
Common Workflow Language reference implementation

Release 3.1.20240508115724

Peter Amstutz and Common Workflow Language Project contributors

May 16, 2024

CONTENTS:

1	Install	3
1.1	cwltool packages	3
1.2	MS Windows users	4
1.3	cwltool development version	4
1.4	Recommended Software	4
2	Run on the command line	5
2.1	Use with boot2docker on macOS	5
2.2	Using uDocker	5
2.3	Using Singularity	6
2.4	Running a tool or workflow from remote or local locations	6
2.5	Overriding workflow requirements at load time	6
2.6	Combining parts of a workflow into a single document	7
2.7	Running only part of a workflow	7
2.8	Visualizing a CWL document	7
2.9	Modeling a CWL document as RDF	8
2.10	Environment Variables in cwltool	8
2.11	Use with GA4GH Tool Registry API	11
2.12	Running MPI-based tools that need to be launched	12
2.13	Enabling Fast Parser (experimental)	13
3	Development	15
3.1	Running tests locally	15
3.2	Import as a module	15
3.3	CWL Tool Control Flow	16
3.4	Extension points	17
3.5	Indices and tables	152
	Python Module Index	153
	Index	155

PyPI:

Conda:

Debian:

Quay.io (Docker):

This is the reference implementation of the [Common Workflow Language open standards](#). It is intended to be feature complete and provide comprehensive validation of CWL files as well as provide other tools related to working with CWL.

cwltool is written and tested for [Python 3.x](#) {x = 6, 8, 9, 10, 11}

The reference implementation consists of two packages. The cwltool package is the primary Python module containing the reference implementation in the cwltool module and console executable by the same name.

The cwlref-runner package is optional and provides an additional entry point under the alias cwl-runner, which is the implementation-agnostic name for the default CWL interpreter installed on a host.

cwltool is provided by the CWL project, [a member project of Software Freedom Conservancy](#) and our [many contributors](#).

Table of Contents

- *cwltool: The reference reference implementation of the Common Workflow Language standards*
 - *Install*
 - * *cwltool packages*
 - * *MS Windows users*
 - * *cwltool development version*
 - * *Recommended Software*
 - *Run on the command line*
 - * *Use with boot2docker on macOS*
 - * *Using uDocker*
 - * *Using Singularity*
 - * *Running a tool or workflow from remote or local locations*
 - * *Overriding workflow requirements at load time*
 - * *Combining parts of a workflow into a single document*
 - * *Running only part of a workflow*
 - * *Visualizing a CWL document*
 - * *Modeling a CWL document as RDF*
 - * *Environment Variables in cwltool*
 - *Leveraging SoftwareRequirements (Beta)*
 - * *Use with GA4GH Tool Registry API*
 - * *Running MPI-based tools that need to be launched*

- * *Enabling Fast Parser (experimental)*
- *Development*
 - * *Running tests locally*
 - * *Import as a module*
 - * *CWL Tool Control Flow*
 - * *Extension points*
 - * *Indices and tables*

INSTALL

1.1 cwltool packages

Your operating system may offer cwltool directly. For [Debian](#), [Ubuntu](#), and similar Linux distribution try

```
sudo apt-get install cwltool
```

If you encounter an error, first try to update package information by using

```
sudo apt-get update
```

If you are running macOS X or other UNIXes and you want to use packages prepared by the conda-forge project, then please follow the install instructions for [conda-forge](#) (if you haven't already) and then

```
conda install -c conda-forge cwltool
```

All of the above methods of installing cwltool use packages that might contain bugs already fixed in newer versions or be missing desired features. If the packaged version of cwltool available to you is too old, then we recommend installing using pip and venv

```
python3 -m venv env      # Create a virtual environment named 'env' in the current_
↪directory
source env/bin/activate  # Activate environment before installing `cwltool`
```

Then install the latest cwlref-runner package from PyPi (which will install the latest cwltool package as well)

```
pip install cwlref-runner
```

If installing alongside another CWL implementation (like `toil-cwl-runner` or `arvados-cwl-runner`) then instead run

```
pip install cwltool
```

1.2 MS Windows users

1. Install Windows Subsystem for Linux 2 and Docker Desktop.
2. Install Debian from the Microsoft Store.
3. Set Debian as your default WSL 2 distro: `wsl --set-default debian`.
4. Return to the Docker Desktop, choose Settings → Resources → WSL Integration and under “Enable integration with additional distros” select “Debian”,
5. Reboot if you have not yet already.
6. Launch Debian and follow the Linux instructions above (`apt-get install cwltool` or use the `venv` method)

Network problems from within WSL2? Try [these instructions](#) followed by `wsl --shutdown`.

1.3 cwltool development version

Or you can skip the direct `pip` commands above and install the latest development version of `cwltool`:

```
git clone https://github.com/common-workflow-language/cwltool.git # clone (copy) the
↪cwltool git repository
cd cwltool                # Change to source directory that git clone just downloaded
pip install .[deps]        # Installs ``cwltool`` from source
cwltool --version          # Check if the installation works correctly
```

Remember, if co-installing multiple CWL implementations, then you need to maintain which implementation `cwl-runner` points to via a symbolic file system link or [another facility](#).

1.4 Recommended Software

We strongly suggested to have the following installed:

- One of the following software container engines
 - [Podman](#)
 - [Docker](#)
 - Singularity/Apptainer: See [Using Singularity](#)
 - `udocker`: See [Using uDocker](#)
- [node.js](#) for evaluating CWL Expressions quickly (required for `udocker` users, optional but recommended for the other container engines).

Without these, some examples in the CWL tutorials at http://www.commonwl.org/user_guide/ may not work.

RUN ON THE COMMAND LINE

Simple command:

```
cwl-runner my_workflow.cwl my_inputs.yaml
```

Or if you have multiple CWL implementations installed and you want to override the default cwl-runner then use:

```
cwltool my_workflow.cwl my_inputs.yaml
```

You can set cwltool options in the environment with CWLTOOL_OPTIONS, these will be inserted at the beginning of the command line:

```
export CWLTOOL_OPTIONS="--debug"
```

2.1 Use with boot2docker on macOS

boot2docker runs Docker inside a virtual machine, and it only mounts Users on it. The default behavior of CWL is to create temporary directories under e.g. /Var which is not accessible to Docker containers.

To run CWL successfully with boot2docker you need to set the --tmpdir-prefix and --tmp-outdir-prefix to somewhere under /Users:

```
$ cwl-runner --tmp-outdir-prefix=/Users/username/project --tmpdir-prefix=/Users/username/  
↳project wc-tool.cwl wc-job.json
```

2.2 Using uDocker

Some shared computing environments don't support Docker software containers for technical or policy reasons. As a workaround, the CWL reference runner supports using the [udocker](https://indigo-dc.github.io/udocker/installation_manual.html) program on Linux using --udocker.

udocker installation: https://indigo-dc.github.io/udocker/installation_manual.html

Run *cwltool* just as you usually would, but with --udocker prior to the workflow path:

```
cwltool --udocker https://github.com/common-workflow-language/common-workflow-language/  
↳raw/main/v1.0/v1.0/test-cwl-out2.cwl https://github.com/common-workflow-language/  
↳common-workflow-language/raw/main/v1.0/v1.0/empty.json
```

As was mentioned in the *Recommended Software* section,

2.3 Using Singularity

cwltool can also use [Singularity](#) version 2.6.1 or later as a Docker container runtime. cwltool with Singularity will run software containers specified in `DockerRequirement` and therefore works with Docker images only, native Singularity images are not supported. To use Singularity as the Docker container runtime, provide `--singularity` command line option to cwltool. With Singularity, cwltool can pass all CWL v1.0 conformance tests, except those involving Docker container ENTRYPOINTS.

Example

```
cwltool --singularity https://github.com/common-workflow-language/common-workflow-
language/raw/main/v1.0/v1.0/cat3-tool-mediumcut.cwl https://github.com/common-workflow-
language/common-workflow-language/raw/main/v1.0/v1.0/cat-job.json
```

2.4 Running a tool or workflow from remote or local locations

cwltool can run tool and workflow descriptions on both local and remote systems via its support for HTTP[S] URLs.

Input job files and Workflow steps (via the *run* directive) can reference CWL documents using absolute or relative local filesystem paths. If a relative path is referenced and that document isn't found in the current directory, then the following locations will be searched: http://www.commonwl.org/v1.0/CommandLineTool.html#Discovering_CWL_documents_on_a_local_filesystem

You can also use `cwlddep` to manage dependencies on external tools and workflows.

2.5 Overriding workflow requirements at load time

Sometimes a workflow needs additional requirements to run in a particular environment or with a particular dataset. To avoid the need to modify the underlying workflow, cwltool supports requirement “overrides”.

The format of the “overrides” object is a mapping of item identifier (workflow, workflow step, or command line tool) to the process requirements that should be applied.

```
cwltool:overrides:
  echo.cwl:
    requirements:
      EnvVarRequirement:
        envDef:
          MESSAGE: override_value
```

Overrides can be specified either on the command line, or as part of the job input document. Workflow steps are identified using the name of the workflow file followed by the step name as a document fragment identifier “#id”. Override identifiers are relative to the top-level workflow document.

```
cwltool --overrides overrides.yml my-tool.cwl my-job.yml
```

```
input_parameter1: value1
input_parameter2: value2
cwltool:overrides:
  workflow.cwl#step1:
    requirements:
```

(continues on next page)

(continued from previous page)

```
EnvVarRequirement:
  envDef:
    MESSAGE: override_value
```

```
cwltool my-tool.cwl my-job-with-overrides.yml
```

2.6 Combining parts of a workflow into a single document

Use `--pack` to combine a workflow made up of multiple files into a single compound document. This operation takes all the CWL files referenced by a workflow and builds a new CWL document with all Process objects (CommandLineTool and Workflow) in a list in the `$graph` field. Cross references (such as `run:` and `source:` fields) are updated to internal references within the new packed document. The top-level workflow is named `#main`.

```
cwltool --pack my-wf.cwl > my-packed-wf.cwl
```

2.7 Running only part of a workflow

You can run a partial workflow with the `--target` (`-t`) option. This takes the name of an output parameter, workflow step, or input parameter in the top-level workflow. You may provide multiple targets.

```
cwltool --target step3 my-wf.cwl
```

If a target is an output parameter, it will only run only the steps that contribute to that output. If a target is a workflow step, it will run the workflow starting from that step. If a target is an input parameter, it will only run the steps connected to that input.

Use `--print-targets` to get a listing of the targets of a workflow. To see which steps will run, use `--print-subgraph` with `--target` to get a printout of the workflow subgraph for the selected targets.

```
cwltool --print-targets my-wf.cwl
```

```
cwltool --target step3 --print-subgraph my-wf.cwl > my-wf-starting-from-step3.cwl
```

2.8 Visualizing a CWL document

The `--print-dot` option will print a file suitable for Graphviz dot program. Here is a bash onliner to generate a Scalable Vector Graphic (SVG) file:

```
cwltool --print-dot my-wf.cwl | dot -Tsvg > my-wf.svg
```

2.9 Modeling a CWL document as RDF

CWL documents can be expressed as RDF triple graphs.

```
cwltool --print-rdf --rdf-serializer=turtle mywf.cwl
```

2.10 Environment Variables in cwltool

This reference implementation supports several ways of setting environment variables for tools, in addition to the standard `EnvVarRequirement`. The sequence of steps applied to create the environment is:

0. If the `--preserve-entire-environment` flag is present, then begin with the current environment, else begin with an empty environment.
1. Add any variables specified by `--preserve-environment` option(s).
2. Set `TMPDIR` and `HOME` per the [CWL v1.0+ CommandLineTool](#) specification.
3. Apply any `EnvVarRequirement` from the `CommandLineTool` description.
4. Apply any manipulations required by any `cwltool:MPIRequirement` extensions.
5. Substitute any secrets required by `Secrets` extension.
6. Modify the environment in response to `SoftwareRequirement` (see below).

2.10.1 Leveraging SoftwareRequirements (Beta)

CWL tools may be decorated with `SoftwareRequirement` hints that cwltool may in turn use to resolve to packages in various package managers or dependency management systems such as [Environment Modules](#).

Utilizing `SoftwareRequirement` hints using cwltool requires an optional dependency, for this reason be sure to use specify the `deps` modifier when installing cwltool. For instance:

```
$ pip install 'cwltool[deps]'
```

Installing cwltool in this fashion enables several new command line options. The most general of these options is `--beta-dependency-resolvers-configuration`. This option allows one to specify a dependency resolver's configuration file. This file may be specified as either XML or YAML and very simply describes various plugins to enable to "resolve" `SoftwareRequirement` dependencies.

Using these hints will allow cwltool to modify the environment in which your tool runs, for example by loading one or more `Environment Modules`. The environment is constructed as above, then the environment may modified by the selected tool resolver. This currently means that you cannot override any environment variables set by the selected tool resolver. Note that the environment given to the configured dependency resolver has the variable `_CWLTOOL` set to `1` to allow introspection.

To discuss some of these plugins and how to configure them, first consider the following `hint` definition for an example CWL tool.

```
SoftwareRequirement:  
  packages:  
    - package: seqtk  
      version:  
        - r93
```

Now imagine deploying cwltool on a cluster with Software Modules installed and that a `seqtk` module is available at version `r93`. This means cluster users likely won't have the binary `seqtk` on their `PATH` by default, but after sourcing this module with the command `modulecmd sh load seqtk/r93 seqtk` is available on the `PATH`. A simple dependency resolvers configuration file, called `dependency-resolvers-conf.yml` for instance, that would enable cwltool to source the correct module environment before executing the above tool would simply be:

```
- type: modules
```

The outer list indicates that one plugin is being enabled, the plugin parameters are defined as a dictionary for this one list item. There is only one required parameter for the plugin above, this is `type` and defines the plugin type. This parameter is required for all plugins. The available plugins and the parameters available for each are documented (incompletely) [here](#). Unfortunately, this documentation is in the context of Galaxy tool requirements instead of CWL SoftwareRequirements, but the concepts map fairly directly.

cwltool is distributed with an example of such `seqtk` tool and sample corresponding job. It could be executed from the cwltool root using a dependency resolvers configuration file such as the above one using the command:

```
cwltool --beta-dependency-resolvers-configuration /path/to/dependency-resolvers-conf.yml \
  ↪ \
    tests/seqtk_seq.cwl \
    tests/seqtk_seq_job.json
```

This example demonstrates both that cwltool can leverage existing software installations and also handle workflows with dependencies on different versions of the same software and libraries. However the above example does require an existing module setup so it is impossible to test this example “out of the box” with cwltool. For a more isolated test that demonstrates all the same concepts - the resolver plugin type `galaxy_packages` can be used.

“Galaxy packages” are a lighter-weight alternative to Environment Modules that are really just defined by a way to lay out directories into packages and versions to find little scripts that are sourced to modify the environment. They have been used for years in Galaxy community to adapt Galaxy tools to cluster environments but require neither knowledge of Galaxy nor any special tools to setup. These should work just fine for CWL tools.

