameter to the internal index of workflow template parameters.

Parameter identifier are expected to be unique. If a parameter with the same identifier as the given parameter already exists in the internal parameter index it will be replaced with the given parameter.

Returns a reference to the object itself.

Parameters **parameter** (*flowserv.model.parameter.base.Parameter*) – Workflow termplate parameter that is added to the internal parameter index.

Return type *flowserv.controller.serial.workflow.base.SerialWorkflow*

`run`(*arguments: Dict*, *workers: Optional[*flowserv.controller.worker.manager.WorkerPool*] = None*, *volumes: Optional[*flowserv.volume.manager.VolumeManager*] = None*) → *flowserv.controller.serial.workflow.result.RunResult*

Execute workflow for the given set of input arguments.

Executes workflow steps in sequence. Terminates early if the execution of a workflow step returns a non-zero value. Uses the given worker factory to create workers for steps that are of class *flowserv.model.workflow.step.ContainerStep*.

Collects results for all executed steps and returns them in the *flowserv.controller.serial.workflow.result.RunResult*.

Parameters

- **arguments** (*dict*) – User-provided arguments for the workflow run.
- **workers** (*flowserv.controller.worker.manager.WorkerPool, default=None*) – Factory for *flowserv.model.workflow.step.ContainerStep* steps. Uses the default worker for all container steps if None.
- **volumes** (*flowserv.volume.manager.VolumeManager*) – Manager for storage volumes that are used by the different workers.

Return type flowserv.controller.worker.result.RunResult

flowserv.controller.serial.workflow.parser module

Parser for serial workflow templates.

flowserv.controller.serial.workflow.parser.`Step`(*identifier: str*, *action: Dict*, *inputs: Optional[List[str]] = None*, *outputs: Optional[List[str]] = None*) → *flowserv.model.workflow.step.WorkflowStep*

Create workflow step instance from dictionary serialization.

The type of the generated workflow step will depend on the elements in the given dictionary serialization.

Raises a ValueError if the given dictionary is not a valid serialization for a workflow step.

Parameters

- **identifier** (*string*) – Unique step name (identifier).
- **action** (*Dict*) – Dictionary serialization for the workflow step.
- **inputs** (*list of string, default=None*) – List of files that are required by the workflow step as inputs.
- **outputs** (*list of string, default=None*) – List of files that are generated by the workflow step as outputs.

Return type *flowserv.model.workflow.step.WorkflowStep*

flowserv.controller.serial.workflow.parser.**parse_template**(*template:* flowserv.model.template.base.WorkflowTemplate, *arguments: Dict*) → Tuple[List[*flowserv.model.workflow.step.ContainerStep*], Dict, List[str]]

Parse a serial workflow template to extract workflow steps and output files.

The expected schema of the workflow specification is as follows:

```
workflow:
    files:
        inputs:
        - "str"
        outputs:
        - "str"
    parameters:
    - name: "scalar"
    steps:
    - name: "str"
      files:
        inputs:
        - "str"
        outputs:
        - "str"
      action: "object depending on the step type"
```

The schema for the action specification for a workflow step is dependent on the step type. For container steps, the expected schema is:

```
action:
    environment: "str"
    commands:
    - "str"
```

Expands template parameter references in the workflow argument specification and the step inputs list. Returns the modified argument list as part of the result.

Parameters **template** (*flowserv.model.template.base.WorkflowTemplate*) – Template for a serial workflow.

Return type tuple of list of flowsert.controller.serial.workflow.step.ContainerStep, dict and list of string

flowserv.controller.serial.workflow.parser.**parse_varnames**(*action: Dict*) → Dict

Parse mapping of function or notebook argument names to the names of variables (e.g., parameters) in the workflow context.

Parameters **action** (*dict*) – Workflow step serialization.

Return type dict

flowserv.controller.serial.workflow.result module

Workflow (step) execution result.

class flowserv.controller.serial.workflow.result.**ExecResult**(*step: flowserv.model.workflow.step.WorkflowStep*, *returncode: typing.Optional[int] = 0*, *stdout: typing.Optional[typing.List[str]] = <factory>*, *stderr: typing.Optional[typing.List[str]] = <factory>*, *exception: typing.Optional[Exception] = None*)

Bases: object

Result of executing a workflow (or a single workflow step). Maintains a returncode to signal success (=0) or error (<>0). If an exception was raised during execution it is captured in the respective property *.exception*. Outputs that were written to standard output and standard error are part of the result object. Outputs are captured as lists of strings.

exception: Optional[Exception] = None

returncode: Optional[int] = 0

stderr: Optional[List[str]]

stdout: Optional[List[str]]

step: *flowserv.model.workflow.step.WorkflowStep*

class flowserv.controller.serial.workflow.result.**RunResult**(*arguments: Dict*)

Bases: object

Result for a serial workflow run. For each executed workflow step the run result maintains the step itself and the *flowserv.controller.serial.workflow.result.ExecResult*. In addition, the run result maintains the context of the workflow that is modified by the executed workflow steps.

Provides properties for easy access to the return code of the final workflow step and the outputs to STDOUT and STDERR.

add(*result:* flowserv.controller.serial.workflow.result.ExecResult)

Add execution result for a workflow step.

Parameters **result** (*flowserv.controller.serial.workflow.result.ExecResult*) – Execution result for the workflow step.

property exception: Exception

Get the exception from the last workflow step that was executed in the workflow run.

The result is None if no workflow step has been executed yet.

Return type Exception

`get`(*var: str*) → Any

Get the value for a given variable from the run context.

Raises a KeyError if the variable is not defined.

Parameters **var** (*string*) – Variable name

Return type any

`property log: List[str]`

Get single list containing the concatenation of STDOUT and STDERR messages.

Return type list of string

`raise_for_status`()

Raise an error if the returncode for this result is not zero.

Will re-raise a cought exception (if set). Otherwise, raises a FlowservError.

`property returncode: int`

Get the return code from the last workflow step that was executed in the workflow run.

The result is None if no workflow step has been executed yet.

Return type int

`property stderr: List[str]`

Get all lines that were written to STDERR by all executed workflow steps.

Return type list of string

`property stdout: List[str]`

Get all lines that were written to STDOUT by all executed workflow steps.

Return type list of string

flowserv.controller.worker package

Submodules

flowserv.controller.worker.base module

Base class for workers that execute workflow steps in different environments. Implementations of the base class may execute workflow commands using the Docker engine or the Python subprocess package.

`class flowserv.controller.worker.base.ContainerWorker`(*variables: Optional[Dict] = None*, *env: Optional[Dict] = None*, *identifier: Optional[str] = None*, *volume: Optional[str] = None*)

Bases: *flowserv.controller.worker.base.Worker*

Execution engine for container steps in a serial workflow. Provides the functionality to expand arguments in the individual command statements. Implementations may differ in the run method that executes the expanded commands.

exec(*step:* flowserv.model.workflow.step.ContainerStep, *context: Dict*, *store:* flowserv.volume.fs.FileSystemStorage) → *flowserv.controller.serial.workflow.result.ExecResult*

Execute a given list of commands that are represented by template strings.

Substitutes parameter and template placeholder occurrences first. Then calls the implementation-specific run method to execute the individual commands.

Note that the container worker expects a file system storage volume.

Parameters

- **step** (*flowserv.controller.serial.workflow.ContainerStep*) – Step in a serial workflow.
- **context** (*dict*) – Dictionary of argument values for parameters in the template.
- **store** (*flowserv.volume.fs.FileSystemStorage*) – Storage volume that contains the workflow run files.

Return type *flowserv.controller.serial.workflow.result.ExecResult*

abstract run(*step:* flowserv.model.workflow.step.ContainerStep, *env: Dict*, *rundir: str*) → *flowserv.controller.serial.workflow.result.ExecResult*

Execute a list of commands in a workflow step.

Parameters

- **step** (*flowserv.controller.serial.workflow.ContainerStep*) – Step in a serial workflow.
- **env** (*dict, default=None*) – Default settings for environment variables when executing workflow steps. May be None.
- **rundir** (*string*) – Path to the working directory of the workflow run.

Return type *flowserv.controller.serial.workflow.result.ExecResult*

class flowserv.controller.worker.base.**Worker**(*identifier: Optional[str] = None*, *volume: Optional[str] = None*)

Bases: object

Worker to execute steps in a serial workflow. For each class of workflow steps a separate worker can be implemented to execute instances of that particular step type.

abstract exec(*step:* flowserv.model.workflow.step.WorkflowStep, *context: Dict*, *store:* flowserv.volume.base.StorageVolume) → *flowserv.controller.serial.workflow.result.ExecResult*

Execute a given workflow step in the current workflow context.

Parameters

- **step** (*flowserv.model.workflow.step.WorkflowStep*) – Step in a serial workflow.
- **context** (*dict*) – Dictionary of variables that represent the current workflow state.
- **store** (*flowserv.volume.base.StorageVolume*) – Storage volume that contains the workflow run files.

Return type *flowserv.controller.serial.workflow.result.ExecResult*

flowserv.controller.worker.code module

Execute a workflow code step.

class flowserv.controller.worker.code.**CodeWorker**(*identifier: Optional[str] = None*, *volume: Optional[str] = None*)

Bases: *flowserv.controller.worker.base.Worker*

Worker to execute workflow steps of type *flowserv.model.workflow.step.CodeStep*

exec(*step:* flowserv.model.workflow.step.CodeStep, *context: Dict*, *store:* flowserv.volume.fs.FileSystemStorage) → *flowserv.controller.serial.workflow.result.ExecResult*

Execute a workflow step of type *flowserv.model.workflow.step.CodeStep* in a given context.

Captures output to STDOUT and STDERR and includes them in the returned execution result.

Note that the code worker expects a file system storage volume.

Parameters

- **step** (*flowserv.model.workflow.step.CodeStep*) – Code step in a serial workflow.
- **context** (*dict*) – Context for the executed code.
- **store** (*flowserv.volume.fs.FileSystemStorage*) – Storage volume that contains the workflow run files.

Return type *flowserv.controller.serial.workflow.result.ExecResult*

class flowserv.controller.worker.code.**OutputStream**(*stream*)

Bases: `object`

Output stream for standard output and standard error streams when executing a Python function.

close()

flush()

write(*text*)

writelines(*iterable*)

flowserv.controller.worker.config module

Collection of helper methods to configure worker engines.

flowserv.controller.worker.config.**java_jvm**() → str

Get path to the Java virtual machine.

Return type string

flowserv.controller.worker.config.**python_interpreter**() → str

Get path to the executable for the Python interpreter in the current environment.

Return type string

flowserv.controller.worker.docker module

Implementation of a workflow step engine that uses the local Docker daemon to execute workflow steps.

class flowserv.controller.worker.docker.**DockerWorker**(*variables: Optional[Dict] = None*, *env: Optional[Dict] = None*, *identifier: Optional[str] = None*, *volume: Optional[str] = None*)

Bases: *flowserv.controller.worker.base.ContainerWorker*

Container step engine that uses the local Docker deamon to execute the commands in a workflow step.

run(*step:* flowserv.model.workflow.step.ContainerStep, *env: Dict*, *rundir: str*) → *flowserv.controller.serial.workflow.result.ExecResult*

Execute a list of commands from a workflow steps synchronously using the Docker engine.

Stops execution if one of the commands fails. Returns the combined result from all the commands that were executed.

Parameters

- **step** (*flowserv.controller.serial.workflow.ContainerStep*) – Step in a serial workflow.
- **env** (*dict, default=None*) – Default settings for environment variables when executing workflow steps. May be None.
- **rundir** (*string*) – Path to the working directory of the workflow run that this step belongs to.

Return type *flowserv.controller.serial.workflow.result.ExecResult*

flowserv.controller.worker.docker.**NOTEBOOK_DOCKER_WORKER = 'nbdocker'**

Default Dockerfile for created papermill containers.

class flowserv.controller.worker.docker.**NotebookDockerWorker**(*env: Optional[Dict] = None*, *identifier: Optional[str] = None*, *volume: Optional[str] = None*)

Bases: *flowserv.controller.worker.base.Worker*

Execution engine for notebook steps in a serial workflow.

exec(*step:* flowserv.model.workflow.step.NotebookStep, *context: Dict*, *store:* flowserv.volume.fs.FileSystemStorage) → *flowserv.controller.serial.workflow.result.ExecResult*

Execute a given notebook workflow step in the current workflow context.

The notebook engine expects a file system storage volume that provides access to the notebook file and any other aditional input files.

Parameters

- **step** (*flowserv.model.workflow.step.NotebookStep*) – Notebook step in a serial workflow.
- **context** (*dict*) – Dictionary of variables that represent the current workflow state.
- **store** (*flowserv.volume.fs.FileSystemStorage*) – Storage volume that contains the workflow run files.

Return type *flowserv.controller.serial.workflow.result.ExecResult*

flowserv.controller.worker.docker.**docker_build**(*name: str*, *requirements: List[str]*) → Tuple[str, List[str]]

Build a Docker image from a standard Python image with `papermill` and the given requirements installed.

Returns the identifier of the created image.

Parameters

- **name** (*string*) – Name for the created image (derived from the workflow step name).
- **requirements** (*list of string*) – List of requirements that will be written to a file `requirements.txt` and installed inside the created Docker image.

Return type string, list of string

flowserv.controller.worker.docker.**docker_run**(*image: str*, *commands: List[str]*, *env: Dict*, *rundir: str*, *result:* flowserv.controller.serial.workflow.result.ExecResult) → *flowserv.controller.serial.workflow.result.ExecResult*

Helper function that executes a list of commands inside a Docker container.

Parameters

- **image** (*string*) – Identifier of the Docker image to run.
- **commands** (*string or list of string*) – Commands that are executed inside the Docker container.
- **result** (*flowserv.controller.serial.workflow.result.ExecResult*) – Result object that will contain the run outputs and status code.

Return type *flowserv.controller.serial.workflow.result.ExecResult*

flowserv.controller.worker.manager module

Factory for workers that implement the *flowserv.controller.worker.base.Worker* class. Workers are used to initiate and control the excution of workflow steps using different execution backends and implementations.

Instances of worker classes are created from a configuration specifications that follow the following schema:

```
definitions:
  keyValuePair:
    description: Key-value pair object.
    properties:
      key:
        description: Value key.
        type: string
      value:
        anyOf:
        - type: integer
        - type: string
        description: Scalar value associated with the key.
    required:
    - key
    - value
    type: object
  workerSpec:
    description: Specification for a worker engine instance.
```

(continues on next page)

(continued from previous page)

```
    properties:
      env:
        description: Key-value pairs for environment variables.
        items:
          $ref: '#/definitions/keyValuePair'
        type: array
      name:
        description: Unique worker identifier.
        type: string
      type:
        description: Worker type identifier
        enum:
        - code
        - docker
        - subprocess
        type: string
      vars:
        description: Key-value pairs for template string variables.
        items:
          $ref: '#/definitions/keyValuePair'
        type: array
      volume:
        description: Storage volume the worker has access to.
        type: string
    required:
    - name
    - type
    type: object
```

flowserv.controller.worker.manager.**Code**(*identifier: Optional[str] = None*, *volume: Optional[str] = None*) → Dict

Get base configuration serialization for a code worker.

Parameters

- **identifier** (*string, default=None*) – Unique worker identifier. If no identifier is given, a new unique identifier will be generated.
- **volume** (*string, default=None*) – Identifier for the storage volume that the worker has access to.

Return type dict

flowserv.controller.worker.manager.**Docker**(*identifier: Optional[str] = None*, *variables: Optional[Dict] = None*, *env: Optional[Dict] = None*, *volume: Optional[str] = None*) → Dict

Get base configuration for a subprocess worker with the given optional arguments.

Parameters

- **identifier** (*string, default=None*) – Unique worker identifier. If no identifier is given, a new unique identifier will be generated.
- **variables** (*dict, default=None*) – Mapping with default values for placeholders in command template strings.

- **env** (*dict, default=None*) – Default settings for environment variables when executing workflow steps. These settings can get overridden by step-specific settings.
- **volume** (*string, default=None*) – Identifier for the storage volume that the worker has access to.

Return type dict

`flowserv.controller.worker.manager.`**`Notebook`**(*identifier: Optional[str] = None*, *volume: Optional[str] = None*) → Dict

Get base configuration serialization for a notebook worker.

Parameters

- **identifier** (*string, default=None*) – Unique worker identifier. If no identifier is given, a new unique identifier will be generated.
- **volume** (*string, default=None*) – Identifier for the storage volume that the worker has access to.

Return type dict

`flowserv.controller.worker.manager.`**`Subprocess`**(*identifier: Optional[str] = None*, *variables: Optional[Dict] = None*, *env: Optional[Dict] = None*, *volume: Optional[str] = None*) → Dict

Get base configuration for a subprocess worker with the given optional arguments.

Parameters

- **identifier** (*string, default=None*) – Unique worker identifier. If no identifier is given, a new unique identifier will be generated.
- **variables** (*dict, default=None*) – Mapping with default values for placeholders in command template strings.
- **env** (*dict, default=None*) – Default settings for environment variables when executing workflow steps. These settings can get overridden by step-specific settings.
- **volume** (*string, default=None*) – Identifier for the storage volume that the worker has access to.

Return type dict

class `flowserv.controller.worker.manager.`**`WorkerPool`**(*workers: Optional[List[Dict]] = []*, *managers: Optional[Dict] = None*)

Bases: `object`

Manager for a pool of worker instances. Workers are responsible for the initiation and control of the execution of steps in a serial workflow.

Workers are instantiated from a dictionary serializations that follows the *workerSpec* schema defined in the *schema.json* file.

`get`(*step:* flowserv.model.workflow.step.WorkflowStep) → *flowserv.controller.worker.base.Worker*

Get the instance of the worker that is associated with the given workflow step.

If no worker specification exists for the given step a default worker is returned. The type of the default worker depends on the type of the workflow step. For code steps, currently only one type of worker exists. For container steps, a sub-process worker is used as the default worker.

Parameters **step** (*flowserv.model.workflow.step.WorkflowStep*) – Step in a serial workflow.

Return type *flowserv.controller.worker.base.Worker*

get_default_worker(*step:* flowserv.model.workflow.step.WorkflowStep) → *flowserv.controller.worker.base.Worker*

Return the default worker depending on the type of the given workflow step.

Parameters **step** (*flowserv.model.workflow.step.WorkflowStep*) – Step in a serial workflow.

Return type *flowserv.controller.worker.base.Worker*

flowserv.controller.worker.manager.**WorkerSpec**(*worker_type: str*, *identifier: Optional[str] = None*, *variables: Optional[Dict] = None*, *env: Optional[Dict] = None*, *volume: Optional[str] = None*) → Dict

Get a serialization for a worker specification.

Parameters

- **worker_type** (*string*) – Unique worker type identifier.
- **identifier** (*string, default=None*) – Unique worker identifier. If no identifier is given, a new unique identifier will be generated.
- **variables** (*dict, default=None*) – Mapping with default values for placeholders in command template strings.
- **env** (*dict, default=None*) – Default settings for environment variables when executing workflow steps. These settings can get overridden by step-specific settings.
- **volume** (*string, default=None*) – Identifier for the storage volume that the worker has access to.

Return type dict

flowserv.controller.worker.manager.**create_worker**(*doc: Dict*) → *flowserv.controller.worker.base.Worker*

Factory pattern for workers.

Create an instance of a worker implementation from a given worker serialization.

Parameters **doc** (*dict*) – Dictionary serialization for a worker.

Return type *flowserv.controller.worker.base.Worker*

flowserv.controller.worker.manager.**default_container_worker** = <**flowserv.controller.worker.subprocess.SubprocessWorker object**>

Serialization label for worker identifier.

flowserv.controller.worker.notebook module

class flowserv.controller.worker.notebook.**NotebookEngine**(*identifier: Optional[str] = None*, *env: Optional[Dict] = None*, *volume: Optional[str] = None*)

Bases: *flowserv.controller.worker.base.Worker*

Execution engine for notebook steps in a serial workflow.

exec(*step:* flowserv.model.workflow.step.NotebookStep, *context: Dict*, *store:* flowserv.volume.fs.FileSystemStorage) → *flowserv.controller.serial.workflow.result.ExecResult*

Execute a given notebook workflow step in the current workflow context.

The notebook engine expects a file system storage volume that provides access to the notebook file and any other aditional input files.

Parameters

- **step** (*flowserv.model.workflow.step.NotebookStep*) – Notebook step in a serial workflow.
- **context** (*dict*) – Dictionary of variables that represent the current workflow state.
- **store** (*flowserv.volume.fs.FileSystemStorage*) – Storage volume that contains the workflow run files.

Return type *flowserv.controller.serial.workflow.result.ExecResult*

flowserv.controller.worker.subprocess module

Workflow step processor that uses the Python subprocess package to execute a given list of commands in a container environment.

`class flowserv.controller.worker.subprocess.SubprocessWorker(variables: Optional[Dict] = None, env: Optional[Dict] = None, identifier: Optional[str] = None, volume: Optional[str] = None)`

Bases: *flowserv.controller.worker.base.ContainerWorker*

Container step engine that uses the subprocess package to execute the commands in a workflow step.

`run(step: flowserv.model.workflow.step.ContainerStep, env: Dict, rundir: str) → flowserv.controller.serial.workflow.result.ExecResult`

Execute a list of shell commands in a workflow step synchronously.

Stops execution if one of the commands fails. Returns the combined result from all the commands that were executed.

Parameters

- **step** (*flowserv.controller.serial.workflow.ContainerStep*) – Step in a serial workflow.
- **env** (*dict, default=None*) – Default settings for environment variables when executing workflow steps. May be None.
- **rundir** (*string*) – Path to the working directory of the workflow run.

Return type *flowserv.controller.serial.workflow.result.ExecResult*

`flowserv.controller.worker.subprocess.append(outstream: List[str], text: str)`

Append the given text to an output stream if the text is not empty.

Submodules

flowserv.controller.base module

Abstract interface for the workflow controller. The controller provides methods to start and cancel the execution of workflows, as well as methods that poll the current state of a workflow.

The aim of an abstract workflow controller is to keep the workflow controller flexible with respect to the processing backend that is being used. The implementation of the controller can either orchestrate the execution of a workflow iteself or be a wrapper around an existing workflow engine. An example for latter is a workflow controller that wrapps around the REANA workflow engine.

The implementation of the controller is responsible for interpreting a given workflow template and a set of template parameter arguments. The controller therefore requires a method for modifying the workflow template with a given set of user-provided template modifiers.

The controller is also responsible for retrieving output files and for providing access to these files.

class flowserv.controller.base.**WorkflowController**

Bases: object

The workflow controller is used to start execution of workflow templates for a given set of template parameter arguments, as well as to poll the state of workflow execution and to cancel execution.

Workflow executions, referred to as runs, are identified by unique run ids that are assigned by components that are outside of the controller. Implementations of the controller are responsible for maintaining a mapping of these run identifiers to any indentifiers that are generated by the workflow engine.

abstract cancel_run(*run_id: str*)

Request to cancel execution of the given run.

Parameters **run_id** (*string*) – Unique run identifier

Raises *flowserv.error.UnknownRunError* –

abstract exec_workflow(*run:* flowserv.model.base.RunObject, *template:* flowserv.model.template.base.WorkflowTemplate, *arguments: Dict*, *staticfs:* flowserv.volume.base.StorageVolume, *config: Optional[Dict] = None*) → Tuple[*flowserv.model.workflow.state.WorkflowState*, *flowserv.volume.base.StorageVolume*]

Initiate the execution of a given workflow template for a set of argument values. Returns the state of the workflow and the path to the directory that contains run result files for successful runs.

The client provides a unique identifier for the workflow run that is being used to retrieve the workflow state in future calls.

If the state of the run handle is not pending, an error is raised.

Parameters

- **run** (*flowserv.model.base.RunObject*) – Handle for the run that is being executed.
- **template** (*flowserv.model.template.base.WorkflowTemplate*) – Workflow template containing the parameterized specification and the parameter declarations.
- **arguments** (*dict*) – Dictionary of argument values for parameters in the template.
- **staticfs** (*flowserv.volume.base.StorageVolume*) – Storage volume that contains the static files from the workflow template.
- **config** (*dict, default=None*) – Optional implementation-specific configuration settings that can be used to overwrite settings that were initialized at object creation.

Return type *flowserv.model.workflow.state.WorkflowState*, *flowserv.volume.base.StorageVolume*

flowserv.model package

Subpackages

flowserv.model.parameter package

Submodules

flowserv.model.parameter.actor module

Declarations for parameters that represent actors (steps) in a workflow. The actual object instances that result from these actors are implementation dependent. For this reason, the values for actor parameters are represented as serialized dictionaries.

class flowserv.model.parameter.actor.**Actor**(*name: str*, *index: Optional[int] = 0*, *label: Optional[str] = None*, *help: Optional[str] = None*, *default: Optional[str] = None*, *required: Optional[bool] = False*, *group: Optional[str] = None*)

Bases: *flowserv.model.parameter.base.Parameter*

Workflow actor parameter type.

cast(*value:* flowserv.model.parameter.actor.ActorValue) → *flowserv.model.parameter.actor.ActorValue*

Ensure that the given value is an instance of the actor value class *flowserv.model.parameter.actor.ActorValue*.

Serializations of workflow steps are implementation dependent. For this reason, the function does not further validate the contents of the serialized workglow step.

Raises an InvalidArgumentError if the argument value is not a dictionary.

Parameters **value** (*flowserv.model.parameter.actor.ActorValue*) – Argument value for an actor parameter.

Return type *flowserv.model.parameter.actor.ActorValue*

static from_dict(*doc: Dict*, *validate: Optional[bool] = True*) → *flowserv.model.parameter.actor.Actor*

Get an actor parameter instance from a given dictionary serialization.

Parameters

- **doc** (*dict*) – Dictionary serialization for string parameter delaration.
- **validate** (*bool, default=True*) – Validate the serialized object if True.

Return type *flowserv.model.parameter.string.String*

Raises *flowserv.error.InvalidParameterError* –

class flowserv.model.parameter.actor.**ActorValue**(*spec: Dict*, *files: Optional[List[*flowserv.model.parameter.files.IOValue*]] = None*)

Bases: `object`

Value for an *flowserv.model.parameter.actor.Actor* parameter. The value contains two main components: (i) the serialization of the workflow step that is used by the workflow engine to create an instance of a workflow step, and (ii) a list of additional input files and directories for the workflow step that will be copied to the workflow run directory.

files: Optional[List[*flowserv.model.parameter.files.IOValue*]] = None

spec: Dict

flowserv.model.parameter.base module

Base class for workflow template parameters. Each parameter has a set of properties that are used to (i) identify the parameter, (ii) define a nested parameter structure, and (iii) render UI forms to collect parameter values.

flowserv.model.parameter.base.**OPTIONAL = ['label', 'help', 'defaultValue', 'group']**

Unique parameter type identifier.

class flowserv.model.parameter.base.**Parameter**(*dtype: str*, *name: str*, *index: Optional[int] = 0*, *label: Optional[str] = None*, *help: Optional[str] = None*, *default: Optional[Any] = None*, *required: Optional[bool] = False*, *group: Optional[str] = None*)

Bases: object

Base class for template parameters. The base class maintains the unique parameter name, the data type identifier, the human-readable label and the description for display purposes, the is required flag, an optional default value, the index position for input form rendering, and the identifier for the parameter group.

Implementing classes have to provide a static .from_dict() method that returns an instance of the class from a dictionary serialization. The dictionary serializations for each class are generated by the .to_dict() method.

abstract cast(*value: Any*) → Any

Validate the given argument value for the parameter type. Returns the argument representation for the value that is used to replace references to the parameter in workflow templates.

Raises an InvalidArgumentError if the given value is not valid for the parameter type.

Parameters **value** (*any*) – User-provided value for a template parameter.

Return type sting, float, or int

Raises *flowserv.error.InvalidArgumentError* –

display_name() → str

Human-readable display name for the parameter. The default display name is the defined label. If no label is defined the parameter name is returned.

Return type str

abstract static from_dict(*cls*, *doc: Dict*, *validate: Optional[bool] = True*) → *flowserv.model.parameter.base.Parameter*

Get instance of implementing class from dictionary serialization.

Parameters

- **doc** (*dict*) – Dictionary serialization for a parameter.
- **validate** (*bool, default=True*) – Validate the serialized object if True.

Return type *flowserv.model.parameter.base.Parameter*

Raises *flowserv.error.InvalidParameterError* –

is_actor() → bool

Test if the parameter is of type Actor.

Return type bool

`is_bool`() → bool

Test if the parameter is of type Bool.

Return type bool

`is_file`() → bool

Test if the parameter is of type File.

Return type bool

`is_float`() → bool

Test if the parameter is of type Float.

Return type bool

`is_int`() → bool

Test if the parameter is of type Int.

Return type bool

`is_list`() → bool

Test if the parameter is of type Array.

Return type bool

`is_numeric`() → bool

Test if the parameter is of type Numeric.

Parameters **para** (*flowserv.model.parameter.base.Parameter*) – Template parameter definition.

Return type bool

`is_record`() → bool

Test if the parameter is of type Record.

Return type bool

`is_select`() → bool

Test if the parameter is of type Select.

Return type bool

`is_string`() → bool

Test if the parameter is of type String.

Return type bool

`prompt`() → str

Get default input prompt for the parameter declaration. The prompt contains an indication of the data type, the parameter name and the default value (if defined).

Return type string

`to_dict`() → Dict

Get dictionary serialization for the parameter declaration. Implementing classes can add elements to the base dictionary.

Return type dict

class flowserv.model.parameter.base.**ParameterGroup**(*name: str*, *title: str*, *index: int*)

Bases: object

Parameter groups are identifiable sets of parameters. These sets are primarily intended for display purposes in the front-end. Therefore, each group has a display name and an index position that defines the sort order for groups.

classmethod from_dict(*doc*, *validate=True*)

Create object instance from dictionary serialization.

Parameters

- **doc** (*dict*) – Dictionary serialization for parameter group handles
- **validate** (*bool, default=True*) – Validate the serialization if True.

Return type *flowserv.model.parameter.base.ParameterGroup*

Raises **ValueError** –

to_dict()

Get dictionary serialization for parameter group handle.

Return type dict

flowserv.model.parameter.boolean module

Declarations for Boolean parameter values. Boolean parameters do not add any additional properties to the base parameter class.

class flowserv.model.parameter.boolean.**Bool**(*name: str*, *index: Optional[int] = 0*, *label: Optional[str] = None*, *help: Optional[str] = None*, *default: Optional[bool] = None*, *required: Optional[bool] = False*, *group: Optional[str] = None*)

Bases: *flowserv.model.parameter.base.Parameter*

Boolean parameter type.

cast(*value: Any*) → Any

Convert the given value into a Boolean value. Converts string values to Boolean True if they match either of the string representations '1', 't' or 'true' (case-insensitive) and to False if the value is None or it matches '', '0', 'f' or 'false'. Raises an error if a given value is not a valid representation for a Boolean value.

Parameters **value** (*any*) – User-provided value for a template parameter.

Return type sting

Raises *flowserv.error.InvalidArgumentError* –

static from_dict(*doc: Dict*, *validate: Optional[bool] = True*) → *flowserv.model.parameter.boolean.Bool*

Get Boolean parameter instance from a dictionary serialization.

Parameters

- **doc** (*dict*) – Dictionary serialization for a Boolean parameter declaration.
- **validate** (*bool, default=True*) – Validate the serialized object if True.

Return type *flowserv.model.parameter.boolean.Bool*

Raises *flowserv.error.InvalidParameterError* –

flowserv.model.parameter.enum module

Declarations for enumeration parameter values. Enumeration parameters contain a list of valid parameter values. These values are defined by a printable 'name' and an associated 'value'.

flowserv.model.parameter.enum.**Option**(*name: str*, *value: Union[str, int]*, *default: Optional[bool] = None*) → Dict

Get a dictionary serialization for an element in the enumeration of valid values for a select parameter.

Parameters

- **name** (*string*) – Option display name.
- **value** (*string or int*) – Returned value if this option is selected.
- **default** (*bool, default=None*) – Indicate if this is the default option for the selection.

Return type dict

class flowserv.model.parameter.enum.**Select**(*name: str*, *values: List[Dict]*, *index: Optional[int] = 0*, *label: Optional[str] = None*, *help: Optional[str] = None*, *default: Optional[bool] = None*, *required: Optional[bool] = False*, *group: Optional[str] = None*)

Bases: *flowserv.model.parameter.base.Parameter*

Enumeration parameter type for select boxes. Extends the base parameter with a list of possible argument values.

cast(*value: Any*) → Any

Ensure that the given value is valid. If the value is not contained in the enumerated list of values an error is raised.

Parameters **value** (*any*) – User-provided value for a template parameter.

Return type sting

Raises *flowserv.error.InvalidArgumentError* –

static from_dict(*doc: Dict*, *validate: Optional[bool] = True*) → *flowserv.model.parameter.enum.Select*

Get select parameter instance from a dictionary serialization.

Parameters

- **doc** (*dict*) – Dictionary serialization for select parameter declaration.
- **validate** (*bool, default=True*) – Validate the serialized object if True.

Return type *flowserv.model.parameter.enum.Select*

Raises *flowserv.error.InvalidParameterError* –

to_dict() → Dict

Get dictionary serialization for the parameter declaration. Adds list of enumerated values to the base serialization.

Return type dict

flowserv.model.parameter.factory module

Factory for parameter declarations. Allows to create instances of parameter declaration classes from dictionary serializations.

class flowserv.model.parameter.factory.**ParameterDeserializer**

Bases: object

Factory for parameter declarations from dictionary serializations. This class is merely a dispatcher that looks at the parameter type in a given serialization and calls the deserialization method of the respective parameter declaration class.

static from_dict(*doc: Dict*, *validate: Optional[bool] = True*) → *flowserv.model.parameter.base.Parameter*

Create instance of parameter declaration class from a given dictionary serialization object.

Parameters

- **doc** (*dict*) – Dictionary serialization for a parameter declaration.
- **validate** (*bool, default=True*) – Validate given serialization if True.

Return type *flowserv.model.parameter.base.Parameter*

Raises *flowserv.error.InvalidParameterError* –

flowserv.model.parameter.files module

Declarations for file parameter values. A file parameter extends the base parameter class with a target path for the file when creating the workflow storage volume.

class flowserv.model.parameter.files.**File**(*name: str*, *index: Optional[int] = 0*, *target: Optional[str] = None*, *label: Optional[str] = None*, *help: Optional[str] = None*, *default: Optional[bool] = None*, *required: Optional[bool] = False*, *group: Optional[str] = None*)

Bases: *flowserv.model.parameter.base.Parameter*

File parameter type. Extends the base parameter with a target path for the file.

cast(*value: Union[*flowserv.volume.base.IOHandle*, Tuple[*flowserv.volume.base.IOHandle*, str]]*)

Get an instance of the InputFile class for a given argument value. The input value can either be a string (filename) or a dictionary.

Parameters **value** (*tuple of flowserv.volume.base.IOHandle, str*) – Handle for a file object and optional user-provided target path. If the target path is None the defined target path is used or the defined default value. If neither is given an error is raised.

Return type *flowserv.model.parameter.files.InputFile*

Raises

- *flowserv.error.InvalidArgumentError* –
- *flowserv.error.UnknownFileError* –

static from_dict(*doc: Dict*, *validate: bool = True*) → *flowserv.model.parameter.files.File*

Get file parameter instance from a dictionary serialization.

Parameters

- **doc** (*dict*) – Dictionary serialization for file parameter declaration.

- **validate** (*bool, default=True*) – Validate the serialized object if True.

Return type *flowserv.model.parameter.files.File*

Raises *flowserv.error.InvalidParameterError* –

`to_dict`() → Dict

Get dictionary serialization for the parameter declaration. Adds target path to the base serialization.

Return type dict

`class flowserv.model.parameter.files.IOValue(target: str)`

Bases: `object`

The IO value represents the value for a template parameter of type 'file'. Implementations will either represent single input files or directories. The `copy` method is used to store the input object in the target storage volume for workflow runs.

`abstract copy(target: flowserv.volume.base.StorageVolume) → List[str]`

Copy the input(s) to the given target storage volume.

Returns the list of copied files.

Return type list of string

`class flowserv.model.parameter.files.InputDirectory(store: flowserv.volume.base.StorageVolume, target: str, source: Optional[str] = None)`

Bases: *flowserv.model.parameter.files.IOValue*

Implementation of the IOValue class for template parameter values that represents a directory on a storage volume. The `copy` method will copy the complete folder to the tartget volume for a workflow run.

`copy(target: flowserv.volume.base.StorageVolume) → List[str]`

Copy the file object to the target volume.

Return type list of string

`class flowserv.model.parameter.files.InputFile(source: flowserv.volume.base.IOHandle, target: str)`

Bases: *flowserv.model.parameter.files.IOValue*

Implementation of the IOValue class for template parameter values that are a single file. Maintains the IOHandle for an input file that can be copied to the tartget volume for a workflow run.

`copy(target: flowserv.volume.base.StorageVolume) → List[str]`

Copy the file object to the target volume.

Return type list of string

flowserv.model.parameter.list module

Declarations for list parameters. List parameter values are lists of values for a single parameter declaration.

`class flowserv.model.parameter.list.Array(name: str, para: flowserv.model.parameter.base.Parameter, index: Optional[int] = 0, label: Optional[str] = None, help: Optional[str] = None, default: Optional[List] = None, required: Optional[bool] = False, group: Optional[str] = None)`

Bases: *flowserv.model.parameter.base.Parameter*

List parameter type to define lists of values that all match the same parameter declaration..

cast(*value: List*) → List

Convert the given value into a list where each value in the given list has been converted using the parameter declaration for the list parameter.

Raises an error if the value is not a list or if the associated parameter declaration raised an exception during cast.

Parameters **value** (*any*) – User-provided value for a template parameter.

Return type list

Raises *flowserv.error.InvalidArgumentError* –

static from_dict(*doc: Dict*, *validate: Optional[bool] = True*) → List

Get list parameter instance from a dictionary serialization.

Parameters

- **doc** (*dict*) – Dictionary serialization for a list parameter declaration.
- **validate** (*bool, default=True*) – Validate the serialized object if True.

Return type flowserv.model.parameter.list.List

Raises *flowserv.error.InvalidParameterError* –

to_dict() → Dict

Get dictionary serialization for the parameter declaration. Adds the serialized value for the list parameter declaration.

Return type dict

flowserv.model.parameter.numeric module

Declarations for numeric parameter values. Numeric parameters can specify ranges of valid values or minimum and maximum values.

class flowserv.model.parameter.numeric.**Boundary**(*value: Optional[Union[int, float, str]] = None*, *is_closed: Optional[bool] = True*)

Bases: `object`

Boundary definition for a range interval. Contains the interval boundary value and a flag defining whether the interval is open (i.e., excludes the defined value) or closed (i.e., includes the defined value).

is_closed: Optional[bool] = True

to_left_boundary() → str

Return a left-boundary representation.

Return type string

to_right_boundary() → str

Return a left-boundary representation.

Return type string

value: Union[int, float, str] = None

class flowserv.model.parameter.numeric.**Float**(*name: str*, *index: Optional[int] = 0*, *label: Optional[str] = None*, *help: Optional[str] = None*, *default: Optional[float] = None*, *required: Optional[bool] = False*, *group: Optional[str] = None*, *min: Optional[Union[int, float, str, flowserv.model.parameter.numeric.Boundary]] = None*, *max: Optional[Union[int, float, str, flowserv.model.parameter.numeric.Boundary]] = None*)

Bases: *flowserv.model.parameter.numeric.Numeric*

Base class for float parameter types.

class flowserv.model.parameter.numeric.**Int**(*name: str*, *index: Optional[int] = 0*, *label: Optional[str] = None*, *help: Optional[str] = None*, *default: Optional[int] = None*, *required: Optional[bool] = False*, *group: Optional[str] = None*, *min: Optional[Union[int, float, str, flowserv.model.parameter.numeric.Boundary]] = None*, *max: Optional[Union[int, float, str, flowserv.model.parameter.numeric.Boundary]] = None*)

Bases: *flowserv.model.parameter.numeric.Numeric*

Base class for integer parameter types.

class flowserv.model.parameter.numeric.**Numeric**(*dtype: str*, *name: str*, *index: Optional[int] = 0*, *label: Optional[str] = None*, *help: Optional[str] = None*, *default: Optional[Union[int, float]] = None*, *required: Optional[bool] = False*, *group: Optional[str] = None*, *constraint: Optional[flowserv.model.parameter.numeric.RangeConstraint] = None*)

Bases: *flowserv.model.parameter.base.Parameter*

Base class for numeric parameter types. Extends the base class with an optional range constraint.

cast(*value: Any*) → Any

Convert the given value into a numeric value. Raises an error if the value cannot be converted to the respective numeric type of if it does not satisfy the optional range constraint.

Parameters **value** (*any*) – User-provided value for a template parameter.

Return type sting, float, or int

Raises *flowserv.error.InvalidArgumentError* –

static from_dict(*doc: Dict*, *validate: Optional[bool] = True*) → *flowserv.model.parameter.numeric.Numeric*

Get numeric parameter instance from a dictionary serialization.

Parameters

- **doc** (*dict*) – Dictionary serialization for numeric parameter.
- **validate** (*bool, default=True*) – Validate the serialized object if True.

Return type *flowserv.model.parameter.numeric.Numeric*

Raises *flowserv.error.InvalidParameterError* –

to_dict() → Dict

Get dictionary serialization for the parameter declaration. Adds a serialization for an optional range constraint to the serialization of the base class.

Return type dict

class flowserv.model.parameter.numeric.**RangeConstraint**(*left_boundary: Dict*, *right_boundary: Dict*)

Bases: object

Range constraint for numeric parameter values. Range constraints are specified as strings following standard interval notation where square brackets denote closed intervals and round brackets denote open intervals. Infinity and negative infinity are represented as 'inf' and '-inf', respectively. They may also be represented as an empty string.

Valid range interval strings are: [x,y], (x,y), [x,y), and (x,y] where x may either be a number, '', or '-inf', and y may either be a number, '', or 'inf'.

classmethod from_string(*value: str*)

Create range constraint instance from string representation.

Parameters **value** (*string*) – String representation for a range constraint.

Return type *flowserv.model.parameter.numeric.RangeConstraint*

Raises **ValueError** –

is_closed() → bool

Returns True if both interval boundaries are closed.

Return type bool

max_value() → Union[int, float]

Get the right boundary value for the interval.

Return type int or float

min_value() → Union[int, float]

Get the left boundary value for the interval.

Return type int or float

to_string() → str

Get string representation for the range constraint.

Return type string

validate(*value: Union[int, float]*) → bool

Validate that the given value is within the interval. Raises an error if the value is not within the interval.

Parameters **value** (*int or float*) – Value that is being tested.

Raises *flowserv.error.InvalidArgumentError* –

flowserv.model.parameter.numeric.**range_constraint**(*left: Optional[Union[int, float, str, flowserv.model.parameter.numeric.Boundary]] = None*, *right: Optional[Union[int, float, str, flowserv.model.parameter.numeric.Boundary]] = None*) → *flowserv.model.parameter.numeric.RangeConstraint*

Create a range constraint instance from a given pair of interval boundaries. Returns None if no boundary is given.

Parameters

- **left** (*flowserv.model.parameter.numeric.IntervalBoundary, default=None*) – Left boundary for range constraint.
- **right** (*flowserv.model.parameter.numeric.IntervalBoundary, default=None*) – Right boundary for range constraint.

Return type *flowserv.model.parameter.numeric.RangeConstraint*

flowserv.model.parameter.record module

Declarations for record parameter values. Records are collections of parameters. Each component (field) of a record is identified by a unique name.

`class flowserv.model.parameter.record.Record`(*name: str*, *fields: List[*flowserv.model.parameter.base.Parameter*]*, *index: Optional[int] = 0*, *label: Optional[str] = None*, *help: Optional[str] = None*, *default: Optional[Dict] = None*, *required: Optional[bool] = False*, *group: Optional[str] = None*)

Bases: *flowserv.model.parameter.base.Parameter*

Record parameter type that associates parameter declarations for record components with unique keys.

`cast`(*value: Any*) → Dict

Convert the given value into a record. Returns a dictionary that is a mapping of filed identifier to the converted values returned by the respective parameter declaration.

Expects a list of dictionaries containing two elements: 'name' and 'value'. The name identifies the record field and the value is the argument value for that field.

Raises an error if the value is not a list, if any of the dictionaries are not well-formed, if required fields are not present in the given list, or if the respective parameter declaration for a record fields raised an exception during cast.

Parameters **value** (*any*) – User-provided value for a template parameter.

Return type sting

Raises *flowserv.error.InvalidArgumentError* –

`static from_dict`(*doc: Dict*, *validate: Optional[bool] = True*) → *flowserv.model.parameter.record.Record*

Get record parameter instance from a given dictionary serialization.

Parameters

- **doc** (*dict*) – Dictionary serialization for record parameter declaration.
- **validate** (*bool, default=True*) – Validate the serialized object if True.

Return type *flowserv.model.parameter.record.Record*

Raises *flowserv.error.InvalidParameterError* –

`to_dict`() → Dict

Get dictionary serialization for the parameter declaration. Adds list of serialized parameter declarations for record fields to the base serialization.

Individual fields are serialized as dictionaries with elements 'name' for the field name and 'para' for the serialized parameter declaration.

Return type dict

flowserv.model.parameter.string module

Declarations for string parameter values. String parameters do not add any additional properties to the base parameter class.

class flowserv.model.parameter.string.**String**(*name: str*, *index: Optional[int] = 0*, *label: Optional[str] = None*, *help: Optional[str] = None*, *default: Optional[str] = None*, *required: Optional[bool] = False*, *group: Optional[str] = None*)

Bases: *flowserv.model.parameter.base.Parameter*

String parameter type.

cast(*value: Any*) → str

Convert the given value into a string value.

Parameters **value** (*any*) – User-provided value for a template parameter.

Return type sting

static from_dict(*doc: Dict*, *validate: Optional[bool] = True*) → *flowserv.model.parameter.string.String*

Get string parameter instance from a given dictionary serialization.

Parameters

- **doc** (*dict*) – Dictionary serialization for string parameter delaration.
- **validate** (*bool, default=True*) – Validate the serialized object if True.

Return type *flowserv.model.parameter.string.String*

Raises *flowserv.error.InvalidParameterError* –

flowserv.model.template package

Submodules

flowserv.model.template.base module

A workflow template contains a workflow specification and an optional list of template parameters. The workflow specification may contain references to template parameters.

Template parameters are common to different backends that excute a workflow. The syntax and structure of the workflow specification is engine specific. The only common part here is that template parameters are referenced using the $[[..]] syntax from within the specifications.

The template parameters can be used to render front-end forms that gather user input for each parameter. Given an association of parameter identifiers to user-provided values, the workflow backend is expected to be able to execute the modified workflow specification in which references to template parameters have been replaced by parameter values.

class flowserv.model.template.base.**WorkflowTemplate**(*workflow_spec*, *parameters*, *parameter_groups=None*, *outputs=None*, *postproc_spec=None*, *result_schema=None*)

Bases: `object`

Workflow templates are parameterized workflow specifications. The template contains the workflow specification, template parameters, and information about benchmark results and post-processing steps.

The syntax and structure of the workflow specification(s) is not further inspected. It is dependent on the workflow controller that is used to execute workflow runs.

The template for a parameterized workflow contains a dictionary of template parameter declarations. Parameter declarations are keyed by their unique identifier in the dictionary.

For benchmark templates, a result schema is defined. This schema is used to store the results of different benchmark runs in a database and to the generate a benchmark leader board.

classmethod from_dict(*doc*, *validate=True*)

Create an instance of the workflow template for a dictionary serialization. The structure of the dictionary is expected to be the same as generated by the to_dict() method of this class. The only mandatory element in the dictionary is the workflow specification.

Parameters

- **doc** (*dict*) – Dictionary serialization of a workflow template
- **validate** (*bool, optional*) – Validate template parameter declarations against the parameter schema if this flag is True.

Return type *flowserv.model.template.base.WorkflowTemplate*

Raises

- *flowserv.error.InvalidTemplateError* –
- *flowserv.error.UnknownParameterError* –

to_dict()

Get dictionary serialization for the workflow template.

Return type dict

validate_arguments(*arguments*)

Ensure that the workflow can be instantiated using the given set of arguments. Raises an error if there are template parameters for which the argument set does not provide a value and that do not have a default value.

Parameters **arguments** (*dict*) – Dictionary of argument values for parameters in the template

Raises *flowserv.error.MissingArgumentError* –

flowserv.model.template.files module

Classes to maintain of workflow output file specifications.

class flowserv.model.template.files.**WorkflowOutputFile**(*source: str*, *title: Optional[str] = None*, *key: Optional[str] = None*, *caption: Optional[str] = None*, *format: Optional[Dict] = None*, *widget: Optional[Dict] = None*)

Bases: object

Information about workflow output files that are accessible as resources for successful workflow runs. The majority of properties that are defined for output files are intended to be used by applications that display the result of workflow runs. For each output file the following information may be specified:

- source: relative path the the file in the run folder
- key: Unique key that is assigned to the resource for dictionary access
- title: optional title for display purposes

- caption: optional caption for display purposes
- format: optional format information for file contents.
- widget: optional instructions for widget used to display file contents.

`classmethod from_dict`(*doc*, *validate=True*)

Create object instance from dictionary serialization.

Parameters

- **doc** (*dict*) – Dictionary serialization for a workflow output file.
- **validate** (*bool, default=True*) – Validate the serialization if True.

Return type *flowserv.model.template.files.WorkflowOutputFile*

Raises `ValueError` –

`to_dict`()

Get dictionary serialization for the output file specification.

Return type dict

flowserv.model.template.parameter module

Collection of helper methods for parameter references in workflow templates.

`class flowserv.model.template.parameter.ParameterIndex`(*parameters: Optional[List[*flowserv.model.parameter.base.Parameter*]] = None*)

Bases: `dict`

Index of parameter declaration. Parameters are indexed by their unique identifier.

`static from_dict`(*doc: Dict*, *validate: Optional[bool] = True*) → *flowserv.model.parameter.base.Parameter*

Create a parameter index from a dictionary serialization. Expects a list of dictionaries, each being a serialized parameter declaration.

Raises an error if parameter indices are not unique.

Parameters

- **doc** (*list*) – List of serialized parameter declarations.
- **validate** (*bool, default=True*) – Validate dictionary serializations if True.

Return type flowserv.model.template.base.ParameterIndex

`set_defaults`(*arguments: Dict*) → Dict

Set default values for parameters that have a default and that are not present in the given arguments dictionary.

Returns a modified copy of the given arguments dictionary.

Parameters **arguments** (*dict*) – Dictionary of user-provided argument values.

Return type dict

sorted()

Get list of parameter declarations sorted by ascending parameter index position.

Return type list(*flowserv.model.parameter.base.Parameter*)

to_dict()

Get dictionary serialization for the parameter declarations.

Return type list

flowserv.model.template.parameter.**VARIABLE**(*name*)

Get string representation containing the reference to a variable with given name. This string is intended to be used as a template parameter reference within workflow specifications in workflow templates.

Parameters **name** (*string*) – Template parameter name

Return type string

flowserv.model.template.parameter.**expand_value**(*value*, *arguments*, *parameters*)

Test whether the string is a reference to a template parameter and (if True) replace the value with the given argument or default value.

In the current implementation template parameters are referenced using $[[..]] syntax.

Parameters

- **value** (*string*) – String value in the workflow specification for a template parameter
- **arguments** (*dict*) – Dictionary that associates template parameter identifiers with argument values
- **parameters** (*flowserv.model.template.parameter.ParameterIndex*) – Dictionary of parameter declarations

Return type any

Raises *flowserv.error.MissingArgumentError* –

flowserv.model.template.parameter.**get_name**(*value*)

Extract the parameter name for a template parameter reference.

Parameters **value** (*string*) – String value in the workflow specification for a template parameter

Return type string

flowserv.model.template.parameter.**get_parameter_references**(*spec*, *parameters=None*)

Get set of parameter identifier that are referenced in the given workflow specification. Adds parameter identifier to the given parameter set.

Parameters

- **spec** (*dict*) – Parameterized workflow specification.
- **parameters** (*set, optional*) – Result set of referenced parameter identifier.

Return type set

Raises *flowserv.error.InvalidTemplateError* –

flowserv.model.template.parameter.**get_value**(*value*, *arguments*)

Get the result value from evaluating a parameter reference expression. Expects a value that satisfies the is_parameter() predicate. If the given expression is unconditional, e.g., $[[name]], the parameter name is the returned result. If the expression is conditional, e.g., $[[name ? x : y]] the argument value for parameter 'name' is tested for being Boolean True or False. Depending on the outcome of the evaluation either x or y are returned.

Note that nested conditional expressions are currently not supported.

Parameters

- **value** (*string*) – Parameter reference string that satisifes the is_parameter() predicate.
- **arguments** (*dict*) – Dictionary of user-provided argument values for template arguments.

Return type string

Raises *flowserv.error.MissingArgumentError* –

`flowserv.model.template.parameter.`**`is_parameter`**(*value*)

Returns True if the given value is a reference to a template parameter.

Parameters **value** (*string*) – String value in the workflow specification for a template parameter

Return type bool

`flowserv.model.template.parameter.`**`placeholders`**(*text: str*) → Set[str]

Get the set of names for all placeholders in the given string.

Placeholders start with '$' following the string.template syntax.

Parameters **text** (*string*) – Template string.

Return type set of string

`flowserv.model.template.parameter.`**`replace_args`**(*spec*, *arguments*, *parameters*)

Replace template parameter references in the workflow specification with their respective values in the argument dictionary or their defined default value. The type of the result is depending on the type of the spec object.

Parameters

- **spec** (*any*) – Parameterized workflow specification.
- **arguments** (*dict*) – Dictionary that associates template parameter identifiers with argument values.
- **parameters** (*flowserv.model.template.parameter.ParameterIndex*) – Dictionary of parameter declarations.

Return type type(spec)

Raises

- *flowserv.error.InvalidTemplateError* –
- *flowserv.error.MissingArgumentError* –

flowserv.model.template.schema module

Definition of schema components for benchmark results. The schema definition is part of the extended workflow template specification that is used to define benchmarks.

`class` `flowserv.model.template.schema.`**`ResultColumn`**(*column_id*, *name*, *dtype*, *path=None*, *required=None*)

Bases: `object`

Column in the result schema of a benchmark. Each column has a unique identifier and unique name. The identifier is used as column name in the database schema. The name is for display purposes in a user interface. The optional path element is used to extract the column value from nested result files.

`cast`(*value*)

Cast the given value to the data type of the column. Will raise ValueError if type cast is not successful.

Parameters **value** (*scalar*) – Expects a scalar value that can be converted to the respective column type.

Return type int, float, or string

`classmethod from_dict`(*doc*, *validate=True*)

Get an instance of the column from the dictionary serialization. Raises an error if the given dictionary does not contain the expected elements as generated by the to_dict() method of the class.

Parameters

- **doc** (*dict*) – Dictionary serialization of a column object
- **validate** (*bool, default=True*) – Validate the serialization if True.

Return type *flowserv.model.template.schema.ResultColumn*

Raises *flowserv.error.InvalidTemplateError* –

`jpath`()

The Json path for a result column is a list of element keys that reference the column value in a nested document. If the internal path variable is not set the column identifier is returned as the only element in the path.

Return type list(string)

`to_dict`()

Get dictionary serialization for the column object.

Return type dict

`class flowserv.model.template.schema.`**ResultSchema**(*result_file*, *columns*, *order_by=None*)

Bases: `object`

The result schema of a benchmark run is a collection of columns. The result schema is used to generate leader boards for benchmarks.

The schema also contains the identifier of the output file that contains the result object. The result object that is generated by each benchmark run is expected to contain a value for each required columns in the schema.

`classmethod from_dict`(*doc*, *validate=True*)

Get an instance of the schema from a dictionary serialization. Raises an error if the given dictionary does not contain the expected elements as generated by the to_dict() method of the class or if the names or identifier of columns are not unique.

Returns None if the given document is None.

Parameters

- **doc** (*dict*) – Dictionary serialization of a benchmark result schema object
- **validate** (*bool, default=True*) – Validate the serialization if True.

Return type *flowserv.model.template.schema.ResultSchema*

Raises *flowserv.error.InvalidTemplateError* –

`get_default_order`()

By default the first column in the schema is used as the sort column. Values in the column are sorted in descending order.

Return type list(*flowserv.model.template.schema.SortColumn*)

`to_dict()`

Get dictionary serialization for the result schema object.

Return type dict

`class flowserv.model.template.schema.SortColumn(column_id, sort_desc=None)`

Bases: `object`

The sort column defines part of an ORDER BY statement that is used to sort benchmark results when creating the benchmark leader board. Each object contains a reference to a result column and a flag indicating the sort order for values in the column.

`classmethod from_dict(doc, validate=True)`

Get an instance of the sort column from the dictionary serialization. Raises an error if the given dictionary does not contain the expected elements as generated by the to_dict() method of the class.

Parameters

- **doc** (*dict*) – Dictionary serialization of a column object
- **validate** (*bool, default=True*) – Validate the serialization if True.

Return type *flowserv.model.template.schema.SortColumn*

Raises *flowserv.error.InvalidTemplateError* –

`to_dict()`

Get dictionary serialization for the sort column object.

Return type dict

flowserv.model.workflow package

Submodules

flowserv.model.workflow.manager module

The workflow repository maintains information about registered workflow templates. For each template additional basic information is stored in the underlying database.

`class flowserv.model.workflow.manager.WorkflowManager(session: sqlalchemy.orm.session.Session, fs: flowserv.volume.base.StorageVolume)`

Bases: `object`

The workflow manager maintains information that is associated with workflow templates in a workflow repository.

`create_workflow(source: str, identifier: Optional[str] = None, name: Optional[str] = None, description: Optional[str] = None, instructions: Optional[str] = None, specfile: Optional[str] = None, manifestfile: Optional[str] = None, engine_config: Optional[Dict] = None, ignore_postproc: Optional[bool] = False, verbose: Optional[bool] = False) → flowserv.model.base.WorkflowObject`

Add new workflow to the repository. The associated workflow template is created in the template repository from either the given source directory or a Git repository. The template repository will raise an error if neither or both arguments are given.

The method will look for a workflow description file in the template base folder with the name flowserv.json, flowserv.yaml, flowserv.yml (in this order). The expected structure of the file is:

```
name: ''
description: ''
instructions: ''
files:
    - source: ''
      target: ''
specfile: '' or workflowSpec: ''
```

An error is raised if both specfile and workflowSpec are present in the description file.

Raises an error if no workflow name is given or if a given workflow name is not unique.

Parameters

- **source** (*string*) – Path to local template, name or URL of the template in the repository.
- **identifier** (*string, default=None*) – Unique user-defined workflow identifier.
- **name** (*string, default=None*) – Unique workflow name.
- **description** (*string, default=None*) – Optional short description for display in workflow listings.
- **instructions** (*string, default=None*) – File containing instructions for workflow users.
- **specfile** (*string, default=None*) – Path to the workflow template specification file (absolute or relative to the workflow directory).
- **manifestfile** (*string, default=None*) – Path to manifest file. If not given an attempt is made to read one of the default manifest file names in the base directory.
- **engine_config** (*dict, default=None*) – Optional configuration settings that will be used as the default when running the workflow and the post-processing workflow.
- **ignore_postproc** (*bool, default=False*) – Ignore post-processing workflow specification if True.
- **verbose** (*bool, default=False*) – Print information about copied files.

Return type *flowserv.model.base.WorkflowObject*

Raises

- ***flowserv.error.ConstraintViolationError*** –
- ***flowserv.error.InvalidTemplateError*** –
- ***flowserv.error.InvalidManifestError*** –
- **ValueError** –

delete_workflow(*workflow_id*)

Delete the workflow with the given identifier.

Parameters **workflow_id** (*string*) – Unique workflow identifier

Raises ***flowserv.error.UnknownWorkflowError*** –

get_workflow(*workflow_id*)

Get handle for the workflow with the given identifier. Raises an error if no workflow with the identifier exists.

Parameters **workflow_id** (*string*) – Unique workflow identifier

Return type *flowserv.model.base.WorkflowObject*

Raises *flowserv.error.UnknownWorkflowError* –

`list_workflows()`

Get a list of descriptors for all workflows in the repository.

Return type list(*flowserv.model.base.WorkflowObject*)

`update_workflow`(*workflow_id*, *name=None*, *description=None*, *instructions=None*)

Update name, description, and instructions for a given workflow.

Raises an error if the given workflow does not exist or if the name is not unique.

Parameters

- **workflow_id** (*string*) – Unique workflow identifier
- **name** (*string, optional*) – Unique workflow name
- **description** (*string, optional*) – Optional short description for display in workflow listings
- **instructions** (*string, optional*) – Text containing detailed instructions for workflow execution

Return type *flowserv.model.base.WorkflowObject*

Raises

- *flowserv.error.ConstraintViolationError* –
- *flowserv.error.UnknownWorkflowError* –

`flowserv.model.workflow.manager.clone`(*source*, *repository=None*)

Clone a workflow template repository. If source points to a directory on local disk it is returned as the 'cloned' source directory. Otherwise, it is assumed that source either references a known template in the global workflow template repository or points to a git repository. The repository is cloned into a temporary directory which is removed when the generator resumes after the workflow has been copied to the local repository.

Returns a tuple containing the path to the resulting template source directory on the local disk and the optional path to the template's manifest file.

Parameters

- **source** (*string*) – The source is either a path to local template directory, an identifer for a template in the global template repository, or the URL for a git repository.
- **repository** (*flowserv.model.workflow.repository.WorkflowRepository,*) – default=None Object providing access to the global workflow repository.

Return type string, string

flowserv.model.workflow.manifest module

Helper functions to read workflow manifest files.

flowserv.model.workflow.manifest.**MANIFEST_FILES = ['flowserv.json', 'flowserv.yaml', 'flowserv.yml']**

Regular expression for file includes in markdown.

class flowserv.model.workflow.manifest.**WorkflowManifest**(*basedir*, *name*, *workflow_spec*, *description=None*, *instructions=None*, *files=None*)

Bases: object

The workflow manifest contains the workflow specification, the workflow metadata (name, description and optional instructions), as well as the list of files that need to be copied when creating a local copy of a workflow template in a repository.

copyfiles(*dst: str*) → List[Tuple[str, str]]

Get list of all template files from the base folder that need to be copied to the template folder of a workflow repository.

The result is a list of tuples specifying the relative file source and target path. The target path for each file is a concatenation of the given destination base directory and the specified target path for the file or folder. If the list of files is undefined in the manifest, the result is a tuple (None, dst) indicating that the full base directory is to be copied to the destination.

Return type list of (string, string)

static load(*basedir*, *manifestfile=None*, *name=None*, *description=None*, *instructions=None*, *specfile=None*, *existing_names={}*)

Read the workflow manifest from file. By default, an attempt is made to read a file with one the following names in the basedir (in the given order): flowserv.json, flowserv.yaml, flowserv.yml. If the manifest file parameter is given the specified file is being read instead.

The parameters name, description, instructions, and specfile are used to override the respective properties in the manifest file.

Raises a ValueError if no manifest file is found or if no name or workflow specification is present in the resulting manifest object.