The cwltool source code repository's test directory is setup with a very simple directory that defines a set of “Galaxy packages” (but really just defines one package named `random-lines`). The directory layout is simply:

```
tests/test_deps_env/
  random-lines/
    1.0/
      env.sh
```

If the `galaxy_packages` plugin is enabled and pointed at the `tests/test_deps_env` directory in cwltool's root and a SoftwareRequirement such as the following is encountered.

```
hints:
  SoftwareRequirement:
    packages:
      - package: 'random-lines'
        version:
          - '1.0'
```

Then cwltool will simply find that `env.sh` file and source it before executing the corresponding tool. That `env.sh` script is only responsible for modifying the job's `PATH` to add the required binaries.

This is a full example that works since resolving “Galaxy packages” has no external requirements. Try it out by executing the following command from cwltool's root directory:

```
cwltool --beta-dependency-resolvers-configuration tests/test_deps_env_resolvers_conf.yml \
  tests/random_lines.cwl \
  tests/random_lines_job.json
```

The resolvers configuration file in the above example was simply:

```
- type: galaxy_packages
  base_path: ./tests/test_deps_env
```

It is possible that the `SoftwareRequirements` in a given CWL tool will not match the module names for a given cluster. Such requirements can be re-mapped to specific deployed packages or versions using another file specified using the resolver plugin parameter *mapping_files*. We will demonstrate this using *galaxy_packages*, but the concepts apply equally well to Environment Modules or Conda packages (described below), for instance.

So consider the resolvers configuration file. (*tests/test_deps_env_resolvers_conf_rewrite.yml*):

```
- type: galaxy_packages
  base_path: ./tests/test_deps_env
  mapping_files: ./tests/test_deps_mapping.yml
```

And the corresponding mapping configuration file (*tests/test_deps_mapping.yml*):

```
- from:
    name: randomLines
    version: 1.0.0-rc1
  to:
    name: random-lines
    version: '1.0'
```

This is saying if cwltool encounters a requirement of `randomLines` at version `1.0.0-rc1` in a tool, to rewrite to our specific plugin as `random-lines` at version `1.0`. cwltool has such a test tool called `random_lines_mapping.cwl` that contains such a source `SoftwareRequirement`. To try out this example with mapping, execute the following command from the cwltool root directory:

```
cwltool --beta-dependency-resolvers-configuration tests/test_deps_env_resolvers_conf_rewrite.yml \
  tests/random_lines_mapping.cwl \
  tests/random_lines_job.json
```

The previous examples demonstrated leveraging existing infrastructure to provide requirements for CWL tools. If instead a real package manager is used cwltool has the opportunity to install requirements as needed. While initial support for Homebrew/Linuxbrew plugins is available, the most developed such plugin is for the [Conda](#) package manager. Conda has the nice properties of allowing multiple versions of a package to be installed simultaneously, not requiring elevated permissions to install Conda itself or packages using Conda, and being cross-platform. For these reasons, cwltool may run as a normal user, install its own Conda environment and manage multiple versions of Conda packages on Linux and Mac OS X.

The Conda plugin can be endlessly configured, but a sensible set of defaults that has proven a powerful stack for dependency management within the Galaxy tool development ecosystem can be enabled by simply passing cwltool the `--beta-conda-dependencies` flag.

With this, we can use the seqtk example above without Docker or any externally managed services - cwltool should install everything it needs and create an environment for the tool. Try it out with the following command:

```
cwltool --beta-conda-dependencies tests/seqtk_seq.cwl tests/seqtk_seq_job.json
```

The CWL specification allows URIs to be attached to `SoftwareRequirement`s that allow disambiguation of package names. If the mapping files described above allow deployers to adapt tools to their infrastructure, this mechanism allows tools to adapt their requirements to multiple package managers. To demonstrate this within the context of the `seqtk`, we can simply break the package name we use and then specify a specific Conda package as follows:

```
hints:
  SoftwareRequirement:
    packages:
      - package: seqtk_seq
        version:
          - '1.2'
    specs:
      - https://anaconda.org/bioconda/seqtk
      - https://packages.debian.org/sid/seqtk
```

The example can be executed using the command:

```
cwltool --beta-conda-dependencies tests/seqtk_seq_wrong_name.cwl tests/seqtk_seq_job.json
```

The plugin framework for managing the resolution of these software requirements as maintained as part of `galaxy-tool-util` - a small, portable subset of the Galaxy project. More information on configuration and implementation can be found at the following links:

- [Dependency Resolvers in Galaxy](#)
- [Conda for \[Galaxy\] Tool Dependencies](#)
- [Mapping Files - Implementation](#)
- [Specifications - Implementation](#)
- [Initial cwltool Integration Pull Request](#)

2.11 Use with GA4GH Tool Registry API

Cwltool can launch tools directly from [GA4GH Tool Registry API](#) endpoints.

By default, cwltool searches <https://dockstore.org/>. Use `--add-tool-registry` to add other registries to the search path.

For example

```
cwltool quay.io/collaboratory/dockstore-tool-bamstats:develop test.json
```

and (defaults to latest when a version is not specified)

```
cwltool quay.io/collaboratory/dockstore-tool-bamstats test.json
```

For this example, grab the `test.json` (and input file) from <https://github.com/CancerCollaboratory/dockstore-tool-bamstats>

```
wget https://dockstore.org/api/api/ga4gh/v2/tools/quay.io%2Fbriandoconnor%2Fdockstore-
↪tool-bamstats/versions/develop/PLAIN-CWL/descriptor/test.json
```

(continues on next page)

(continued from previous page)

```
wget https://github.com/CancerCollaboratory/dockstore-tool-bamstats/raw/develop/rna.
↳ SRR948778.bam
```

2.12 Running MPI-based tools that need to be launched

Cwltool supports an extension to the CWL spec <http://commonwl.org/cwltool#MPIRequirement>. When the tool definition has this in its `requirements/hints` section, and cwltool has been run with `--enable-ext`, then the tool's command line will be extended with the commands needed to launch it with `mpirun` or similar. You can specify the number of processes to start as either a literal integer or an expression (that will result in an integer). For example:

```
#!/usr/bin/env cwl-runner
cwlVersion: v1.1
class: CommandLineTool
$namespaces:
  cwltool: "http://commonwl.org/cwltool#"
requirements:
  cwltool:MPIRequirement:
    processes: $(inputs.nproc)
inputs:
  nproc:
    type: int
```

Interaction with containers: the `MPIRequirement` currently prepends its commands to the front of the command line that is constructed. If you wish to run a containerized application in parallel, for simple use cases, this does work with Singularity, depending upon the platform setup. However, this combination should be considered “alpha” – please do report any issues you have! This does not work with Docker at the moment. (More precisely, you get n copies of the same single process image run at the same time that cannot communicate with each other.)

The host-specific parameters are configured in a simple YAML file (specified with the `--mpi-config-file` flag). The allowed keys are given in the following table; all are optional.

Key	Type	De- fault	Description
runner	str	“mpirun”	The primary command to use.
nproc_flag	str	“-n”	Flag to set number of processes to start.
default_nproc	int	1	Default number of processes.
extra_flags	List[str]	[]	A list of any other flags to be added to the runner's command line before the <code>baseCommand</code> .
env_pass	List[str]	[]	A list of environment variables that should be passed from the host environment through to the tool (e.g., giving the node list as set by your scheduler).
env_pass_	List[str]	[]	A list of python regular expressions that will be matched against the host's environment. Those that match will be passed through.
env_set	Mapping[str, str]	{}	A dictionary whose keys are the environment variables set and the values being the values.

2.13 Enabling Fast Parser (experimental)

For very large workflows, *cwltool* can spend a lot of time in initialization, before the first step runs. There is an experimental flag `--fast-parser` which can dramatically reduce the initialization overhead, however as of this writing it has several limitations:

- Error reporting in general is worse than the standard parser, you will want to use it with workflows that you know are already correct.
- It does not check for dangling links (these will become runtime errors instead of loading errors)
- Several other cases fail, as documented in <https://github.com/common-workflow-language/cwltool/pull/1720>

DEVELOPMENT

3.1 Running tests locally

- Running basic tests (/tests):

To run the basic tests after installing *cwltool* execute the following:

```
pip install -rtest-requirements.txt
pytest    ## N.B. This requires node.js or docker to be available
```

To run various tests in all supported Python environments, we use *tox*. To run the test suite in all supported Python environments first clone the complete code repository (see the `git clone` instructions above) and then run the following in the terminal: `pip install "tox<4"; tox -p`

List of all environment can be seen using: `tox --listenvs` and running a specific test env using: `tox -e <env name>` and additionally run a specific test using this format: `tox -e py310-unit -- -v tests/test_examples.py::test_scandeps`

- Running the entire suite of CWL conformance tests:

The GitHub repository for the CWL specifications contains a script that tests a CWL implementation against a wide array of valid CWL files using the *cwlttest* program

Instructions for running these tests can be found in the Common Workflow Language Specification repository at https://github.com/common-workflow-language/common-workflow-language/blob/main/CONFORMANCE_TESTS.md.

3.2 Import as a module

Add

```
import cwltool
```

to your script.

The easiest way to use *cwltool* to run a tool or workflow from Python is to use a Factory

```
import cwltool.factory
fac = cwltool.factory.Factory()

echo = fac.make("echo.cwl")
result = echo(inp="foo")

# result["out"] == "foo"
```

3.3 CWL Tool Control Flow

Technical outline of how cwltool works internally, for maintainers.

1. Use `CWL load_tool()` to load document.
 1. Fetches the document from file or URL
 2. Applies preprocessing (syntax/identifier expansion and normalization)
 3. Validates the document based on `cwlVersion`
 4. If necessary, updates the document to the latest spec
 5. Constructs a `Process` object using `make_tool()` callback. This yields a `CommandLineTool`, `Workflow`, or `ExpressionTool`. For workflows, this recursively constructs each workflow step.
 6. To construct custom types for `CommandLineTool`, `Workflow`, or `ExpressionTool`, provide a custom `make_tool()`
2. Iterate on the `job()` method of the `Process` object to get back runnable jobs.
 1. `job()` is a generator method (uses the Python iterator protocol)
 2. Each time the `job()` method is invoked in an iteration, it returns one of: a runnable item (an object with a `run()` method), `None` (indicating there is currently no work ready to run) or end of iteration (indicating the process is complete.)
 3. Invoke the runnable item by calling `run()`. This runs the tool and gets output.
 4. An output callback reports the output of a process.
 5. `job()` may be iterated over multiple times. It will yield all the work that is currently ready to run and then yield `None`.
3. `Workflow` objects create a corresponding `WorkflowJob` and `WorkflowJobStep` objects to hold the workflow state for the duration of the job invocation.
 1. The `WorkflowJob` iterates over each `WorkflowJobStep` and determines if the inputs the step are ready.
 2. When a step is ready, it constructs an input object for that step and iterates on the `job()` method of the workflow job step.
 3. Each runnable item is yielded back up to top-level run loop
 4. When a step job completes and receives an output callback, the job outputs are assigned to the output of the workflow step.
 5. When all steps are complete, the intermediate files are moved to a final workflow output, intermediate directories are deleted, and the workflow's output callback is called.
4. `CommandLineTool job()` objects yield a single runnable object.
 1. The `CommandLineTool job()` method calls `make_job_runner()` to create a `CommandLineJob` object
 2. The job method configures the `CommandLineJob` object by setting public attributes
 3. The job method iterates over file and directories inputs to the `CommandLineTool` and creates a "path map".
 4. Files are mapped from their "resolved" location to a "target" path where they will appear at tool invocation (for example, a location inside a Docker container.) The target paths are used on the command line.
 5. Files are staged to targets paths using either Docker volume binds (when using containers) or symlinks (if not). This staging step enables files to be logically rearranged or renamed independent of their source layout.

6. The `run()` method of `CommandLineJob` executes the command line tool or Docker container, waits for it to complete, collects output, and makes the output callback.

3.4 Extension points

The following functions can be passed to `main()` to override or augment the listed behaviors.

executor

```
executor(tool, job_order_object, runtimeContext, logger)
(Process, Dict[Text, Any], RuntimeContext) -> Tuple[Dict[Text, Any], Text]
```

An implementation of the top-level workflow execution loop should synchronously run a process object to completion and return the output object.

versionfunc

```
()
() -> Text
```

Return version string.

logger_handler

```
logger_handler
logging.Handler
```

Handler object for logging.

The following functions can be set in `LoadingContext` to override or augment the listed behaviors.

fetcher_constructor

```
fetcher_constructor(cache, session)
(Dict[unicode, unicode], requests.sessions.Session) -> Fetcher
```

Construct a `Fetcher` object with the supplied cache and HTTP session.

resolver

```
resolver(document_loader, document)
(Loader, Union[Text, dict[Text, Any]]) -> Text
```

Resolve a relative document identifier to an absolute one that can be fetched.

The following functions can be set in `RuntimeContext` to override or augment the listed behaviors.

construct_tool_object

```
construct_tool_object(toolpath_object, loadingContext)
(ImmutableMapping[Text, Any], LoadingContext) -> Process
```

Hook to construct a `Process` object (eg `CommandLineTool`) object from a document.

select_resources

```
selectResources(request)
(Dict[str, int], RuntimeContext) -> Dict[Text, int]
```

Take a resource request and turn it into a concrete resource assignment.

make_fs_access

```
make_fs_access(basedir)
(Text) -> StdFsAccess
```

Return a file system access object.

In addition, when providing custom subclasses of Process objects, you can override the following methods:

CommandLineTool.make_job_runner

```
make_job_runner(RuntimeContext)
(RuntimeContext) -> Type[JobBase]
```

Create and return a job runner object (this implements concrete execution of a command line tool).

Workflow.make_workflow_step

```
make_workflow_step(toolpath_object, pos, loadingContext, parentworkflowProv)
(Dict[Text, Any], int, LoadingContext, Optional[ProvenanceProfile]) -> WorkflowStep
```

Create and return a workflow step object.