Parameters

- **basedir** (*string*) – Path to the base directory containing the workflow files. This directory is used when reading the manifest file (if not given as argument) and the instructions file (if not given as argument).
- **manifestfile** (*string, default=None*) – Path to manifest file. If not given an attempt is made to read one of the default manifest file names in the base directory.
- **name** (*string*) – Unique workflow name
- **description** (*string*) – Optional short description for display in workflow listings
- **instructions** (*string*) – File containing instructions for workflow users.
- **specfile** (*string*) – Path to the workflow template specification file (absolute or relative to the workflow directory)
- **existing_names** (*set, default=set()*) – Set of names for existing projects.

Return type *flowserv.model.workflow.manifest.WorkflowManifest*

Raises IOError, OSError, ValueError, *flowserv.error.InvalidManifestError* –

`template()`

Get workflow template instance for the workflow specification that is included in the manifest.

Return type flowserv.model.template.base.Workflowtemplate

`flowserv.model.workflow.manifest.getfile(basedir, manifest_value, user_argument)`

Get name for a file that is referenced in a workflow manifest. If the user argument is given it overrides the respective value in the manifest. For user arguments we first assume that the path references a file on disk, either as absolute path or as a path relative to the current working directory. If no file exists at the specified location an attempt is made to read the file relative to the base directory. For manifest values, they are always assumed to be relative to the base directory.

Parameters

- **basedir** (*string*)
- **manifest_value** (*string*) – Relative path to the file in the base directory.
- **user_argument** (*string*) – User provided value that overrides the manifest value. This value can be None.

Return type string

`flowserv.model.workflow.manifest.read_instructions(filename: str)` → str

Read instruction text from a given file. If the filename is None the result will be None as well.

Return type string

`flowserv.model.workflow.manifest.unique_name(name, existing_names)`

Ensure that the workflow name in the project metadata is not empty, not longer than 512 character, and unique.

Parameters

- **name** (*string*) – Workflow name in manifest or given by user.
- **existing_names** (*set*) – Set of names for existing projects.

Raises *flowserv.error.ConstraintViolationError* –

flowserv.model.workflow.repository module

Helper class to access the repository of workflow templates.

`class flowserv.model.workflow.repository.WorkflowRepository(templates: Optional[List[Dict]] = None)`

Bases: `object`

Repository for workflow specifications. The repository is currently maintained as a Json file on GitHub. The file contains an array of objects. Each object describes an installable workflow template with the following elements:

- id: unique human-readable template identifier
- description: short description
- url: Url to a git repository that contains the workflow files
- **manifest: optional (relative) path for manifest file in the workflow** repository.

get(*identifier: str*) → Tuple[str, str, Dict]

Get the URL, the optional (relative) manifest file path, and optional additional arguments (e.g., the branch name) for the repository entry with the given identifier. If no entry matches the identifier it is returned as the function result.

Returns a tuple of (url, manifestpath, args). If the manifest element is not present in the repository entry the second value is None. If no arguments are present the result is an empty dictionary.

Parameters **identifier** (*string*) – Workflow template identifier or repository URL.

Return type string, string, dict

list() → List[Tuple[str, str, str]]

Get list of tuples containing the template identifier, descriptions, and repository URL.

Return type list

flowserv.model.workflow.state module

Definition of workflow states. The classes in this module represent the different possible states of a workflow run. There are four different states: (PENDING) the workflow run has been submitted and is waiting to start running, (RUNNING) the workflow is actively executing at the moment, (ERROR) workflow execution was interrupted by an error or canceled by the user, (SUCCESS) the workflow run completed successfully.

Contains default methods to (de-)serialize workflow state.

flowserv.model.workflow.state.**CANCELED = ['canceled at user request']**

Definition of state type identifier.

flowserv.model.workflow.state.**STATE_SUCCESS = 'SUCCESS'**

Short cut to list of active states.

class flowserv.model.workflow.state.**StateCanceled**(*created_at: str*, *started_at: Optional[str] = None*, *stopped_at: Optional[str] = None*, *messages: Optional[List[str]] = None*)

Bases: *flowserv.model.workflow.state.WorkflowState*

Cancel state representation for a workflow run. The workflow has three timestamps: the workflow creation time, workflow run start time and the time when the workflow was canceled. The state also maintains an optional list of messages.

class flowserv.model.workflow.state.**StateError**(*created_at: str*, *started_at: Optional[str] = None*, *stopped_at: Optional[str] = None*, *messages: Optional[List[str]] = None*)

Bases: *flowserv.model.workflow.state.WorkflowState*

Error state representation for a workflow run. The workflow has three timestamps: the workflow creation time, workflow run start time and the time at which the error occured (ot workflow was canceled). The state also maintains an optional list of error messages.

class flowserv.model.workflow.state.**StatePending**(*created_at: Optional[str] = None*)

Bases: *flowserv.model.workflow.state.WorkflowState*

State representation for a pending workflow that is waiting to start running. The workflow has only one timestamp representing the workflow creation time.

cancel(*messages: Optional[List[str]] = None*) → *flowserv.model.workflow.state.StateCanceled*

Get instance of canceled state for a pending wokflow.

Since the workflow did not start to run the started_at timestamp is set to the current time just like the stopped_at timestamp.

Parameters **messages** (*list(string), optional*) – Optional list of messages

Return type *flowserv.model.workflow.state.StateCanceled*

error(*messages: Optional[List[str]] = None*) → *flowserv.model.workflow.state.StateError*

Get instance of error state for a pending wokflow. If the exception that caused the workflow execution to terminate is given it will be used to create the list of error messages.

Since the workflow did not start to run the started_at timestamp is set to the current time just like the stopped_at timestamp.

Parameters **messages** (*list(string), optional*) – Optional list of error messages

Return type *flowserv.model.workflow.state.StateError*

start() → *flowserv.model.workflow.state.StateRunning*

Get instance of running state with the same create at timestamp as this state and the started at with the current timestamp.

Return type *flowserv.model.workflow.state.StateRunning*

success(*files: Optional[List[str]] = None*) → *flowserv.model.workflow.state.StateSuccess*

Get instance of success state for a competed wokflow.

Parameters **files** (*list(string), default=None*) – Optional list of created files (relative path).

Return type *flowserv.model.workflow.state.StateSuccess*

class flowserv.model.workflow.state.**StateRunning**(*created_at: str*, *started_at: Optional[str] = None*)

Bases: *flowserv.model.workflow.state.WorkflowState*

State representation for a active workflow run. The workflow has two timestamps: the workflow creation time and the workflow run start time.

cancel(*messages: Optional[List[str]] = None*) → *flowserv.model.workflow.state.StateCanceled*

Get instance of class cancel state for a running wokflow.

Parameters **messages** (*list(string), optional*) – Optional list of messages

Return type *flowserv.model.workflow.state.StateCanceled*

error(*messages: Optional[List[str]] = None*) → *flowserv.model.workflow.state.StateError*

Get instance of error state for a running wokflow. If the exception that caused the workflow execution to terminate is given it will be used to create the list of error messages.

Parameters **messages** (*list(string), optional*) – Optional list of error messages

Return type *flowserv.model.workflow.state.StateError*

success(*files: Optional[List[str]] = None*) → *flowserv.model.workflow.state.StateSuccess*

Get instance of success state for a competed wokflow.

Parameters **files** (*list(string), default=None*) – Optional list of created files (relative path).

Return type *flowserv.model.workflow.state.StateSuccess*

class flowserv.model.workflow.state.**StateSuccess**(*created_at: str*, *started_at: str*, *finished_at: Optional[str] = None*, *files: Optional[List[str]] = None*)

Bases: *flowserv.model.workflow.state.WorkflowState*

Success state representation for a workflow run. The workflow has three timestamps: the workflow creation time, workflow run start time and the time when the workflow execution finished. The state also maintains handles to any files that were created by the workflow run.

class flowserv.model.workflow.state.**WorkflowState**(*type_id: str*, *created_at: Optional[str] = None*)

Bases: object

The base class for workflow states contains the state type identifier that is used by the different state type methods. The state also maintains the timestamp of workflow run creation. Subclasses will add additional timestamps and properties.

is_active() → bool

A workflow is in active state if it is either pending or running.

Return type bool

is_canceled() → bool

Returns True if the workflow state is of type CANCELED.

Return type bool

is_error() → bool

Returns True if the workflow state is of type ERROR.

Return type bool

is_pending() → bool

Returns True if the workflow state is of type PENDING.

Return type bool

is_running() → bool

Returns True if the workflow state is of type RUNNING.

Return type bool

is_success() → bool

Returns True if the workflow state is of type SUCCESS.

Return type bool

flowserv.model.workflow.state.**deserialize_state**(*doc*)

Create instance of workflow state from a given dictionary serialization.

Parameters **doc** (*dict*) – Serialization if the workflow state

Return type *flowserv.model.workflow.state.WorkflowState*

Raises

- **KeyError** –
- **ValueError** –

flowserv.model.workflow.state.**serialize_state**(*state*)

Create dictionary serialization if a given workflow state.

Parameters **state** (*flowserv.model.workflow.state.WorkflowState*) – Workflow state

Return type dict

flowserv.model.workflow.step module

Definitions for the different types of steps in a serial workflow. At this point we distinguish three types of workflow steps:

flowserv.model.workflow.step.FunctionStep and *flowserv.model.workflow.step.ContainerStep* :class: *flowserv.model.workflow.step.NotebookStep*.

A *flowserv.model.workflow.step.CodeStep* is used to execute a given function within the workflow context. The code is executed within the same thread and environment as the flowserv engine. Code steps are intended for minor actions (e.g., copying of files or reading results from previous workflow steps). For these actions it would cause too much overhead to create an external Python script that is run as a subprocess or a Docker container image.

A *flowserv.model.workflow.step.ContainerStep* is a workflow step that is executed in a separate container-like environment. The environment can either be a subprocess with specific environment variable settings or a Docker container.

class flowserv.model.workflow.step.**CodeStep**(*identifier: str*, *func: Callable*, *arg: Optional[str] = None*, *varnames: Optional[Dict] = None*, *inputs: Optional[List[str]] = None*, *outputs: Optional[List[str]] = None*)

Bases: *flowserv.model.workflow.step.WorkflowStep*

Workflow step that executes a given Python function.

The function is evaluated using the current state of the workflow arguments. If the executed function returns a result, the returned object can be added to the arguments. That is, the argument dictionary is updated and the added object is availble for the following workflows steps.

exec(*context: Dict*)

Execute workflow step using the given arguments.

The given set of input arguments may be modified by the return value of the evaluated function.

Parameters **context** (*dict*) – Mapping of parameter names to their current value in the workflow executon state. These are the global variables in the execution context.

class flowserv.model.workflow.step.**ContainerStep**(*identifier: str*, *image: str*, *commands: Optional[List[str]] = None*, *env: Optional[Dict] = None*, *inputs: Optional[List[str]] = None*, *outputs: Optional[List[str]] = None*)

Bases: *flowserv.model.workflow.step.WorkflowStep*

Workflow step that is executed in a container environment. Contains a reference to the container identifier and a list of command line statements that are executed in a given environment.

add(*cmd: str*) → *flowserv.model.workflow.step.ContainerStep*

Append a given command line statement to the list of commands in the workflow step.

Returns a reference to the object itself.

Parameters **cmd** (*string*) – Command line statement

Return type flowserv.model.workflow.serial.Step

flowserv.model.workflow.step.**FunctionStep**

alias of *flowserv.model.workflow.step.CodeStep*

class flowserv.model.workflow.step.**NotebookStep**(*identifier: str*, *notebook: str*, *output: Optional[str] = None*, *requirements: Optional[List[str]] = None*, *params: Optional[List[str]] = None*, *varnames: Optional[Dict] = None*, *inputs: Optional[List[str]] = None*, *outputs: Optional[List[str]] = None*)

Bases: *flowserv.model.workflow.step.WorkflowStep*

cli_command(*context: Dict*) → str

Get command to run notebbok using papermill from command line.

This method is used when running a notebook inside a Docker container.

Parameters **context** (*dict*) – Mapping of parameter names to their current value in the workflow executon state. These are the global variables in the execution context.

exec(*context: Dict*, *rundir: str*)

Execute the notebook using papermill in the given workflow context.

Parameters

- **context** (*dict*) – Mapping of parameter names to their current value in the workflow executon state. These are the global variables in the execution context.
- **rundir** (*string*) – Directory for the workflow run that contains all the run files.

class flowserv.model.workflow.step.**WorkflowStep**(*identifier: str*, *step_type: int*, *inputs: Optional[List[str]] = None*, *outputs: Optional[List[str]] = None*)

Bases: `object`

Base class for the different types of steps (actor) in a serial workflow.

We distinguish several workflow steps including steps that are executed in a container-like environment and steps that directly execute Python code.

The aim of this base class is to provide functions to distinguish between these two types of steps and to maintain properties that are common to all steps.

Each step in a serial workflow has a unique identifier (name) and optional lists of input files and output files. All files are specified as relative path expressions (keys).

is_code_step() → bool

True if the workflow step is of type *flowserv.model.workflow.step.CodeStep*.

Return type bool

is_container_step() → bool

True if the workflow step is of type *flowserv.model.workflow.step.ContainerStep*.

Return type bool

is_notebook_step() → bool

True if the workflow step is of type :class:'flowserv.model.workflow.step.NotebookStep'.

Return type bool

property name: str

Synonym for the step identifier.

Return type string

flowserv.model.workflow.step.**output_notebook**(*name: str*, *input: str*) → str

Generate name for output notebook.

If an output name is given it is returned as it is. Otherwise, the name of the input notebook will have the suffix `.ipynb` replaced by `.out.ipynb`. If the input notebook does not have a suffix `.ipynb` the suffix `.out.ipynb` is appended to the input notebook name.

Parameters

- **name** (*string*) – User-provided name for the output notebook. This value may be None.
- **input** (*string*) – Name of the input notebook.

Return type string

Submodules

flowserv.model.auth module

The authentication and authorization module contains methods to authorize users that have logged in to the system as well as methods to authorize that a given user can execute a requested action.

class flowserv.model.auth.**Auth**(*session*)

Bases: `object`

Base class for authentication and authorization methods. Different authorization policies should override the methods of this class.

authenticate(*api_key*)

Get the unique user identifier that is associated with the given API key. Raises an error if the API key is None or if it is not associated with a valid login.

Parameters **api_key** (*string*) – Unique API access token assigned at login

Return type *flowserv.model.base.User*

Raises *flowserv.error.UnauthenticatedAccessError* –

group_or_run_exists(*group_id=None*, *run_id=None*)

Test whether the given group or run exists. Raises an error if they don't exist or if no parameter or both parameters are given.

Returns the group identifier for the run. If group_id is given the value is returned as the result. If the run_id is given the group identifier is retrieved as part of the database query.

Parameters

- **group_id** (*string, optional*) – Unique workflow group identifier
- **run_id** (*string, optional*) – Unique run identifier

Raises

- **ValueError** –
- *flowserv.error.UnknownRunError* –
- *flowserv.error.UnknownWorkflowGroupError* –

abstract is_group_member(*user_id*, *group_id=None*, *run_id=None*)

Verify that the given user is member of a workflow group. The group is identified either by the given group identifier or by the identifier for a run that is associated with the group.

Expects that exactly one of the two optional identifier is given. Raises a ValueError if both identifier are None or both are not None. Raises an error if the workflow group or the run is unknown.

Parameters

- **user_id** (*string*) – Unique user identifier
- **group_id** (*string, optional*) – Unique workflow group identifier
- **run_id** (*string, optional*) – Unique run identifier

Return type bool

Raises

- **ValueError** –
- *flowserv.error.UnknownRunError* –
- *flowserv.error.UnknownWorkflowGroupError* –

class flowserv.model.auth.**DefaultAuthPolicy**(*session*)

Bases: *flowserv.model.auth.Auth*

Default implementation for the API's authorization methods.

is_group_member(*user_id*, *group_id=None*, *run_id=None*)

Verify that the given user is member of a workflow group. The group is identified either by the given group identifier or by the identifier for a run that is associated with the group.

Expects that exactly one of the two optional identifier is given. Raises a ValueError if both identifier are None or both are not None.

Parameters

- **user_id** (*string*) – Unique user identifier
- **group_id** (*string, optional*) – Unique workflow group identifier
- **run_id** (*string, optional*) – Unique run identifier

Return type bool

Raises

- **ValueError** –
- *flowserv.error.UnknownRunError* –
- *flowserv.error.UnknownWorkflowGroupError* –

class flowserv.model.auth.**OpenAccessAuth**(*session*)

Bases: *flowserv.model.auth.Auth*

Implementation for the API's authorization policy that gives full access to any registered user.

is_group_member(*user_id*, *group_id=None*, *run_id=None*)

Anyone has access to a workflow group. This method still ensures that the combination of argument values is valid and that the group or run exists.

Parameters

- **user_id** (*string*) – Unique user identifier

- **group_id** (*string, optional*) – Unique workflow group identifier
- **run_id** (*string, optional*) – Unique run identifier

Return type bool

Raises `ValueError` –

`flowserv.model.auth.`**open_access**(*session*)

Create an open access policy object.

flowserv.model.base module

Define the classes in the Object-Relational Mapping.

class `flowserv.model.base.`**APIKey**(***kwargs*)

Bases: `sqlalchemy.orm.decl_api.Base`

API key assigned to a user at login.

expires

user_id

value

class `flowserv.model.base.`**FileObject**(***kwargs*)

Bases: `sqlalchemy.orm.decl_api.Base`

The file handle base class provides additional methods to access a file and its properties.

created_at = Column(None, String(length=32), table=None, nullable=False, default=ColumnDefault(<function utc_now>))

file_id = Column(None, String(length=32), table=None, primary_key=True, nullable=False, default=ColumnDefault(<function get_unique_identifier>))

key = Column(None, String(length=1024), table=None, nullable=False)

mime_type = Column(None, String(length=64), table=None)

name = Column(None, String(length=512), table=None, nullable=False)

size = Column(None, Integer(), table=None, nullable=False)

class `flowserv.model.base.`**GroupObject**(***kwargs*)

Bases: `sqlalchemy.orm.decl_api.Base`

A workflow group associates a set of users with a workflow template. It allows to define a group-specific set of parameters for the template.

engine_config

group_id

members

name

owner

owner_id

parameters

runs

uploads

workflow

workflow_id

workflow_spec

class flowserv.model.base.**JsonObject**(**args*, ***kwargs*)

Bases: sqlalchemy.sql.type_api.TypeDecorator

Decorator for objects that are stored as serialized JSON strings.

cache_ok = True

Indicate if statements using this ExternalType are "safe to cache".

The default value None will emit a warning and then not allow caching of a statement which includes this type. Set to False to disable statements using this type from being cached at all without a warning. When set to True, the object's class and selected elements from its state will be used as part of the cache key. For example, using a TypeDecorator:

```
class MyType(TypeDecorator):
    impl = String

    cache_ok = True

    def __init__(self, choices):
        self.choices = tuple(choices)
        self.internal_only = True
```

The cache key for the above type would be equivalent to:

```
>>> MyType(["a", "b", "c"])._static_cache_key
(<class '__main__.MyType'>, ('choices', ('a', 'b', 'c')))
```

The caching scheme will extract attributes from the type that correspond to the names of parameters in the __init__() method. Above, the "choices" attribute becomes part of the cache key but "internal_only" does not, because there is no parameter named "internal_only".

The requirements for cacheable elements is that they are hashable and also that they indicate the same SQL rendered for expressions using this type every time for a given cache value.

To accommodate for datatypes that refer to unhashable structures such as dictionaries, sets and lists, these objects can be made "cacheable" by assigning hashable structures to the attributes whose names correspond with the names of the arguments. For example, a datatype which accepts a dictionary of lookup values may publish this as a sorted series of tuples. Given a previously un-cacheable type as:

```
class LookupType(UserDefinedType):
    '"a custom type that accepts a dictionary as a parameter.

    this is the non-cacheable version, as "self.lookup" is not
    hashable.
```

(continues on next page)

(continued from previous page)

```
    '''

    def __init__(self, lookup):
        self.lookup = lookup

    def get_col_spec(self, **kw):
        return "VARCHAR(255)"

    def bind_processor(self, dialect):
        # ...  works with "self.lookup" ...
```

Where "lookup" is a dictionary. The type will not be able to generate a cache key:

```
>>> type_ = LookupType({"a": 10, "b": 20})
>>> type_._static_cache_key
<stdin>:1: SAWarning: UserDefinedType LookupType({'a': 10, 'b': 20}) will not
produce a cache key because the ``cache_ok`` flag is not set to True.
Set this flag to True if this type object's state is safe to use
in a cache key, or False to disable this warning.
symbol('no_cache')
```

If we **did** set up such a cache key, it wouldn't be usable. We would get a tuple structure that contains a dictionary inside of it, which cannot itself be used as a key in a "cache dictionary" such as SQLAlchemy's statement cache, since Python dictionaries aren't hashable:

```
>>> # set cache_ok = True
>>> type_.cache_ok = True

>>> # this is the cache key it would generate
>>> key = type_._static_cache_key
>>> key
(<class '__main__.LookupType'>, ('lookup', {'a': 10, 'b': 20}))

>>> # however this key is not hashable, will fail when used with
>>> # SQLAlchemy statement cache
>>> some_cache = {key: "some sql value"}
Traceback (most recent call last): File "<stdin>", line 1,
in <module> TypeError: unhashable type: 'dict'
```

The type may be made cacheable by assigning a sorted tuple of tuples to the ".lookup" attribute:

```
class LookupType(UserDefinedType):
    '''a custom type that accepts a dictionary as a parameter.

    The dictionary is stored both as itself in a private variable,
    and published in a public variable as a sorted tuple of tuples,
    which is hashable and will also return the same value for any
    two equivalent dictionaries.  Note it assumes the keys and
    values of the dictionary are themselves hashable.

    '''
```

(continues on next page)

(continued from previous page)

```
    cache_ok = True

    def __init__(self, lookup):
        self._lookup = lookup

        # assume keys/values of "lookup" are hashable; otherwise
        # they would also need to be converted in some way here
        self.lookup = tuple(
            (key, lookup[key]) for key in sorted(lookup)
        )

    def get_col_spec(self, **kw):
        return "VARCHAR(255)"

    def bind_processor(self, dialect):
        # ...  works with "self._lookup" ...
```

Where above, the cache key for `LookupType({"a": 10, "b": 20})` will be:

```
>>> LookupType({"a": 10, "b": 20})._static_cache_key
(<class '__main__.LookupType'>, ('lookup', (('a', 10), ('b', 20))))
```

New in version 1.4.14: - added the `cache_ok` flag to allow some configurability of caching for `TypeDecorator` classes.

New in version 1.4.28: - added the `ExternalType` mixin which generalizes the `cache_ok` flag to both the `TypeDecorator` and `UserDefinedType` classes.

See also:

sql_caching

impl

alias of `sqlalchemy.sql.sqltypes.Unicode`

process_bind_param(*value*, *dialect*)

Expects a JSON serializable object.

process_literal_param(*value*, *dialect*)

Expects a JSON serializable object.

process_result_value(*value*, *dialect*)

Create JSON object from string serialization.

class flowserv.model.base.**PasswordRequest**(***kwargs*)

Bases: `sqlalchemy.orm.decl_api.Base`

Unique identifier associated with a password reset request.

expires

request_id

user_id

class flowserv.model.base.**RunFile**(***kwargs*)

Bases: *flowserv.model.base.FileObject*

File resources that are created by successful workflow runs.

created_at

file_id

key

mime_type

name

run

run_id

size

class flowserv.model.base.**RunMessage**(***kwargs*)

Bases: sqlalchemy.orm.decl_api.Base

Log for messages created by workflow runs. Primarily used for error messages by now.

message

pos

run

run_id

class flowserv.model.base.**RunObject**(***kwargs*)

Bases: sqlalchemy.orm.decl_api.Base

Workflow runs may be triggered by workflow group members or they represent post-processing workflows. In the latter case the group identifier is None.

arguments

created_at

ended_at

files

get_file(*by_id=None*, *by_key=None*)

Get handle for identified file. A file can either be identified by the unique identifier or file key (i.e., relative path name). Returns None if the file is not found.

Raises a ValueError if an invalid combination of parameters is given.

Parameters

- **by_id** (*string*) – Unique file identifier.
- **by_key** (*string*) – Relative path to the file in the run directory.

Return type *flowserv.model.base.RunFile*

group

group_id

is_active()

A run is in active state if it is either pending or running.

Return type bool

is_canceled()

Returns True if the run state is of type CANCELED.

Return type bool

is_error()

Returns True if the run state is of type ERROR.

Return type bool

is_pending()

Returns True if the run state is of type PENDING.

Return type bool

is_running()

Returns True if the run state is of type RUNNING.

Return type bool

is_success()

Returns True if the run state is of type SUCCESS.

Return type bool

log

outputs()

Get specification of output file properties. The result is a dictionary of workflow output file specifications keyed by either the user-specified key or the file source. If the workflow template does not contain any output file specifications the result is an empty dictionary.

If the run is associated with a group, then the output file specification of the associated workflow is returned. If the run is a post-processing run the optional output specification in the post- processing workflow template is returned.

Returns dict(string

Return type *flowserv.model.template.files.WorkflowOutputFile*)

result

run_id

started_at

state()

Get an instance of the workflow state for the given run.

Return type *flowserv.model.workflow.state.WorkflowState*

state_type

workflow

workflow_id

class flowserv.model.base.**UploadFile**(***kwargs*)

Bases: *flowserv.model.base.FileObject*

Uploaded files are assigned to individual workflow groups. Each file is assigned a unique identifier.

created_at

file_id

group

group_id

key

mime_type

name

size

class flowserv.model.base.**User**(***kwargs*)

Bases: sqlalchemy.orm.decl_api.Base

Each user that registers with the application has a unique identifier and a unique user name associated with them.

For users that are logged into the system the user handle contains the API key that was assigned during login..

active

api_key

groups

is_logged_in()

Test if the user API key is set as an indicator of whether the user is currently logged in or not.

Return type bool

name

password_request

secret

user_id

class flowserv.model.base.**WorkflowObject**(***kwargs*)

Bases: sqlalchemy.orm.decl_api.Base

Each workflow has a unique name, an optional short descriptor and long instruction text. The five main components of the template are (i) the workflow specification, (ii) the list of parameter declarations, (iii) an optional post-processing workflow specification, (iv) the optional grouping of parameters into sets, and (v) the result schema for workflows that generate metrics for individual workflow runs.

With each workflow a reference to the latest run containing post-processing results is maintained. The value is NULL if no post-porcessing workflow is defined for the template or if it has not been executed yet. The post- processing key contains the sorted list of identifier for the runs that were used as input to generate the post-processing results.

description

engine_config

get_template(*workflow_spec=None*, *parameters=None*)

Get template for the workflow. The optional parameters allow to override the default values with group-specific values.

Parameters

- **workflow_spec** (*dict, default=None*) – Modified workflow specification.
- **parameters** (*dict(flowserv.model.parameter.base.Parameter)*) – Modified wokflow parameter list.

Return type *flowserv.model.template.base.WorkflowTemplate*

groups

ignore_postproc

instructions

name

outputs

parameter_groups

parameters

postproc_ranking

postproc_run_id

postproc_spec

ranking()

Get list of identifier for runs in the current ranking sorted by their rank.

Return type list(string)

result_schema

property run_postproc

Returns True iff the result schema and post-processing workflow are defined and the ignore_postproc flag is False.

Return type bool

runs

workflow_id

workflow_spec

class flowserv.model.base.**WorkflowOutputs**(**args*, ***kwargs*)

Bases: sqlalchemy.sql.type_api.TypeDecorator

Decorator for workflow output file specifications that are stored as serialized Json objects.

cache_ok = True

Indicate if statements using this `ExternalType` are "safe to cache".

The default value `None` will emit a warning and then not allow caching of a statement which includes this type. Set to `False` to disable statements using this type from being cached at all without a warning. When set to `True`, the object's class and selected elements from its state will be used as part of the cache key. For example, using a `TypeDecorator`:

```
class MyType(TypeDecorator):
    impl = String

    cache_ok = True

    def __init__(self, choices):
        self.choices = tuple(choices)
        self.internal_only = True
```

The cache key for the above type would be equivalent to:

```
>>> MyType(["a", "b", "c"])._static_cache_key
(<class '__main__.MyType'>, ('choices', ('a', 'b', 'c')))
```

The caching scheme will extract attributes from the type that correspond to the names of parameters in the `__init__()` method. Above, the "choices" attribute becomes part of the cache key but "internal_only" does not, because there is no parameter named "internal_only".

The requirements for cacheable elements is that they are hashable and also that they indicate the same SQL rendered for expressions using this type every time for a given cache value.

To accommodate for datatypes that refer to unhashable structures such as dictionaries, sets and lists, these objects can be made "cacheable" by assigning hashable structures to the attributes whose names correspond with the names of the arguments. For example, a datatype which accepts a dictionary of lookup values may publish this as a sorted series of tuples. Given a previously un-cacheable type as:

```
class LookupType(UserDefinedType):
    '"a custom type that accepts a dictionary as a parameter.

    this is the non-cacheable version, as "self.lookup" is not
    hashable.

    '"

    def __init__(self, lookup):
        self.lookup = lookup

    def get_col_spec(self, **kw):
        return "VARCHAR(255)"

    def bind_processor(self, dialect):
        # ...  works with "self.lookup" ...
```

Where "lookup" is a dictionary. The type will not be able to generate a cache key:

```
>>> type_ = LookupType({"a": 10, "b": 20})
>>> type_._static_cache_key
```

(continues on next page)

(continued from previous page)

```
<stdin>:1: SAWarning: UserDefinedType LookupType({'a': 10, 'b': 20}) will not
produce a cache key because the ``cache_ok`` flag is not set to True.
Set this flag to True if this type object's state is safe to use
in a cache key, or False to disable this warning.
symbol('no_cache')
```

If we **did** set up such a cache key, it wouldn't be usable. We would get a tuple structure that contains a dictionary inside of it, which cannot itself be used as a key in a "cache dictionary" such as SQLAlchemy's statement cache, since Python dictionaries aren't hashable:

```
>>> # set cache_ok = True
>>> type_.cache_ok = True

>>> # this is the cache key it would generate
>>> key = type_._static_cache_key
>>> key
(<class '__main__.LookupType'>, ('lookup', {'a': 10, 'b': 20}))

>>> # however this key is not hashable, will fail when used with
>>> # SQLAlchemy statement cache
>>> some_cache = {key: "some sql value"}
Traceback (most recent call last): File "<stdin>", line 1,
in <module> TypeError: unhashable type: 'dict'
```

The type may be made cacheable by assigning a sorted tuple of tuples to the ".lookup" attribute:

```
class LookupType(UserDefinedType):
    '"a custom type that accepts a dictionary as a parameter.

    The dictionary is stored both as itself in a private variable,
    and published in a public variable as a sorted tuple of tuples,
    which is hashable and will also return the same value for any
    two equivalent dictionaries.  Note it assumes the keys and
    values of the dictionary are themselves hashable.

    '"

    cache_ok = True

    def __init__(self, lookup):
        self._lookup = lookup

        # assume keys/values of "lookup" are hashable; otherwise
        # they would also need to be converted in some way here
        self.lookup = tuple(
            (key, lookup[key]) for key in sorted(lookup)
        )

    def get_col_spec(self, **kw):
        return "VARCHAR(255)"

    def bind_processor(self, dialect):
        # ...  works with "self._lookup" ...
```

Where above, the cache key for `LookupType({"a": 10, "b": 20})` will be:

```
>>> LookupType({"a": 10, "b": 20})._static_cache_key
(<class '__main__.LookupType'>, ('lookup', (('a', 10), ('b', 20))))
```

New in version 1.4.14: - added the `cache_ok` flag to allow some configurability of caching for `TypeDecorator` classes.

New in version 1.4.28: - added the `ExternalType` mixin which generalizes the `cache_ok` flag to both the `TypeDecorator` and `UserDefinedType` classes.