3.4.1 cwltool Command Line Options

cwltool

Reference executor for Common Workflow Language standards. Not for production use.

```
usage: cwltool [-h] [--basedir BASEDIR] [--outdir OUTDIR] [--log-dir LOG_DIR]
               [--parallel]
               [--preserve-environment ENVVAR | --preserve-entire-environment]
               [--rm-container | --leave-container]
               [--cidfile-dir CIDFILE_DIR] [--cidfile-prefix CIDFILE_PREFIX]
               [--tmpdir-prefix TMPDIR_PREFIX]
               [--tmp-outdir-prefix TMP_OUTDIR_PREFIX | --cachedir CACHEDIR]
               [--rm-tmpdir | --leave-tmpdir]
               [--move-outputs | --leave-outputs | --copy-outputs]
               [--enable-pull | --disable-pull]
               [--rdf-serializer RDF_SERIALIZER] [--eval-timeout EVAL_TIMEOUT]
               [--provenance PROVENANCE] [--enable-user-provenance]
               [--disable-user-provenance] [--enable-host-provenance]
               [--disable-host-provenance] [--orcid ORCID]
               [--full-name CWL_FULL_NAME]
               [--print-rdf | --print-dot | --print-pre | --print-deps | --print-input-
-> deps | --pack | --version | --validate | --print-supported-versions | --print-subgraph
-> | --print-targets | --make-template]
               [--strict | --non-strict] [--skip-schemas]
               [--no-doc-cache | --doc-cache]
               [--verbose | --no-warnings | --quiet | --debug]
               [--write-summary WRITE_SUMMARY] [--strict-memory-limit]
               [--strict-cpu-limit] [--timestamps] [--js-console]
```

(continues on next page)

(continued from previous page)

```

[--disable-js-validation]
[--js-hint-options-file JS_HINT_OPTIONS_FILE]
[--user-space-docker-cmd CMD | --udocker | --singularity | --podman | --
↪no-container]
[--tool-help] [--relative-deps {primary,cwd}] [--enable-dev]
[--enable-ext] [--enable-color | --disable-color]
[--default-container DEFAULT_CONTAINER] [--no-match-user]
[--custom-net CUSTOM_NET]
[--enable-ga4gh-tool-registry | --disable-ga4gh-tool-registry]
[--add-ga4gh-tool-registry GA4GH_TOOL_REGISTRIES]
[--on-error {stop,continue}]
[--compute-checksum | --no-compute-checksum]
[--relax-path-checks] [--force-docker-pull] [--no-read-only]
[--overrides OVERRIDES]
[--target TARGET | --single-step SINGLE_STEP | --single-process SINGLE_
↪PROCESS]
[--mpi-config-file MPI_CONFIG_FILE]
[cwl_document] ...

```

cwl_document

path or URL to a CWL Workflow, CommandLineTool, or ExpressionTool. If the *inputs_object* has a *cwl:tool* field indicating the path or URL to the *cwl_document*, then the *cwl_document* argument is optional.

inputs_object

path or URL to a YAML or JSON formatted description of the required input values for the given *cwl_document*.

-h, --help

show this help message and exit

--basedir <basedir>**--outdir <outdir>**

Output directory. The default is the current directory.

--log-dir <log_dir>

Log your tools stdout/stderr to this location outside of container This will only log stdout/stderr if you specify stdout/stderr in their respective fields or capture it as an output

--parallel

Run jobs in parallel.

--preserve-environment <envvar>

Preserve specific environment variable when running CommandLineTools. May be provided multiple times. By default PATH is preserved when not running in a container.

--preserve-entire-environment

Preserve all environment variables when running CommandLineTools without a software container.

--rm-container

Delete Docker container used by jobs after they exit (default)

--leave-container

Do not delete Docker container used by jobs after they exit

--cidfile-dir <cidfile_dir>

Store the Docker container ID into a file in the specified directory.

--cidfile-prefix <cidfile_prefix>

Specify a prefix to the container ID filename. Final file name will be followed by a timestamp. The default is no prefix.

--tmpdir-prefix <tmpdir_prefix>

Path prefix for temporary directories. If `--tmpdir-prefix` is not provided, then the prefix for temporary directories is influenced by the value of the `TMPDIR`, `TEMP`, or `TMP` environment variables. Taking those into consideration, the current default is `/tmp/`.

--tmp-outdir-prefix <tmp_outdir_prefix>

Path prefix for intermediate output directories. Defaults to the value of `--tmpdir-prefix`.

--cachedir <cachedir>

Directory to cache intermediate workflow outputs to avoid recomputing steps. Can be very helpful in the development and troubleshooting of CWL documents.

--rm-tmpdir

Delete intermediate temporary directories (default)

--leave-tmpdir

Do not delete intermediate temporary directories

--move-outputs

Move output files to the workflow output directory and delete intermediate output directories (default).

--leave-outputs

Leave output files in intermediate output directories.

--copy-outputs

Copy output files to the workflow output directory and don't delete intermediate output directories.

--enable-pull

Try to pull Docker images

--disable-pull

Do not try to pull Docker images

--rdf-serializer <rdf_serializer>

Output RDF serialization format used by `--print-rdf` (one of turtle (default), n3, nt, xml)

--eval-timeout <eval_timeout>

Time to wait for a Javascript expression to evaluate before giving an error, default 60s.

--provenance <provenance>

Save provenance to specified folder as a Research Object that captures and aggregates workflow execution and data products.

--enable-user-provenance

Record user account info as part of provenance.

--disable-user-provenance

Do not record user account info in provenance.

--enable-host-provenance

Record host info as part of provenance.

--disable-host-provenance

Do not record host info in provenance.

--orcid <orcid>

Record user ORCID identifier as part of provenance, e.g. <https://orcid.org/0000-0002-1825-0097> or 0000-0002-1825-0097. Alternatively the environment variable ORCID may be set.

--full-name <cwl_full_name>

Record full name of user as part of provenance, e.g. Josiah Carberry. You may need to use shell quotes to preserve spaces. Alternatively the environment variable CWL_FULL_NAME may be set.

--print-rdf

Print corresponding RDF graph for workflow and exit

--print-dot

Print workflow visualization in graphviz format and exit

--print-pre

Print CWL document after preprocessing.

--print-deps

Print CWL document dependencies.

--print-input-deps

Print input object document dependencies.

--pack

Combine components into single document and print.

--version

Print version and exit

--validate

Validate CWL document only.

--print-supported-versions

Print supported CWL specs.

--print-subgraph

Print workflow subgraph that will execute. Can combined with `-target` or `-single-step`

--print-targets

Print targets (output parameters)

--make-template

Generate a template input object

--strict

Strict validation (unrecognized or out of place fields are error)

--non-strict

Lenient validation (ignore unrecognized fields)

--skip-schemas

Skip loading of schemas

--no-doc-cache

Disable disk cache for documents loaded over HTTP

--doc-cache

Enable disk cache for documents loaded over HTTP

--verbose

Default logging

--no-warnings

Only print errors.

--quiet

Only print warnings and errors.

--debug

Print even more logging

--write-summary <write_summary>, **-w** <write_summary>

Path to write the final output JSON object to. Default is stdout.

--strict-memory-limit

When running with software containers and the Docker engine, pass either the calculated memory allocation from ResourceRequirements or the default of 1 gigabyte to Docker's `--memory` option.

--strict-cpu-limit

When running with software containers and the Docker engine, pass either the calculated cpu allocation from ResourceRequirements or the default of 1 core to Docker's `--cpu` option. Requires docker version `>= v1.13`.

--timestamps

Add timestamps to the errors, warnings, and notifications.

--js-console

Enable javascript console output

--disable-js-validation

Disable javascript validation.

--js-hint-options-file <js_hint_options_file>

File of options to pass to jshint. This includes the added option "includewarnings".

--user-space-docker-cmd <cmd>

(Linux/OS X only) Specify the path to udocker. Implies `--udocker`

--udocker

(Linux/OS X only) Use the udocker runtime for running containers (equivalent to `--user-space-docker-cmd=udocker`).

--singularity

Use Singularity or Apptainer runtime for running containers. Requires Singularity v2.6.1+ and Linux with kernel version v3.18+ or with overlayfs support backported.

--podman

Use Podman runtime for running containers.

--no-container

Do not execute jobs in a Docker container, even when *DockerRequirement* is specified under *hints*.

--tool-help

Print command line help for tool

--relative-deps {primary,cwd}

When using `--print-deps`, print paths relative to primary file or current working directory.

--enable-dev

Enable loading and running unofficial development versions of the CWL standards.

--enable-ext

Enable loading and running 'cwltool:' extensions to the CWL standards.

--enable-color

Enable logging color (default enabled)

--disable-color

Disable colored logging (default false)

--default-container <default_container>

Specify a default software container to use for any CommandLineTool without a DockerRequirement.

--no-match-user

Disable passing the current uid to *docker run --user*

--custom-net <custom_net>

Passed to *docker run* as the '`--net`' parameter when NetworkAccess is true, which is its default setting.

--enable-ga4gh-tool-registry

Enable tool resolution using GA4GH tool registry API

--disable-ga4gh-tool-registry

Disable tool resolution using GA4GH tool registry API

--add-ga4gh-tool-registry <ga4gh_tool_registries>

Add a GA4GH tool registry endpoint to use for resolution, default [`'https://dockstore.org/api'`]

--on-error {stop,continue}

Desired workflow behavior when a step fails. One of 'stop' (do not submit any more steps) or 'continue' (may submit other steps that are not downstream from the error). Default is 'stop'.

--compute-checksum

Compute checksum of contents while collecting outputs

--no-compute-checksum

Do not compute checksum of contents while collecting outputs

--relax-path-checks

Relax requirements on path names to permit spaces and hash characters.

--force-docker-pull

Pull latest software container image even if it is locally present

--no-read-only

Do not set root directory in the container as read-only

--overrides <overrides>

Read process requirement overrides from file.

--target <target>, **-t** <target>

Only execute steps that contribute to listed targets (can be provided more than once).

--single-step <single_step>

Only executes a single step in a workflow. The input object must match that step's inputs. Can be combined with `--print-subgraph`.

--single-process <single_process>

Only executes the underlying Process (CommandLineTool, ExpressionTool, or sub-Workflow) for the given step in a workflow. This will not include any step-level processing: 'scatter', 'when'; and there will be no processing of step-level 'default', or 'valueFrom' input modifiers. However, requirements/hints from the step or parent workflow(s) will be inherited as usual. The input object must match that Process's inputs.

--mpi-config-file <mpi_config_file>

Platform specific configuration for MPI (parallel launcher, its flag etc). See README section 'Running MPI-based tools' for details of the format.

3.4.2 Loops

The `cwltool:Loop` requirement enables workflow-level looping of a step. It is valid only under requirements of a `WorkflowStep`. Unlike other CWL requirements, `Loop` requirement is not propagated to inner steps.

The `cwltool:Loop` is not compatible with `scatter` and `when`. Combining a `cwltool:Loop` requirement with a `scatter` or a `when` clause in the same step will produce an error.

The loop condition

The `loopWhen` field controls loop termination. It is an expansion of the CWL v1.2 `when` construct, which controls conditional execution. This is an expression that must be evaluated with inputs bound to the step input object and outputs produced in the last step execution, and returns a boolean value. It is an error if this expression returns a value other than true or false. For example:

```
example:
  run:
    class: ExpressionTool
    inputs:
      i1: int
    outputs:
      o1: int
    expression: >
      ${return {'o1': inputs.i1 + 1};}
  in:
    i1: i1
  out: [o1]
  requirements:
    cwltool:Loop:
      loopWhen: $(inputs.i1 < 10)
      loop:
        i1: o1
      outputMethod: last
```

This loop executes until the counter `i1` reaches the value of 10, and then terminates. Note that if the `loopWhen` condition evaluates to false prior to the first iteration, the loop is skipped. The value assumed by the output fields depends on the specified `outputMethod`, as described below.

The loop field

The `loop` field defines the input parameters of the loop iterations after the first one (inputs of the first iteration are the step input parameters). If no loop rule is specified for a given step `in` field, the initial value is kept constant among all iterations.

The `LoopInput` is basically a reduced version of the `WorkflowStepInput` structure with the possibility to include outputs of the previous step execution in the `valueFrom` expression.

Field	Re-require	Type	Description
<code>id</code>	optional	string	It must reference the <code>id</code> of one of the elements in the <code>in</code> field of the step.
<code>loop</code>	optional	string string	Specifies one or more of the step output parameters that will provide input to the loop iterations after the first one (inputs of the first iteration are the step input parameters).
<code>link</code>	optional	Link	The method to use to merge multiple inbound links into a single array. If not specified, the default method is <code>merge_nested</code> .
<code>pick</code>	optional	Pick-Value	The method to use to choose non-null elements among multiple sources.
<code>valueFrom</code>	optional	string Expression	To use <code>valueFrom</code> , <code>StepInputExpressionRequirement</code> must be specified in the workflow or workflow step requirements. If <code>valueFrom</code> is a constant string value, use this as the value for this input parameter. If <code>valueFrom</code> is a parameter reference or expression, it must be evaluated to yield the actual value to be assigned to the input field. The <code>self</code> value in the parameter reference or expression must be null if there is no <code>loopSource</code> field, or the value of the parameter(s) specified in the <code>loopSource</code> field. The value of <code>inputs</code> in the parameter reference or expression must be the input object to the previous iteration of the workflow step (or the initial inputs for the first iteration).

Loop output modes

The `outputMethod` field specifies the desired method of dealing with loop outputs. It behaves similarly to the `scatterMethod` field. For the sake of simplicity, there can be a single `outputMethod` field for each step instead of specifying a different behaviour for each output element. The `outputMethod` field can take two possible values: `last` or `all`.

The `last` output mode propagates only the last computed element to the subsequent steps when the loop terminates. When a loop with an `outputMethod` equal to `last` is skipped, each output assumes a `null` value.

This is the most recurrent behaviour and it is typical of the optimization processes, when a step must iterate until a desired precision is reached. For example:

```
optimization:
  in:
    a: a
    prev_a:
      default: ${ return inputs.a - (2 * inputs.threshold) }
      threshold: threshold
  run: optimize.cwl
  out: [a]
  requirements:
    cwltool:Loop:
```

(continues on next page)

(continued from previous page)

```

loopWhen: ${ return (inputs.a - inputs.prev_a) > inputs.threshold)
loop:
  a: a
  prev_a:
    valueFrom: $(inputs.a)
outputMethod: last

```

This loop keeps optimizing the initial a value until the error value falls below a given (constant) `threshold`. Then, the last values of a will be propagated.

The `all` output mode propagates a single array with all output values to the subsequent steps when the loop terminates. When a loop with an `outputMethod` equal to `all` is skipped, each output assumes a `[]` value.

This behaviour is needed when a recurrent simulation produces loop-carried results, but the subsequent steps need to know the total amount of computed values to proceed. For example:

```

simulation:
  in:
    a: a
    day:
      default: 0
    max_day: max_day
  run: simulate.cwl
  out: [a]
  requirements:
    cwltool:Loop:
      loopWhen: ${ return inputs.day < inputs.max_day }
      loop:
        a: a
        day:
          valueFrom: $(inputs.day + 1)
      outputMethod: all

```

In this case, subsequent steps can start processing outputs even before the `simulation` step terminates. When a loop with an `outputMethod` equal to `last` is skipped, each output assumes a `null` value.

Loop-independent iterations

If a `cwltool:Loop` comes with loop-independent iterations, i.e. if each iteration does not depend on the result produced by the previous ones, all iterations can be processed concurrently. For example:

```

example:
  run: inner.cwl
  in:
    i1: i1
  out: [o1]
  requirements:
    cwltool:Loop:
      loopWhen: $(inputs.i1 < 10)
      loop:
        i1:
          valueFrom: $(inputs.i1 + 1)
      outputMethod: all

```

Since each iteration of this loop only depends on the input field `i1`, all its iterations can be processed in parallel if there is enough computing power.

3.4.3 Provenance capture

It is possible to capture the full provenance of a workflow execution to a folder, including intermediate values:

```
cwltool --provenance revsort-run-1/ tests/wf/revsort.cwl tests/wf/revsort-job.json
```

Who executed the workflow?