See also:

sql_caching

impl

alias of `sqlalchemy.sql.sqltypes.Unicode`

process_bind_param(*value*, *dialect*)

Expects a list of workflow output file objects.

process_literal_param(*value*, *dialect*)

Expects a list of workflow output file objects.

process_result_value(*value*, *dialect*)

Create workflow output file list from JSON serialization.

class flowserv.model.base.**WorkflowParameterGroups**(**args*, ***kwargs*)

Bases: `sqlalchemy.sql.type_api.TypeDecorator`

Decorator for workflow parameters groups that are stored as serialized Json objects.

cache_ok = True

Indicate if statements using this `ExternalType` are "safe to cache".

The default value `None` will emit a warning and then not allow caching of a statement which includes this type. Set to `False` to disable statements using this type from being cached at all without a warning. When set to `True`, the object's class and selected elements from its state will be used as part of the cache key. For example, using a `TypeDecorator`:

```
class MyType(TypeDecorator):
    impl = String

    cache_ok = True

    def __init__(self, choices):
        self.choices = tuple(choices)
        self.internal_only = True
```

The cache key for the above type would be equivalent to:

```
>>> MyType(["a", "b", "c"])._static_cache_key
(<class '__main__.MyType'>, ('choices', ('a', 'b', 'c')))
```

The caching scheme will extract attributes from the type that correspond to the names of parameters in the `__init__()` method. Above, the "choices" attribute becomes part of the cache key but "internal_only" does not, because there is no parameter named "internal_only".

The requirements for cacheable elements is that they are hashable and also that they indicate the same SQL rendered for expressions using this type every time for a given cache value.

To accommodate for datatypes that refer to unhashable structures such as dictionaries, sets and lists, these objects can be made "cacheable" by assigning hashable structures to the attributes whose names correspond with the names of the arguments. For example, a datatype which accepts a dictionary of lookup values may publish this as a sorted series of tuples. Given a previously un-cacheable type as:

```
class LookupType(UserDefinedType):
    '"a custom type that accepts a dictionary as a parameter.

    this is the non-cacheable version, as "self.lookup" is not
    hashable.

    '"

    def __init__(self, lookup):
        self.lookup = lookup

    def get_col_spec(self, **kw):
        return "VARCHAR(255)"

    def bind_processor(self, dialect):
        # ...  works with "self.lookup" ...
```

Where "lookup" is a dictionary. The type will not be able to generate a cache key:

```
>>> type_ = LookupType({"a": 10, "b": 20})
>>> type_._static_cache_key
<stdin>:1: SAWarning: UserDefinedType LookupType({'a': 10, 'b': 20}) will not
produce a cache key because the ``cache_ok`` flag is not set to True.
Set this flag to True if this type object's state is safe to use
in a cache key, or False to disable this warning.
symbol('no_cache')
```

If we **did** set up such a cache key, it wouldn't be usable. We would get a tuple structure that contains a dictionary inside of it, which cannot itself be used as a key in a "cache dictionary" such as SQLAlchemy's statement cache, since Python dictionaries aren't hashable:

```
>>> # set cache_ok = True
>>> type_.cache_ok = True

>>> # this is the cache key it would generate
>>> key = type_._static_cache_key
>>> key
(<class '__main__.LookupType'>, ('lookup', {'a': 10, 'b': 20}))

>>> # however this key is not hashable, will fail when used with
>>> # SQLAlchemy statement cache
>>> some_cache = {key: "some sql value"}
Traceback (most recent call last): File "<stdin>", line 1,
in <module> TypeError: unhashable type: 'dict'
```

The type may be made cacheable by assigning a sorted tuple of tuples to the ".lookup" attribute:

```
class LookupType(UserDefinedType):
    '"a custom type that accepts a dictionary as a parameter.

    The dictionary is stored both as itself in a private variable,
    and published in a public variable as a sorted tuple of tuples,
    which is hashable and will also return the same value for any
    two equivalent dictionaries.  Note it assumes the keys and
    values of the dictionary are themselves hashable.

    '"


    cache_ok = True

    def __init__(self, lookup):
        self._lookup = lookup

        # assume keys/values of "lookup" are hashable; otherwise
        # they would also need to be converted in some way here
        self.lookup = tuple(
            (key, lookup[key]) for key in sorted(lookup)
        )

    def get_col_spec(self, **kw):
        return "VARCHAR(255)"

    def bind_processor(self, dialect):
        # ...  works with "self._lookup" ...
```

Where above, the cache key for `LookupType({"a": 10, "b": 20})` will be:

```
>>> LookupType({"a": 10, "b": 20})._static_cache_key
(<class '__main__.LookupType'>, ('lookup', (('a', 10), ('b', 20))))
```

New in version 1.4.14: - added the `cache_ok` flag to allow some configurability of caching for `TypeDecorator` classes.

New in version 1.4.28: - added the `ExternalType` mixin which generalizes the `cache_ok` flag to both the `TypeDecorator` and `UserDefinedType` classes.

See also:

sql_caching

impl

alias of `sqlalchemy.sql.sqltypes.Unicode`

process_bind_param(*value*, *dialect*)

Expects a list of workflow module objects.

process_literal_param(*value*, *dialect*)

Expects a list of workflow module objects.

process_result_value(*value*, *dialect*)

Create workflow module list from JSON serialization.

class flowserv.model.base.**WorkflowParameters**(**args*, ***kwargs*)

Bases: sqlalchemy.sql.type_api.TypeDecorator

Decorator for workflow parameters that are stored as serialized Json objects.

cache_ok = True

Indicate if statements using this ExternalType are "safe to cache".

The default value None will emit a warning and then not allow caching of a statement which includes this type. Set to False to disable statements using this type from being cached at all without a warning. When set to True, the object's class and selected elements from its state will be used as part of the cache key. For example, using a TypeDecorator:

```
class MyType(TypeDecorator):
    impl = String

    cache_ok = True

    def __init__(self, choices):
        self.choices = tuple(choices)
        self.internal_only = True
```

The cache key for the above type would be equivalent to:

```
>>> MyType(["a", "b", "c"])._static_cache_key
(<class '__main__.MyType'>, ('choices', ('a', 'b', 'c')))
```

The caching scheme will extract attributes from the type that correspond to the names of parameters in the __init__() method. Above, the "choices" attribute becomes part of the cache key but "internal_only" does not, because there is no parameter named "internal_only".

The requirements for cacheable elements is that they are hashable and also that they indicate the same SQL rendered for expressions using this type every time for a given cache value.

To accommodate for datatypes that refer to unhashable structures such as dictionaries, sets and lists, these objects can be made "cacheable" by assigning hashable structures to the attributes whose names correspond with the names of the arguments. For example, a datatype which accepts a dictionary of lookup values may publish this as a sorted series of tuples. Given a previously un-cacheable type as:

```
class LookupType(UserDefinedType):
    '"a custom type that accepts a dictionary as a parameter.

    this is the non-cacheable version, as "self.lookup" is not
    hashable.

    '"

    def __init__(self, lookup):
        self.lookup = lookup

    def get_col_spec(self, **kw):
        return "VARCHAR(255)"

    def bind_processor(self, dialect):
        # ...  works with "self.lookup" ...
```

Where "lookup" is a dictionary. The type will not be able to generate a cache key:

```
>>> type_ = LookupType({"a": 10, "b": 20})
>>> type_._static_cache_key
<stdin>:1: SAWarning: UserDefinedType LookupType({'a': 10, 'b': 20}) will not
produce a cache key because the ``cache_ok`` flag is not set to True.
Set this flag to True if this type object's state is safe to use
in a cache key, or False to disable this warning.
symbol('no_cache')
```

If we **did** set up such a cache key, it wouldn't be usable. We would get a tuple structure that contains a dictionary inside of it, which cannot itself be used as a key in a "cache dictionary" such as SQLAlchemy's statement cache, since Python dictionaries aren't hashable:

```
>>> # set cache_ok = True
>>> type_.cache_ok = True

>>> # this is the cache key it would generate
>>> key = type_._static_cache_key
>>> key
(<class '__main__.LookupType'>, ('lookup', {'a': 10, 'b': 20}))

>>> # however this key is not hashable, will fail when used with
>>> # SQLAlchemy statement cache
>>> some_cache = {key: "some sql value"}
Traceback (most recent call last): File "<stdin>", line 1,
in <module> TypeError: unhashable type: 'dict'
```

The type may be made cacheable by assigning a sorted tuple of tuples to the ".lookup" attribute:

```
class LookupType(UserDefinedType):
    '"a custom type that accepts a dictionary as a parameter.

    The dictionary is stored both as itself in a private variable,
    and published in a public variable as a sorted tuple of tuples,
    which is hashable and will also return the same value for any
    two equivalent dictionaries.  Note it assumes the keys and
    values of the dictionary are themselves hashable.

    '"

    cache_ok = True

    def __init__(self, lookup):
        self._lookup = lookup

        # assume keys/values of "lookup" are hashable; otherwise
        # they would also need to be converted in some way here
        self.lookup = tuple(
            (key, lookup[key]) for key in sorted(lookup)
        )

    def get_col_spec(self, **kw):
        return "VARCHAR(255)"
```

(continues on next page)

(continued from previous page)

```
    def bind_processor(self, dialect):
        # ...  works with "self._lookup" ...
```

Where above, the cache key for `LookupType({"a": 10, "b": 20})` will be:

```
>>> LookupType({"a": 10, "b": 20})._static_cache_key
(<class '__main__.LookupType'>, ('lookup', (('a', 10), ('b', 20))))
```

New in version 1.4.14: - added the `cache_ok` flag to allow some configurability of caching for `TypeDecorator` classes.

New in version 1.4.28: - added the `ExternalType` mixin which generalizes the `cache_ok` flag to both the `TypeDecorator` and `UserDefinedType` classes.

See also:

sql_caching

impl

alias of `sqlalchemy.sql.sqltypes.Unicode`

process_bind_param(*value*, *dialect*)

Expects a dictionary of parameter declarations.

process_literal_param(*value*, *dialect*)

Expects a dictionary of parameter declarations.

process_result_value(*value*, *dialect*)

Create parameter index from JSON serialization.

class flowserv.model.base.**WorkflowRankingRun**(***kwargs*)

Bases: `sqlalchemy.orm.decl_api.Base`

Identifier of a run that was input for a workflow's post-processing run. Maintains the run identifier and the ranking position. Note that this class does not provide direct access to the handles for the post-processing input runs.

rank

run_id

workflow

workflow_id

class flowserv.model.base.**WorkflowResultSchema**(**args*, ***kwargs*)

Bases: `sqlalchemy.sql.type_api.TypeDecorator`

Decorator for the workflow result schema that is stored as serialized Json object.

cache_ok = True

Indicate if statements using this `ExternalType` are "safe to cache".

The default value `None` will emit a warning and then not allow caching of a statement which includes this type. Set to `False` to disable statements using this type from being cached at all without a warning. When set to `True`, the object's class and selected elements from its state will be used as part of the cache key. For example, using a `TypeDecorator`:

```
class MyType(TypeDecorator):
    impl = String

    cache_ok = True

    def __init__(self, choices):
        self.choices = tuple(choices)
        self.internal_only = True
```

The cache key for the above type would be equivalent to:

```
>>> MyType(["a", "b", "c"])._static_cache_key
(<class '__main__.MyType'>, ('choices', ('a', 'b', 'c')))
```

The caching scheme will extract attributes from the type that correspond to the names of parameters in the `__init__()` method. Above, the "choices" attribute becomes part of the cache key but "internal_only" does not, because there is no parameter named "internal_only".

The requirements for cacheable elements is that they are hashable and also that they indicate the same SQL rendered for expressions using this type every time for a given cache value.

To accommodate for datatypes that refer to unhashable structures such as dictionaries, sets and lists, these objects can be made "cacheable" by assigning hashable structures to the attributes whose names correspond with the names of the arguments. For example, a datatype which accepts a dictionary of lookup values may publish this as a sorted series of tuples. Given a previously un-cacheable type as:

```
class LookupType(UserDefinedType):
    '"a custom type that accepts a dictionary as a parameter.

    this is the non-cacheable version, as "self.lookup" is not
    hashable.

    '"

    def __init__(self, lookup):
        self.lookup = lookup

    def get_col_spec(self, **kw):
        return "VARCHAR(255)"

    def bind_processor(self, dialect):
        # ...  works with "self.lookup" ...
```

Where "lookup" is a dictionary. The type will not be able to generate a cache key:

```
>>> type_ = LookupType({"a": 10, "b": 20})
>>> type_._static_cache_key
<stdin>:1: SAWarning: UserDefinedType LookupType({'a': 10, 'b': 20}) will not
produce a cache key because the ``cache_ok`` flag is not set to True.
Set this flag to True if this type object's state is safe to use
in a cache key, or False to disable this warning.
symbol('no_cache')
```

If we **did** set up such a cache key, it wouldn't be usable. We would get a tuple structure that contains a dictionary inside of it, which cannot itself be used as a key in a "cache dictionary" such as SQLAlchemy's

statement cache, since Python dictionaries aren't hashable:

```
>>> # set cache_ok = True
>>> type_.cache_ok = True

>>> # this is the cache key it would generate
>>> key = type_._static_cache_key
>>> key
(<class '__main__.LookupType'>, ('lookup', {'a': 10, 'b': 20}))

>>> # however this key is not hashable, will fail when used with
>>> # SQLAlchemy statement cache
>>> some_cache = {key: "some sql value"}
Traceback (most recent call last): File "<stdin>", line 1,
in <module> TypeError: unhashable type: 'dict'
```

The type may be made cacheable by assigning a sorted tuple of tuples to the ".lookup" attribute:

```
class LookupType(UserDefinedType):
    '"a custom type that accepts a dictionary as a parameter.

    The dictionary is stored both as itself in a private variable,
    and published in a public variable as a sorted tuple of tuples,
    which is hashable and will also return the same value for any
    two equivalent dictionaries.  Note it assumes the keys and
    values of the dictionary are themselves hashable.

    '"

    cache_ok = True

    def __init__(self, lookup):
        self._lookup = lookup

        # assume keys/values of "lookup" are hashable; otherwise
        # they would also need to be converted in some way here
        self.lookup = tuple(
            (key, lookup[key]) for key in sorted(lookup)
        )

    def get_col_spec(self, **kw):
        return "VARCHAR(255)"

    def bind_processor(self, dialect):
        # ...  works with "self._lookup" ...
```

Where above, the cache key for `LookupType({"a": 10, "b": 20})` will be:

```
>>> LookupType({"a": 10, "b": 20})._static_cache_key
(<class '__main__.LookupType'>, ('lookup', (('a', 10), ('b', 20))))
```

New in version 1.4.14: - added the `cache_ok` flag to allow some configurability of caching for `TypeDecorator` classes.

New in version 1.4.28: - added the `ExternalType` mixin which generalizes the `cache_ok` flag to both the

TypeDecorator and UserDefinedType classes.

See also:

sql_caching

impl

alias of sqlalchemy.sql.sqltypes.Unicode

process_bind_param(*value*, *dialect*)

Expects a workflow result schema object.

process_literal_param(*value*, *dialect*)

Expects a workflow result schema object.

process_result_value(*value*, *dialect*)

Create result schema from JSON serialization.

flowserv.model.base.**by_pos**(*msg*)

Helper to sort log messages by position.

flowserv.model.base.**group_member = Table('group_member', MetaData(), Column('group_id', String(length=32), ForeignKey('workflow_group.group_id'), table=<group_member>), Column('user_id', String(length=32), ForeignKey('api_user.user_id'), table=<group_member>), schema=None)**

Each API user has a unique internal identifier and a password. If the active flag is True the user is active, otherwise the user has registered but not been activated or the user been deleted. In either case an inactive user is not permitted to login). Each user has a unique name. This name is the identifier that is visible to the user and that is used for display

flowserv.model.constraint module

Collection of methods to check and enforce constraints that are currently defined for the objects that are stored in the database.

flowserv.model.constraint.**validate_identifier**(*identifier: str*) → bool

Validate the given identifier to ensure that: (i) it's length is between 1 and 32, and (ii) it only contains letters (A-Z), digites, or underscore.

If the idnentifier is None it is considered valid. If an invalid identifier is given a ValueError will be raised.

Returns True if the identifier is valid.

Parameters **identifier** (*string*) – Unique identifier string or None

Raises **ValueError** –

flowserv.model.constraint.**validate_name**(*name*)

Validate the given name. Raises an error if the given name violates the current constraints for names. The constraints are:

- no empty or missing names
- names can be at most 512 characters long

Parameters **name** (*string*) – Name that is being validated

Raises *flowserv.error.ConstraintViolationError* –

flowserv.model.database module

Wrapper for database connections. The wrapper is used to open database sessions as well as to create a fresh database.

class flowserv.model.database.**DB**(*connect_url: str*, *web_app: Optional[bool] = False*, *echo: Optional[bool] = False*)

Bases: object

Wrapper to establish a database connection and create the database schema.

init() → *flowserv.model.database.DB*

Create all tables in the database model schema. This will also register the default user. The password for the user is a random UUID since the default user is not expected to login (but be used only in open access policies).

session()

Create a new database session instance. The sessoin is wrapped by a context manager to properly manage the session scope.

Return type *flowserv.model.database.SessionScope*

flowserv.model.database.**SQLITE_DB**(*dirname: str*, *filename: Optional[str] = 'test.db'*)

Get connection Url for a databse file.

class flowserv.model.database.**SessionScope**(*session*)

Bases: object

Context manager for providing transactional scope around a series of database operations.

close()

Close the session.

open()

Get a reference to the database session object.

flowserv.model.database.**TEST_DB**(*dirname: str*, *filename: Optional[str] = 'test.db'*)

Get connection Url for a databse file.

flowserv.model.files module

Classes for files that are part of workflow run inputs and outputs. Information about these files that are maintained on storage volumes is also stored in the database.

This module also defines the folder structure on the storage volume for workflows and their associated resources.

The folder structure is currently as follows:

```
/workflows/                  : Base directory
    {workflow_id}            : Folder for individual workflow
        groups/              : Folder for workflow groups
            {group_id}       : Folder for individual group
                files/       : Uploaded files for workflow group
                    {file_id}: Folder for uploaded file
        runs/                : Folder for all workflow runs
            {run_id}         : Result files for individual runs
        static/
```

class flowserv.model.files.**FileHandle**(*name: str*, *mime_type: str*, *fileobj:* flowserv.volume.base.IOHandle)

Bases: *flowserv.volume.base.IOHandle*

Handle for a file that is stored in the database. Extends the file object with the base file name and the mime type.

The implementation is a wrapper around a file object to make the handle agnostic to the underlying storage mechanism.

open() → IO

Get an BytesIO buffer containing the file content. If the associated file object is a path to a file on disk the file is being read.

Return type io.BytesIO

Raises *flowserv.error.UnknownFileError* –

size() → int

Get size of the file in the number of bytes.

Return type int

flowserv.model.files.**group_uploaddir**(*workflow_id: str*, *group_id: str*) → str

Get base directory for files that are uploaded to a workflow group.

Parameters

- **workflow_id** (*string*) – Unique workflow identifier
- **group_id** (*string*) – Unique workflow group identifier

Return type string

flowserv.model.files.**io_file**(*data: Union[List, Dict]*, *format: Optional[str] = None*) → *flowserv.volume.base.IOBuffer*

Write simple text to given bytes buffer.

Parameters

- **data** (*list or dict*) – List of strings or dictionary.
- **format** (*string, default=None*) – Format identifier.

Return type *flowserv.volume.base.IOBuffer*

flowserv.model.files.**run_basedir**(*workflow_id: str*, *run_id: str*) → str

Get path to the base directory for all files that are maintained for a workflow run.

Parameters

- **workflow_id** (*string*) – Unique workflow identifier
- **run_id** (*string*) – Unique run identifier

Return type string

flowserv.model.files.**run_tmpdir**() → str

Get path to a temporary workflow run directory.

Return type string

flowserv.model.files.**workflow_basedir**(*workflow_id: str*) → str

Get base directory containing associated files for the workflow with the given identifier.

Parameters **workflow_id** (*string*) – Unique workflow identifier

Return type string

flowserv.model.files.**workflow_groupdir**(*workflow_id: str*, *group_id: str*) → str

Get base directory containing files that are associated with a workflow group.

Parameters

- **workflow_id** (*string*) – Unique workflow identifier
- **group_id** (*string*) – Unique workflow group identifier

Return type string

flowserv.model.files.**workflow_staticdir**(*workflow_id: str*) → str

Get base directory containing static files that are associated with a workflow template.

Parameters **workflow_id** (*string*) – Unique workflow identifier

Return type string

flowserv.model.group module

The workflow group manager provides functionality to create and maintain workflow groups. All information about groups is maintained in the underlying database.

class flowserv.model.group.**WorkflowGroupManager**(*session: sqlalchemy.orm.session.Session*, *fs:* flowserv.volume.base.StorageVolume, *users: Optional[*flowserv.model.user.UserManager*] = None*)

Bases: object

Manager for workflow groups that associate a set of users with a set of workflow runs. The manager provides functionality to interact with the underlying database for creating and maintaining workflow groups.

create_group(*workflow_id: str*, *name: str*, *parameters: List[*flowserv.model.parameter.base.Parameter*]*, *workflow_spec: Dict*, *user_id: Optional[str] = None*, *members: Optional[List[str]] = None*, *engine_config: Optional[Dict] = None*, *identifier: Optional[str] = None*)

Create a new group for a given workflow. Within each workflow, the names of groups are expected to be unique.

The workflow group may define additional parameters for the template. The full (modifued or original) parameter list is stored with the group together with the workflow specification.

A group may have a list of users that are members. Membership can be used to control which users are allowed to execute the associated workflow and to upload/view files. The user that creates the group, identified by user_id parameter, is always part of the initial list of group members.

If a list of members is given it is ensured that each identifier in the list references an existing user.

Parameters

- **workflow_id** (*string*) – Unique workflow identifier
- **name** (*string*) – Group name
- **user_id** (*string*) – Unique identifier of the user that created the group
- **parameters** (*list(flowserv.model.parameter.base.Parameter)*) – List of workflow template parameter declarations that may be specific to the group
- **workflow_spec** (*dict*) – Workflow specification
- **members** (*list(string), optional*) – Optional list of user identifiers for other group members

- **engine_config** (*dict, default=None*) – Optional configuration settings that will be used as the default when running a workflow.
- **identifier** (*string, default=None*) – Optional user-provided group identifier.

Return type *flowserv.model.base.GroupObject*

Raises

- *flowserv.error.ConstraintViolationError* –
- *flowserv.error.UnknownUserError* –

`delete_file`(*group_id*, *file_id*)

Delete uploaded group file with given identifier. Raises an error if the group or file does not exist.

Parameters

- **group_id** (*string*) – Unique group identifier
- **file_id** (*string*) – Unique file identifier

Raises

- *flowserv.error.UnknownWorkflowGroupError* –
- *flowserv.error.UnknownFileError* –

`delete_group`(*group_id*)

Delete the given workflow group and all associated resources.

Parameters **group_id** (*string*) – Unique group identifier

Raises *flowserv.error.UnknownWorkflowGroupError* –

`get_group`(*group_id*)

Get handle for the workflow group with the given identifier.

Parameters **group_id** (*string*) – Unique group identifier

Return type *flowserv.model.base.GroupObject*

Raises *flowserv.error.UnknownWorkflowGroupError* –

`get_uploaded_file`(*group_id: str*, *file_id: str*) → *flowserv.model.files.FileHandle*

Get handle for an uploaded group file with the given identifier. Raises an error if the group or the file does not exists.

Returns the file handle and an object that provides read access to the file contents. The object may either be the path to the file on disk or a IOHandle.

Parameters

- **group_id** (*string*) – Unique group identifier
- **file_id** (*string*) – Unique file identifier

Return type *flowserv.model.files.FileHandle*

Raises

- *flowserv.error.UnknownWorkflowGroupError* –
- *flowserv.error.UnknownFileError* –

list_groups(*workflow_id=None*, *user_id=None*)

Get a listing of group descriptors. If the user identifier is given, only those groups are returned that the user is a member of. If the workflow identifier is given, only groups for the given workflow are included.

Parameters

- **workflow_id** (*string, optional*) – Unique workflow identifier
- **user_id** (*string, optional*) – Unique user identifier

Return type list(*flowserv.model.base.GroupObject*)

list_uploaded_files(*group_id*)

Get list of file handles for all files that have been uploaded to a given workflow group.

Parameters **group_id** (*string*) – Unique group identifier

Return type list(*flowserv.model.base.UploadFile*)

Raises ***flowserv.error.UnknownWorkflowGroupError*** –

update_group(*group_id*, *name=None*, *members=None*)

Update the name and/or list of members for a workflow group.

Parameters

- **group_id** (*string*) – Unique group identifier
- **name** (*string, optional*) – Unique user identifier
- **members** (*list(string), optional*) – List of user identifier for group members

Return type *flowserv.model.base.GroupObject*

Raises

- ***flowserv.error.ConstraintViolationError*** –
- ***flowserv.error.UnknownUserError*** –
- ***flowserv.error.UnknownWorkflowGroupError*** –

upload_file(*group_id: str*, *file:* flowserv.volume.base.IOHandle, *name: str*)

Upload a new file for a workflow group. This will create a copy of the given file in the file store that is associated with the group. The file will be places in a unique folder inside the groups upload folder.

Raises an error if the given file name is invalid.

Parameters

- **group_id** (*string*) – Unique group identifier
- **file** (*flowserv.volume.base.IOHandle*) – File object (e.g., uploaded via HTTP request)
- **name** (*string*) – Name of the file

Return type *flowserv.model.base.UploadFile*

Raises

- ***flowserv.error.ConstraintViolationError*** –
- ***flowserv.error.UnknownWorkflowGroupError*** –

flowserv.model.ranking module

This module contains the ranking manager that is responsible for maintaining and querying analytics results for individual workflows.

class flowserv.model.ranking.**RankingManager**(*session*)

Bases: object

The ranking manager maintains leader boards for individual workflows. Analytics results for each workflow are maintaind in separate tables. The schema of thoses tables is defined by the result schema of the respective workflow template.

get_ranking(*workflow*, *order_by=None*, *include_all=False*)

Query the underlying database to retrieve a result ranking for a given workflow.

Parameters

- **workflow** (*flowserv.model.base.WorkflowObject*) – Handle for workflow.
- **order_by** (*list(flowserv.model.template.schema.SortColumn), optional*) – Use the given attribute to sort run results. If not given, the schema default sort order is used
- **include_all** (*bool, optional*) – Include at most one entry per group in the result if False

Return type list(*flowserv.model.ranking.RunResult*)

class flowserv.model.ranking.**RunResult**(*run_id*, *group_id*, *group_name*, *created_at*, *started_at*, *finished_at*, *values*)

Bases: object

Handle for analytics results of a successful workflow run. Maintains the run identifier, run timestamps, group information, and a dictionary containing the generated values. The elements in the dictionary are defined by the result schema in the respective workflow template.

exectime()

The execution time for the workflow run is the difference between the start timestamp and the end timestamp.

Return type datetime.timedelta

get(*name*)

Get the result value for the schema attribute with the given name.

Parameters **name** (*string*) – Name of attribute in the result schema.

Return type string, int, or float

flowserv.model.run module

The run manager is used to create, delete, query, and update information about workflow runs in an underlying database.

class flowserv.model.run.**RunManager**(*session: sqlalchemy.orm.session.Session*, *fs:* flowserv.volume.base.StorageVolume)

Bases: object

The run manager maintains workflow runs. It provides methods the create, delete, and retrieve runs. the manager also provides the functionality to update the state of workflow runs.

create_run(*workflow=None*, *group=None*, *arguments=None*, *runs=None*)

Create a new entry for a run that is in pending state. Returns a handle for the created run.

A run is either created for a group (i.e., a grop submission run) or for a workflow (i.e., a post-processing run). Only one of the two parameters is expected to be None.

Parameters

- **workflow** (*flowserv.model.base.WorkflowObject, default=None*) – Workflow handle if this is a post-processing run.
- **group** (*flowserv.model.base.GroupObject*) – Group handle if this is a group sumbission run.
- **arguments** (*list*) – List of argument values for parameters in the template.
- **runs** (*list(string), default=None*) – List of run identifier that define the input for a post-processing run.

Return type *flowserv.model.base.RunObject*

Raises

- **ValueError** –
- ***flowserv.error.MissingArgumentError*** –

delete_obsolete_runs(*date: str*, *state: Optional[str] = None*) → int

Delete all workflow runs that were created before the given date. The optional state parameter allows to further restrict the list of deleted runs to those that were created before the given date and that are in the give state.

Parameters

- **date** (*string*) – Filter for run creation date.
- **state** (*string, default=None*) – Filter for run state.

Return type int

delete_run(*run_id*)

Delete the entry for the given run from the underlying database.

Parameters **run_id** (*string*) – Unique run identifier

Raises ***flowserv.error.UnknownRunError*** –

get_run(*run_id: str*) → *flowserv.model.base.RunObject*

Get handle for the given run from the underlying database. Raises an error if the run does not exist.

Parameters **run_id** (*string*) – Unique run identifier

Return type *flowserv.model.base.RunObject*

Raises ***flowserv.error.UnknownRunError*** –

get_runarchive(*run_id: str*) → *flowserv.model.files.FileHandle*

Get tar archive containing all result files for a given workflow run. Raises UnknownRunError if the run is not in SUCCESS state.

Parameters **run_id** (*string*) – Unique run identifier.

Return type *flowserv.model.files.FileHandle*

Raises ***flowserv.error.UnknownRunError*** –

get_runfile(*run_id: str*, *file_id: Optional[str] = None*, *key: Optional[str] = None*) → *flowserv.model.files.FileHandle*

Get handle and file object for a given run result file. The file is either identified by the unique file identifier or the file key. Raises an error if the specified file does not exist.

Parameters

- **run_id** (*string*) – Unique run identifier.
- **file_id** (*string*) – Unique file identifier.

Return type *flowserv.model.files.FileHandle*

Raises

- ***flowserv.error.UnknownFileError*** –
- **ValueError** –

list_obsolete_runs(*date: str*, *state: Optional[str] = None*) → List[*flowserv.model.base.RunObject*]

List all workflow runs that were created before the given date. The optional state parameter allows to further restrict the list of returned runs to those that were created before the given date and that are in the give state.

Parameters

- **date** (*string*) – Filter for run creation date.
- **state** (*string, default=None*) – Filter for run state.

Return type list(*flowserv.model.base.RunObject*)

list_runs(*group_id*, *state=None*)

Get list of run handles for all runs that are associated with a given workflow group.

Parameters

- **group_id** (*string, optional*) – Unique workflow group identifier
- **state** (*string or list(string), default=None*) – Run state query. If given, only those runs that are in the given state(s) will be returned.

Return type list(*flowserv.model.base.RunObject*)

update_run(*run_id: str*, *state:* flowserv.model.workflow.state.WorkflowState, *runstore: Optional[*flowserv.volume.base.StorageVolume*] = None*)

Update the state of the given run. This method does check if the state transition is valid. Transitions are valid for active workflows, if the transition is (a) from pending to running or (b) to an inactive state. Invalid state transitions will raise an error.

For successful runs a reference to the storage volume and the run directory containing the result files has to be given.

Parameters

- **run_id** (*string*) – Unique identifier for the run
- **state** (*flowserv.model.workflow.state.WorkflowState*) – New workflow state
- **runstore** (*flowserv.volume.base.StorageVolume, default=None*) – Storage volume containing the run (result) files for a successful workflow run.

Return type *flowserv.model.base.RunObject*

Raises

- ***flowserv.error.ConstraintViolationError*** –

- *flowserv.error.UnknownRunError* –

flowserv.model.run.**read_run_results**(*run:* flowserv.model.base.RunObject, *schema:* flowserv.model.template.schema.ResultSchema, *runstore:* flowserv.volume.base.StorageVolume)

Read the run results from the result file that is specified in the workflow result schema. If the file is not found we currently do not raise an error.