Optional parameters are available to capture information about *who* executed the workflow *where*:

```
cwltool --orcid https://orcid.org/0000-0002-1825-0097 \
--full-name "Alice W Land" \
--enable-user-provenance --enable-host-provenance \
--provenance revsort-run-1/ \
tests/wf/revsort.cwl tests/wf/revsort-job.json
```

These parameters are opt-in as they track person-identifiable information. The options `--enable-user-provenance` and `--enable-host-provenance` will pick up account/machine info from where `cwltool` is executed (e.g. UNIX username). This may get the full name of the user wrong, in which case `--full-name` can be supplied.

For consistent tracking it is recommended to apply for an [ORCID](#) identifier and provide it as above, since `--enable-user-provenance` and `--enable-host-provenance` are only able to identify the local machine account.

It is possible to set the shell environment variables `ORCID` and `CWL_FULL_NAME` to avoid supplying `--orcid` or `--full-name` for every workflow run, for instance by augmenting the `~/ .bashrc` or equivalent:

```
export ORCID=https://orcid.org/0000-0002-1825-0097
export CWL_FULL_NAME="Stian Soiland-Reyes"
```

Care should be taken to preserve spaces when setting `--full-name` or `CWL_FULL_NAME`.

CWLProv folder structure

The CWLProv folder structure under `revsort-run-1` is a [Research Object](#) that conforms to the [RO BagIt profile](#) and contains [PROV](#) traces detailing the execution of the workflow and its steps.

A rough overview of the CWLProv folder structure:

- `bagit.txt` - bag marker for [BagIt](#).
- `bag-info.txt` - minimal bag metadata. The `External-Identifier` key shows which `arcp` can be used as base URI within the folder bag.
- `manifest-*.txt` - checksums of files under `data/` (algorithms subject to change)
- `tagmanifest-*.txt` - checksums of the remaining files (algorithms subject to change)
- `metadata/manifest.json` - [Research Object manifest](#) as JSON-LD. Types and relates files within bag.
- `metadata/provenance/primary.cwlprov*` - [PROV](#) trace of main workflow execution in alternative [PROV](#) and RDF formats
- `data/` - bag payload, workflow/step input/output data files (content-addressable)

- `data/32/327fc7aedf4f6b69a42a7c8b808dc5a7aff61376` - a data item with checksum `327fc7aedf4f6b69a42a7c8b808dc5a7aff61376` (checksum algorithm is subject to change)
- `workflow/packed.cwl` - The `cwltool --pack` standalone version of the executed workflow
- `workflow/primary-job.json` - Job input for use with `packed.cwl` (references `data/*`)
- `snapshot/` - Direct copies of original files used for execution, but may have broken relative/absolute paths

See the [CWLProv paper](#) for more details.

Research Object manifest

The file `metadata/manifest.json` follows the structure defined for [Research Object Bundles](#) - but note that `.ro/` is instead called `metadata/` as this conforms to the [RO BagIt profile](#).

Some of the keys of the CWLProv manifest are explained below:

```
"@context": [
  {
    "@base": "arcp://uuid,67f38794-d24a-435f-bd4a-0242a56a581b/metadata/"
  },
  "https://w3id.org/bundle/context"
]
```

This [JSON-LD context](#) enables consumers to alternatively consume the JSON file as Linked Data with absolute identifiers. The key for that is the `@base` which means URIs within this JSON file are relative to the `metadata/` folder within this Research Object bag, and the external JSON-LD.

Output from `cwltool` should follow the JSON structure shown beyond; however interested consumer may alternatively parse it as JSON-LD with a RDF triple store like [Apache Jena](#) for further querying.

The manifest lists which software version created the Research Object - we will hear more from this UUID later:

```
"createdBy": {
  "uri": "urn:uuid:7c9d9e88-666b-4977-85f4-c02da08a942d",
  "name": "cwltool 1.0.20180416145054"
}
```

Secondly the manifest lists the person who “authored the run” - that is put the workflow and inputs together with `cwltool`:

```
"authoredBy": {
  "orcid": "https://orcid.org/0000-0002-1825-0097",
  "name": "Stian Soiland-Reyes"
}
```

Note that the author of the workflow run may differ from the author of the workflow definition.

The list of aggregates are the main resources that this Research Object transports:

```
"aggregates": [
  {
    "uri": "urn:hash::sha1:53870991af88a6d678cbeed3255bb65993c52925",
    ...
  },
  { "provenance/primary.cwlprov.xml",
    ...
  }
]
```

(continues on next page)

(continued from previous page)

```

    },
    {
      "uri": "../workflow/packed.cwl",
      "createdBy": {
        "uri": "urn:uuid:7c9d9e88-666b-4977-85f4-c02da08a942d",
        "name": "cwltool 1.0.20180416145054"
      },
      "conformsTo": "https://w3id.org/cwl/",
      "mediatype": "text/x+yaml; charset=UTF-8",
      "createdOn": "2018-04-16T18:27:09.513824"
    },
    {
      "uri": "../snapshot/hello-workflow.cwl",
      "conformsTo": "https://w3id.org/cwl/",
      "mediatype": "text/x+yaml; charset=UTF-8",
      "createdOn": "2018-04-04T13:29:55.717707"
    }
  }

```

Beyond being a listing of file names and identifiers, this also lists formats and light-weight provenance. We note that the CWL file is marked to conform to the <https://w3id.org/cwl/> CWL specification.

Some of the files like `packed.cwl` have been created by `cwltool` as part of the run, while others have been created before the run “outside”. Note that `cwltool` is currently unable to extract the original authors and contributors of the original files, this is planned for future versions.

Under `annotations` we see that the main point of this whole research object (/ aka `arcp://uuid,67f38794-d24a-435f-bd4a-0242a56a581b/`) is to describe something called `urn:uuid:67f38794-d24a-435f-bd4a-0242a56a581b`:

```

"annotations": [
  {
    "about": "urn:uuid:67f38794-d24a-435f-bd4a-0242a56a581b",
    "content": "/",
    "oa:motivatedBy": {
      "@id": "oa:describing"
    }
  }
],

```

We will later see that this is the UUID for the workflow run. A workflow run is an *activity*, something that happens - it can't be directly saved to a file. However it can be *described* in different ways, in this case as CWLProv provenance:

```

{
  "about": "urn:uuid:67f38794-d24a-435f-bd4a-0242a56a581b",
  "content": [
    "provenance/primary.cwlprov.xml",
    "provenance/primary.cwlprov.nt",
    "provenance/primary.cwlprov.ttl",
    "provenance/primary.cwlprov.provn",
    "provenance/primary.cwlprov.jsonld",
    "provenance/primary.cwlprov.json"
  ],
  "oa:motivatedBy": {
    "@id": "http://www.w3.org/ns/prov#has_provenance"
  }
}

```

Finally the research object wants to highlight the workflow file:

```
{
  "about": "workflow/packed.cwl",
  "oa:motivatedBy": {
    "@id": "oa:highlighting"
  }
},
```

And links the run ID 67f38794.. to the `primary-job.json` and packed.cwl:

```
{
  "about": "urn:uuid:67f38794-d24a-435f-bd4a-0242a56a581b",
  "content": [
    "workflow/packed.cwl",
    "workflow/primary-job.json"
  ],
  "oa:motivatedBy": {
    "@id": "oa:linking"
  }
}
```

Note: `oa:motivatedBy` in CWLProv are subject to change.

PROV profile

The underlying model and information of the **PROV** files under `metadata/provenance` is the same, but is made available in multiple serialization formats:

- `primary.cwlprov.provn` – PROV-N Textual Provenance Notation
- `primary.cwlprov.xml` – PROV-XML
- `primary.cwlprov.json` – PROV-JSON
- `primary.cwlprov.jsonld` – PROV-O as JSON-LD (@context subject to change)
- `primary.cwlprov.ttl` – PROV-O as RDF Turtle
- `primary.cwlprov.nt` – PROV-O as RDF N-Triples

The below extracts use the PROV-N syntax for brevity.

CWLPROV namespaces

Note that the identifiers must be expanded with the defined prefix-es when comparing across serializations. These set which vocabularies (“namespaces”) are used by the CWLProv statements:

```
prefix data <urn:hash::sha1:>
prefix input <arcp://uuid,0e6cb79e-fe70-4807-888c-3a61b9bf232a/workflow/primary-job.json
  ↪#>
prefix cwlprov <https://w3id.org/cwl/prov#>
prefix wfprov <http://purl.org/wf4ever/wfprov#>
prefix sha256 <nih:sha-256;>
prefix schema <http://schema.org/>
prefix wfdesc <http://purl.org/wf4ever/wfdesc#>
```

(continues on next page)

(continued from previous page)

```

prefix orcid <https://orcid.org/>
prefix researchobject <arcp://uuid,0e6cb79e-fe70-4807-888c-3a61b9bf232a/>
prefix id <urn:uuid:>
prefix wf <arcp://uuid,0e6cb79e-fe70-4807-888c-3a61b9bf232a/workflow/packed.cwl#>
prefix foaf <http://xmlns.com/foaf/0.1/>

```

Note that the `arcp` base URI will correspond to the UUID of each main workflow run.

Account who launched cwltool

If `--enable-user-provenance` was used, the local machine account (e.g. Windows or UNIX user name) who started `cwltool` is tracked:

```

agent(id:855c6823-bbe7-48a5-be37-b0f07f20c495, [foaf:accountName="stain", prov:type=
  ↳ 'foaf:OnlineAccount', prov:label="stain"])

```

It is assumed that the account was under the control of the named person (in PROV terms “actedOnBehalfOf”):

```

agent(id:433df002-2584-462a-80b0-cf90b97e6e07, [prov:label="Stian Soiland-Reyes",
  prov:type='prov:Person', foaf:account='id:8815e39c-9711-4105-bf52-dbc016c8028f'])
actedOnBehalfOf(id:8815e39c-9711-4105-bf52-dbc016c8028f, id:433df002-2584-462a-80b0-
  ↳ cf90b97e6e07, -)

```

However we do not have an identifier for neither the account or the person, so every `cwltool` run will yield new UUIDs.

With `--enable-user-provenance` it is possible to associate the account with a hostname:

```

agent(id:855c6823-bbe7-48a5-be37-b0f07f20c495, [cwlprov:hostname="biggie", prov:type=
  ↳ 'foaf:OnlineAccount', prov:location="biggie"])

```

Note that the hostname is often non-global or variable (e.g. on cloud instances or virtual machines), and thus may be unreliable when considering `cwltool` executions on multiple hosts.

If the `--orcid` parameter or ORCID shell variable is included, then the person associated with the local machine account is uniquely identified, no matter where the workflow was executed:

```

agent(orcid:0000-0002-1825-0097, [prov:type='prov:Person', prov:label="Stian Soiland-
  ↳ Reyes",
  foaf:account='id:855c6823-bbe7-48a5-be37-b0f07f20c495'])
actedOnBehalfOf(id:855c6823-bbe7-48a5-be37-b0f07f20c495, orcid:0000-0002-1825-0097, -)

```

The running of `cwltool` itself makes it the workflow engine. It is the machine account who launched the `cwltool` (not necessarily the person behind it):

```

agent(id:7c9d9e88-666b-4977-85f4-c02da08a942d, [prov:type='prov:SoftwareAgent',
  ↳ prov:type='wfprov:WorkflowEngine', prov:label="cwltool 1.0.20180416145054"])
wasStartedBy(id:855c6823-bbe7-48a5-be37-b0f07f20c495, -, id:9c3d4d1f-473d-468f-a6f2-
  ↳ 1ef4de571a7f, 2018-04-16T18:27:09.428090)

```

Starting a workflow

The main job of the cwltool execution is to run a workflow, here the activity for workflow/packed.cwl#main:

```
activity(id:67f38794-d24a-435f-bd4a-0242a56a581b, 2018-04-16T18:27:09.428165, -,
  ↳[prov:type='wfprov:WorkflowRun', prov:label="Run of workflow/packed.cwl#main"])
wasStartedBy(id:67f38794-d24a-435f-bd4a-0242a56a581b, -, id:7c9d9e88-666b-4977-85f4-
  ↳c02da08a942d, 2018-04-16T18:27:09.428285)
```

Now what is that workflow again? Well a tiny bit of prospective provenance is included:

```
entity(wf:main, [prov:type='prov:Plan', prov:type='wfdesc:Workflow', prov:label=
  ↳"Prospective provenance"])
entity(wf:main, [prov:label="Prospective provenance", wfdesc:hasSubProcess='wf:main/step0
  ↳'])
entity(wf:main/step0, [prov:type='wfdesc:Process', prov:type='prov:Plan'])
```

But we can also expand the wf identifiers to find that we are talking about arcp://uuid, 0e6cb79e-fe70-4807-888c-3a61b9bf232a/workflow/packed.cwl# - that is the main workflow in the file workflow/packed.cwl of the Research Object.

Running workflow steps

A workflow will contain some steps, each execution of these are again nested activities:

```
activity(id:6c7c04ea-dcc8-40d2-92a4-7705f7286756, -, -, [prov:type='wfprov:ProcessRun',
  ↳prov:label="Run of workflow/packed.cwl#main"])
wasStartedBy(id:6c7c04ea-dcc8-40d2-92a4-7705f7286756, -, id:67f38794-d24a-435f-bd4a-
  ↳0242a56a581b, 2018-04-16T18:27:09.430883)
activity(id:a583b025-9a16-49ce-8515-f3249eb2aacf, -, -, [prov:type='wfprov:ProcessRun',
  ↳prov:label="Run of workflow/packed.cwl#main/step0"])
wasAssociatedWith(id:a583b025-9a16-49ce-8515-f3249eb2aacf, -, wf:main/step0)
```

Again we see the link back to the workflow plan, the workflow execution of #main/step0 in this case. Note that depending on scattering etc there might be multiple activities for a single step in the workflow definition.

Data inputs (usage)

This activities uses some data at the input message:

```
activity(id:a583b025-9a16-49ce-8515-f3249eb2aacf, -, -, [prov:type='wfprov:ProcessRun',
  ↳prov:label="Run of workflow/packed.cwl#main/step0"])
used(id:a583b025-9a16-49ce-8515-f3249eb2aacf,
  ↳data:53870991af88a6d678cbeed3255bb65993c52925, 2018-04-16T18:27:09.433743, [prov:role=
  ↳'wf:main/step0/message'])
```

Data files within a workflow execution are identified using urn:hash::sha1: URIs derived from their sha1 checksum (checksum algorithm and prefix subject to change):

```
entity(data:53870991af88a6d678cbeed3255bb65993c52925, [prov:type='wfprov:Artifact',
  ↳prov:value="Hei7"])
```

Small values (typically those provided on the command line) may be present as *prov:value*. The corresponding data/file within the Research Object has a content-addressable filename based on the checksum; but it is also possible to look up this independent from the corresponding metadata/manifest.json aggregation:

```
"aggregates": [
  {
    "uri": "urn:hash::sha1:53870991af88a6d678cbeed3255bb65993c52925",
    "bundledAs": {
      "uri": "arcp://uuid,0e6cb79e-fe70-4807-888c-3a61b9bf232a/data/53/
↪53870991af88a6d678cbeed3255bb65993c52925",
      "folder": "/data/53/",
      "filename": "53870991af88a6d678cbeed3255bb65993c52925"
    }
  },
]
```

Data outputs (generation)

Similarly a step typically generates some data, here response:

```
activity(id:a583b025-9a16-49ce-8515-f3249eb2aacf, -, -, [prov:type='wfprov:ProcessRun', ↵
↪prov:label="Run of workflow/packed.cwl#main/step0"])
wasGeneratedBy(data:53870991af88a6d678cbeed3255bb65993c52925, id:a583b025-9a16-49ce-8515-
↪f3249eb2aacf, 2018-04-16T18:27:09.438236, [prov:role='wf:main/step0/response'])
```

In the hello world example this is interesting because it is the same data output as-is, but typically the outputs will each have different checksums (and thus different identifiers).