Parameters

- **run** (*flowserv.model.base.RunObject*) – Handle for a workflow run.
- **schema** (*flowserv.model.template.schema.ResultSchema*) – Workflow result schema specification that contains the reference to the result file key.
- **runstore** (*flowserv.volume.base.StorageVolume*) – Storage volume containing the run (result) files for a successful workflow run.

flowserv.model.run.**store_run_files**(*run:* flowserv.model.base.RunObject, *files: List[str]*, *source:* flowserv.volume.base.StorageVolume, *target:* flowserv.volume.base.StorageVolume) → List[*flowserv.model.base.RunFile*]

Create list of output files for a successful run. The list of files depends on whether files are specified in the workflow specification or not. If files are specified only those files are included in the returned lists. Otherwise, all result files that are listed in the run state are returned.

Parameters

- **run** (*flowserv.model.base.RunObject*) – Handle for a workflow run.
- **files** (*list of string*) – List of result files for a successful workflow run.
- **source** (*flowserv.volume.base.StorageVolume*) – Storage volume containing the run (result) files for a successful workflow run.
- **target** (*flowserv.volume.base.StorageVolume*) – Storage volume for persiting run result files.

Return type list of RunObject, list of string

flowserv.model.run.**validate_state_transition**(*current_state: str*, *target_state: str*, *valid_states: List[str]*)

Validate that a transition from current state to target state is permitted. The list of valid state identifier determines the current states that are permitted to transition to the target state. If an invalid transition is detected an error is raised.

Parameters

- **current_state** (*str*) – Identifier for the current run state.
- **target_state** (*str*) – Identifier for the target workflow state.
- **valid_states** (*list of string*) – List of valid source states for the anticipated target state.

flowserv.model.user module

Base class to manage information about users that are registered with the API. Each user has a unique identifier and a display name. The identifier is generated automatically during user registration. It is used internally to reference a user.

The user manager contains methods to login and logout users. At login, the user is assigned an API key that can be used for authentication and authorization in requests to the API. Each key has a fixed lifespan after which it becomes invalid. If a user logs out the API key is invalidated immediately.

class flowserv.model.user.**UserManager**(*session*, *token_timeout: Optional[int] = 86400*)

Bases: object

The user manager registers new users and handles requests to reset a user password. The user manager also allows users to login and to logout. A user that is logged in has an API key associated with them. This key is valid until a timeout period has passed. When the user logs out the API key is invalidated. API keys are stored in an underlying database.

activate_user(*user_id*)

Activate the user with the given identifier. A user is active if the respective active flag in the underlying database is set to 1.

Parameters **user_id** (*string*) – Unique user identifier

Return type *flowserv.model.base.User*

Raises *flowserv.error.UnknownUserError* –

get_user(*user_id*, *active=None*)

Get handle for specified user. The active parameter allows to put an additional constraint on the value of the active property for the user.

Raises an unknown user error if no matching user exists.

Return type *flowserv.model.base.User*

Raises *flowserv.error.UnknownUserError* –

list_users(*prefix=None*)

Get a listing of registered users. The optional query string is used to filter users whose name starts with the given string.

Parameters **prefix** (*string, default=None*) – Prefix string to filter users based on their name.

Return type list(*flowserv.model.base.User*)

Raises *flowserv.error.UnauthenticatedAccessError* –

login_user(*username*, *password*)

Authorize a given user and assign an API key for them. If the user is unknown or the given credentials do not match those in the database an unknown user error is raised.

Returns the API key that has been associated with the user identifier.

Parameters

- **username** (*string*) – Unique name (i.e., email address) that the user provided when they registered
- **password** (*string*) – User password specified during registration (in plain text)

Return type *flowserv.model.base.User*

Raises *flowserv.error.UnknownUserError* –

`logout_user`(*api_key*)

Invalidate the API key for the given user. This will logout the user.

Parameters **user** (*flowserv.model.base.User*) – Handle for user that is being logged out

Return type *flowserv.model.base.User*

`register_user`(*username*, *password*, *verify=False*)

Create a new user for the given username. Raises an error if a user with that name already is registered. Returns the internal unique identifier for the created user.

The verify flag allows to create active or inactive users. An inactive user cannot login until they have been activated. This option is intended for scenarios where the user receives an email after they register that contains a verification/activation link to ensure that the provided email address is valid.

Parameters

- **username** (*string*) – User email address that is used as the username
- **password** (*string*) – Password used to authenticate the user
- **verify** (*bool, optional*) – Determines whether the created user is active or inactive

Return type *flowserv.model.base.User*

Raises

- *flowserv.error.ConstraintViolationError* –
- *flowserv.error.DuplicateUserError* –

`request_password_reset`(*username*)

Request a password reset for the user with a given name. Returns the request identifier that is required as an argument to reset the password. The result is always going to be the identifier string independently of whether a user with the given username is registered or not.

Invalidates all previous password reset requests for the user.

Parameters **username** (*string*) – User email that was provided at registration

Return type string

`reset_password`(*request_id*, *password*)

Reset the password for the user that made the given password reset request. Raises an error if no such request exists or if the request has timed out.

Parameters

- **request_id** (*string*) – Unique password reset request identifier
- **password** (*string*) – New user password

Return type *flowserv.model.base.User*

Raises

- *flowserv.error.ConstraintViolationError* –
- *flowserv.error.UnknownRequestError* –

`flowserv.model.user.validate_password`(*password*)

Validate a given password. Raises constraint violation error if an invalid password is given.

Currently, the only constraint for passwords is that they are not empty

Parameters **password** (*string*) – User password for authentication

Raises *flowserv.error.ConstraintViolationError* –

flowserv.service package

Subpackages

flowserv.service.files package

Submodules

flowserv.service.files.base module

Interface for the workflow user group files API component that defines methods to access, delete, and upload files for workflow groups.

class flowserv.service.files.base.**UploadFileService**

Bases: object

API component that provides methods to access, delete and upload files for workflow user groups.

abstract delete_file(*group_id: str*, *file_id: str*)

Delete file with given identifier that was previously uploaded.

Raises errors if the file or the workflow group does not exist or if the user is not authorized to delete the file.

Parameters

- **group_id** (*string*) – Unique workflow group identifier
- **file_id** (*string*) – Unique file identifier

Raises

- *flowserv.error.UnauthorizedAccessError* –
- *flowserv.error.UnknownFileError* –

abstract get_uploaded_file(*group_id: str*, *file_id: str*) → IO

Get handle for file with given identifier that was uploaded to the workflow group.

Currently we do allow downloads for non-submission members (i.e., the user identifier is optional).

Parameters

- **group_id** (*string*) – Unique workflow group identifier
- **file_id** (*string*) – Unique file identifier

Return type io.BytesIO

Raises

- *flowserv.error.UnauthorizedAccessError* –
- *flowserv.error.UnknownFileError* –
- *flowserv.error.UnknownWorkflowGroupError* –

abstract list_uploaded_files(*group_id: str*) → Dict

Get a listing of all files that have been uploaded for the given workflow group.

Parameters **group_id** (*string*) – Unique workflow group identifier

Return type dict

Raises

- *flowserv.error.UnauthorizedAccessError* –
- *flowserv.error.UnknownWorkflowGroupError* –

abstract upload_file(*group_id: str*, *file:* flowserv.volume.base.IOHandle, *name: str*) → Dict

Create a file for a given workflow group.

Parameters

- **group_id** (*string*) – Unique workflow group identifier
- **file** (*flowserv.volume.base.IOHandle*) – File object (e.g., uploaded via HTTP request)
- **name** (*string*) – Name of the file

Return type dict

Raises

- *flowserv.error.ConstraintViolationError* –
- *flowserv.error.UnauthorizedAccessError* –
- *flowserv.error.UnknownWorkflowGroupError* –

flowserv.service.files.local module

The workflow user group files API component provides methods to access, delete, and upload files for workflow groups.

class flowserv.service.files.local.**LocalUploadFileService**(*group_manager:* flowserv.model.group.WorkflowGroupManager, *auth:* flowserv.model.auth.Auth, *user_id: Optional[str] = None*, *serializer: Optional[*flowserv.view.files.UploadFileSerializer*] = None*)

Bases: *flowserv.service.files.base.UploadFileService*

API component that provides methods to access, delete and upload files for workflow user groups.

delete_file(*group_id: str*, *file_id: str*)

Delete file with given identifier that was previously uploaded.

Raises errors if the file or the workflow group does not exist or if the user is not authorized to delete the file.

Parameters

- **group_id** (*string*) – Unique workflow group identifier
- **file_id** (*string*) – Unique file identifier

Raises

- *flowserv.error.UnauthorizedAccessError* –

- *flowserv.error.UnknownFileError* –

get_uploaded_file(*group_id: str*, *file_id: str*) → IO

Get IO buffer for file with given identifier that was uploaded to the workflow group.

Parameters

- **group_id** (*string*) – Unique workflow group identifier
- **file_id** (*string*) – Unique file identifier
- **user_id** (*string, optional*) – Unique user identifier

Return type io.BytesIO

Raises

- *flowserv.error.UnauthorizedAccessError* –
- *flowserv.error.UnknownFileError* –
- *flowserv.error.UnknownWorkflowGroupError* –

get_uploaded_file_handle(*group_id: str*, *file_id: str*) → *flowserv.model.files.FileHandle*

Get handle for file with given identifier that was uploaded to the workflow group.

Returns the file handle for the uploaded file.

Parameters

- **group_id** (*string*) – Unique workflow group identifier
- **file_id** (*string*) – Unique file identifier
- **user_id** (*string, optional*) – Unique user identifier

Return type flowserv.model.base.files.FileHandle

Raises

- *flowserv.error.UnauthorizedAccessError* –
- *flowserv.error.UnknownFileError* –
- *flowserv.error.UnknownWorkflowGroupError* –

list_uploaded_files(*group_id: str*) → Dict

Get a listing of all files that have been uploaded for the given workflow group.

Parameters **group_id** (*string*) – Unique workflow group identifier

Return type dict

Raises

- *flowserv.error.UnauthorizedAccessError* –
- *flowserv.error.UnknownWorkflowGroupError* –

upload_file(*group_id: str*, *file:* flowserv.volume.base.IOHandle, *name: str*) → Dict

Create a file for a given workflow group.

Parameters

- **group_id** (*string*) – Unique workflow group identifier
- **file** (*flowserv.volume.base.IOHandle*) – File object (e.g., uploaded via HTTP request)
- **name** (*string*) – Name of the file

Return type dict

Raises

- *flowserv.error.ConstraintViolationError* –
- *flowserv.error.UnauthorizedAccessError* –
- *flowserv.error.UnknownWorkflowGroupError* –

flowserv.service.files.remote module

API service component that provides functionality to access, delete, and upload files at a remote RESTful API.

class flowserv.service.files.remote.**RemoteUploadFileService**(*descriptor:* flowserv.service.descriptor.ServiceDescriptor)

Bases: *flowserv.service.files.base.UploadFileService*

API component that provides methods to access, delete and upload files for workflow user groups at a remote RESTful API.

delete_file(*group_id: str*, *file_id: str*)

Delete file with given identifier that was previously uploaded.

Raises errors if the file or the workflow group does not exist or if the user is not authorized to delete the file.

Parameters

- **group_id** (*string*) – Unique workflow group identifier
- **file_id** (*string*) – Unique file identifier

Raises

- *flowserv.error.UnauthorizedAccessError* –
- *flowserv.error.UnknownFileError* –

get_uploaded_file(*group_id: str*, *file_id: str*) → IO

Get handle for file with given identifier that was uploaded to the workflow group.

Currently we do allow downloads for non-submission members (i.e., the user identifier is optional).

Parameters

- **group_id** (*string*) – Unique workflow group identifier
- **file_id** (*string*) – Unique file identifier

Return type io.BytesIO

Raises

- *flowserv.error.UnauthorizedAccessError* –
- *flowserv.error.UnknownFileError* –
- *flowserv.error.UnknownWorkflowGroupError* –

list_uploaded_files(*group_id: str*) → Dict

Get a listing of all files that have been uploaded for the given workflow group.

Parameters **group_id** (*string*) – Unique workflow group identifier

Return type dict

Raises

- *flowserv.error.UnauthorizedAccessError* –
- *flowserv.error.UnknownWorkflowGroupError* –

`upload_file`(*group_id: str*, *file:* flowserv.volume.base.IOHandle, *name: str*) → Dict

Create a file for a given workflow group.

Parameters

- **group_id** (*string*) – Unique workflow group identifier
- **file** (*flowserv.volume.base.IOHandle*) – File object (e.g., uploaded via HTTP request)
- **name** (*string*) – Name of the file

Return type dict

Raises

- *flowserv.error.ConstraintViolationError* –
- *flowserv.error.UnauthorizedAccessError* –
- *flowserv.error.UnknownWorkflowGroupError* –

flowserv.service.group package

Submodules

flowserv.service.group.base module

Base class for the workflow user group API component that provides methods to access, create, and manipulate workflow groups.

`class flowserv.service.group.base.`**`WorkflowGroupService`**

Bases: `object`

API component that provides methods to access and manipulate workflow groups.

`abstract create_group`(*workflow_id: str*, *name: str*, *members: Optional[List[str]] = None*, *parameters: Optional[List[*flowserv.model.parameter.base.Parameter*]] = None*, *identifier: Optional[str] = None*) → Dict

Create a new user group for a given workflow. Each group has a a unique name for the workflow, a group owner, and a list of additional group members.

Workflow groups also define variants of the original workflow template by allowing to specify a list of additional template parameters.

Parameters

- **workflow_id** (*string*) – Unique workflow identifier
- **name** (*string*) – Unique team name
- **members** (*list(string), default=None*) – List of user identifier for group members
- **parameters** (*list of flowserv.model.parameter.base.Parameter, default=None*) – Optional list of parameter declarations that are used to modify the template parameters for submissions of the created group.

- **engine_config** (*dict, default=None*) – Optional configuration settings that will be used as the default when running a workflow.
- **identifier** (*string, default=None*) – Optional user-provided group identifier.

Return type dict

abstract delete_group(*group_id: str*)

Delete a given workflow group and all associated runs and uploaded files. If the user is not a member of the group an unauthorized access error is raised.

Parameters **group_id** (*string*) – Unique workflow group identifier

abstract get_group(*group_id: str*) → Dict

Get handle for workflow group with the given identifier.

Parameters **group_id** (*string*) – Unique workflow group identifier

Return type dict

abstract list_groups(*workflow_id: Optional[str] = None*) → Dict

Get a listing of all workflow groups. The result contains only those groups that the user is a member of. If the workflow identifier is given as an additional filter, then the result contains a user's groups for that workflow only.

Parameters **workflow_id** (*string, optional*) – Unique workflow identifier

Return type dict

abstract update_group(*group_id: str*, *name: Optional[str] = None*, *members: Optional[List[str]] = None*) → Dict

Update the name for the workflow group with the given identifier.

Parameters

- **group_id** (*string*) – Unique workflow group identifier
- **name** (*string, optional*) – New workflow group name
- **members** (*list(string), optional*) – Modified list of team members

Return type dict

flowserv.service.group.local module

The workflow user group API component provides methods to access, create, and manipulate workflow groups.

class flowserv.service.group.local.**LocalWorkflowGroupService**(*group_manager:* flowserv.model.group.WorkflowGroupManager, *workflow_repo:* flowserv.model.workflow.manager.WorkflowManager, *backend:* flowserv.controller.base.WorkflowController, *run_manager:* flowserv.model.run.RunManager, *auth:* flowserv.model.auth.Auth, *user_id: Optional[str] = None*, *serializer: Optional[*flowserv.view.group.WorkflowGroupSerializer*] = None*)

Bases: *flowserv.service.group.base.WorkflowGroupService*

API component that provides methods to access and manipulate workflow groups.

create_group(*workflow_id: str*, *name: str*, *members: Optional[List[str]] = None*, *parameters: Optional[List[*flowserv.model.parameter.base.Parameter*]] = None*, *engine_config: Optional[Dict] = None*, *identifier: Optional[str] = None*) → Dict

Create a new user group for a given workflow. Each group has a a unique name for the workflow, a group owner, and a list of additional group members.

Workflow groups also define variants of the original workflow template by allowing to specify a list of additional template parameters.

Parameters

- **workflow_id** (*string*) – Unique workflow identifier
- **name** (*string*) – Unique team name
- **members** (*list(string), default=None*) – List of user identifier for group members
- **parameters** (*list of flowserv.model.parameter.base.Parameter, default=None*) – Optional list of parameter declarations that are used to modify the template parameters for submissions of the created group.
- **engine_config** (*dict, default=None*) – Optional configuration settings that will be used as the default when running a workflow.
- **identifier** (*string, default=None*) – Optional user-provided group identifier.

Return type dict

delete_group(*group_id: str*)

Delete a given workflow group and all associated runs and uploaded files. If the user is not a member of the group an unauthorized access error is raised.

Parameters **group_id** (*string*) – Unique workflow group identifier

get_group(*group_id: str*) → Dict

Get handle for workflow group with the given identifier.

Parameters **group_id** (*string*) – Unique workflow group identifier

Return type dict

list_groups(*workflow_id: Optional[str] = None*) → Dict

Get a listing of all workflow groups. The result contains only those groups that the user is a member of. If the workflow identifier is given as an additional filter, then the result contains a user's groups for that workflow only.

Parameters **workflow_id** (*string, optional*) – Unique workflow identifier

Return type dict

update_group(*group_id: str*, *name: Optional[str] = None*, *members: Optional[List[str]] = None*) → Dict

Update the name for the workflow group with the given identifier.

Parameters

- **group_id** (*string*) – Unique workflow group identifier
- **name** (*string, optional*) – New workflow group name
- **members** (*list(string), optional*) – Modified list of team members

Return type dict

flowserv.service.group.remote module

Implementation of the group API component that provides access to workflow user groups via a RESTful API.

class flowserv.service.group.remote.**RemoteWorkflowGroupService**(*descriptor:* flowserv.service.descriptor.ServiceDescriptor, *labels: Optional[Dict] = None*)

Bases: *flowserv.service.group.base.WorkflowGroupService*

API component that provides methods to access and manipulate workflow groups via a RESTful API.

create_group(*workflow_id: str*, *name: str*, *members: Optional[List[str]] = None*, *parameters: Optional[List[*flowserv.model.parameter.base.Parameter*]] = None*, *engine_config: Optional[Dict] = None*, *identifier: Optional[str] = None*) → Dict

Create a new user group for a given workflow. Each group has a a unique name for the workflow, and an optional list of additional group members. The group owener will be the authenticated user based on the access token that is provided in the request header.

Workflow groups also define variants of the original workflow template by allowing to specify a list of additional template parameters.

Parameters

- **workflow_id** (*string*) – Unique workflow identifier
- **name** (*string*) – Unique team name
- **members** (*list(string), default=None*) – List of user identifier for group members
- **parameters** (*list of flowserv.model.parameter.base.Parameter, default=None*) – Optional list of parameter declarations that are used to modify the template parameters for submissions of the created group.
- **engine_config** (*dict, default=None*) – Optional configuration settings that will be used as the default when running a workflow.
- **identifier** (*string, default=None*) – Optional user-provided group identifier.

Return type dict

delete_group(*group_id: str*)

Delete a given workflow group and all associated runs and uploaded files. If the user is not a member of the group an unauthorized access error is raised.

Parameters **group_id** (*string*) – Unique workflow group identifier

get_group(*group_id: str*) → Dict

Get handle for workflow group with the given identifier.

Parameters **group_id** (*string*) – Unique workflow group identifier

Return type dict

list_groups(*workflow_id: Optional[str] = None*) → Dict

Get a listing of all workflow groups. The result contains only those groups that the user is a member of. If the workflow identifier is given as an additional filter, then the result contains a user's groups for that workflow only.

Parameters **workflow_id** (*string, optional*) – Unique workflow identifier

Return type dict

update_group(*group_id: str*, *name: Optional[str] = None*, *members: Optional[List[str]] = None*) → Dict

Update the name for the workflow group with the given identifier.

Parameters

- **group_id** (*string*) – Unique workflow group identifier
- **name** (*string, optional*) – New workflow group name
- **members** (*list(string), optional*) – Modified list of team members

Return type dict

flowserv.service.postproc package

Submodules

flowserv.service.postproc.base module

Definitions of constants for post=processing workflows. This module also contains helper functions that prepare the input data for post-porcessing workflows.

flowserv.service.postproc.base.**LABEL_FILES = 'files'**

Fixed set of parameter declarations for post-processing workflows. Contains only the declaration for the runs folder.

flowserv.service.postproc.base.**RUNS_FILE = 'runs.json'**

Labels for metadata objects in the run listing.

flowserv.service.postproc.base.**prepare_postproc_data**(*input_files: List[str]*, *ranking: List[*flowserv.model.ranking.RunResult*]*, *run_manager:* flowserv.model.run.RunManager, *store:* flowserv.volume.base.StorageVolume)

Create input files for post-processing steps for a given set of runs.

Creates files for a post-processing run in a given base directory on a storage volume. The resulting directory contains files for each run in a given ranking. For each run a sub-folder with the run identifier as the directory name is created. Each folder contains copies of result files for the run for those files that are specified in the input files list. A file `runs.json` in the base directory lists the runs in the ranking together with their group name.

Parameters

- **input_files** (*list(string)*) – List of identifier for benchmark run output files that are copied into the input directory for each submission.
- **ranking** (*list(flowserv.model.ranking.RunResult)*) – List of runs in the current result ranking
- **run_manager** (*flowserv.model.run.RunManager*) – Manager for workflow runs
- **store** (*flowserv.volume.base.StorageVolume*) – Target storage volume where the created post-processing files are stored.

flowserv.service.postproc.client module

Helper classes for post-processing client that need to access the run results that are included in the result folder that is provided as input to post-processing workflows.

class flowserv.service.postproc.client.**Run**(*run_id*, *name*, *files*)

Bases: object

Run result object containing the unique run identifier, the run name, and the list of resource files that were generated by the run and that are available to the post-processing workflow.

get_file(*name*)

Get the path to the run resource file with the given name.

Parameters **name** (*string*) – Unique file name

Return type string

class flowserv.service.postproc.client.**Runs**(*basedir*)

Bases: object

List of run result handles that are available to the post-processing workflow. The order of runs in the list reflects the order in the result ranking.

at_rank(*index*)

Get result handle for run at the given rank.

Parameters **index** (*int*)

Return type *flowserv.service.postproc.client.Run*

get_run(*run_id*)

Get result handle for the run with the given identifier. Returns None if the run identifier is unknown.

Parameters **run_id** (*string*) – Unique run identifier

Return type *flowserv.service.postproc.client.Run*

flowserv.service.run package

Submodules

flowserv.service.run.argument module

Helper functions for run arguments.

flowserv.service.run.argument.**deserialize_arg**(*doc: Dict*) → Tuple[str, Any]

Get the parameter name and argument value from a run argument.

Parameters **doc** (*dict*) – User-provided argument value for a run parameter.

Return type string, any

flowserv.service.run.argument.**deserialize_fh**(*doc: Dict*) → Tuple[str, str]

Get the file identifier and optional target path from an input file argument.

Parameters **doc** (*dict*) – Input file argument value.

Return type string, string

flowserv.service.run.argument.**is_fh**(*value: Any*) → bool

Check whether an argument value is a serialization of an input file. Expects a dictionary with the following schema:

{'type': '$file', 'value': {'fileId': 'string', 'targetPath': 'string'}}

The target path is optional.

Parameters **value** (*any*) – User provided argument value.

Return type bool

flowserv.service.run.argument.**serialize_arg**(*name: str*, *value: Any*) → Dict

Get serialization for a run argument.

Parameters

- **name** (*string*) – Unique parameter identifier.
- **value** (*any*) – Argument value.

Return type dict

flowserv.service.run.argument.**serialize_fh**(*file_id: str*, *target: Optional[str] = None*) → Dict

Get the file identifier and optional target path from an input file argument.

Parameters **doc** (*dict*) – Input file argument value.

Return type dict

flowserv.service.run.base module

Interface for the API service component provides methods that execute, access, and manipulate workflow runs and their results.

class flowserv.service.run.base.**RunService**

Bases: object

API component that provides methods to start, access, and manipulate workflow runs and their resources.

abstract cancel_run(*run_id: str*, *reason: Optional[str] = None*) → Dict

Cancel the run with the given identifier. Returns a serialization of the handle for the canceled run.

Raises an unauthorized access error if the user does not have the necessary access rights to cancel the run.

Parameters

- **run_id** (*string*) – Unique run identifier
- **reason** (*string, optional*) – Optional text describing the reason for cancelling the run

Return type dict

abstract delete_run(*run_id: str*) → Dict

Delete the run with the given identifier.

Raises an unauthorized access error if the user does not have the necessary access rights to delete the run.

Parameters

- **run_id** (*string*) – Unique run identifier
- **flowserv.error.InvalidRunStateError**

abstract get_result_archive(*run_id: str*) → IO

Get compressed tar-archive containing all result files that were generated by a given workflow run. If the run is not in sucess state a unknown resource error is raised.

Raises an unauthorized access error if the user does not have read access to the run.

Parameters **run_id** (*string*) – Unique run identifier

Return type io.BytesIO

abstract get_result_file(*run_id: str*, *file_id: str*) → IO

Get file handle for a resource file that was generated as the result of a successful workflow run.

Raises an unauthorized access error if the user does not have read access to the run.

Parameters

- **run_id** (*string*) – Unique run identifier.
- **file_id** (*string*) – Unique result file identifier.

Return type *flowserv.model.files.FileHandle*

abstract get_run(*run_id: str*) → Dict

Get handle for the given run.

Raises an unauthorized access error if the user does not have read access to the run.

Parameters **run_id** (*string*) – Unique run identifier

Return type dict

abstract list_runs(*group_id: str*, *state: Optional[str] = None*)

Get a listing of all run handles for the given workflow group.

Raises an unauthorized access error if the user does not have read access to the workflow group.

Parameters

- **group_id** (*string*) – Unique workflow group identifier
- **state** (*string, default=None*) – State identifier query

Return type dict

abstract start_run(*group_id: str*, *arguments: List[Dict]*, *config: Optional[Dict] = None*) → Dict

Start a new workflow run for the given group. The user provided arguments are expected to be a list of (key,value)-pairs. The key value identifies the template parameter. The data type of the value depends on the type of the parameter.

Returns a serialization of the handle for the started run.

Raises an unauthorized access error if the user does not have the necessary access to modify the workflow group.

Parameters

- **group_id** (*string*) – Unique workflow group identifier
- **arguments** (*list(dict)*) – List of user provided arguments for template parameters.
- **config** (*dict, default=None*) – Optional implementation-specific configuration settings that can be used to overwrite settings that were initialized at object creation.

Return type dict

flowserv.service.run.local module

The workflow run API component provides methods that execute, access, and manipulate workflow runs and their results. The local run service accesses all resources directly via a the database model.

class flowserv.service.run.local.**LocalRunService**(*run_manager:* flowserv.model.run.RunManager, *group_manager:* flowserv.model.group.WorkflowGroupManager, *ranking_manager:* flowserv.model.ranking.RankingManager, *backend:* flowserv.controller.base.WorkflowController, *fs:* flowserv.volume.base.StorageVolume, *auth:* flowserv.model.auth.Auth, *user_id: Optional[str] = None*, *serializer: Optional[*flowserv.view.run.RunSerializer*] = None*)

Bases: *flowserv.service.run.base.RunService*

API component that provides methods to start, access, and manipulate workflow runs and their resources. Uses the database model classes to access and manipulate the database state.

cancel_run(*run_id: str*, *reason: Optional[str] = None*) → Dict

Cancel the run with the given identifier. Returns a serialization of the handle for the canceled run.

Raises an unauthorized access error if the user does not have the necessary access rights to cancel the run.

Parameters

- **run_id** (*string*) – Unique run identifier
- **reason** (*string, optional*) – Optional text describing the reason for cancelling the run

Return type dict

Raises

- *flowserv.error.UnauthorizedAccessError* –
- *flowserv.error.UnknownRunError* –
- *flowserv.error.InvalidRunStateError* –

delete_run(*run_id: str*) → Dict

Delete the run with the given identifier.

Raises an unauthorized access error if the user does not have the necessary access rights to delete the run.

Parameters **run_id** (*string*) – Unique run identifier

Raises

- *flowserv.error.UnauthorizedAccessError* –
- *flowserv.error.UnknownRunError* –
- *flowserv.error.InvalidRunStateError* –

get_result_archive(*run_id: str*) → *flowserv.model.files.FileHandle*

Get compressed tar-archive containing all result files that were generated by a given workflow run. If the run is not in sucess state a unknown resource error is raised.

Raises an unauthorized access error if the user does not have read access to the run.

Parameters **run_id** (*string*) – Unique run identifier

Return type *flowserv.model.files.FileHandle*

Raises

- ***flowserv.error.UnauthorizedAccessError*** –
- ***flowserv.error.UnknownRunError*** –
- ***flowserv.error.UnknownFileError*** –

get_result_file(*run_id: str*, *file_id: str*) → *flowserv.model.files.FileHandle*

Get file handle for a resource file that was generated as the result of a successful workflow run.

Raises an unauthorized access error if the user does not have read access to the run.

Parameters

- **run_id** (*string*) – Unique run identifier.
- **file_id** (*string*) – Unique result file identifier.

Return type *flowserv.model.files.FileHandle*

Raises

- ***flowserv.error.UnauthorizedAccessError*** –
- ***flowserv.error.UnknownRunError*** –
- ***flowserv.error.UnknownFileError*** –

get_run(*run_id: str*) → Dict

Get handle for the given run.

Raises an unauthorized access error if the user does not have read access to the run.

Parameters **run_id** (*string*) – Unique run identifier

Return type dict

Raises

- ***flowserv.error.UnauthorizedAccessError*** –
- ***flowserv.error.UnknownRunError*** –

list_runs(*group_id: str*, *state: Optional[str] = None*)

Get a listing of all run handles for the given workflow group.

Raises an unauthorized access error if the user does not have read access to the workflow group.

Parameters **group_id** (*string*) – Unique workflow group identifier

Return type dict

Raises

- ***flowserv.error.UnauthorizedAccessError*** –
- ***flowserv.error.UnknownWorkflowGroupError*** –

start_run(*group_id: str*, *arguments: List[Dict]*, *config: Optional[Dict] = None*) → Dict

Start a new workflow run for the given group. The user provided arguments are expected to be a list of (name,value)-pairs. The name identifies the template parameter. The data type of the value depends on the type of the parameter.

Returns a serialization of the handle for the started run.

Raises an unauthorized access error if the user does not have the necessary access to modify the workflow group.

Parameters

- **group_id** (*string*) – Unique workflow group identifier
- **arguments** (*list(dict)*) – List of user provided arguments for template parameters.
- **config** (*dict, default=None*) – Optional implementation-specific configuration settings that can be used to overwrite settings that were initialized at object creation.

Return type dict

Raises

- *flowserv.error.InvalidArgumentError* –
- *flowserv.error.MissingArgumentError* –
- *flowserv.error.UnauthorizedAccessError* –
- *flowserv.error.UnknownFileError* –
- *flowserv.error.UnknownParameterError* –
- *flowserv.error.UnknownWorkflowGroupError* –

update_run(*run_id: str*, *state:* flowserv.model.workflow.state.WorkflowState, *runstore: Optional[*flowserv.volume.base.StorageVolume*] = None*)

Update the state of the given run. For runs that are in a SUCCESS state the workflow evaluation ranking is updated (if a result schema is defined for the corresponding template). If the ranking results change, an optional post-processing step is executed (synchronously). These changes occur before the state of the workflow is updated in the underlying database.

All run result files are maintained in a engine-specific storage volume before being moved to the file storage. For runs that are incative the `runstore` parameter is expected to reference the run folder.

Parameters

- **run_id** (*string*) – Unique identifier for the run
- **state** (*flowserv.model.workflow.state.WorkflowState*) – New workflow state.
- **runstore** (*flowserv.volume.base.StorageVolume, default=None*) – Storage volume containing the run (result) files for a successful workflow run.

Raises *flowserv.error.ConstraintViolationError* –

flowserv.service.run.local.**run_postproc_workflow**(*workflow:* flowserv.model.base.WorkflowObject, *ranking: List[*flowserv.model.ranking.RunResult*]*, *keys: List[str]*, *run_manager:* flowserv.model.run.RunManager, *tmpstore:* flowserv.volume.base.StorageVolume, *staticfs:* flowserv.volume.base.StorageVolume, *backend:* flowserv.controller.base.WorkflowController)

Run post-processing workflow for a workflow template.

Parameters

- **workflow** (*flowserv.model.base.WorkflowObject*) – Handle for the workflow that triggered the post-processing workflow run.
- **ranking** (*list(flowserv.model.ranking.RunResult)*) – List of runs in the current result ranking.

- **keys** (*list of string*) – Sorted list of run identifier for runs in the ranking.
- **run_manager** (*flowserv.model.run.RunManager*) – Manager for workflow runs
- **tmpstore** (*flowserv.volume.base.StorageVolume*) – Temporary storage volume where the created post-processing files are stored. This volume will be erased after the workflow is started.
- **staticfs** (*flowserv.volume.base.StorageVolume*) – Storage volume that contains the static files from the workflow template.
- **backend** (*flowserv.controller.base.WorkflowController*) – Backend that is used to execute the post-processing workflow.

flowserv.service.run.remote module

Implementation for the API service component provides methods that execute, access, and manipulate workflow runs and their results. The remote service API provides access to run resources at a RESTful API.

`class flowserv.service.run.remote.RemoteRunService(descriptor: flowserv.service.descriptor.ServiceDescriptor, labels: Optional[Dict] = None)`

Bases: *flowserv.service.run.base.RunService*

API component that provides methods to start, access, and manipulate workflow runs and their resources at a remote RESTful API.

`cancel_run(run_id: str, reason: Optional[str] = None)` → Dict

Cancel the run with the given identifier. Returns a serialization of the handle for the canceled run.

Raises an unauthorized access error if the user does not have the necessary access rights to cancel the run.

Parameters

- **run_id** (*string*) – Unique run identifier
- **reason** (*string, optional*) – Optional text describing the reason for cancelling the run

Return type dict

`delete_run(run_id: str)` → Dict

Delete the run with the given identifier.

Raises an unauthorized access error if the user does not have the necessary access rights to delete the run.

Parameters **run_id** (*string*) – Unique run identifier

`get_result_archive(run_id: str)` → IO

Get compressed tar-archive containing all result files that were generated by a given workflow run. If the run is not in sucess state a unknown resource error is raised.

Raises an unauthorized access error if the user does not have read access to the run.

Parameters **run_id** (*string*) – Unique run identifier

Return type io.BytesIO

`get_result_file(run_id: str, file_id: str)` → IO

Get file handle for a resource file that was generated as the result of a successful workflow run.

Raises an unauthorized access error if the user does not have read access to the run.

Parameters

- **run_id** (*string*) – Unique run identifier.
- **file_id** (*string*) – Unique result file identifier.

Return type *flowserv.model.files.FileHandle*

`get_run`(*run_id: str*) → Dict

Get handle for the given run.

Raises an unauthorized access error if the user does not have read access to the run.

Parameters **run_id** (*string*) – Unique run identifier

Return type dict

`list_runs`(*group_id: str*, *state: Optional[str] = None*)

Get a listing of all run handles for the given workflow group.

Raises an unauthorized access error if the user does not have read access to the workflow group.

Parameters

- **group_id** (*string*) – Unique workflow group identifier
- **state** (*string, default=None*) – State identifier query

Return type dict

`start_run`(*group_id: str*, *arguments: List[Dict]*) → Dict

Start a new workflow run for the given group. The user provided arguments are expected to be a list of (key,value)-pairs. The key value identifies the template parameter. The data type of the value depends on the type of the parameter.

Returns a serialization of the handle for the started run.

Raises an unauthorized access error if the user does not have the necessary access to modify the workflow group.

Parameters

- **group_id** (*string*) – Unique workflow group identifier
- **arguments** (*list(dict)*) – List of user provided arguments for template parameters.

Return type dict

flowserv.service.user package

Submodules

flowserv.service.user.base module

Interface for API methods that interact with the user manager.

`class flowserv.service.user.base.UserService`

Bases: `object`

Specification of methods that handle user login and logout as well as registration and activation of new users.

abstract activate_user(*user_id: str*) → Dict

Activate a new user with the given identifier.

Parameters **user_id** (*string*) – Unique user name

Return type dict

abstract list_users(*query: Optional[str] = None*) → Dict

Get a listing of registered users. The optional query string is used to filter users whose name starts with the given string.

Parameters **query** (*string, default=None*) – Prefix string to filter users based on their name.

Return type dict

abstract login_user(*username: str*, *password: str*) → Dict

Get handle for user with given credentials. Raises error if the user is unknown or if invalid credentials are provided.

Parameters

- **username** (*string*) – Unique name of registered user
- **password** (*string*) – User password (in plain text)

Return type dict

abstract logout_user(*api_key: str*) → Dict

Logout given user.

Parameters **api_key** (*string*) – API key for user that is being logged out.

Return type dict

abstract register_user(*username: str*, *password: str*, *verify: Optional[bool] = False*) → Dict

Create a new user for the given username and password. Raises an error if a user with that name already exists or if the user name is ivalid (e.g., empty or too long).

Returns success object if user was registered successfully.

Parameters

- **username** (*string*) – User email address that is used as the username
- **password** (*string*) – Password used to authenticate the user
- **verify** (*bool, default=False*) – Determines whether the created user is active or inactive

Return type dict

abstract request_password_reset(*username: str*) → Dict

Request to reset the password for the user with the given name. The result contains a unique request identifier for the user to send along with their new password.

Parameters **username** (*string*) – Unique user login name

Return type dict

abstract reset_password(*request_id: str*, *password: str*) → Dict

Reset the password for the user that made the given password reset request. Raises an error if no such request exists or if the request has timed out.

Returns the serialization of the user handle.

Parameters

- **request_id** (*string*) – Unique password reset request identifier
- **password** (*string*) – New user password

Return type dict

abstract whoami_user(*api_key: str*) → Dict

Get serialization of the given user.

Parameters **api_key** (*string*) – API key for a logged-in user.

Return type dict

flowserv.service.user.local module

Implementation of user service API methods that access and manipulate user resources as well as access tokens. This implementation is intended for an API class that provides access to the flowserv objects directly via a local database connection.

class flowserv.service.user.local.**LocalUserService**(*manager:* flowserv.model.user.UserManager, *auth:* flowserv.model.auth.Auth, *serializer: Optional[*flowserv.view.user.UserSerializer*] = None*)

Bases: *flowserv.service.user.base.UserService*

Implement methods that handle user login and logout as well as registration and activation of new users.

activate_user(*user_id: str*) → Dict

Activate a new user with the given identifier.

Parameters **user_id** (*string*) – Unique user name

Return type dict

Raises **flowserv.model.base.error.UnknownUserError** –

list_users(*query: Optional[str] = None*) → Dict

Get a listing of registered users. The optional query string is used to filter users whose name starts with the given string.

Parameters **query** (*string, default=None*) – Prefix string to filter users based on their name.

Return type dict

login_user(*username: str*, *password: str*) → Dict

Get handle for user with given credentials. Raises error if the user is unknown or if invalid credentials are provided.

Parameters

- **username** (*string*) – Unique name of registered user
- **password** (*string*) – User password (in plain text)

Return type dict

Raises **flowserv.model.base.error.UnknownUserError** –

logout_user(*api_key: str*) → Dict

Logout given user.

Parameters **api_key** (*string*) – API key for user that is being logged out.

Return type dict

Raises *flowserv.error.UnauthenticatedAccessError* –

register_user(*username: str*, *password: str*, *verify: Optional[bool] = False*) → Dict

Create a new user for the given username and password. Raises an error if a user with that name already exists or if the user name is ivalid (e.g., empty or too long).

Returns success object if user was registered successfully.

Parameters

- **username** (*string*) – User email address that is used as the username
- **password** (*string*) – Password used to authenticate the user
- **verify** (*bool, default=False*) – Determines whether the created user is active or inactive

Return type dict

Raises

- *flowserv.error.ConstraintViolationError* –
- *flowserv.error.DuplicateUserError* –

request_password_reset(*username: str*) → Dict

Request to reset the password for the user with the given name. The result contains a unique request identifier for the user to send along with their new password.

Parameters **username** (*string*) – Unique user login name

Return type dict

reset_password(*request_id: str*, *password: str*) → Dict

Reset the password for the user that made the given password reset request. Raises an error if no such request exists or if the request has timed out.

Returns the serialization of the user handle.

Parameters

- **request_id** (*string*) – Unique password reset request identifier
- **password** (*string*) – New user password

Return type dict

Raises

- *flowserv.error.ConstraintViolationError* –
- *flowserv.error.UnknownRequestError* –

whoami_user(*api_key: str*) → Dict

Get serialization of the given user.

Parameters **api_key** (*string*) – API key for a logged-in user.

Return type dict

flowserv.service.user.remote module

Implementation of user service API methods that access and manipulate user resources as well as access tokens. This implementation provides access to flowserv objects via a remote RESTful API.

class flowserv.service.user.remote.**RemoteUserService**(*descriptor:* flowserv.service.descriptor.ServiceDescriptor, *labels: Optional[Dict] = None*)

Bases: *flowserv.service.user.base.UserService*

HTTP client for a RESTful API to access flowserv API resources.

activate_user(*user_id: str*) → Dict

Activate a new user with the given identifier.

Parameters **user_id** (*string*) – Unique user name

Return type dict

list_users(*query: Optional[str] = None*) → Dict

Get a listing of registered users. The optional query string is used to filter users whose name starts with the given string.

Parameters **query** (*string, default=None*) – Prefix string to filter users based on their name.

Return type dict

login_user(*username: str*, *password: str*) → Dict

Get handle for user with given credentials. Raises error if the user is unknown or if invalid credentials are provided.

Parameters

- **username** (*string*) – Unique name of registered user
- **password** (*string*) – User password (in plain text)

Return type dict

logout_user(*api_key: str*) → Dict

Logout given user.

Parameters **api_key** (*string*) – API key for user that is being logged out.

Return type dict

register_user(*username: str*, *password: str*, *verify: Optional[bool] = False*) → Dict

Create a new user for the given username and password. Raises an error if a user with that name already exists or if the user name is ivalid (e.g., empty or too long).

Returns success object if user was registered successfully.

Parameters

- **username** (*string*) – User email address that is used as the username
- **password** (*string*) – Password used to authenticate the user
- **verify** (*bool, default=False*) – Determines whether the created user is active or inactive

Return type dict

request_password_reset(*username: str*) → Dict

Request to reset the password for the user with the given name. The result contains a unique request identifier for the user to send along with their new password.

Parameters **username** (*string*) – Unique user login name

reset_password(*request_id: str*, *password: str*) → Dict

Reset the password for the user that made the given password reset request. Raises an error if no such request exists or if the request has timed out.

Returns the serialization of the user handle.

Parameters

- **request_id** (*string*) – Unique password reset request identifier
- **password** (*string*) – New user password

Return type dict

whoami_user(*api_key: str*) → Dict

Get serialization of the given user.

Parameters **api_key** (*string*) – API key for a logged-in user.

Return type dict

flowserv.service.workflow package

Submodules

flowserv.service.workflow.base module

Base class for the workflow API component that provides methods to create and access workflows and workflow result rankings.

class flowserv.service.workflow.base.**WorkflowService**

Bases: object

API component that provides methods to access workflows and workflow evaluation rankings (benchmark leader boards).

abstract get_ranking(*workflow_id: str*, *order_by: Optional[List[*flowserv.model.template.schema.SortColumn*]] = None*, *include_all: Optional[bool] = False*) → Dict

Get serialization of the evaluation ranking for the given workflow.

Parameters

- **workflow_id** (*string*) – Unique workflow identifier
- **order_by** (*list(flowserv.model.template.schema.SortColumn), default=None*) – Use the given attribute to sort run results. If not given, the schema default sort order is used
- **include_all** (*bool, default=False*) – Include all entries (True) or at most one entry (False) per user group in the returned ranking

Return type dict

abstract get_result_archive(*workflow_id: str*) → IO

Get compressed tar-archive containing all result files that were generated by the most recent post-processing workflow. If the workflow does not have a post-processing step or if the post-processing workflow run is not in SUCCESS state, a unknown resource error is raised.

Parameters **workflow_id** (*string*) – Unique workflow identifier

Return type io.BytesIO

abstract get_result_file(*workflow_id: str*, *file_id: str*) → IO

Get file handle for a file that was generated as the result of a successful post-processing workflow run.

Parameters

- **workflow_id** (*string*) – Unique workflow identifier
- **file_id** (*string*) – Unique resource file identifier

Return type *flowserv.model.files.FileHandle*

Raises

- ***flowserv.error.UnknownWorkflowError*** –
- ***flowserv.error.UnknownFileError*** –

abstract get_workflow(*workflow_id: str*) → Dict

Get serialization of the handle for the given workflow.

Parameters **workflow_id** (*string*) – Unique workflow identifier

Return type dict

abstract list_workflows() → Dict

Get serialized listing of descriptors for all workflows in the repository.

Return type dict

flowserv.service.workflow.local module

Workflow API component for a service that is running locally. The local API provides additional methods to create, delete and update a workflow. These functions are not available via the remote service API.

class flowserv.service.workflow.local.**LocalWorkflowService**(*workflow_repo: flowserv.model.workflow.manager.WorkflowManager*, *ranking_manager: flowserv.model.ranking.RankingManager*, *group_manager: flowserv.model.group.WorkflowGroupManager*, *run_manager: flowserv.model.run.RunManager*, *user_id: Optional[str] = None*, *serializer: Optional[flowserv.view.workflow.WorkflowSerializer] = None*)

Bases: ***flowserv.service.workflow.base.WorkflowService***

API component that provides methods to access workflows and workflow evaluation rankings (benchmark leader boards). The local API component extends the base class with functionality to create, delete, and update workflow templates.

create_workflow(*source: str*, *identifier: Optional[str] = None*, *name: Optional[str] = None*, *description: Optional[str] = None*, *instructions: Optional[str] = None*, *specfile: Optional[str] = None*, *manifestfile: Optional[str] = None*, *engine_config: Optional[Dict] = None*, *ignore_postproc: Optional[bool] = False*, *verbose: Optional[bool] = False*) → Dict

Create a new workflow in the repository. If the workflow template includes a result schema the workflow is also registered with the ranking manager.

Raises an error if the given workflow name is not unique.

Parameters

- **source** (*string*) – Path to local template, name or URL of the template in the repository.
- **name** (*string*) – Unique workflow name
- **description** (*string, default=None*) – Optional short description for display in workflow listings.
- **instructions** (*string, default=None*) – Text containing detailed instructions for running the workflow.
- **specfile** (*string, default=None*) – Path to the workflow template specification file (absolute or relative to the workflow directory).
- **manifestfile** (*string, default=None*) – Path to manifest file. If not given an attempt is made to read one of the default manifest file names in the base directory.
- **engine_config** (*dict, default=None*) – Optional configuration settings that will be used as the default when running the workflow and the post-processing workflow.
- **ignore_postproc** (*bool, default=False*) – Ignore post-processing workflow specification if True.
- **verbose** (*bool, default=False*) – Print information about source and target volume and the files that are being copied.

Return type dict

Raises

- ***flowserv.error.ConstraintViolationError*** –
- ***flowserv.error.InvalidTemplateError*** –
- **ValueError** –

delete_workflow(*workflow_id: str*)

Delete the workflow with the given identifier.

Parameters **workflow_id** (*string*) – Unique workflow identifier.

Raises ***flowserv.error.UnknownWorkflowError*** –

get_ranking(*workflow_id: str*, *order_by: Optional[List[*flowserv.model.template.schema.SortColumn*]] = None*, *include_all: Optional[bool] = False*) → Dict

Get serialization of the evaluation ranking for the given workflow. Returns None if the workflow does not have a result schema.

Parameters

- **workflow_id** (*string*) – Unique workflow identifier.
- **order_by** (*list(flowserv.model.template.schema.SortColumn), default=None*) – Use the given attribute to sort run results. If not given, the schema default sort order is used.

- **include_all** (*bool, default=False*) – Include all entries (True) or at most one entry (False) per user group in the returned ranking.

Return type dict

Raises *flowserv.error.UnknownWorkflowError* –

get_result_archive(*workflow_id: str*) → *flowserv.model.files.FileHandle*

Get compressed tar-archive containing all result files that were generated by the most recent post-processing workflow. If the workflow does not have a post-processing step or if the post-processing workflow run is not in SUCCESS state, a unknown resource error is raised.

Parameters **workflow_id** (*string*) – Unique workflow identifier.

Return type *flowserv.model.files.FileHandle*

Raises

- *flowserv.error.UnknownWorkflowError* –
- *flowserv.error.UnknownFileError* –

get_result_file(*workflow_id: str*, *file_id: str*) → *flowserv.model.files.FileHandle*

Get file handle for a file that was generated as the result of a successful post-processing workflow run.

Parameters

- **workflow_id** (*string*) – Unique workflow identifier.
- **file_id** (*string*) – Unique resource file identifier.

Return type *flowserv.model.files.FileHandle*

Raises

- *flowserv.error.UnknownWorkflowError* –
- *flowserv.error.UnknownFileError* –

get_workflow(*workflow_id: str*) → Dict

Get serialization of the handle for the given workflow.

Parameters **workflow_id** (*string*) – Unique workflow identifier.

Return type dict

Raises *flowserv.error.UnknownWorkflowError* –

list_workflows() → Dict

Get serialized listing of descriptors for all workflows in the repository.

Return type dict

update_workflow(*workflow_id: str*, *name: Optional[str] = None*, *description: Optional[str] = None*, *instructions: Optional[str] = None*) → Dict

Update name, description, and instructions for a given workflow. Returns the serialized handle for the updated workflow.

Raises an error if the given workflow does not exist or if the name is not unique.

Parameters

- **workflow_id** (*string*) – Unique workflow identifier.
- **name** (*string, default=None*) – Unique workflow name.

- **description** (*string, default=None*) – Optional short description for display in workflow listings.
- **instructions** (*string, default=None*) – Text containing detailed instructions for workflow execution.

Return type dict

Raises

- *flowserv.error.ConstraintViolationError* –
- *flowserv.error.UnknownWorkflowError* –

flowserv.service.workflow.remote module

Implementation for the API service component that provides access to workflow handles and ranking results. This implementation is for the service that accesses a remote RESTful API.

class flowserv.service.workflow.remote.**RemoteWorkflowService**(*descriptor:* flowserv.service.descriptor.ServiceDescriptor)

Bases: *flowserv.service.workflow.base.WorkflowService*

API component that provides methods to access workflows and workflow evaluation rankings (benchmark leader boards) via a RESTful API.

get_ranking(*workflow_id: str*, *order_by: Optional[List[*flowserv.model.template.schema.SortColumn*]] = None*, *include_all: Optional[bool] = False*) → Dict

Get serialization of the evaluation ranking for the given workflow.

Parameters

- **workflow_id** (*string*) – Unique workflow identifier
- **order_by** (*list(flowserv.model.template.schema.SortColumn), default=None*) – Use the given attribute to sort run results. If not given, the schema default sort order is used
- **include_all** (*bool, default=False*) – Include all entries (True) or at most one entry (False) per user group in the returned ranking

Return type dict

get_result_archive(*workflow_id: str*) → IO

Get compressed tar-archive containing all result files that were generated by a given workflow run. If the run is not in sucess state a unknown resource error is raised.

Raises an unauthorized access error if the user does not have read access to the run.

Parameters **run_id** (*string*) – Unique run identifier

Return type io.BytesIO

get_result_file(*workflow_id: str*, *file_id: str*) → IO

Get file handle for a resource file that was generated as the result of a successful workflow run.

Raises an unauthorized access error if the user does not have read access to the run.

Parameters

- **run_id** (*string*) – Unique run identifier.
- **file_id** (*string*) – Unique result file identifier.

Return type *flowserv.model.files.FileHandle*

get_workflow(*workflow_id: str*) → Dict

Get serialization of the handle for the given workflow.

Parameters **workflow_id** (*string*) – Unique workflow identifier

Return type dict

Raises ***flowserv.error.UnknownWorkflowError*** –

list_workflows() → Dict

Get serialized listing of descriptors for all workflows in the repository.

Return type dict

Submodules

flowserv.service.api module

Base class to access all API resources. Different types of resourecs are managed by different components of the API.

class flowserv.service.api.**API**(*service:* flowserv.service.descriptor.ServiceDescriptor, *workflow_service:* flowserv.service.workflow.base.WorkflowService, *group_service:* flowserv.service.workflow.base.WorkflowService, *upload_service:* flowserv.service.files.base.UploadFileService, *run_service:* flowserv.service.run.base.RunService, *user_service:* flowserv.service.user.base.UserService)

Bases: object

The API object is a wrapper for the following API components:

- groups()
- runs()
- service()
- uploads()
- users()
- workflows()

groups() → *flowserv.service.group.base.WorkflowGroupService*

Get API service component that provides functionality to access and manipulate workflows groups.

Return type flowserv.service.group.WorkflowGroupService

runs() → *flowserv.service.run.base.RunService*

Get API service component that provides functionality to access workflows runs.

Return type flowserv.service.run.RunService

server() → *flowserv.service.descriptor.ServiceDescriptor*

Get API component for the service descriptor.

Return type flowserv.service.server.Service

uploads() → *flowserv.service.files.base.UploadFileService*

Get API service component that provides functionality to access, delete, and upload files for workflows groups.

Return type flowserv.service.files.UploadFileService

users() → *flowserv.service.user.base.UserService*

Get instance of the user service component.

Return type flowserv.service.user.UserService

workflows() → *flowserv.service.workflow.base.WorkflowService*

Get API service component that provides functionality to access workflows and workflow leader boards.

Return type flowserv.service.workflow.WorkflowService

class flowserv.service.api.**APIFactory**(*defaults: Dict*)

Bases: *flowserv.controller.base.WorkflowController*, *flowserv.config.Config*

Factory pattern for creating API instances. Extends the workflow controller with a __call__ method that returns a context manager for creating new instances of either a local or remote service API.

login(*username: str*, *password: str*)

Authenticate the user using the given credentials. Updates the internal configuration with the returned access token.

logout()

Delete an access token from the internal configuration.

flowserv.service.descriptor module

API component that provides information about the service iteself.

class flowserv.service.descriptor.**ServiceDescriptor**(*doc: Optional[Dict] = None*, *routes: Optional[Dict] = {'files:delete': 'uploads/{userGroupId}/files/{fileId}', 'files:download': 'uploads/{userGroupId}/files/{fileId}', 'files:list': 'uploads/{userGroupId}/files', 'files:upload': 'uploads/{userGroupId}/files', 'groups:create': 'workflows/{workflowId}/groups', 'groups:delete': 'groups/{userGroupId}', 'groups:get': 'groups/{userGroupId}', 'groups:list': 'groups', 'groups:runs': 'groups/{userGroupId}/runs?state={state}', 'groups:update': 'groups/{userGroupId}', 'leaderboard': 'workflows/{workflowId}/leaderboard?orderBy={orderBy}&includeAll=*, 'runs:cancel': 'runs/{runId}', 'runs:delete': 'runs/{runId}', 'runs:download:archive': 'runs/{runId}/downloads/archive', 'runs:download:file': 'runs/{runId}/downloads/files/{fileId}', 'runs:get': 'runs/{runId}', 'runs:start': 'groups/{userGroupId}/runs', 'service': '', 'users:activate': 'users/activate', 'users:list': 'users', 'users:login': 'users/login', 'users:logout': 'users/logout', 'users:pwd:request': 'users/password/request', 'users:pwd:reset': 'users/password/reset', 'users:register': 'users/register', 'users:whoami': 'users/whoami', 'workflows:download:archive': 'workflows/{workflowId}/downloads/archive', 'workflows:download:file': 'workflows/{workflowId}/downloads/files/{fileId}', 'workflows:get': 'workflows/{workflowId}', 'workflows:groups': 'workflows/{workflowId}/groups', 'workflows:list': 'workflows'}*, *serializer: Optional[flowserv.view.descriptor.ServiceDescriptorSerializer] = None*)

Bases: `object`

API component that provides the API sevice descriptor that contains the basic information and supported route patterns for the service.

static from_config(*env: Dict*, *username: Optional[str] = None*) → *flowserv.service.descriptor.ServiceDescriptor*

Get descriptor with basic information from values in the given configuration settings.

Parameters

- **env** (*dict, default=None*) – Dictionary that provides access to configuration parameter values.
- **username** (*string, default=None*) – Optional name for an authenticated user.

Return type *flowserv.service.descriptor.ServiceDescriptor*

routes() → Dict

Get dictionary of supported API routes. The returned dictionary maps unique route identifiers their route Url pattern.

Return type dict

to_dict() → Dict

Get serialization of descriptor containing the basic information about the API. If the user provided a valid access token then the user name will be set and included in the serialized object. If no user name is present in the returned dictionary the user is not authenticated or authentication is not configured (i.e., open access).

Return type dict

urls(*key: str*, ***kwargs*) → str

Get the full Url for the route with the given key.

Parameters

- **key** (*string*) – Url route pattern key.
- **kwargs** (*dict*) – Optional key word arguments to replace Url pattern variables.

Return type string

flowserv.service.local module

Helper classes method to create instances of the API components. The local API operates directly on the flowserv database (as opposed to the remote API that interacts with a local API via a RESTfule API) and provides the ability to execute workflows on the local machine using an associated workflow engine.

All API components use the same underlying database connection. The connection object is under the control of of a context manager to ensure that the connection is closed properly after every API request has been handled.

class flowserv.service.local.**LocalAPIFactory**(*env: Optional[Dict] = None*, *db: Optional[*flowserv.model.database.DB*] = None*, *engine: Optional[*flowserv.controller.base.WorkflowController*] = None*, *user_id: Optional[str] = None*)

Bases: *flowserv.service.api.APIFactory*

Factory for context manager that create local API instances. Provides a wrapper around the database and the workflow engine.

cancel_run(*run_id: str*)

Request to cancel execution of the given run.

Parameters **run_id** (*string*) – Unique run identifier

Raises ***flowserv.error.UnknownRunError*** –

exec_workflow(*run:* flowserv.model.base.RunObject, *template:* flowserv.model.template.base.WorkflowTemplate, *arguments: Dict*, *staticfs:* flowserv.volume.base.StorageVolume, *config: Optional[Dict] = None*) → Tuple[*flowserv.model.workflow.state.WorkflowState*, *flowserv.volume.base.StorageVolume*]

Initiate the execution of a given workflow template for a set of argument values. Returns the state of the workflow and the path to the directory that contains run result files for successful runs.

The client provides a unique identifier for the workflow run that is being used to retrieve the workflow state in future calls.

If the state of the run handle is not pending, an error is raised.

Parameters

- **run** (*flowserv.model.base.RunObject*) – Handle for the run that is being executed.
- **template** (*flowserv.model.template.base.WorkflowTemplate*) – Workflow template containing the parameterized specification and the parameter declarations.
- **arguments** (*dict*) – Dictionary of argument values for parameters in the template.
- **staticfs** (*flowserv.volume.base.StorageVolume*) – Storage volume that contains the static files from the workflow template.
- **config** (*dict, default=None*) – Optional implementation-specific configuration settings that can be used to overwrite settings that were initialized at object creation.

Return type *flowserv.model.workflow.state.WorkflowState*, *flowserv.volume.base.StorageVolume*

`class flowserv.service.local.SessionManager(env: Dict, db: flowserv.model.database.DB, engine: flowserv.controller.base.WorkflowController, fs: flowserv.volume.base.StorageVolume, user_id: str, access_token: str)`

Bases: `object`

Context manager that creates a local API and controls the database session that is used by all the API components.

`flowserv.service.local.init_backend(api: flowserv.service.api.APIFactory) → flowserv.controller.base.WorkflowController`

Create an instance of the workflow engine based on the given configuration settings. The workflow engine receives a reference to the API factory for callback operations that modify the global database state.

Parameters

- **env** (*dict, default=None*) – Dictionary that provides access to configuration parameter values.
- **api** (*flowserv.service.api.APIFactory*) – Reference to tha API factory for callbacks that modify the global database state.

Return type *flowserv.controller.base.WorkflowController*

`flowserv.service.local.init_db(env: Dict) → flowserv.model.database.DB`

Create an instance of the database object based on the given configuration settings. Sets the respective variables to the default value if not set.

Parameters **env** (*dict, default=None*) – Dictionary that provides access to configuration parameter values.

Return type *flowserv.model.database.DB*

flowserv.service.remote module

Helper functions for the remote service client.

`flowserv.service.remote.delete(url: str)`

Send DELETE request to given URL.

Parameters **url** (*string*) – Request URL.

`flowserv.service.remote.download_file(url: str) → IO`

Download a remote file.

Parameters **url** (*string*) – Request URL.

Return type io.BytesIO

flowserv.service.remote.**get**(*url: str*) → Dict

Send GET request to given URL and return the JSON body.

Parameters **url** (*string*) – Request URL.

Return type dict

flowserv.service.remote.**headers**() → Dict

Get dictionary of header elements for HTTP requests to a remote API.

Return type dict

flowserv.service.remote.**post**(*url: str*, *files: Optional[List] = None*, *data: Optional[Dict] = None*) → Dict

Send POST request with given (optional) body to a URL. Returns the JSON body from the response.

Parameters

- **url** (*string*) – Request URL.
- **data** (*dict, default=None*) – Optional request body.

Return type dict

flowserv.service.remote.**put**(*url: str*, *data: Optional[Dict] = None*) → Dict

Send PUT request with given (optional) body to a URL. Returns the JSON body from the response.

Parameters

- **url** (*string*) – Request URL.
- **data** (*dict, default=None*) – Optional request body.

Return type dict

flowserv.tests package

Submodules

flowserv.tests.controller module

Helper methods and classes for unit test for components of the benchmark modules.

class flowserv.tests.controller.**StateEngine**(*fs: Optional[*flowserv.volume.base.StorageVolume*] = None*, *state: Optional[*flowserv.model.workflow.state.WorkflowState*] = None*)

Bases: *flowserv.controller.base.WorkflowController*

Workflow controller for test purposes. Maintains a dictionary of run states. Allows to modify the state of maintained runs

cancel_run(*run_id: str*)

Request to cancel execution of the given run.

Parameters **run_id** (*string*) – Unique run identifier.

`configuration`() → List

Get a list of tuples with the names of additional configuration variables and their current values.

Return type list((string, string))

`error`(*run_id: str*, *messages: Optional[List[str]] = None*) → *flowserv.model.workflow.state.WorkflowState*

Set the run with the given identifier into error state.

Parameters

- **run_id** (*string*) – Unique run identifier
- **messages** (*list(string), optional*) – Default error messages

Return type *flowserv.model.workflow.state.WorkflowState*

`exec_workflow`(*run:* flowserv.model.base.RunObject, *template:* flowserv.model.template.base.WorkflowTemplate, *arguments: dict*, *staticfs:* flowserv.volume.base.StorageVolume, *config: Optional[Dict] = None*) → Tuple[*flowserv.model.workflow.state.WorkflowState*, *flowserv.volume.base.StorageVolume*]

Fake execute method that returns the workflow state that the was provided when the object was instantiated. Ignores all given arguments.