The step is ended:

```
wasEndedBy(id:a583b025-9a16-49ce-8515-f3249eb2aacf, -, id:67f38794-d24a-435f-bd4a-
↪0242a56a581b, 2018-04-16T18:27:09.438482)
```

In this case the step output is also a workflow output response, so the data is also generated by the workflow activity:

```
activity(id:67f38794-d24a-435f-bd4a-0242a56a581b, 2018-04-16T18:27:09.428165, -, ↵
↪[prov:type='wfprov:WorkflowRun', prov:label="Run of workflow/packed.cwl#main"])
wasGeneratedBy(data:53870991af88a6d678cbeed3255bb65993c52925, id:67f38794-d24a-435f-bd4a-
↪0242a56a581b, 2018-04-16T18:27:09.439323, [prov:role='wf:main/response'])
```

Ending the workflow

Finally the overall workflow #main also ends:

```
activity(id:67f38794-d24a-435f-bd4a-0242a56a581b, 2018-04-16T18:27:09.428165, -, ↵
↪[prov:type='wfprov:WorkflowRun', prov:label="Run of workflow/packed.cwl#main"])
agent(id:7c9d9e88-666b-4977-85f4-c02da08a942d, [prov:type='prov:SoftwareAgent', ↵
↪prov:type='wfprov:WorkflowEngine', prov:label="cwltool 1.0.20180416145054"])
wasEndedBy(id:67f38794-d24a-435f-bd4a-0242a56a581b, -, id:7c9d9e88-666b-4977-85f4-
↪c02da08a942d, 2018-04-16T18:27:09.445785)
```

Note that the end of the outer cwltool activity is not recorded, as cwltool is still running at the point of writing out this provenance.

Currently the provenance trace do not distinguish executions within nested workflows; it is planned that these will be tracked in separate files under `metadata/provenance/`.

3.4.4 Python version support policy

cwltool will always support Python 3 versions that are officially supported by the Python Software Foundation.

For versions that are no longer supported by the Python Software Foundation (or “upstream” for short), *cwltool* support also extends to the latest Python versions included in the following major Linux distributions:

- Debian (stable)
- Ubuntu (LTS release standard support)

This means that users may need to install a newer version of Python from their Linux distributor if the default version is too old.

If there is a conflict between a third party package dependency which has dropped support for a Python version that *cwltool* should support according to this policy, then possible options (such as pinning the dependency, eliminating the dependency, or changing Python version support of *cwltool*) should be discussed among the *cwltool* maintainers and downstream users before making the decision to drop support for a Python version before the date outlined in this policy. The reasoning for dropping support for a Python version should be outlined here.

As of 2023-08-14, here are approximate *cwltool* support periods for Python versions (*EOL* == “End of Life”, the end of the support period by that provider):

Python	cwltool end of support
2.7	ended 2020-01 (upstream EOL)
3.5	ended 2020-10 (upstream EOL)
3.6	ended 2023-08-31 (change in cwltool policy)
3.7	ended 2023-07-27 (upstream EOL)
3.8	2024-10-14 (upstream EOL)
3.9	2025-10-01 (upstream EOL)
3.10	2027-04-01 (Ubuntu 22.04 LTS EOL)
3.11	2027-10-01 (upstream EOL)
3.12	2028-10-01 (planned upstream EOL)
3.13	2029-10-01 (planned upstream EOL)

Python version of supported Linux distributions, for reference (as of August 2023)

Python Version	Linux distros where it is a supported version
3.6	Ubuntu 18.04 LTS
3.7	Debian 10
3.8	Ubuntu 20.04 LTS
3.9	Debian 11, Ubuntu 20.04 LTS
3.10	Ubuntu 22.04 LTS
3.11	Debian 12
3.12	Debian 13 (planned)

3.4.5 Process generator

Experimental feature and unofficial extension to the CWL standards.

A process generator is a CWL Process type that executes a concrete CWL process (CommandLineTool, Workflow or ExpressionTool) which produces CWL files as output, then executes the CWL that was generated.

The intention is to have a formalized way to express a pre-processing or bootstrapping step in which a CWL description is generated by another program (such as from a template, or conversion from another workflow language).

The ProcessGenerator is a subtype of CWL process, so it must define its inputs and outputs. The “run” field is similar to the “run” field of a workflow step – it specifies a tool to run that will create new CWL as output.

```
- name: ProcessGenerator
  type: record
  inVocab: true
  extends: cwl:Process
  documentRoot: true
  fields:
    - name: class
      jsonldPredicate:
        _id: "@type"
        _type: "@vocab"
      type: string
    - name: run
      type: [string, cwl:Process]
      jsonldPredicate:
        _id: "cwl:run"
        _type: "@id"
      subscope: run
      doc: |
        Specifies the process to run.
```

Process generator example (pytoolgen.cwl)

```
#!/usr/bin/env cwl-runner
cwlVersion: v1.0
$namespaces:
  cwltool: "http://commonwl.org/cwltool#"
class: cwltool:ProcessGenerator
inputs:
  script: string
  dir: Directory
outputs: {}
run:
  class: CommandLineTool
  inputs:
    script: string
    dir: Directory
  outputs:
    runProcess:
      type: File
      outputBinding:
        glob: main.cwl
  requirements:
```

(continues on next page)

(continued from previous page)

```

InlineJavascriptRequirement: {}
cwltool:LoadListingRequirement:
  loadListing: shallow_listing
InitialWorkDirRequirement:
  listing: |
    ${
      var v = inputs.dir.listing;
      v.push({entryname: "inp.py", entry: inputs.script});
      return v;
    }
arguments: [python, inp.py]
stdout: main.cwl

```

The process generator has two required inputs: “script” and “dir”. It runs the command line tool listed inline in “run” with the input object, which is required to have those parameters. Note: the input object may contain additional parameters which are intended for the generated CWL when it is executed.

The command line tool populates the working directory using InitialWorkDirRequirement. It uses the listing from ‘dir’ and adds a new file literal called “inp.py” which contains the text from the input parameter “script”. Then it runs “python inp.py”.

The output of this command line tool is the File parameter “runProcess”. In this example, the “inp.py” script, when run, is expected to print the CWL description to standard output, which will be captured in the “runProcess” output parameter.

Next, the ProcessGenerator will load file in the “runProcess” parameter, which in this example is “main.cwl”. Finally, it will execute the process with input object that was originally provided to the process generator.

The output of the generated script is used as the output for ProcessGenerator as a whole.

Here’s an example (zing.cwl) that uses pytoolgen.cwl.

```

#!/usr/bin/env cwltool
{cwl:tool: pytoolgen.cwl, script: {${include: "#attachment-1"}, dir: {class: Directory,
location: .}}
--- |
import os
import sys
print("""
cwlVersion: v1.0
class: CommandLineTool
inputs:
  zing: string
outputs: {}
arguments: [echo, ${inputs.zing}]
""")

```

The first line `#!/usr/bin/env cwltool` means that this file can be given the executable bit (+x) and then run directly.

This is a multi-part YAML file. The first section is a CWL input object.

The input object uses “cwl:tool” to indicate that this input object should be used as input to execute “pytoolgen.cwl”.

The parameter `script: {${include: "#attachment-1"}` takes the text from the second part of the file (following the YAML division marker `--- |`) and assigns it as a string value to “script”.

The “dir” parameter is not doing much in this example, but by capturing the whole directory it allows the Python script to refer to files in the current directory.

In this example the script is trivially printing CWL as a string, but of course could do something much more complex: generate code from a template, select among several possible workflows based on the input, convert from another workflow language, etc.

When this is executed, the following steps happen:

1. pytoolgen.py is loaded and executed with the 1st part of the file as the input object
2. The “script” parameter contains the contents of the second part. The inline command line tool creates a file called “inp.py” with the contents of “script”
3. The inline command line tool runs python on “inp.py” and collects the output, which is CWL description for a trivial “echo” tool.
4. It loads the CWL description and executes it with any additional parameters declared in the input object or command line.

Example runs

Note: requires cwltool flags --enable-ext and --enable-dev

You can set these with the environment parameter CWLTOOL_OPTIONS

```
$ export CWLTOOL_OPTIONS="--enable-dev --enable-ext"

$ ./zing.cwl
INFO /home/peter/work/cwltool/venv3/bin/cwltool 3.1.20211112163758
INFO Resolved './zing.cwl' to 'file:///home/peter/work/cwltool/tests/wf/generator/zing.
↪cwl'
INFO [job d3626216-d7d8-4322-bc21-4d469634cc9a] /tmp/8sez90gb$ python \
    inp.py > /tmp/8sez90gb/main.cwl
INFO [job d3626216-d7d8-4322-bc21-4d469634cc9a] completed success
usage: ./zing.cwl [-h] --zing ZING [job_order]
./zing.cwl: error: the following arguments are required: --zing
```

```
$ ./zing.cwl --zing blurf
INFO /home/peter/work/cwltool/venv3/bin/cwltool 3.1.20211112163758
INFO Resolved './zing.cwl' to 'file:///home/peter/work/cwltool/tests/wf/generator/zing.
↪cwl'
INFO [job a580b69d-2b88-4268-904e-ed105ba7c85e] /tmp/ujff239o$ python \
    inp.py > /tmp/ujff239o/main.cwl
INFO [job a580b69d-2b88-4268-904e-ed105ba7c85e] completed success
INFO [job main.cwl] /tmp/f_7bxncq$ echo \
    blurf
blurf
INFO [job main.cwl] completed success
{
  "runProcess": {
    "location": "file:///home/peter/work/cwltool/tests/wf/generator/main.cwl",
    "basename": "main.cwl",
    "class": "File",
    "checksum": "sha1$8c160b680fb2cededef3228a53425e595b8cdf48",
    "size": 111,
```

(continues on next page)

(continued from previous page)

```

    "path": "/home/peter/work/cwltool/tests/wf/generator/main.cwl"
  }
}
INFO Final process status is success

$ echo "zing: zoop" > job.yml
$ ./zing.cwl job.yml
INFO /home/peter/work/cwltool/venv3/bin/cwltool 3.1.20211112163758
INFO Resolved './zing.cwl' to 'file:///home/peter/work/cwltool/tests/wf/generator/zing.
↪cwl'
INFO [job 9073a083-dc79-4719-8762-1c024480605c] /tmp/meeo3d19$ python \
inp.py > /tmp/meeo3d19/main.cwl
INFO [job 9073a083-dc79-4719-8762-1c024480605c] completed success
INFO [job main.cwl] /tmp/2pqdz5nq$ echo \
zoop
zoop
INFO [job main.cwl] completed success
{
  "runProcess": {
    "location": "file:///home/peter/work/cwltool/tests/wf/generator/main.cwl",
    "basename": "main.cwl",
    "class": "File",
    "checksum": "sha1$8c160b680fb2cededef3228a53425e595b8cdf48",
    "size": 111,
    "path": "/home/peter/work/cwltool/tests/wf/generator/main.cwl"
  }
}
INFO Final process status is success

```

3.4.6 API Reference

This page contains auto-generated API reference documentation¹.

cwltool

Reference implementation of the CWL standards.

Subpackages

cwltool.cwlprov

Stores Research Object including provenance.

¹ Created with [sphinx-autoapi](#)

Submodules

`cwltool.cwlprov.provenance_constants`

Module Contents

```
cwltool.cwlprov.provenance_constants.__citation__ =  
'https://doi.org/10.5281/zenodo.1208477'  
  
cwltool.cwlprov.provenance_constants.CWLPROV_VERSION = 'https://w3id.org/cwl/prov/0.6.0'  
  
cwltool.cwlprov.provenance_constants.METADATA = 'metadata'  
  
cwltool.cwlprov.provenance_constants.DATA = 'data'  
  
cwltool.cwlprov.provenance_constants.WORKFLOW = 'workflow'  
  
cwltool.cwlprov.provenance_constants.SNAPSHOT = 'snapshot'  
  
cwltool.cwlprov.provenance_constants.MAIN  
  
cwltool.cwlprov.provenance_constants.PROVENANCE  
  
cwltool.cwlprov.provenance_constants.LOGS  
  
cwltool.cwlprov.provenance_constants.WFDESC  
  
cwltool.cwlprov.provenance_constants.WFPROV  
  
cwltool.cwlprov.provenance_constants.WF4EVER  
  
cwltool.cwlprov.provenance_constants.RO  
  
cwltool.cwlprov.provenance_constants.ORE  
  
cwltool.cwlprov.provenance_constants.FOAF  
  
cwltool.cwlprov.provenance_constants.SCHEMA  
  
cwltool.cwlprov.provenance_constants.CWLPROV  
  
cwltool.cwlprov.provenance_constants.ORCID  
  
cwltool.cwlprov.provenance_constants.UUID  
  
cwltool.cwlprov.provenance_constants.ENCODING = 'UTF-8'  
  
cwltool.cwlprov.provenance_constants.TEXT_PLAIN  
  
cwltool.cwlprov.provenance_constants.Hasher  
  
cwltool.cwlprov.provenance_constants.SHA1 = 'sha1'  
  
cwltool.cwlprov.provenance_constants.SHA256 = 'sha256'  
  
cwltool.cwlprov.provenance_constants.SHA512 = 'sha512'  
  
cwltool.cwlprov.provenance_constants.USER_UUID  
  
cwltool.cwlprov.provenance_constants.ACCOUNT_UUID
```

`cwltool.cwlprov.provenance_profile`

Module Contents

Classes

<i>ProvenanceProfile</i>	Provenance profile.
--------------------------	---------------------

Functions

<i>copy_job_order</i> (job, job_order_object)	Create copy of job object for provenance.
---	---

`cwltool.cwlprov.provenance_profile.copy_job_order(job, job_order_object)`

Create copy of job object for provenance.

Parameters

- `job` (`Union[cwltool.process.Process, cwltool.utils.JobsType]`)
- `job_order_object` (`cwltool.utils.CWLObjectType`)

Return type

`cwltool.utils.CWLObjectType`

```
class cwltool.cwlprov.provenance_profile.ProvenanceProfile(research_object, full_name,
                                                         host_provenance, user_provenance,
                                                         orcid, fsaccess, run_uuid=None)
```

Provenance profile.

Populated as the workflow runs.

Parameters

- `research_object` (`cwltool.cwlprov.ro.ResearchObject`)
- `full_name` (`str`)
- `host_provenance` (`bool`)
- `user_provenance` (`bool`)
- `orcid` (`str`)
- `fsaccess` (`cwltool.stdfsaccess.StdFsAccess`)
- `run_uuid` (`Optional[uuid.UUID]`)

`__str__()`

Represent this Provenance profile as a string.