Parameters

- **run** (*flowserv.model.base.RunObject*) – Handle for the run that is being executed.
- **template** (*flowserv.model.template.base.WorkflowTemplate*) – Workflow template containing the parameterized specification and the parameter declarations.
- **arguments** (*dict*) – Dictionary of argument values for parameters in the template.
- **staticfs** (*flowserv.volume.base.StorageVolume*) – Storage volume that contains the static files from the workflow template.
- **config** (*dict, default=None*) – Optional implementation-specific configuration settings that can be used to overwrite settings that were initialized at object creation.

Return type *flowserv.model.workflow.state.WorkflowState*, *flowserv.volume.base.StorageVolume*

`start`(*run_id: str*) → *flowserv.model.workflow.state.WorkflowState*

Set the run with the given identifier into running state. Returns the modified workflow state.

Parameters **run_id** (*string*) – Unique run identifier

Return type *flowserv.model.workflow.state.WorkflowState*

`success`(*run_id: str*, *files: Optional[List[str]] = None*) → *flowserv.model.workflow.state.WorkflowState*

Set the default state to SUCCESS.

Parameters

- **run_id** (*string*) – Unique run identifier
- **files** (*list(string), default=None*) – List of created resource files (relative paths).

Return type *flowserv.model.workflow.state.WorkflowState*

flowserv.tests.model module

Helper method for creating database objects.

flowserv.tests.model.**create_group**(*session*, *workflow_id*, *users*)

Create a new workflow group in the database. Expects a workflow identifier and a list of user identifier. Returns the identifier for the created group.

Parameters

- **session** (*sqlalchemy.orm.session.Session*) – Database session.
- **workflow_id** (*string*) – Unique workflow identifier.
- **users** (*list*) – List of unique user identifier.

Return type string

flowserv.tests.model.**create_run**(*session*, *workflow_id*, *group_id*)

Create a new group run. Returns the run identifier.

Parameters

- **session** (*sqlalchemy.orm.session.Session*) – Database session.
- **workflow_id** (*string*) – Unique workflow identifier.
- **group_id** (*string*) – Unique group identifier.

Return type string

flowserv.tests.model.**create_user**(*session*, *active=True*)

Create a new user in the database. User identifier, name and password are all the same UUID. Returns the user identifier.

Parameters

- **session** (*sqlalchemy.orm.session.Session*) – Database session.
- **active** (*bool, default=True*) – User activation flag.

Return type string

flowserv.tests.model.**create_workflow**(*session*, *workflow_spec={}*, *result_schema=None*)

Create a new workflow handle for a given workflow specification. Returns the workflow identifier.

Parameters

- **session** (*sqlalchemy.orm.session.Session*) – Database session.
- **workflow_spec** (*dict, default=dict()*) – Optional workflow specification.
- **result_schema** (*dict, default=None*) – Optional result schema.

Return type string

flowserv.tests.model.**success_run**(*database:* flowserv.model.database.DB, *fs:* flowserv.volume.base.StorageVolume, *basedir: str*) → Tuple[str, str, str, str]

Create a successful run with two result files:

- A.json
- results/B.json

Returns the identifier of the created workflow, group, run, and user.

flowserv.tests.remote module

Implementation of the remote client for test purposes.

class flowserv.tests.remote.**RemoteTestClient**(*runcount=5*, *error=None*)

Bases: *flowserv.controller.remote.client.RemoteClient*

Implementation of the remote workflow engine client. Simulates the execution of a workflow. The remote workflow initially is in pending state. The first call to the get_workflow_state method will return a workflow in running state without actually stating any workflow execution. The next N calls to get_workflow_state will then simulate a runnign workflow. When the method is then called next either successful run or an error run is returned.

create_workflow(*run:* flowserv.model.base.RunObject, *template:* flowserv.model.template.base.WorkflowTemplate, *arguments: Dict*, *staticfs:* flowserv.volume.base.StorageVolume) → *flowserv.controller.remote.client.RemoteWorkflowHandle*

Create a new instance of a workflow from the given workflow template and user-provided arguments.

Parameters

- **run** (*flowserv.model.base.RunObject*) – Handle for the run that is being executed.
- **template** (*flowserv.model.template.base.WorkflowTemplate*) – Workflow template containing the parameterized specification and the parameter declarations.
- **arguments** (*dict*) – Dictionary of argument values for parameters in the template.
- **staticfs** (*flowserv.volume.base.StorageVolume*) – Storage volume that contains the static files from the workflow template.

Return type *flowserv.controller.remote.client.RemoteWorkflowHandle*

get_workflow_state(*workflow_id: str*, *current_state:* flowserv.model.workflow.state.WorkflowState) → *flowserv.model.workflow.state.WorkflowState*

Get information about the current state of a given workflow.

Note, if the returned result is SUCCESS the workflow resource files may not have been initialized properly. This will be done by the workflow controller. The timestamps, however, should be set accurately.

Parameters

- **workflow_id** (*string*) – Unique workflow identifier
- **current_state** (*flowserv.model.workflw.state.WorkflowState*) – Last known state of the workflow by the workflow controller

Return type flowserv.model.workflw.state.WorkflowState

stop_workflow(*workflow_id: str*)

Stop the execution of the workflow with the given identifier.

Sets the state to None to raise an error the next time the workflow state is polled.

Parameters **workflow_id** (*string*) – Unique workflow identifier

flowserv.tests.serialize module

Helper methods to test object serialization.

flowserv.tests.serialize.**validate_file_handle**(*doc*)

Validate serialization of a file handle.

Parameters **doc** (*dict*) – File handle serialization

Raises **ValueError** –

flowserv.tests.serialize.**validate_file_listing**(*doc*, *count*)

Validate serialization of a file listing. The count parameter gives the expected number of files in the listing.

Parameters

- **doc** (*dict*) – Listing of file handle serializations
- **count** (*int*) – Expected number of files in the listing

Raises **ValueError** –

flowserv.tests.serialize.**validate_group_handle**(*doc*)

Validate serialization of a workflow group handle.

Parameters **doc** (*dict*) – Workflow group handle serialization

Raises **ValueError** –

flowserv.tests.serialize.**validate_group_listing**(*doc*)

Validate serialization of a workflow group listing.

Parameters **doc** (*dict*) – Listing of workflow group descriptor serializations

Raises **ValueError** –

flowserv.tests.serialize.**validate_para_module**(*doc*)

Validate serialization of a workflow parameter module handle.

Parameters **doc** (*dict*) – Workflow parameter module handle serialization

Raises **ValueError** –

flowserv.tests.serialize.**validate_parameter**(*doc*)

Validate serialization of a workflow parameter.

Parameters **doc** (*dict*) – Parameter serialization

Raises **ValueError** –

flowserv.tests.serialize.**validate_ranking**(*doc*)

Validate serialization of a workflow evaluation ranking.

Parameters **doc** (*dict*) – Ranking serialization

Raises **ValueError** –

flowserv.tests.serialize.**validate_reset_request**(*doc*)

Validate serialization of a user password reset request.

Parameters **doc** (*dict*) – Reset request response serialization

Raises **ValueError** –

flowserv.tests.serialize.**validate_run_descriptor**(*doc*)

Validate serialization of run descriptor.

Parameters **doc** (*dict*) – Run handle serialization

Raises **ValueError** –

flowserv.tests.serialize.**validate_run_handle**(*doc*, *state*)

Validate serialization of a run handle.

Parameters

- **doc** (*dict*) – Run handle serialization
- **state** (*string*) – Expected run state

Raises **ValueError** –

flowserv.tests.serialize.**validate_run_listing**(*doc*)

Validate serialization of a workflow run listing.

Parameters **doc** (*dict*) – Serialization for listing of workflow run descriptors

Raises **ValueError** –

flowserv.tests.serialize.**validate_user_handle**(*doc*, *login*, *inactive=False*)

Validate serialization of a user handle. Serialization depends on whether the user is currently logged in or not.

Parameters

- **doc** (*dict*) – User handle serialization
- **login** (*bool*) – Flag indicating whether the handle is for a user that is logged in
- **inactive** (*bool, optional*) – Flag indicating whether the user account has been activated yet

Raises **ValueError** –

flowserv.tests.serialize.**validate_user_listing**(*doc*)

Validate serialization of a user listing.

Parameters **doc** (*dict*) – Serialization for listing of user descriptors

Raises **ValueError** –

flowserv.tests.serialize.**validate_workflow_handle**(*doc*)

Validate serialization of a workflow handle. Here we distinguish between handles that have optional elements (description and instructions) and those that have not.

Parameters **doc** (*dict*) – Workflow handle serialization.

Raises **ValueError** –

flowserv.tests.serialize.**validate_workflow_listing**(*doc*)

Validate serialization of a workflow descriptor listing.

Parameters **doc** (*dict*) – Serialization for listing of workflow descriptors

Raises **ValueError** –

flowserv.tests.service module

Helper methods to initialize the database state via the service API.

flowserv.tests.service.**create_group**(*api*, *workflow_id*, *users=None*)

Create a new group for the given workflow.

Parameters

- **api** (*flowserv.service.api.API*) – Service API manager.
- **workflow_id** (*string*) – Unique workflow identifier.
- **users** (*list(string)*) – Identifier for group members.

Return type string

flowserv.tests.service.**create_ranking**(*api*, *workflow_id*, *count*)

Create a ranking with n groups for the Hello World benchmark having a successful run each. Returns the group identifier in order of creation. The avg_len value is increased as groups are created and the max_len value is decreased.

Parameters

- **api** (*flowserv.service.api.API*) – Service API manager.
- **workflow_id** (*string*) – Unique workflow identifier.
- **user_id** (*string*) – Identifier for the group owner.
- **count** (*int*) – Number of groups that are created for the workflow.

Return type list(string)

flowserv.tests.service.**create_user**(*api*)

Register a new user with the API and return the unique user identifier.

Parameters **api** (*flowserv.service.api.API*) – Service API manager.

Return type string

flowserv.tests.service.**create_workflow**(*api*, *source*, *specfile=None*)

Start a new workflow for a given template.

flowserv.tests.service.**start_hello_world**(*api*, *group_id*)

Start a new run for the Hello World template. Returns the run identifier and the identifier for the input file.

Parameters

- **api** (*flowserv.service.api.API*) – Service API manager.
- **group_id** (*string*) – Unique group identifier.

Return type string, string

flowserv.tests.service.**start_run**(*api*, *group_id*, *arguments=[]*, *config=None*)

Start a new workflow run for a given group. Returns the identifier of the started run.

Parameters

- **api** (*flowserv.service.api.API*) – Service API manager.
- **group_id** (*string*) – Unique group identifier.
- **user_id** (*string*) – Unique user identifier.

- **arguments** (*list, default=None*) – Optional arguments to run the workflow.
- **config** (*dict, default=None*) – Optional configuration settings for the workflow run.

Return type string

`flowserv.tests.service.`**`upload_file`**(*api*, *group_id*, *file*)

Upload an input file for a workflow run. returns the file identifier.

Parameters

- **api** (*flowserv.service.api.API*) – Service API manager.
- **group_id** (*string*) – Unique group identifier.
- **file** (*IOHandle*) – Uploaded file.

Return type string

`flowserv.tests.service.`**`write_results`**(*runstore: flowserv.volume.base.StorageVolume*, *files: Tuple[Union[dict, list], str, str]*)

Create a result files for a workflow run.

Parameters

- **runstore** (*flowserv.volume.base.StorageVolume*) – Storage volume for the run (result) files of a successful workflow run.
- **files** (*list*) – List of 3-tuples containing the file data, format, and relative path.

flowserv.tests.worker module

Helper methods for worker classes unit tests.

`flowserv.tests.worker.`**`a_plus_b`**(*a: int*, *b: int*) → int

Simple helper function for testing code steps.

Returns the sum of the two arguments.

`flowserv.tests.worker.`**`multi_by_x`**(*filename: str*, *x: int*) → int

Read input file with single integer value (in Json format) and multiplies the value with the given x.

Expects a Json object with format: {“value”: v}

Returns the multiplication result.

flowserv.util package

Submodules

flowserv.util.core module

Collection of general utility functions.

`flowserv.util.core.`**`get_unique_identifier`**() → str

Create a new unique identifier.

Return type string

flowserv.util.core.**import_obj**(*import_path: str*) → Union[Callable, Type]

Import an object (function or class) from a given package path.

Parameters **import_path** (*string*) – Full package target path for the imported object. Assumes that path components are separated by '.'.

Return type callable or class

flowserv.util.core.**jquery**(*doc: Dict*, *path: List[str]*) → Any

Json query to extract the value at the given path in a nested dictionary object.

Returns None if the element that is specified by the path does not exist.

Parameters

- **doc** (*dict*) – Nested dictionary
- **path** (*list(string)*) – List of elements in the query path

Return type any

flowserv.util.core.**stacktrace**(*ex*) → List[str]

Get list of strings representing the stack trace for a given exception.

Parameters **ex** (*Exception*) – Exception that was raised by flowServ code

Return type list of string

flowserv.util.core.**validate_doc**(*doc: typing.Dict*, *mandatory: typing.Optional[typing.List[str]] = None*, *optional: typing.Optional[typing.List[str]] = None*, *exception: typing.Optional[typing.Type] = <class 'ValueError'>*)

Raises error if a dictionary contains labels that are not in the given label lists or if there are labels in the mandatory list that are not in the dictionary. Returns the given dictionary (if valid).

Parameters

- **doc** (*dict*) – Dictionary serialization of an object
- **mandatory** (*list(string), default=None*) – List of mandatory labels for the dictionary serialization
- **optional** (*list(string), optional*) – List of optional labels for the dictionary serialization
- **exception** (*Error, default=ValueError*) – Error class that is raised if validation fails. By default, a ValueError is raised.

Return type dict

Raises **ValueError** –

flowserv.util.datetime module

Utility methods for date time conversion.

flowserv.util.datetime.**to_datetime**(*timestamp: str*) → datetime.datetime

Converts a timestamp string in ISO format into a datatime object.

Parameters **timstamp** (*string*) – Timestamp in ISO format

Returns Datetime object

Return type datetime.datetime

flowserv.util.datetime.**utc_now**() → str

Get the current time in UTC timezone as a string in ISO format.

Return type string

flowserv.util.files module

Helper methods for the reproducible open benchmark platform. Provides methods to (i) read and write files in JSON and YAML format, (ii) create directories, (iii) validate dictionaries, and (iv) to create of unique identifiers.

flowserv.util.files.**cleardir**(*directory: str*)

Remove all files in the given directory.

Parameters **directory** (*string*) – Path to directory that is being created.

flowserv.util.files.**dirname**(*key: str*) → str

Get the parent directory for a given file identifier.

Parameters **key** (*str*) – Relative file path expression.

Return type str

flowserv.util.files.**filepath**(*key: str*, *sep: Optional[str] = '/'*) → str

Convert a given file path to a local path.

Replaces the default path separator '/' with the OS-specific separator if it is different from the default one.

Parameters

- **key** (*str*) – Relative file path expression.
- **sep** (*string, default=OS file path separator*) – OS-specific file path separator.

Return type str

flowserv.util.files.**join**(**args*) → str

Concatenate a list of values using the key path separator '/'.

Parameters **args** (*list*) – List of argument values.

Return type str

flowserv.util.files.**read_buffer**(*filename: str*) → IO

Read content from specified file into a BytesIO buffer.

Parameters **filename** (*string*) – Path tpo file on disk.

Return type io.BytesIO

flowserv.util.files.**read_object**(*filename: str*, *format: Optional[str] = None*) → Dict

Load a Json object from a file. The file may either be in Yaml or in Json format.

Parameters

- **filename** (*string or io.BytesIO*) – Path to file on disk
- **format** (*string, optional*) – Optional file format identifier. The default is YAML.

Return type dict

Raises **ValueError** –

flowserv.util.files.**write_object**(*filename: str*, *obj: Union[Dict, List]*, *format: Optional[str] = None*)

Write given dictionary to file as Json object.

Parameters

- **filename** (*string*) – Path to output file
- **obj** (*dict*) – Output object

Raises **ValueError** –

flowserv.util.serialize module

Helper methods for object serialization and deserialization.

flowserv.util.serialize.**to_dict**(*args: List[Dict]*) → Dict

Convert a list of serialized key-value pairs into a dictionary that maps the keys to their values.

Parameters **args** (*list*) – List of dictionary serializations for key-value pairs.

Return type dict

flowserv.util.serialize.**to_kvp**(*key: str*, *value: str*) → Dict

Serialize a key-value pair into a dictionary.

Parameters

- **key** (*string*) – Key value
- **value** (*string*) – Associate value for the key.

Return type dict

flowserv.util.ssh module

SSH client interface for interacting with remote servers using the paramiko package.

class flowserv.util.ssh.**SSHClient**(*hostname: str*, *port: Optional[int] = None*, *timeout: Optional[float] = None*, *look_for_keys: Optional[bool] = False*, *sep: Optional[str] = '/'*)

Bases: object

SSH client that allows to run remote commands and access files.

close()

Close the SSH client.

exec_cmd(*command*) → str

Execute command on the remote server.

Returns output from STDOUT. Raises an error if command execution on the remote server failed (as indicated by the program exit code).

Parameters **command** (*string*) – Command line string that is executed on the remote server.

Return type string

sftp() → paramiko.sftp_client.SFTPClient

Get SFTP client.

Return type paramiko.SFTPClient

property ssh_client: paramiko.client.SSHClient

Get an active instance of the SSH Client.

Return type paramiko.SSHClient

walk(*dirpath: str*) → List[str]

Get recursive listing of all files in a given directory.

Returns a list of relative path expressions for files in the directory.

If `dirpath` does not reference a directory the result is None.

Parameters **dirpath** (*string*) – Path to a directory on the remote server.

Return type list of string

flowserv.util.ssh.**paramiko_ssh_client**(*hostname: str*, *port: Optional[int] = None*, *timeout: Optional[float] = None*, *look_for_keys: Optional[bool] = False*) → paramiko.client.SSHClient

Helper function to create a paramiko SSH Client.

This separate function is primarily intended to make patching easier for unit testing.

Parameters

- **hostname** (*string*) – Server to connect to.
- **port** (*int, default=None*) – Server port to connect to.
- **timeout** (*float, default=None*) – Optional timeout (in seconds) for the TCP connect.
- **look_for_keys** (*bool, default=False*) – Set to True to enable searching for discoverable private key files in `~/.ssh/`.

Return type paramiko.SSHClient

flowserv.util.ssh.**ssh_client**(*hostname: str*, *port: Optional[int] = None*, *timeout: Optional[float] = None*, *look_for_keys: Optional[bool] = False*, *sep: Optional[str] = '/'*) → *flowserv.util.ssh.SSHClient*

Context manager for the flowserv SSHCilent.

Parameters

- **hostname** (*string*) – Server to connect to.
- **port** (*int, default=None*) – Server port to connect to.
- **timeout** (*float, default=None*) – Optional timeout (in seconds) for the TCP connect.
- **look_for_keys** (*bool, default=False*) – Set to True to enable searching for discoverable private key files in `~/.ssh/`.
- **sep** (*string, default='/'*) – Path separator used by the remote file system.

Return type paramiko.SSHClient

flowserv.util.ssh.**walk**(*client: paramiko.sftp_client.SFTPClient*, *dirpath: str*, *prefix: Optional[str] = None*, *sep: Optional[str] = '/'*) → List[str]

Recursively scan contents of a remote directory.

Returns a list of tuples that contain the relative sub-directory path and the file name for all files. The sub-directory path for files in the `dirpath` is None.

If `dirpath` does not reference a directory the result is None.

Parameters

- **client** (*paramiko.SFTPClient*) – SFTP client.
- **dirpath** (*string*) – Path to a directory on the remote server.
- **prefix** (*string, default=None*) – Prefix path for the current (sub-)directory.
- **sep** (*string, default='/'*) – Path separator used by the remote file system.

Return type list of tuples of (string, string)

flowserv.view package

Submodules

flowserv.view.descriptor module

Serializer for the service descriptor.

class flowserv.view.descriptor.**ServiceDescriptorSerializer**

Bases: object

Default serializer for the service descriptor.

from_config(*env: Dict*, *username: Optional[str] = None*) → Dict

Get serialized descriptor with basic information from values in the given configuration settings.

Parameters

- **env** (*dict, default=None*) – Dictionary that provides access to configuration parameter values.
- **username** (*string, default=None*) – Optional name for an authenticated user.

Return type dict

get_name(*doc: Dict*) → str

Get the name value from the given document.

Parameters **doc** (*dict*) – Serialization of a service descriptor.

Return type string

get_routes(*doc: Dict*, *routes: Dict*) → str

Get the user name from the given document.

Parameters

- **doc** (*dict*) – Serialization of a service descriptor.
- **routes** (*dict*) – Dictionary with Url patterns for supported API routes. This will override patterns that may be defined in the given serialized descriptor.

Return type string

get_url(*doc: Dict*) → str

Get the base Url from the given document.

Parameters **doc** (*dict*) – Serialization of a service descriptor.

Return type string

get_username(*doc: Dict*) → str

Get the user name from the given document.

Parameters **doc** (*dict*) – Serialization of a service descriptor.

Return type string

get_version(*doc: Dict*) → str

Get the version information from the given document.

Parameters **doc** (*dict*) – Serialization of a service descriptor.

Return type string

service_descriptor(*name: str*, *version: str*, *url: str*, *routes: Dict*, *username: Optional[str] = None*) → Dict

Serialization of the service descriptor. The descriptor contains the service name, version, and a list of route patterns. The optional user name indicates whether a request for the service descriptor contained a valid access token. If the user name is not None it will be included in the service descriptor.

Parameters

- **name** (*string*) – Service name.
- **version** (*string*) – Service version number.
- **url** (*string*) – Base Url for the service API. This is the prefix for all Url routes.
- **username** (*string, default=None*) – Name of the user that was authenticated by a given access token.

Return type dict

flowserv.view.files module

Serializer for uploaded workflow user group files.

class flowserv.view.files.**UploadFileSerializer**

Bases: object

Default serializer for handles and listings of files that were uploaded for a workflow groups.

file_handle(*group_id: str*, *fh:* flowserv.model.base.FileObject) → Dict

Get serialization for a file handle.

Parameters

- **group_id** (*string*) – Unique workflow group identifier
- **fh** (*flowserv.model.base.FileObject*) – File handle

Return type dict

file_listing(*group_id: str*, *files: List[*flowserv.model.base.FileObject*]*) → Dict

Get serialization for listing of uploaded files for a given workflow group.

Parameters

- **group_id** (*string*) – Unique workflow group identifier
- **files** (*list(flowserv.model.base.FileObject)*) – List of file handle

Return type dict

flowserv.view.group module

Serializer for workflow user groups.

class flowserv.view.group.**WorkflowGroupSerializer**(*files: Optional[flowserv.view.files.UploadFileSerializer] = None*, *runs: Optional[flowserv.view.run.RunSerializer] = None*)

Bases: object

Default serializer for workflow user groups.

group_descriptor(*group:* flowserv.model.base.GroupObject) → Dict

Get serialization for a workflow group descriptor. The descriptor contains the group identifier, name, and the base list of HATEOAS references.

Parameters **group** (*flowserv.model.base.GroupObject*) – Workflow group handle

Return type dict

group_handle(*group:* flowserv.model.base.GroupObject, *runs: Optional[List[*flowserv.model.base.RunObject*]] = None*) → Dict

Get serialization for a workflow group handle.

Parameters

- **group** (*flowserv.model.base.GroupObject*) – Workflow group handle
- **runs** (*list of flowserv.model.base.RunObject, default=None*) – Optional list of run handles for an authenticated user.

Return type dict

group_listing(*groups: List[*flowserv.model.base.GroupObject*]*) → Dict

Get serialization of a workflow group descriptor list.

Parameters **groups** (*list(flowserv.model.base.GroupObject)*) – List of descriptors for workflow groups

Return type dict

flowserv.view.run module

Serializer for workflow runs.

class flowserv.view.run.**RunSerializer**

Bases: object

Serializer for workflow runs.

run_descriptor(*run:* flowserv.model.base.RunObject) → Dict

Get serialization for a run descriptor. The descriptor contains the run identifier, state, timestampls, and the base list of HATEOAS references.

Parameters **run** (*flowserv.model.base.RunObject*) – Run decriptor

Return type dict

run_handle(*run:* flowserv.model.base.RunObject, *group: Optional[*flowserv.model.base.GroupObject*] = None*) → Dict

Get serialization for a run handle. The run handle extends the run descriptor with the run arguments, the parameter declaration taken from the workflow group handle (since it may differ from the parameter list of the workflow), and additional information associated with the run state.

Parameters

- **run** (*flowserv.model.base.RunObject*) – Workflow run handle
- **group** (*flowserv.model.base.GroupObject, default=None*) – Workflow group handle. Missing for post-processing workflows

Return type dict

run_listing(*runs: List[*flowserv.model.base.RunObject*]*) → Dict

Get serialization for a list of run handles.

Parameters **runs** (*list(flowserv.model.base.RunObject)*) – List of run handles

Return type dict

flowserv.view.user module

Serializer for user resources.

class flowserv.view.user.**UserSerializer**

Bases: object

Default serializer for user resources.

reset_request(*request_id*)

Serialization for requested identifier to rest a user password.

Parameters **request_id** (*string*) – Unique request identifier

Return type dict

user(*user*, *include_token=True*)

Serialization for user handle. Contains the user name and the access token if the user is logged in.

Parameters

- **user** (*flowserv.model.base.User*) – Handle for a registered user
- **include_token** (*bool, optional*) – Include API tokens for logged in users if True

Return type dict

user_listing(*users*)

Serialize a list of user handles.

Parameters **users** (*list(flowserv.model.base.User)*) – List of user handles

Return type dict

flowserv.view.validate module

flowserv.view.validate.**validator**(*key: str*) → jsonschema.validators.Draft7Validator

Get Json schema validator for a specific API resource. The resource is identified by its unique key.

Parameters **key** (*string*) – Unique resource key in the schema definition.

Return type jsonschema.Draft7Validator

flowserv.view.workflow module

Serializer for workflow resources.

class flowserv.view.workflow.**WorkflowSerializer**(*groups: Optional[*flowserv.view.group.WorkflowGroupSerializer*] = None*, *runs: Optional[*flowserv.view.run.RunSerializer*] = None*)

Bases: object

Default serializer for workflow resource objects. Defines the methods that are used to serialize workflow descriptors, handles, and listing.

workflow_descriptor(*workflow:* flowserv.model.base.WorkflowObject) → Dict

Get dictionary serialization containing the descriptor of a workflow resource.

Parameters **workflow** (*flowserv.model.base.WorkflowObject*) – Workflow descriptor.

Return type dict

workflow_handle(*workflow:* flowserv.model.base.WorkflowObject, *postproc: Optional[*flowserv.model.base.RunObject*] = None*, *groups: Optional[List[*flowserv.model.base.GroupObject*]] = None*) → Dict

Get dictionary serialization containing the handle of a workflow resource.

Parameters

- **workflow** (*flowserv.model.base.WorkflowObject*) – Workflow handle
- **postproc** (*flowserv.model.base.RunObject*) – Handle for workflow post-porcessing run.
- **groups** (*list(flowserv.model.base.GroupObject), default=None*) – Optional list of descriptors for workflow groups for an authenticated user.

Return type dict

workflow_leaderboard(*workflow:* flowserv.model.base.WorkflowObject, *ranking: List[*flowserv.model.ranking.RunResult*]*, *postproc: Optional[*flowserv.model.base.RunObject*] = None*) → Dict

Get dictionary serialization for a workflow evaluation leaderboard.

Parameters

- **workflow** (*flowserv.model.base.WorkflowObject*) – Workflow handle
- **leaderboard** (*flowserv.model.ranking.ResultRanking*) – List of entries in the workflow evaluation leaderboard
- **postproc** (*flowserv.model.base.RunObject*) – Handle for workflow post-porcessing run.

Return type dict

workflow_listing(*workflows: List[*flowserv.model.base.WorkflowObject*]*) → Dict

Get dictionary serialization of a workflow listing.

Parameters **workflows** (*list(flowserv.model.base.WorkflowObject)*) – List of workflow descriptors

Return type dict

flowserv.volume package

Submodules

flowserv.volume.base module

Base classes for workflow runtime storage volumes.

class flowserv.volume.base.**IOBuffer**(*buf: IO*)

Bases: *flowserv.volume.base.IOHandle*

Implementation of the file object interface for bytes IO buffers.

open() → IO

Get the associated BytesIO buffer.

Return type io.BytesIO

size() → int

Get size of the file in the number of bytes.

Return type int

class flowserv.volume.base.**IOHandle**

Bases: object

Wrapper around different file objects (i.e., files on disk or files in object stores). Provides functionality to load file content as a bytes buffer and to write file contents to disk.

abstract open() → IO

Get file contents as a BytesIO buffer.

Return type io.BytesIO

Raises *flowserv.error.UnknownFileError* –

abstract size() → int

Get size of the file in the number of bytes.

Return type int

class flowserv.volume.base.**StorageVolume**(*identifier: Optional[str] = None*)

Bases: object

The runtime storage volume provides access to a file system-like object for storing and retrieving files and folders that are required or produced by a workflow step.

Storage volumes are used to provide a copy of the required run files for a workflow step. Each volume has a unique identifier that is used to keep track which files and file versions are available in the volume.

abstract close()

Close any open connection and release all resources when workflow execution is done.

copy(*src: Union[str, List[str]]*, *store:* flowserv.volume.base.StorageVolume, *dst: Optional[str] = None*, *verbose: Optional[bool] = False*) → List[str]

Copy the file or folder at the source path of this storage volume to the given storage volume.

The source path is relative to the base directory for the workflow run.

Returns the list of files that were copied.

Parameters

- **src** (*string or list of string*) – Relative source path(s) for downloaded files and directories.
- **store** (*flowserv.volume.base.StorageValue*) – Storage volume for destination files.
- **dst** (*string, default=None*) – Destination folder for downloaded files.
- **verbose** (*bool, default=False*) – Print information about source and target volume and the files that are being copied.

Return type list of string

abstract delete(*key: str*) → int

Delete file or folder with the given key.

Parameters **key** (*str*) – Path to a file object in the storage volume.

abstract describe() → str

Get short descriptive string about the storage volume for display purposes.

Return type str

abstract erase()

Erase the storage volume base directory and all its contents.

abstract get_store_for_folder(*key: str*, *identifier: Optional[str] = None*) → *flowserv.volume.base.StorageVolume*

Get storage volume for a sob-folder of the given volume.

Parameters

- **key** (*string*) – Relative path to sub-folder. The concatenation of the base folder for this storage volume and the given key will form te new base folder for the returned storage volume.
- **identifier** (*string, default=None*) – Unique volume identifier.

Return type *flowserv.volume.base.StorageVolume*

abstract load(*key: str*) → *flowserv.volume.base.IOHandle*

Load a file object at the source path of this volume store.

Returns a file handle that can be used to open and read the file.

Parameters **key** (*str*) – Path to a file object in the storage volume.

Return type *flowserv.volume.base.IOHandle*

abstract mkdir(*path: str*)

Create the directory with the given (relative) path and all of its parent directories.

Does not raise an error if the directory exists.

Parameters **path** (*string*) – Relative path to a directory in the storage volume.

abstract store(*file:* flowserv.volume.base.IOHandle, *dst: str*)

Store a given file object at the destination path of this volume store.

Parameters

- **file** (*flowserv.volume.base.IOHandle*) – File-like object that is being stored.
- **dst** (*str*) – Destination path for the stored object.

abstract to_dict() → Dict

Get dictionary serialization for the storage volume.

The returned serialization can be used by the volume factory to generate a new instance of this volume store.

Return type dict

abstract walk(*src: str*) → List[Tuple[str, *flowserv.volume.base.IOHandle*]]

Get list of all files at the given source path.

If the source path references a single file the returned list will contain a single entry. If the source specifies a folder the result contains a list of all files in that folder and the subfolders.