Return type

`str`

`generate_prov_doc()`

Add basic namespaces.

Return type

Tuple[str, prov.model.ProvDocument]

evaluate(*process, job, job_order_object, research_obj*)

Evaluate the nature of job.

Parameters

- **process** (cwltool.process.Process)
- **job** (cwltool.utils.JobsType)
- **job_order_object** (cwltool.utils.CWLObjectType)
- **research_obj** (cwltool.cwlprov.ro.ResearchObject)

Return type

None

record_process_start(*process, job, process_run_id=None*)**Parameters**

- **process** (cwltool.process.Process)
- **job** (cwltool.utils.JobsType)
- **process_run_id** (Optional[str])

Return type

Optional[str]

start_process(*process_name, when, process_run_id=None*)

Record the start of each Process.

Parameters

- **process_name** (str)
- **when** (datetime.datetime)
- **process_run_id** (Optional[str])

Return type

str

record_process_end(*process_name, process_run_id, outputs, when*)**Parameters**

- **process_name** (str)
- **process_run_id** (str)
- **outputs** (Union[cwltool.utils.CWLObjectType, MutableSequence[cwltool.utils.CWLObjectType], None])
- **when** (datetime.datetime)

Return type

None

declare_file(*value*)**Parameters****value** (cwltool.utils.CWLObjectType)

Return type

Tuple[prov.model.ProvEntity, prov.model.ProvEntity, str]

declare_directory(value)

Register any nested files/directories.

Parameters**value** (cwltool.utils.CWLObjectType)**Return type**

prov.model.ProvEntity

declare_string(value)

Save as string in UTF-8.

Parameters**value** (str)**Return type**

Tuple[prov.model.ProvEntity, str]

declare_artefact(value)

Create data artefact entities for all file objects.

Parameters**value** (Any)**Return type**

prov.model.ProvEntity

used_artefacts(job_order, process_run_id, name=None)

Add used() for each data artefact.

Parameters

- **job_order** (Union[cwltool.utils.CWLObjectType, List[cwltool.utils.CWLObjectType]])
- **process_run_id** (str)
- **name** (Optional[str])

Return type

None

generate_output_prov(final_output, process_run_id, name)

Call wasGeneratedBy() for each output,copy the files into the RO.

Parameters

- **final_output** (Union[cwltool.utils.CWLObjectType, MutableSequence[cwltool.utils.CWLObjectType], None])
- **process_run_id** (Optional[str])
- **name** (Optional[str])

Return type

None

prospective_prov(job)

Create prospective prov recording as wfdesc prov:Plan.

Parameters`job (cwltool.utils.JobsType)`**Return type**

None

activity_has_provenance(*activity*, *prov_ids*)Add <http://www.w3.org/TR/prov-aq/> relations to nested PROV files.**Parameters**

- **activity** (*str*)
- **prov_ids** (*Sequence[prov.identifier.Identifier]*)

Return type

None

finalize_prov_profile(*name*)

Transfer the provenance related files to the RO.

Parameters**name** (*Optional[str]*)**Return type**

List[prov.identifier.QualifiedName]

cwltool.cwlprov.ro

Stores class definition of ResearchObject and WritableBagFile.

Module Contents**Classes**

*ResearchObject*CWLProv Research Object.

class cwltool.cwlprov.ro.**ResearchObject**(*fsaccess*, *temp_prefix_ro*='tmp', *orcid*='', *full_name*='')

CWLProv Research Object.

Parameters

- **fsaccess** (*cwltool.stdfsaccess.StdFsAccess*)
- **temp_prefix_ro** (*str*)
- **orcid** (*str*)
- **full_name** (*str*)

self_check()

Raise ValueError if this RO is closed.

Return type

None

__str__()

Represent this RO as a string.

Return type

str

user_provenance(*document*)

Add the user provenance.

Parameters

document (*prov.model.ProvDocument*)

Return type

None

add_tagfile(*path*, *timestamp=None*)

Add tag files to our research object.

Parameters

- **path** (*str*)
- **timestamp** (*Optional[datetime.datetime]*)

Return type

None

add_uri(*uri*, *timestamp=None*)

Parameters

- **uri** (*str*)
- **timestamp** (*Optional[datetime.datetime]*)

Return type

cwltool.cwlprov.Aggregate

add_annotation(*about*, *content*, *motivated_by='oa:describing'*)

Cheap URI relativize for current directory and /.

Parameters

- **about** (*str*)
- **content** (*List[str]*)
- **motivated_by** (*str*)

Return type

str

generate_snapshot(*prov_dep*)

Copy all of the CWL files to the snapshot/ directory.

Parameters

prov_dep (*cwltool.utils.CWLObjectType*)

Return type

None

has_data_file(*sha1hash*)

Confirm the presence of the given file in the RO.

Parameters

`sha1hash` (*str*)

Return type

bool

`add_data_file`(*from_fp*, *timestamp=None*, *content_type=None*)

Copy inputs to data/ folder.

Parameters

- `from_fp` (*IO[Any]*)
- `timestamp` (*Optional[datetime.datetime]*)
- `content_type` (*Optional[str]*)

Return type

str

`add_to_manifest`(*rel_path*, *checksums*)

Add files to the research object manifest.

Parameters

- `rel_path` (*str*)
- `checksums` (*Dict[str, str]*)

Return type

None

`cwltool.cwlprov.writablebagfile`

Stores class definition of ResearchObject and WritableBagFile.

Module Contents

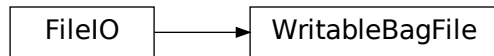
Classes

<code>WritableBagFile</code>	Writes files in research object.
------------------------------	----------------------------------

Functions

<code>write_bag_file</code> (<i>research_object</i> , <i>path</i> [, <i>encoding</i>])	Write the bag file into our research object.
<code>open_log_file_for_activity</code> (<i>research_object</i> , <i>uuid_uri</i>)	Begin the per-activity log.
<code>close_ro</code> (<i>research_object</i> [, <i>save_to</i>])	Close the Research Object, optionally saving to specified folder.
<code>packed_workflow</code> (<i>research_object</i> , <i>packed</i>)	Pack CWL description to generate re-runnable CWL object in RO.
<code>create_job</code> (<i>research_object</i> , <i>builder_job</i> [, <i>is_output</i>])	Generate the new job object with RO specific relative paths.

```
class cwltool.cwlprov.writablebagfile.WritableBagFile(research_object, rel_path)
    Bases: io.FileIO
```



Writes files in research object.

Parameters

- **research_object** (`cwltool.cwlprov.ro.ResearchObject`)
- **rel_path** (`str`)

write(*b*)

Write some content to the Bag.

Parameters

b (*Any*)

Return type

`int`

close()

Flush and close this stream.

Finalize checksums and manifests.

Return type

`None`

seekable()

Return False, seeking is not supported.

Return type

`bool`

readable()

Return False, reading is not supported.

Return type

`bool`

truncate(*size=None*)

Resize the stream, only if we haven't started writing.

Parameters

size (*Optional* [`int`])

Return type

`int`

```
cwltool.cwlprov.writablebagfile.write_bag_file(research_object, path, encoding=ENCODING)
```

Write the bag file into our research object.

Parameters

- **research_object** (`cwltool.cwlprov.ro.ResearchObject`)
- **path** (*str*)
- **encoding** (*Optional[str]*)

Return type

Union[`io.TextIOWrapper`, `WritableBagFile`]

`cwltool.cwlprov.writablebagfile.open_log_file_for_activity(research_object, uuid_uri)`

Begin the per-activity log.

Parameters

- **research_object** (`cwltool.cwlprov.ro.ResearchObject`)
- **uuid_uri** (*str*)

Return type

Union[`io.TextIOWrapper`, `WritableBagFile`]

`cwltool.cwlprov.writablebagfile.close_ro(research_object, save_to=None)`

Close the Research Object, optionally saving to specified folder.

Closing will remove any temporary files used by this research object. After calling this method, this `ResearchObject` instance can no longer be used, except for no-op calls to `.close()`.

The 'saveTo' folder should not exist - if it does, it will be deleted.

It is safe to call this function multiple times without the 'saveTo' argument, e.g. within a `try..finally` block to ensure the temporary files of this Research Object are removed.

Parameters

- **research_object** (`cwltool.cwlprov.ro.ResearchObject`)
- **save_to** (*Optional[str]*)

Return type

None

`cwltool.cwlprov.writablebagfile.packed_workflow(research_object, packed)`

Pack CWL description to generate re-runnable CWL object in RO.

Parameters

- **research_object** (`cwltool.cwlprov.ro.ResearchObject`)
- **packed** (*str*)

Return type

None

`cwltool.cwlprov.writablebagfile.create_job(research_object, builder_job, is_output=False)`

Generate the new job object with RO specific relative paths.

Parameters

- **research_object** (`cwltool.cwlprov.ro.ResearchObject`)
- **builder_job** (`cwltool.utils.CWLObjectType`)
- **is_output** (*bool*)

Return type

`cwltool.utils.CWLObjectType`

Package Contents

Classes

<i>Aggregate</i>	RO Aggregate class.
<i>AuthoredBy</i>	RO AuthoredBy class.

Functions

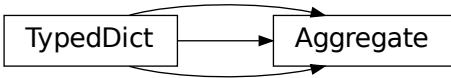
<i>checksum_copy</i> (src_file[, dst_file, hasher, buffersize])	Compute checksums while copying a file.
---	---

Attributes

<i>Annotation</i>

cwltool.cwlprov.**Annotation**

class cwltool.cwlprov.**Aggregate**
Bases: TypedDict



RO Aggregate class.

uri: `str | None`

bundledAs: `Dict[str, Any] | None`

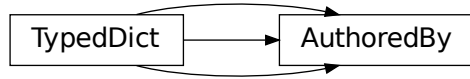
mediatype: `str | None`

conformsTo: `str | List[str] | None`

createdOn: `str | None`

createdBy: `Dict[str, str] | None`

class cwltool.cwlprov.**AuthoredBy**
Bases: TypedDict



RO AuthoredBy class.

orcid: `str` | `None`

name: `str` | `None`

uri: `str` | `None`

`cwltool.cwlprov.checksum_copy(src_file, dst_file=None, hasher=None, buffersize=1024 * 1024)`

Compute checksums while copying a file.

Parameters

- **src_file** (`IO[Any]`)
- **dst_file** (`Optional[IO[Any]]`)
- **hasher** (`Optional[Callable[[], hashlib._Hash]]`)
- **buffersize** (`int`)

Return type

`str`

Submodules

`cwltool.__main__`

Default entryptpoint for the cwltool module.

`cwltool.argparser`

Command line argument parsing for cwltool.

Module Contents

Classes

<i>FSAction</i>	Base action for our custom actions.
<i>FSAppendAction</i>	Appending version of the base action for our custom actions.
<i>FileAction</i>	Base action for our custom actions.
<i>DirectoryAction</i>	Base action for our custom actions.
<i>FileAppendAction</i>	Appending version of the base action for our custom actions.
<i>DirectoryAppendAction</i>	Appending version of the base action for our custom actions.
<i>AppendAction</i>	An argparse action that clears the default values if any value is provided.

Functions

<i>arg_parser()</i>	
<i>get_default_args()</i>	Get default values of cwltool's command line options.
<i>add_argument(toolparser, name, inptype, records[, ...])</i>	
<i>generate_parser(toolparser, tool, namemap, records[, ...])</i>	Generate an ArgumentParser for the given CWL Process.

`cwltool.argparser.arg_parser()`

Return type

`argparse.ArgumentParser`

`cwltool.argparser.get_default_args()`

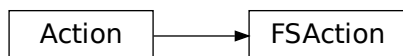
Get default values of cwltool's command line options.

Return type

`Dict[str, Any]`

class `cwltool.argparser.FSAction(option_strings, dest, nargs=None, urljoin=urllib.parse.urljoin, base_uri="", **kwargs)`

Bases: `argparse.Action`



Base action for our custom actions.

Parameters

- **option_strings** (*List*[*str*])
- **dest** (*str*)
- **nargs** (*Any*)
- **urljoin** (*Callable*[[*str*, *str*], *str*])
- **base_uri** (*str*)
- **kwargs** (*Any*)

objclass: *str* | *None*

__call__ (*parser*, *namespace*, *values*, *option_string=None*)

Parameters

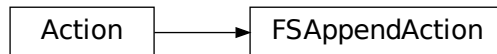
- **parser** (*argparse.ArgumentParser*)
- **namespace** (*argparse.Namespace*)
- **values** (*Union*[*str*, *Sequence*[*Any*], *None*])
- **option_string** (*Optional*[*str*])

Return type

None

```
class cwltool.argparser.FSAppendAction(option_strings, dest, nargs=None, urljoin=urllib.parse.urljoin,
                                     base_uri="", **kwargs)
```

Bases: *argparse.Action*



Appending version of the base action for our custom actions.

Parameters

- **option_strings** (*List*[*str*])
- **dest** (*str*)
- **nargs** (*Any*)
- **urljoin** (*Callable*[[*str*, *str*], *str*])
- **base_uri** (*str*)
- **kwargs** (*Any*)

objclass: *str* | *None*

__call__ (*parser*, *namespace*, *values*, *option_string=None*)

Parameters

- **parser** (*argparse.ArgumentParser*)

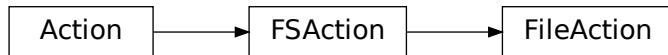
- `namespace` (*argparse.Namespace*)
- `values` (*Union[str, Sequence[Any], None]*)
- `option_string` (*Optional[str]*)

Return type

None

```
class cwltool.argparser.FileAction(option_strings, dest, nargs=None, urljoin=urllib.parse.urljoin,  
                                  base_uri="", **kwargs)
```

Bases: *FSAction*



Base action for our custom actions.

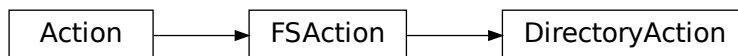
Parameters

- `option_strings` (*List[str]*)
- `dest` (*str*)
- `nargs` (*Any*)
- `urljoin` (*Callable[[str, str], str]*)
- `base_uri` (*str*)
- `kwargs` (*Any*)

objclass: *str* | *None* = 'File'

```
class cwltool.argparser.DirectoryAction(option_strings, dest, nargs=None, urljoin=urllib.parse.urljoin,  
                                       base_uri="", **kwargs)
```

Bases: *FSAction*



Base action for our custom actions.

Parameters

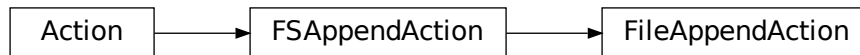
- `option_strings` (*List[str]*)
- `dest` (*str*)
- `nargs` (*Any*)

- `urljoin(Callable[[str, str], str])`
- `base_uri(str)`
- `kwargs(Any)`

objclass: `str | None = 'Directory'`

```
class cwltool.argparser.FileAppendAction(option_strings, dest, nargs=None, urljoin=urllib.parse.urljoin,
                                         base_uri="", **kwargs)
```

Bases: [FSAppendAction](#)



Appending version of the base action for our custom actions.

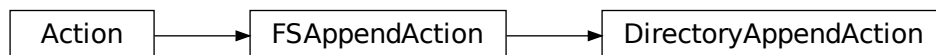
Parameters

- `option_strings(List[str])`
- `dest(str)`
- `nargs(Any)`
- `urljoin(Callable[[str, str], str])`
- `base_uri(str)`
- `kwargs(Any)`

objclass: `str | None = 'File'`

```
class cwltool.argparser.DirectoryAppendAction(option_strings, dest, nargs=None,
                                              urljoin=urllib.parse.urljoin, base_uri="", **kwargs)
```

Bases: [FSAppendAction](#)

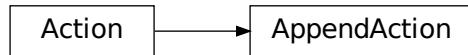


Appending version of the base action for our custom actions.

Parameters

- `option_strings(List[str])`
- `dest(str)`
- `nargs(Any)`
- `urljoin(Callable[[str, str], str])`

```
    • base_uri (str)
    • kwargs (Any)
objclass: str | None = 'Directory'
class cwltool.argparser.AppendAction(option_strings, dest, nargs=None, **kwargs)
Bases: argparse.Action
```



An argparse action that clears the default values if any value is provided.

Parameters

- **option_strings** (*List*[*str*])
- **dest** (*str*)
- **nargs** (*Any*)
- **kwargs** (*Any*)

```
__call__(parser, namespace, values, option_string=None)
```

Parameters

- **parser** ([argparse.ArgumentParser](#))
- **namespace** ([argparse.Namespace](#))
- **values** (*Union*[*str*, *Sequence*[*Any*], *None*])
- **option_string** (*Optional*[*str*])

Return type

None

```
cwltool.argparser.add_argument(toolparser, name, inptype, records, description="", default=None,
                               input_required=True, urljoin=urllib.parse.urljoin, base_uri="")
```

Parameters

- **toolparser** ([argparse.ArgumentParser](#))
- **name** (*str*)
- **inptype** (*Any*)
- **records** (*List*[*str*])
- **description** (*str*)
- **default** (*Any*)
- **input_required** (*bool*)
- **urljoin** (*Callable*[[*str*, *str*], *str*])

- **base_uri** (*str*)

Return type

None

`cwltool.argparser.generate_parser(toolparser, tool, namemap, records, input_required=True, urljoin=urllib.parse.urljoin, base_uri="")`

Generate an ArgumentParser for the given CWL Process.

Parameters

- **toolparser** (*argparse.ArgumentParser*)
- **tool** (*cwltool.process.Process*)
- **namemap** (*Dict[str, str]*)
- **records** (*List[str]*)
- **input_required** (*bool*)
- **urljoin** (*Callable[[str, str], str]*)
- **base_uri** (*str*)

Return type*argparse.ArgumentParser***cwltool.builder**

Command line builder.

Module Contents**Classes**

<i>Builder</i>	Helper class to construct a command line from a CWL CommandLineTool.
----------------	--

Functions

<i>content_limit_respected_read_bytes(f)</i>	Read a file as bytes, respecting the CONTENT_LIMIT.
<i>content_limit_respected_read(f)</i>	Read a file as a string, respecting the CONTENT_LIMIT.
<i>substitute(value, replace)</i>	Perform CWL SecondaryFilesDSL style substitution.

Attributes

INPUT_OBJ_VOCAB

`cwltool.builder.INPUT_OBJ_VOCAB: Dict[str, str]`

`cwltool.builder.content_limit_respected_read_bytes(f)`

Read a file as bytes, respecting the CONTENT_LIMIT.

Parameters

`f (IO[bytes])` – file handle

Returns

the file contents

Raises

WorkflowException – if the file is too large

Return type

`bytes`

`cwltool.builder.content_limit_respected_read(f)`

Read a file as a string, respecting the CONTENT_LIMIT.

Parameters

`f (IO[bytes])` – file handle

Returns

the file contents

Raises

WorkflowException – if the file is too large

Return type

`str`

`cwltool.builder.substitute(value, replace)`

Perform CWL SecondaryFilesDSL style substitution.

Parameters

- **value** (`str`)
- **replace** (`str`)

Return type

`str`

class `cwltool.builder.Builder`(*job, files, bindings, schemaDefs, names, requirements, hints, resources, mutation_manager, formatgraph, make_fs_access, fs_access, job_script_provider, timeout, debug, js_console, force_docker_pull, loadListing, outdir, tmpdir, stagedir, cwlVersion, container_engine*)

Bases: `cwltool.utils.HasReqsHints`



Helper class to construct a command line from a CWL CommandLineTool.

Parameters

- **job** (*cwltool.utils.CWLObjectType*)
- **files** (*List[cwltool.utils.CWLObjectType]*)
- **bindings** (*List[cwltool.utils.CWLObjectType]*)
- **schemaDefs** (*MutableMapping[str, cwltool.utils.CWLObjectType]*)
- **names** (*schema_salad.avro.schema.Names*)
- **requirements** (*List[cwltool.utils.CWLObjectType]*)
- **hints** (*List[cwltool.utils.CWLObjectType]*)
- **resources** (*Dict[str, Union[int, float]]*)
- **mutation_manager** (*Optional[cwltool.mutation.MutationManager]*)
- **formatgraph** (*Optional[rdflib.Graph]*)
- **make_fs_access** (*Type[cwltool.stdfsaccess.StdFsAccess]*)
- **fs_access** (*cwltool.stdfsaccess.StdFsAccess*)
- **job_script_provider** (*Optional[cwltool.software_requirements.DependenciesConfiguration]*)
- **timeout** (*float*)
- **debug** (*bool*)
- **js_console** (*bool*)
- **force_docker_pull** (*bool*)
- **loadListing** (*cwltool.utils.LoadListingType*)
- **outdir** (*str*)
- **tmpdir** (*str*)
- **stagedir** (*str*)
- **cwlVersion** (*str*)
- **container_engine** (*str*)

build_job_script (*commands*)

Parameters

commands (*List[str]*)

Return type

Optional[str]

bind_input(*schema*, *datum*, *discover_secondaryFiles*, *lead_pos*=None, *tail_pos*=None)

Bind an input object to the command line.

Raises

- **ValidationException** – in the event of an invalid type union
- **WorkflowException** – if a CWL Expression (“position”, “required”, “pattern”, “format”) evaluates to the wrong type or if a required secondary file is missing

Parameters

- **schema** (*cwltool.utils.CWLObjectType*)
- **datum** (*Union[cwltool.utils.CWLObjectType, List[cwltool.utils.CWLObjectType]]*)
- **discover_secondaryFiles** (*bool*)
- **lead_pos** (*Optional[Union[int, List[int]]]*)
- **tail_pos** (*Optional[Union[str, List[int]]]*)

Return type

List[MutableMapping[str, Union[str, List[int]]]]

tostr(*value*)

Represent an input parameter as a string.

Raises

WorkflowException – if the item is a File or Directory and the “path” is missing.

Parameters

value (*Union[MutableMapping[str, str], Any]*)

Return type

str

generate_arg(*binding*)

Parameters

binding (*cwltool.utils.CWLObjectType*)

Return type

List[str]

do_eval(*ex*, *context*=None, *recursive*=False, *strip_whitespace*=True)

Parameters

- **ex** (*Optional[cwltool.utils.CWLOutputType]*)
- **context** (*Optional[Any]*)
- **recursive** (*bool*)
- **strip_whitespace** (*bool*)

Return type

Optional[cwltool.utils.CWLOutputType]

cwltool.checker

Static checking of CWL workflow connectivity.

Module Contents

Functions

<code>check_types</code> (srctype, sinktype, linkMerge, valueFrom)	Check if the source and sink types are correct.
<code>merge_flatten_type</code> (src)	Return the merge flattened type of the source type.
<code>can_assign_src_to_sink</code> (src, sink[, strict])	Check for identical type specifications, ignoring extra keys like inputBinding.
<code>missing_subset</code> (fullset, subset)	
<code>static_checker</code> (workflow_inputs, workflow_outputs, ...)	Check if all source and sink types of a workflow are compatible before run time.
<code>check_all_types</code> (src_dict, sinks, sourceField, ...)	Given a list of sinks, check if their types match with the types of their sources.
<code>circular_dependency_checker</code> (step_inputs)	Check if a workflow has circular dependency.
<code>get_dependency_tree</code> (step_inputs)	Get the dependency tree in the form of adjacency list.
<code>processDFS</code> (adjacency, traversal_path, processed, cycles)	Perform depth first search.
<code>get_step_id</code> (field_id)	Extract step id from either input or output fields.
<code>is_conditional_step</code> (param_to_step, parm_id)	
<code>is_all_output_method_loop_step</code> (param_to_step, parm_id)	Check if a step contains a http://commonwl.org/cwltool#Loop requirement with <i>all</i> outputMethod.
<code>loop_checker</code> (steps)	Check http://commonwl.org/cwltool#Loop requirement compatibility with other directives.

Attributes

`SrcSink`

`cwltool.checker.check_types`(srctype, sinktype, linkMerge, valueFrom)

Check if the source and sink types are correct.

Raises

`WorkflowException` – If there is an unrecognized linkMerge type

Parameters

- **srctype** (`cwltool.utils.SinkType`)
- **sinktype** (`cwltool.utils.SinkType`)
- **linkMerge** (`Optional[str]`)
- **valueFrom** (`Optional[str]`)

Return type

Union[Literal[pass], Literal[warning], Literal[exception]]

`cwltool.checker.merge_flatten_type(src)`

Return the merge flattened type of the source type.

Parameters**src** (*cwltool.utils.SinkType*)**Return type***cwltool.utils.CWLObjectType*`cwltool.checker.can_assign_src_to_sink(src, sink, strict=False)`Check for identical type specifications, ignoring extra keys like `inputBinding`.

In non-strict comparison, at least one source type must match one sink type, except for 'null'. In strict comparison, all source types must match at least one sink type.

Parameters

- **src** (*cwltool.utils.SinkType*) – admissible source types
- **sink** (*Optional[cwltool.utils.SinkType]*) – admissible sink types
- **strict** (*bool*)

Return type*bool*`cwltool.checker.missing_subset(fullset, subset)`**Parameters**

- **fullset** (*List[Any]*)
- **subset** (*List[Any]*)

Return type*List[Any]*`cwltool.checker.static_checker(workflow_inputs, workflow_outputs, step_inputs, step_outputs, param_to_step)`

Check if all source and sink types of a workflow are compatible before run time.

Raises**ValidationException** – If any incompatibilities are detected.**Parameters**

- **workflow_inputs** (*List[cwltool.utils.CWLObjectType]*)
- **workflow_outputs** (*MutableSequence[cwltool.utils.CWLObjectType]*)
- **step_inputs** (*MutableSequence[cwltool.utils.CWLObjectType]*)
- **step_outputs** (*List[cwltool.utils.CWLObjectType]*)
- **param_to_step** (*Dict[str, cwltool.utils.CWLObjectType]*)

Return type

None

`cwltool.checker.SrcSink`

`cwltool.checker.check_all_types(src_dict, sinks, sourceField, param_to_step)`

Given a list of sinks, check if their types match with the types of their sources.

Raises

- **WorkflowException** – if there is an unrecognized linkMerge value (from `check_types()`)
- **ValidationException** – if a sourceField is missing

Parameters

- **src_dict** (`Dict[str, cwltool.utils.CWLObjectType]`)
- **sinks** (`MutableSequence[cwltool.utils.CWLObjectType]`)
- **sourceField** (`Union[Literal[source], Literal[outputSource]]`)
- **param_to_step** (`Dict[str, cwltool.utils.CWLObjectType]`)

Return type

`Dict[str, List[SrcSink]]`

`cwltool.checker.circular_dependency_checker(step_inputs)`

Check if a workflow has circular dependency.

Raises

ValidationException – If a circular dependency is detected.

Parameters

step_inputs (`List[cwltool.utils.CWLObjectType]`)

Return type

`None`

`cwltool.checker.get_dependency_tree(step_inputs)`

Get the dependency tree in the form of adjacency list.

Parameters

step_inputs (`List[cwltool.utils.CWLObjectType]`)

Return type

`Dict[str, List[str]]`

`cwltool.checker.processDFS(adjacency, traversal_path, processed, cycles)`

Perform depth first search.

Parameters

- **adjacency** (`Dict[str, List[str]]`)
- **traversal_path** (`List[str]`)
- **processed** (`List[str]`)
- **cycles** (`List[List[str]]`)

Return type

`None`

`cwltool.checker.get_step_id(field_id)`

Extract step id from either input or output fields.

Parameters

field_id (`str`)

Return type`str``cwltool.checker.is_conditional_step(param_to_step, parm_id)`**Parameters**

- `param_to_step` (`Dict[str, cwltool.utils.CWLObjectType]`)
- `parm_id` (`str`)

Return type`bool``cwltool.checker.is_all_output_method_loop_step(param_to_step, parm_id)`

Check if a step contains a <http://commonwl.org/cwltool#Loop> requirement with *all* outputMethod.

Parameters

- `param_to_step` (`Dict[str, cwltool.utils.CWLObjectType]`)
- `parm_id` (`str`)

Return type`bool``cwltool.checker.loop_checker(steps)`

Check <http://commonwl.org/cwltool#Loop> requirement compatibility with other directives.

Raises

ValidationException – If there is an incompatible combination between `cwltool:loop` and ‘scatter’ or ‘when’.

Parameters

`steps` (`Iterator[MutableMapping[str, Any]]`)

Return type`None``cwltool.command_line_tool`

Implementation of CommandLineTool.

Module Contents

Classes

<code>PathCheckingMode</code>	What characters are allowed in path names.
<code>ExpressionJob</code>	Job for <code>ExpressionTool</code> .
<code>ExpressionTool</code>	Abstract CWL Process.
<code>AbstractOperation</code>	Abstract CWL Process.
<code>CallbackJob</code>	Callback Job class, used by <code>CommandLineTool.job()</code> .
<code>CommandLineTool</code>	Abstract CWL Process.

Functions

<i>remove_path</i> (f)	
<i>revmap_file</i> (builder, outdir, f)	Remap a file from internal path to external path.
<i>check_adjust</i> (accept_re, builder, file_o)	Map files to assigned path inside a container.
<i>check_valid_locations</i> (fs_access, ob)	

Attributes

<i>OutputPortsType</i>

```
class cwltool.command_line_tool.PathCheckingMode(*args, **kws)
    Bases: enum.Enum
```



What characters are allowed in path names.
We have the strict (default) mode and the relaxed mode.