Parameters **src** (*str*) – Source path specifying a file or folder.

Return type list of tuples (str, *flowserv.volume.base.IOHandle*)

flowserv.volume.base.**copy_files**(*src: Union[str, List[str]]*, *source:* flowserv.volume.base.StorageVolume, *dst: str*, *target:* flowserv.volume.base.StorageVolume, *verbose: Optional[bool] = False*) → List[str]

Copy files and folders at the source path (path) of a given source storage volume to the destination path (path) of a target storage volume.

Returns the list of files that were copied.

Parameters

- **src** (*str or list of string*) – Path specifying the source file(s) or folder(s).
- **source** (*flowserv.volume.base.StorageValue*) – Storage volume for source files.
- **dst** (*string*) – Destination path for copied files.
- **target** (*flowserv.volume.base.StorageValue*) – Storage volume for destination files.
- **verbose** (*bool, default=False*) – Print information about source and target volume and the files that are being copied.

Return type list of string

flowserv.volume.factory module

Factory pattern for file stores.

flowserv.volume.factory.**Volume**(*doc: Dict*) → *flowserv.volume.base.StorageVolume*

Factory pattern to create storage volume instances for the service API.

Expects a serialization object that contains at least the volume type `type`.

Parameters **doc** (*dict*) – Serialization dictionary that provides access to storage volume type and the implementation-specific volume parameters.

Return type *flowserv.volume.base.StorageVolume*

flowserv.volume.fs module

File system workflow storage volume. Maintains workflow run files in a folder on the local file system.

class flowserv.volume.fs.**FSFile**(*filename: str*, *raise_error: Optional[bool] = True*)

Bases: *flowserv.volume.base.IOHandle*

Implementation of the IO object handle interface for files that are stored on the file system.

open() → IO

Get file contents as a BytesIO buffer.

Return type io.BytesIO

Raises *flowserv.error.UnknownFileError* –

size() → int

Get size of the file in the number of bytes.

Return type int

flowserv.volume.fs.**FStore**(*basedir: str*, *identifier: Optional[str] = None*) → Dict

Get configuration object for a file system storage volume.

Parameters

- **basedir** (*string*) – Google Cloud Storage bucket identifier.
- **identifier** (*string, default=None*) – Optional storage volume identifier.

Return type dict

class flowserv.volume.fs.**FileSystemStorage**(*basedir: str*, *identifier: Optional[str] = None*)

Bases: *flowserv.volume.base.StorageVolume*

The file system storage volume provides access to workflow run files that are maintained in a run directory on the local file system.

close()

The file system runtime manager has no connections to close or resources to release.

delete(*key: str*)

Delete file or folder with the given key.

Parameters **key** (*str*) – Path to a file object in the storage volume.

describe() → str

Get short descriptive string about the storage volume for display purposes.

Return type str

erase()

Erase the storage volume base directory and all its contents.

static from_dict(*doc*) → *flowserv.volume.fs.FileSystemStorage*

Get file system storage volume instance from dictionary serialization.

Parameters **doc** (*dict*) – Dictionary serialization as returned by the `to_dict()` method.

Return type *flowserv.volume.fs.FileSystemStorage*

get_store_for_folder(*key: str*, *identifier: Optional[str] = None*) → *flowserv.volume.base.StorageVolume*

Get storage volume for a sob-folder of the given volume.

Parameters

- **key** (*string*) – Relative path to sub-folder. The concatenation of the base folder for this storage volume and the given key will form te new base folder for the returned storage volume.
- **identifier** (*string, default=None*) – Unique volume identifier.

Return type *flowserv.volume.base.StorageVolume*

load(*key: str*) → *flowserv.volume.base.IOHandle*

Load a file object at the source path of this volume store.

Returns a file handle that can be used to open and read the file.

Parameters **key** (*str*) – Path to a file object in the storage volume.

Return type *flowserv.volume.base.IOHandle*

mkdir(*path: str*)

Create the directory with the given (relative) path and all of its parent directories.

Does not raise an error if the directory exists.

Parameters **path** (*string*) – Relative path to a directory in the storage volume.

path(**args*) → pathlib.Path

Get a file system path object for a file or directory that is given by a list of path components relative to the base directory of the storage volume.

Parameters **args** (*list of string*) – List of path components that are joined with the base directory of the storage volume to generate the path object.

Return type pathlib.Path

store(*file:* flowserv.volume.base.IOHandle, *dst: str*)

Store a given file object at the destination path of this volume store.

Parameters

- **file** (*flowserv.volume.base.IOHandle*) – File-like object that is being stored.
- **dst** (*str*) – Destination path for the stored object.

to_dict() → Dict

Get dictionary serialization for the storage volume.

The returned serialization can be used by the volume factory to generate a new instance of this volume store.

Return type dict

walk(*src: str*) → List[Tuple[str, *flowserv.volume.base.IOHandle*]]

Get list of all files at the given source path.

If the source path references a single file the returned list will contain a single entry. If the source specifies a folder the result contains a list of all files in that folder and the subfolders.

Parameters **src** (*str*) – Source path specifying a file or folder.

Return type list of tuples (str, *flowserv.volume.base.IOHandle*)

flowserv.volume.fs.**walkdir**(*dirname: str*, *prefix: str*, *files: List[Tuple[str,* flowserv.volume.base.IOHandle*]]*) → List[Tuple[str, *flowserv.volume.base.IOHandle*]]

Recursively add all files in a given source folder to a file upload list. The elements in the list are tuples of file object and relative target path.

Parameters

- **dirname** (*string*) – Path to folder of the local file system.
- **prefix** (*string*) – Relative destination path for all files in the folder.
- **files** (*list of (string, flowserv.volume.base.IOHandle)*) – Pairs of file objects and their relative target path for upload to a file store.

flowserv.volume.gc module

Implementation of the `flowserv.volume.base.GCVolume` for the use of Google Cloud File Store buckets.

For testing the GCBucket the Google Cloud credentials have to be configured. Set up authentication by creating a service account and setting the environment variable *GOOGLE_APPLICATION_CREDENTIALS*. See the documentation for more details: https://cloud.google.com/storage/docs/reference/libraries#setting_up_authentication

flowserv.volume.gc.**GCBucket**(*bucket: str*, *prefix: Optional[str] = None*, *identifier: Optional[str] = None*) → Dict

Get configuration object for Google Cloud storage volume.

Parameters

- **bucket** (*string*) – Google Cloud Storage bucket identifier.
- **prefix** (*string, default=None*) – Key-prefix for all files.
- **identifier** (*string, default=None*) – Optional storage volume identifier.

Return type dict

class flowserv.volume.gc.**GCFile**(*client: flowserv.volume.gc.GCClient*, *bucket_name: str*, *key: str*)

Bases: *flowserv.volume.base.IOHandle*

Implementation of the file object interface for files that are stored on Google Cloud Storage buckets.

open() → IO

Get file contents as a BytesIO buffer.

Return type io.BytesIO

size() → int

Get size of the file in the number of bytes.

Return type int

class flowserv.volume.gc.**GCVolume**(*bucket_name: str*, *prefix: Optional[str] = None*, *identifier: Optional[str] = None*)

Bases: *flowserv.volume.base.StorageVolume*

Implementation of the storage volume class for Google Cloud File Store buckets.

close()

The Google Cloud client resource does not need to be closed.

`delete`(*key: str*)

Delete file or folder with the given key.

Parameters **key** (*str*) – Path to a file object in the storage volume.

`delete_objects`(*keys: Iterable[str]*)

Delete objects with the given identifier.

Parameters **keys** (*iterable of string*) – Unique identifier for objects that are being deleted.

`describe`() → str

Get short descriptive string about the storage volume for display purposes.

Return type str

`erase`()

Erase the storage volume base directory and all its contents.

`static from_dict`(*doc*) → *flowserv.volume.gc.GCVolume*

Get Google Cloud storage volume instance from dictionary serialization.

Parameters **doc** (*dict*) – Dictionary serialization as returned by the `to_dict()` method.

Return type *flowserv.volume.gc.GCVolume*

`get_store_for_folder`(*key: str*, *identifier: Optional[str] = None*) → *flowserv.volume.base.StorageVolume*

Get storage volume for a sob-folder of the given volume.

Parameters

- **key** (*string*) – Relative path to sub-folder. The concatenation of the base folder for this storage volume and the given key will form te new base folder for the returned storage volume.
- **identifier** (*string, default=None*) – Unique volume identifier.

Return type *flowserv.volume.base.StorageVolume*

`load`(*key: str*) → *flowserv.volume.base.IOHandle*

Load a file object at the source path of this volume store.

Returns a file handle that can be used to open and read the file.

Parameters **key** (*str*) – Path to a file object in the storage volume.

Return type *flowserv.volume.base.IOHandle*

`mkdir`(*path: str*)

Create the directory with the given (relative) path and all of its parent directories.

For bucket stores no directories need to be created prior to accessing them.

Parameters **path** (*string*) – Relative path to a directory in the storage volume.

`query`(*filter: str*) → Iterable[str]

Get identifier for objects that match a given prefix.

Parameters **filter** (*str*) – Prefix query for object identifiers.

Return type iterable of string

store(*file:* flowserv.volume.base.IOHandle, *dst: str*)

Store a given file object at the destination path of this volume store.

Parameters

- **file** (*flowserv.volume.base.IOHandle*) – File-like object that is being stored.
- **dst** (*str*) – Destination path for the stored object.

to_dict() → Dict

Get dictionary serialization for the storage volume.

The returned serialization can be used by the volume factory to generate a new instance of this volume store.

Return type dict

walk(*src: str*) → List[Tuple[str, *flowserv.volume.base.IOHandle*]]

Get list of all files at the given source path.

If the source path references a single file the returned list will contain a single entry. If the source specifies a folder the result contains a list of all files in that folder and the subfolders.

Parameters **src** (*str*) – Source path specifying a file or folder.

Return type list of tuples (str, *flowserv.volume.base.IOHandle*)

flowserv.volume.gc.**GC_STORE = 'gc'**

Type alias for Google Cloud storage bucket objects.

flowserv.volume.gc.**get_google_client**()

Helper method to get instance of the Google Cloud Storage client. This method was added to support mocking for unit tests.

flowserv.volume.manager module

Manager for storage volumes. The volume manager is associated with a workflow run. It maintains information about the files and directories that are available to the workflow run in the virtual workflow environment.

flowserv.volume.manager.**DefaultVolume**(*basedir: str*) → *flowserv.volume.manager.VolumeManager*

Helper method to create a volume manager with a single file system store as the default store.

Parameters **basedir** (*str*) – Base directory for the created file system store.

Return type *flowserv.volume.manager.VolumeManager*

class flowserv.volume.manager.**VolumeManager**(*stores: List[Dict], files: Optional[Dict[str, List[str]]] = None*)

Bases: object

The volume manager maintains information about storage volumes and the files that are available to the workers during workflow execution at each volume. The volume manager is the main component that maintains a virtual runtime environment in which all workers have access to their required input files.

The manager also acts as a factory for volume stores. When the manager is instantiated all storage volumes are specified via their dictionary serialization. The respective volume instances are only created when they are first accessed.

get(*identifier: str*) → *flowserv.volume.base.StorageVolume*

Get the instance for the storage volume with the given identifier.

identifier: str Unique storage volume identifier.

Return type *flowserv.volume.base.StorageVolume*

prepare(*store:* flowserv.volume.base.StorageVolume, *inputs: List[str]*, *outputs: List[str]*)

Prepare the storage volume for a worker.

Ensures that the input files that are needed by the worker are available in their latest version at the given volume store.

Raises a ValueError if a specified input file does not exist.

Parameters

- **store** (*flowserv.volume.base.StorageVolume*) – Storage volume that is being prepared.
- **inputs** (*list of string*) – Relative path (keys) of required input files for a workflow step.
- **outputs** (*list of string*) – Relative path (keys) of created output files by a workflow step.

update(*store:* flowserv.volume.base.StorageVolume, *files: List[str]*)

Update the availability index for workflow files.

The update method is used by a worker to signal the successful execution of a workflow step. The given files specify the output files that were generated by the worker. The store identifier references the volume store that now contains the latest version of these files.

Raises a ValueError if the specified storage volume does not exist.

Parameters

- **store** (*flowserv.volume.base.StorageVolume*) – Storage volume that contains the latest versions for the given files.
- **files** (*list of str*) – List of relative path (keys) for output files that were generated by a successful workflow step.

flowserv.volume.manager.**exact_match**(*s1: str*, *s2: str*) → bool

Test if two strings are exact matches.

Parameters

- **s1** (*string*) – Left side string of the comparison.
- **s2** (*string*) – Right side string of the comparison.

Return type bool

flowserv.volume.manager.**prefix_match**(*value: str*, *prefix: str*) → bool

Test of the given string value starts with a given prefix.

Parameters

- **value** (*str*) – Value for which the prefix is evaluated.
- **prefix** (*str*) – Prefix string that is tested for the given value.

Return type bool

flowserv.volume.s3 module

Implementation of the `flowserv.volume.base.GCVolume` for the use of AWS S3 buckets.

When using the S3Bucket the AWS credentials have to be configured. See the documentation for more details: https://docs.aws.amazon.com/cli/latest/userguide/cli-chap-configure.html

`flowserv.volume.s3.`**`S3Bucket`**(*bucket: str*, *prefix: Optional[str] = None*, *identifier: Optional[str] = None*) → Dict

Type identifier for storage volume serializations.

class `flowserv.volume.s3.`**`S3File`**(*bucket: flowserv.volume.s3.S3Bucket*, *key: str*)

Bases: *flowserv.volume.base.IOHandle*

Implementation of the file object interface for files that are stored on S3 object bucktes.

`open`() → IO

Get file contents as a BytesIO buffer.

Return type io.BytesIO

`size`() → int

Get size of the file in the number of bytes.

Return type int

class `flowserv.volume.s3.`**`S3Volume`**(*bucket_id: str*, *prefix: Optional[str] = None*, *identifier: Optional[str] = None*)

Bases: *flowserv.volume.base.StorageVolume*

Implementation of the bucket interface for AWS S3 buckets.

`close`()

The AWS S3 bucket resource does not need to be closed.

`delete`(*key: str*)

Delete file or folder with the given key.

Parameters **key** (*str*) – Path to a file object in the storage volume.

`describe`() → str

Get short descriptive string about the storage volume for display purposes.

Return type str

`erase`()

Erase the storage volume base directory and all its contents.

`static from_dict`(*doc*) → *flowserv.volume.s3.S3Volume*

Get S3 bucket storage volume instance from dictionary serialization.

Parameters **doc** (*dict*) – Dictionary serialization as returned by the `to_dict()` method.

Return type *flowserv.volume.s3.S3Volume*

`get_store_for_folder`(*key: str*, *identifier: Optional[str] = None*) → *flowserv.volume.base.StorageVolume*

Get storage volume for a sob-folder of the given volume.

Parameters

- **key** (*string*) – Relative path to sub-folder. The concatenation of the base folder for this storage volume and the given key will form te new base folder for the returned storage volume.
- **identifier** (*string, default=None*) – Unique volume identifier.

Return type *flowserv.volume.base.StorageVolume*

`load`(*key: str*) → *flowserv.volume.base.IOHandle*

Load a file object at the source path of this volume store.

Returns a file handle that can be used to open and read the file.

Parameters **key** (*str*) – Path to a file object in the storage volume.

Return type *flowserv.volume.base.IOHandle*

`mkdir`(*path: str*)

Create the directory with the given (relative) path and all of its parent directories.

For bucket stores no directories need to be created prior to accessing them.

Parameters **path** (*string*) – Relative path to a directory in the storage volume.

`query`(*filter: str*) → Iterable[str]

Get identifier for objects that match a given prefix.

Parameters **filter** (*str*) – Prefix query for object identifiers.

Return type iterable of string

`store`(*file:* flowserv.volume.base.IOHandle, *dst: str*)

Store a given file object at the destination path of this volume store.

Parameters

- **file** (*flowserv.volume.base.IOHandle*) – File-like object that is being stored.
- **dst** (*str*) – Destination path for the stored object.

`to_dict`() → Dict

Get dictionary serialization for the storage volume.

The returned serialization can be used by the volume factory to generate a new instance of this volume store.

Return type dict

`walk`(*src: str*) → List[Tuple[str, *flowserv.volume.base.IOHandle*]]

Get list of all files at the given source path.

If the source path references a single file the returned list will contain a single entry. If the source specifies a folder the result contains a list of all files in that folder and the subfolders.

Parameters **src** (*str*) – Source path specifying a file or folder.

Return type list of tuples (str, *flowserv.volume.base.IOHandle*)

flowserv.volume.ssh module

Workflow storage manager that uses a SSH client to connect to a remote server where run files are maintained.

class flowserv.volume.ssh.**RemoteStorage**(*client:* flowserv.util.ssh.SSHClient, *remotedir: str*, *identifier: Optional[str] = None*)

Bases: *flowserv.volume.base.StorageVolume*

File storage volume that connects to a remote server via sftp.

close()

Close the SSH connection when workflow execution is done.

delete(*key: str*)

Delete file or folder with the given key.

Parameters **key** (*str*) – Path to a file object in the storage volume.

describe() → str

Get short descriptive string about the storage volume for display purposes.

Return type str

erase()

Erase the storage volume base directory and all its contents.

static from_dict(*doc*) → *flowserv.volume.ssh.RemoteStorage*

Get remote storage volume instance from dictionary serialization.

Parameters **doc** (*dict*) – Dictionary serialization as returned by the `to_dict()` method.

Return type *flowserv.volume.ssh.RemoteStorage*

get_store_for_folder(*key: str*, *identifier: Optional[str] = None*) → *flowserv.volume.base.StorageVolume*

Get storage volume for a sob-folder of the given volume.

Parameters

- **key** (*string*) – Relative path to sub-folder. The concatenation of the base folder for this storage volume and the given key will form te new base folder for the returned storage volume.
- **identifier** (*string, default=None*) – Unique volume identifier.

Return type *flowserv.volume.base.StorageVolume*

load(*key: str*) → *flowserv.volume.base.IOHandle*

Load a file object at the source path of this volume store.

Returns a file handle that can be used to open and read the file.

Parameters **key** (*str*) – Path to a file object in the storage volume.

Return type *flowserv.volume.base.IOHandle*

mkdir(*path: str*)

Create the directory with the given (relative) path and all of its parent directories.

For bucket stores no directories need to be created prior to accessing them.

Parameters **path** (*string*) – Relative path to a directory in the storage volume.

store(*file:* flowserv.volume.base.IOHandle, *dst: str*)

Store a given file object at the destination path of this volume store.

Parameters

- **file** (*flowserv.volume.base.IOHandle*) – File-like object that is being stored.
- **dst** (*str*) – Destination path for the stored object.

to_dict() → Dict

Get dictionary serialization for the storage volume.

The returned serialization can be used by the volume factory to generate a new instance of this volume store.

Return type dict

walk(*src: str*) → List[Tuple[str, *flowserv.volume.base.IOHandle*]]

Get list of all files at the given source path.

If the source path references a single file the returned list will contain a single entry. If the source specifies a folder the result contains a list of all files in that folder and the subfolders.

Parameters **src** (*str*) – Source path specifying a file or folder.

Return type list of tuples (str, *flowserv.volume.base.IOHandle*)

class flowserv.volume.ssh.**SFTPFile**(*filename: str*, *client:* flowserv.util.ssh.SSHClient)

Bases: *flowserv.volume.base.IOHandle*

Implementation of the IO object handle interface for files that are stored on a remote file system.

open() → IO

Get file contents as a BytesIO buffer.

Return type io.BytesIO

Raises *flowserv.error.UnknownFileError* –

size() → int

Get size of the file in the number of bytes.

Return type int

flowserv.volume.ssh.**Sftp**(*remotedir: str*, *hostname: str*, *port: Optional[int] = None*, *timeout: Optional[float] = None*, *look_for_keys: Optional[bool] = False*, *sep: Optional[str] = '/'*, *identifier: Optional[str] = None*) → Dict

Get configuration object for a remote server storage volume that is accessed via sftp.

Parameters

- **remotedir** (*string*) – Base directory for stored files on the remote server.
- **hostname** (*string*) – Server to connect to.
- **port** (*int, default=None*) – Server port to connect to.
- **timeout** (*float, default=None*) – Optional timeout (in seconds) for the TCP connect.
- **look_for_keys** (*bool, default=False*) – Set to True to enable searching for discoverable private key files in `~/.ssh/`.
- **sep** (*string, default='/'*) – Path separator used by the remote file system.
- **identifier** (*string, default=None*) – Unique storage volume identifier.

Return type dict

flowserv.volume.ssh.**sftp_mkdir**(*client: paramiko.sftp_client.SFTPClient*, *dirpath: str*)

Create a directory on the remote server.

client: paramiko.SFTPClient SFTP client.

dirpath: string Path to the created directory on the remote server.

7.1.2 Submodules

flowserv.config module

Collection of configuration parameters for different components of flowserv.

The module contains helper classes to get configuration values from environment variables and to customize the configuration settings.

flowserv.config.**API_DEFAULTDIR**() → str

The default API base directory is a subfolder in the users data cache directory.

The default user cache directory is different for different OS's. We use appdirs.user_cache_dir to get the directory.

Return type string

flowserv.config.**API_URL**(*env: Dict*) → str

Get the base URL for the API from the respective environment variables 'FLOWSERV_API_HOST', 'FLOWSERV_API_PATH', and 'FLOWSERV_API_PORT' in the given configuration dictionary.

Parameters **env** (*dict*) – Configuration object that provides access to configuration parameters in the environment.

Return type string

flowserv.config.**APP**() → str

Get the value for the FLOWSERV_APP variable from the environment. Raises a missing configuration error if the value is not set. :rtype: string

flowserv.config.**AUTH_OPEN = 'open'**

Default user.

class flowserv.config.**Config**(*defaults: Optional[Dict] = None*)

Bases: dict

Helper class that extends a dictionary with dedicated methods to set individual parameters in the flowserv configuration.

Methods with lower case names are setters for configuration parameters. All setters return a reference to the configuration object itself to allows chanining the setter calls.

auth() → *flowserv.config.Config*

Set the authentication method to the default value that requires authentication.

Return type *flowserv.config.Config*

basedir(*path: str*) → *flowserv.config.Config*

Set the flowserv base directory.

Parameters **path** (*string*) – Path to the base directory for all workflow files.

Return type *flowserv.config.Config*

database(*url: str*) → *flowserv.config.Config*

Set the database connect Url.

Parameters **url** (*string*) – Database connect Url.

Return type *flowserv.config.Config*

multiprocess_engine() → *flowserv.config.Config*

Set configuration to use the serial multi-porcess workflow controller as the default backend.

Return type *flowserv.config.Config*

open_access() → *flowserv.config.Config*

Set the authentication method to open access.

Return type *flowserv.config.Config*

run_async() → *flowserv.config.Config*

Set the run asynchronous flag to True.

Return type *flowserv.config.Config*

run_sync() → *flowserv.config.Config*

Set the run asynchronous flag to False.

Return type *flowserv.config.Config*

token_timeout(*timeout: int*) → *flowserv.config.Config*

Set the authentication token timeout interval.

Return type *flowserv.config.Config*

volume(*config: dict*) → *flowserv.config.Config*

Set configuration object for the file storage volume.

Parameters **config** (*dict*) – Volume configuration information for the storage volume factory.

Return type *flowserv.config.Config*

webapp() → *flowserv.config.Config*

Set the web app flag to True.

Return type *flowserv.config.Config*

flowserv.config.**DEFAULT_POLL_INTERVAL = 2**

Environment variables for the command line interface.

flowserv.config.**DEFAULT_USER = '00000000'**

Environment variables that control the configuration of the workflow controllers.

flowserv.config.**FLOWSERV_API_PROTOCOL = 'FLOWSERV_API_PROTOCOL'**

Default values for environment variables.

flowserv.config.**FLOWSERV_AUTH = 'FLOWSERV_AUTH'**

Default values for environment variables.

flowserv.config.**FLOWSERV_WEBAPP = 'FLOWSERV_WEBAPP'**

Environment variables that are used to configure the file store that is used to maintain files for workflow templates, user group uploads, and workflow runs.

flowserv.config.**REMOTE_CLIENT = 'remote'**

Environment variable that contains the database connection string.

flowserv.config.**env**() → *flowserv.config.Config*

Get configuration parameters from the environment.

flowserv.config.**read_config_obj**(*filename: Union[str, Dict]*) → Dict

Read configuration object from a file.

This function only attempts to read an object from disk if the type of the filename argument is string.

Parameters **filename** (*str or dict*) – Path to file on disk.

Return type dict

flowserv.config.**to_bool**(*value: Any*) → bool

Convert given value to Boolean. Only of the string matches 'True' (ignoring case) the result will be True.

Parameters **value** (*any*) – Expects a string representation of a Boolean value.

Return type bool

flowserv.config.**to_float**(*value: Any*) → float

Convert given value to float.

Parameters **value** (*any*) – Expects an integer, float, or a string representation of an integer or float value.

Return type int

flowserv.config.**to_int**(*value: Any*) → int

Convert given value to integer.

Parameters **value** (*any*) – Expects an integer value or a string representation of an integer value.

Return type int

flowserv.error module

Exceptions that are raised by the various components of the reproducible open benchmark platform.

exception flowserv.error.**ConstraintViolationError**(*message: str*)

Bases: *flowserv.error.FlowservError*

Exception raised when an (implicit) constraint is violated by a requested operation. Example constraints are (i) names that are expected to be unique, (ii) names that cannot have more than n characters long, etc.

exception flowserv.error.**DuplicateArgumentError**(*identifier: str*)

Bases: *flowserv.error.ConstraintViolationError*

Exception indicating that a given argument for a workflow run is not unique.

exception flowserv.error.**DuplicateUserError**(*user_id: str*)

Bases: *flowserv.error.ConstraintViolationError*

Exception indicating that a given user already exists.

exception flowserv.error.**FlowservError**(*message: str*)

Bases: Exception

Base exception indicating that a component of the reproducible and reusable data analysis server encountered an error situation.

exception flowserv.error.**InvalidArgumentError**(*message: str*)

Bases: *flowserv.error.ConstraintViolationError*

Exception indicating that a given template parameter argument value is not valid.

exception flowserv.error.**InvalidConfigurationError**(*name: str*, *value: Any*)

Bases: *flowserv.error.FlowservError*

Error indicating that the value for a mandatory environment variable is invalid.

exception flowserv.error.**InvalidManifestError**(*message: str*)

Bases: *flowserv.error.ConstraintViolationError*

Exception indicating that a given workflow manifest is invalid or has missing elements.

exception flowserv.error.**InvalidParameterError**(*message: str*)

Bases: *flowserv.error.ConstraintViolationError*

Exception indicating that a given template parameter is invalid.

exception flowserv.error.**InvalidRunStateError**(*state*, *resulting_state=None*)

Bases: *flowserv.error.ConstraintViolationError*

Exception indicating that an attempt to modify the state of a run was made that is not allowed in the current run state or that would result in an illegal sequence of workflow states.

exception flowserv.error.**InvalidTemplateError**(*message: str*)

Bases: *flowserv.error.ConstraintViolationError*

Exception indicating that a given workflow template is invalid or has missing elements.

exception flowserv.error.**MissingArgumentError**(*identifier: str*)

Bases: *flowserv.error.ConstraintViolationError*

Exception indicating that a required parameter in a workflow template has no argument given for a workflow run.

exception flowserv.error.**MissingConfigurationError**(*name: str*)

Bases: *flowserv.error.FlowservError*

Error indicating that the value for a mandatory environment variable is not set.

exception flowserv.error.**UnauthenticatedAccessError**

Bases: *flowserv.error.FlowservError*

This exception is raised if an unauthenticated user attempts to access or manipulate application resources.

exception flowserv.error.**UnauthorizedAccessError**

Bases: *flowserv.error.FlowservError*

This exception is raised if an authenticated user attempts to access or manipulate application resources that they have not authorization to.

exception flowserv.error.**UnknownFileError**(*file_id: str*)

Bases: *flowserv.error.UnknownObjectError*

Exception indicating that a given file identifier is unknown.

exception flowserv.error.**UnknownObjectError**(*obj_id: str*, *type_name: Optional[str] = 'object'*)

Bases: *flowserv.error.FlowservError*

Generic error for references to unknown objects.

exception flowserv.error.**UnknownParameterError**(*identifier: str*)

Bases: *flowserv.error.UnknownObjectError*

Exception indicating that a workflow specification references a parameter that is not defined for a given template.

exception flowserv.error.**UnknownRequestError**(*request_id: str*)

Bases: *flowserv.error.UnknownObjectError*

Exception indicating that a given password reset request identifier is unknown.

exception flowserv.error.**UnknownRunError**(*identifier: str*)

Bases: *flowserv.error.UnknownObjectError*

Exception indicating that a given run identifier does not reference a known workflow run.

exception flowserv.error.**UnknownUserError**(*user_id: str*)

Bases: *flowserv.error.UnknownObjectError*

Exception indicating that a given user identifier is unknown.

exception flowserv.error.**UnknownWorkflowError**(*workflow_id: str*)

Bases: *flowserv.error.UnknownObjectError*

Exception indicating that a given workflow identifier is unknown.

exception flowserv.error.**UnknownWorkflowGroupError**(*group_id: str*)

Bases: *flowserv.error.UnknownObjectError*

Exception indicating that a given workflow group identifier is unknown.

flowserv.scanner module

Simple scanner classes to collect values (e.g., for template parameters) from different sources (e.g. standard input). The Scanner class implements the methods that are used to convert input values into the different data types that are supported for parameter declarations.

class flowserv.scanner.**InputReader**

Bases: *flowserv.scanner.TokenReader*

Token reader that reads tokens from standard input.

next_token()

Read token from standard input.

Return type string

class flowserv.scanner.**ListReader**(*tokens*)

Bases: *flowserv.scanner.TokenReader*

Token reader that is initialized with a list of values. Returns tokens from the list until the end of the list is reached.

next_token()

Return next token from the token list. If the end of the list has been reached None is returned.

Return type string

class flowserv.scanner.**Scanner**(*reader=None*)

Bases: object

Scanner that converts input tokens into the simple scalar data types boolean, float, file, integer and string that are supported in the template parameter declarations.

next_bool(*default_value=None*)

Return next token as boolean. If the read token is an empty string the given default value is returned.

Raises ValueError if the token value cannot be converted to boolean.

Any of the following values will be recognized as valid boolean values. All values are case-insensitive:

- True: [true, yes, y, t, 1]
- False:[false, no, n, f, 0]

Parameters **default_value** (*bool, optional*) – Default value that is returned if the read token is an empty string

Return type bool

next_file(*default_value=None*)

Return next token as string representing a file name. There are no tests performed to ensure whether the given value represents a valid path name or not (since the definition of a valid path name is very much dependent on the OS).

Parameters **default_value** (*string, optional*) – Default value that is returned if the read token is an empty string

Return type string

next_float(*default_value=None*)

Return next token as float. Raises ValueError if the token value cannot be converted to float.

Parameters **default_value** (*float, optional*) – Default value that is returned if the read token is an empty string

Return type float

next_int(*default_value=None*)

Return next token as integer. Raises ValueError if the token value cannot be converted to integer.

Parameters **default_value** (*int, optional*) – Default value that is returned if the read token is an empty string

Return type int

next_string(*default_value=None*)

Return next token as string.

Parameters **default_value** (*string, optional*) – Default value that is returned if the read token is an empty string

Return type string

class flowserv.scanner.**TokenReader**

Bases: object

Abstract token reader class that is used by the scanner to get the next input token.

abstract next_token()

Read the next token.

Return type string

flowserv.version module

Information about the current version of the flowServ package.

PYTHON MODULE INDEX

f

INDEX

D

E

F

G

H

I

J

K

L

M

N

O

P

S

T

U

V

W