STRICT

Accepts names that contain one or more of the following:

<code>\w</code>	unicode word characters this includes most characters that can be part of a word in any language, as well as numbers and the underscore
<code>.</code>	a literal period
<code>+</code>	a literal plus sign
<code>,</code>	a literal comma
<code>-</code>	a literal minus sign
<code>:</code>	a literal colon
<code>@</code>	a literal at-symbol
<code>]</code>	a literal end-square-bracket
<code>^</code>	a literal caret symbol
<code>\u2600-\u26FF</code>	matches a single character in the range between (index 9728) and (index 9983)
<code>\U0001f600-\U0001f64f</code>	matches a single character in the range between (index 128512) and (index 128591)

Note: the following characters are intentionally not included:

1. reserved words in POSIX: `!, {, }`
2. POSIX metacharacters listed in the CWL standard as okay to reject: `|, &, ;, <, >, (,), $, `, ", ' , <space>, <tab>, <newline>`.
(In accordance with <https://www.commonwl.org/v1.0/CommandLineTool.html#File> under “path”)
3. POSIX path separator: `\`
(also listed at <https://www.commonwl.org/v1.0/CommandLineTool.html#File> under “path”)
4. Additional POSIX metacharacters: `*, ?, [, #, ~, =, %`.

TODO: switch to <https://pypi.org/project/regex/> and use `\p{Extended_Pictographic}` instead of the manual emoji ranges

RELAXED

Accept anything.

```
class cwltool.command_line_tool.ExpressionJob(builder, script, output_callback, requirements, hints,
                                              outdir=None, tmpdir=None)
```

Job for [ExpressionTool](#).

Parameters

- **builder** ([cwltool.builder.Builder](#))
- **script** ([str](#))
- **output_callback** ([Optional\[cwltool.utils.OutputCallbackType\]](#))
- **requirements** ([List\[cwltool.utils.CWLObjectType\]](#))
- **hints** ([List\[cwltool.utils.CWLObjectType\]](#))
- **outdir** ([Optional\[str\]](#))
- **tmpdir** ([Optional\[str\]](#))

```
run(runtimeContext, tmpdir_lock=None)
```

Parameters

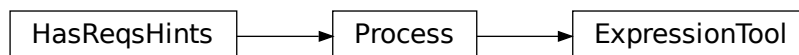
- **runtimeContext** ([cwltool.context.RuntimeContext](#))
- **tmpdir_lock** ([Optional\[threading.Lock\]](#))

Return type

None

```
class cwltool.command_line_tool.ExpressionTool(toolpath_object, loadingContext)
```

Bases: [cwltool.process.Process](#)



Abstract CWL Process.

Parameters

- **toolpath_object** (*ruamel.yaml.comments.CommentedList*)
- **loadingContext** (*cwltool.context.LoadingContext*)

job(*job_order, output_callbacks, runtimeContext*)

Parameters

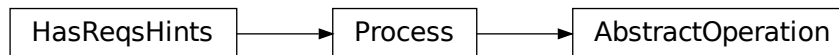
- **job_order** (*cwltool.utils.CWLObjectType*)
- **output_callbacks** (*Optional[cwltool.utils.OutputCallbackType]*)
- **runtimeContext** (*cwltool.context.RuntimeContext*)

Return type

Generator[*ExpressionJob*, None, None]

class *cwltool.command_line_tool.AbstractOperation*(*toolpath_object, loadingContext*)

Bases: *cwltool.process.Process*



Abstract CWL Process.

Parameters

- **toolpath_object** (*ruamel.yaml.comments.CommentedList*)
- **loadingContext** (*cwltool.context.LoadingContext*)

job(*job_order, output_callbacks, runtimeContext*)

Parameters

- **job_order** (*cwltool.utils.CWLObjectType*)
- **output_callbacks** (*Optional[cwltool.utils.OutputCallbackType]*)
- **runtimeContext** (*cwltool.context.RuntimeContext*)

Return type

cwltool.utils.JobsGeneratorType

cwltool.command_line_tool.remove_path(*f*)

Parameters

f (*cwltool.utils.CWLObjectType*)

Return type

None

cwltool.command_line_tool.revmap_file(*builder, outdir, f*)

Remap a file from internal path to external path.

For Docker, this maps from the path inside the container to the path outside the container. Recognizes files in the pathmapper or remaps internal output directories to the external directory.

Parameters

- **builder** (`cwltool.builder.Builder`)
- **outdir** (*str*)
- **f** (`cwltool.utils.CWLObjectType`)

Return type

`Optional[cwltool.utils.CWLObjectType]`

class `cwltool.command_line_tool.CallbackJob`(*job*, *output_callback*, *cachebuilder*, *jobcache*)

Callback Job class, used by `CommandLineTool.job()`.

Parameters

- **job** (`CommandLineTool`)
- **output_callback** (`Optional[cwltool.utils.OutputCallbackType]`)
- **cachebuilder** (`cwltool.builder.Builder`)
- **jobcache** (*str*)

run(*runtimeContext*, *tmpdir_lock*=None)

Parameters

- **runtimeContext** (`cwltool.context.RuntimeContext`)
- **tmpdir_lock** (`Optional[threading.Lock]`)

Return type

None

`cwltool.command_line_tool.check_adjust`(*accept_re*, *builder*, *file_o*)

Map files to assigned path inside a container.

We need to also explicitly walk over input, as implicit reassignment doesn't reach everything in `builder.bindings`

Parameters

- **accept_re** (`Pattern[str]`)
- **builder** (`cwltool.builder.Builder`)
- **file_o** (`cwltool.utils.CWLObjectType`)

Return type

`cwltool.utils.CWLObjectType`

`cwltool.command_line_tool.check_valid_locations`(*fs_access*, *ob*)

Parameters

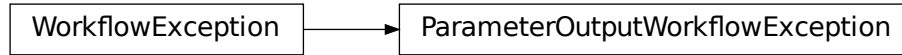
- **fs_access** (`cwltool.stdfsaccess.StdFsAccess`)
- **ob** (`cwltool.utils.CWLObjectType`)

Return type

None

`cwltool.command_line_tool.OutputPortsType`

exception `cwltool.command_line_tool.ParameterOutputWorkflowException(msg, port, **kwargs)`
 Bases: `cwltool.errors.WorkflowException`

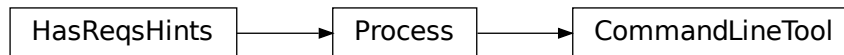


Common base class for all non-exit exceptions.

Parameters

- **msg** (*str*)
- **port** (`cwltool.utils.CWLObjectType`)
- **kwargs** (*Any*)

class `cwltool.command_line_tool.CommandLineTool(toolpath_object, loadingContext)`
 Bases: `cwltool.process.Process`



Abstract CWL Process.

Parameters

- **toolpath_object** (`ruamel.yaml.comments.CommentedList`)
- **loadingContext** (`cwltool.context.LoadingContext`)

make_job_runner (`runtimeContext`)

Parameters

runtimeContext (`cwltool.context.RuntimeContext`)

Return type

Type[`cwltool.job.JobBase`]

static make_path_mapper (`reffiles, stagedir, runtimeContext, separateDirs`)

Parameters

- **reffiles** (`List[cwltool.utils.CWLObjectType]`)
- **stagedir** (*str*)
- **runtimeContext** (`cwltool.context.RuntimeContext`)
- **separateDirs** (*bool*)

Return type*cwltool.pathmapper.PathMapper***updatePathmap**(*outdir*, *pathmap*, *fn*)

Update a PathMapper with a CWL File or Directory object.

Parameters

- **outdir** (*str*)
- **pathmap** (*cwltool.pathmapper.PathMapper*)
- **fn** (*cwltool.utils.CWLObjectType*)

Return type

None

job(*job_order*, *output_callbacks*, *runtimeContext*)**Parameters**

- **job_order** (*cwltool.utils.CWLObjectType*)
- **output_callbacks** (*Optional[cwltool.utils.OutputCallbackType]*)
- **runtimeContext** (*cwltool.context.RuntimeContext*)

Return typeGenerator[Union[*cwltool.job.JobBase*, *CallbackJob*], None, None]**collect_output_ports**(*ports*, *builder*, *outdir*, *rcode*, *compute_checksum=True*, *jobname=""*,
readers=None)**Parameters**

- **ports** (*Union[ruamel.yaml.comments.CommentSeq, Set[cwltool.utils.CWLObjectType]]*)
- **builder** (*cwltool.builder.Builder*)
- **outdir** (*str*)
- **rcode** (*int*)
- **compute_checksum** (*bool*)
- **jobname** (*str*)
- **readers** (*Optional[MutableMapping[str, cwltool.utils.CWLObjectType]]*)

Return type

OutputPortsType

collect_output(*schema*, *builder*, *outdir*, *fs_access*, *compute_checksum=True*)**Parameters**

- **schema** (*cwltool.utils.CWLObjectType*)
- **builder** (*cwltool.builder.Builder*)
- **outdir** (*str*)
- **fs_access** (*cwltool.stdfsaccess.StdFsAccess*)
- **compute_checksum** (*bool*)

Return type
Optional[cwltool.utils.CWLOutputType]

cwltool.context

Shared context objects that replace use of kwargs.

Module Contents

Classes

<i>ContextBase</i>	Shared kwargs based initializer for <i>RuntimeContext</i> and <i>LoadingContext</i> .
<i>LoadingContext</i>	Shared kwargs based initializer for <i>RuntimeContext</i> and <i>LoadingContext</i> .
<i>RuntimeContext</i>	Shared kwargs based initializer for <i>RuntimeContext</i> and <i>LoadingContext</i> .

Functions

<i>make_tool_notimpl</i> (toolpath_object, loadingContext)	Fake implementation of the make tool function.
<i>log_handler</i> (outdir, base_path_logs, stdout_path, ...)	Move logs from log location to final output.
<i>set_log_dir</i> (outdir, log_dir, subdir_name)	Set the log directory.
<i>getdefault</i> (val, default)	Return the val using the default as backup in case the val is None.

Attributes

<i>default_make_tool</i>

class cwltool.context.ContextBase(*kwargs=None*)
Shared kwargs based initializer for *RuntimeContext* and *LoadingContext*.

Parameters
kwargs (*Optional[Dict[str, Any]]*)

cwltool.context.make_tool_notimpl(*toolpath_object*, *loadingContext*)
Fake implementation of the make tool function.

Parameters

- **toolpath_object** (*ruamel.yaml.comments.CommentedList*)
- **loadingContext** (*LoadingContext*)

Return type*cwltool.process.Process*`cwltool.context.default_make_tool``cwltool.context.log_handler(outdir, base_path_logs, stdout_path, stderr_path)`

Move logs from log location to final output.

Parameters

- `outdir` (*str*)
- `base_path_logs` (*str*)
- `stdout_path` (*Optional[str]*)
- `stderr_path` (*Optional[str]*)

Return type

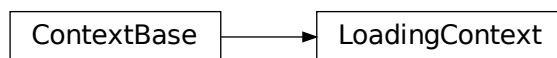
None

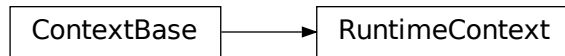
`cwltool.context.set_log_dir(outdir, log_dir, subdir_name)`

Set the log directory.

Parameters

- `outdir` (*str*)
- `log_dir` (*str*)
- `subdir_name` (*str*)

Return type*str*`class cwltool.context.LoadingContext(kwargs=None)`Bases: *ContextBase*Shared kwargs based initializer for *RuntimeContext* and *LoadingContext*.**Parameters**`kwargs` (*Optional[Dict[str, Any]]*)**copy()**Return a copy of this *LoadingContext*.**Return type***LoadingContext*`class cwltool.context.RuntimeContext(kwargs=None)`Bases: *ContextBase*



Shared kwargs based initializer for *RuntimeContext* and *LoadingContext*.

Parameters

kwargs (*Optional*[*Dict*[*str*, *Any*]])

outdir: *str* | *None*

tmpdir: *str* = ''

tmpdir_prefix: *str*

tmp_outdir_prefix: *str* = ''

stagedir: *str* = ''

get_outdir()

Return *outdir* or create one with *tmp_outdir_prefix*.

Return type

str

get_tmpdir()

Return *tmpdir* or create one with *tmpdir_prefix*.

Return type

str

get_stagedir()

Return *stagedir* or create one with *tmpdir_prefix*.

Return type

str

create_tmpdir()

Create a temporary directory that respects *tmpdir_prefix*.

Return type

str

create_outdir()

Create a temporary directory that respects *tmp_outdir_prefix*.

Return type

str

copy()

Return a copy of this *RuntimeContext*.

Return type

RuntimeContext

`cwltool.context.getdefault(val, default)`

Return the `val` using the `default` as backup in case the `val` is `None`.

Parameters

- `val` (*Any*)
- `default` (*Any*)

Return type

Any

`cwltool.cuda`

Support utilities for CUDA.

Module Contents

Functions

<code>cuda_version_and_device_count()</code>	Determine the CUDA version and number of attached CUDA GPUs.
<code>cuda_check(cuda_req, requestCount)</code>	

`cwltool.cuda.cuda_version_and_device_count()`

Determine the CUDA version and number of attached CUDA GPUs.

Return type

`Tuple[str, int]`

`cwltool.cuda.cuda_check(cuda_req, requestCount)`

Parameters

- `cuda_req` (`cwltool.utils.CWLObjectType`)
- `requestCount` (*int*)

Return type

int

`cwltool.cwlrdf`

Module Contents

Functions

gather(tool, ctx)

printrdf(wflow, ctx, style)

Serialize the CWL document into a string, ready for printing.

lastpart(uri)

dot_with_parameters(g, stdout)

dot_without_parameters(g, stdout)

printdot(wf, ctx, stdout)

`cwltool.cwlrdf.gather(tool, ctx)`

Parameters

- **tool** (`cwltool.process.Process`)
- **ctx** (`schema_salad.utils.ContextType`)

Return type

`rdflib.Graph`

`cwltool.cwlrdf.printrdf(wflow, ctx, style)`

Serialize the CWL document into a string, ready for printing.

Parameters

- **wflow** (`cwltool.process.Process`)
- **ctx** (`schema_salad.utils.ContextType`)
- **style** (`str`)

Return type

`str`

`cwltool.cwlrdf.lastpart(uri)`

Parameters

uri (`Any`)

Return type

`str`

`cwltool.cwlrdf.dot_with_parameters(g, stdout)`

Parameters

- **g** (`rdflib.Graph`)
- **stdout** (`Union[TextIO, codecs.StreamWriter]`)

Return type

`None`

`cwltool.cwlrdf.dot_without_parameters(g, stdout)`

Parameters

- `g` (`rdflib.Graph`)
- `stdout` (`Union[TextIO, codecs.StreamWriter]`)

Return type

None

`cwltool.cwlrdf.printdot(wf, ctx, stdout)`

Parameters

- `wf` (`cwltool.process.Process`)
- `ctx` (`schema_salad.utils.ContextType`)
- `stdout` (`IO[str]`)

Return type

None

`cwltool.cwlviewer`

Visualize a CWL workflow.

Module Contents

Classes

`CWLViewer`

Produce similar images with the <https://github.com/common-workflow-language/cwlviewer>.

`class cwltool.cwlviewer.CWLViewer(rdf_description)`

Produce similar images with the <https://github.com/common-workflow-language/cwlviewer>.

Parameters

`rdf_description` (`str`)

`get_dot_graph()`

Get the dot graph object.

Return type

`pydot.Graph`

`dot()`

Get the graph as graphviz.

Return type

`str`

cwltool.docker

Enables Docker software containers via the {u,}docker or podman runtimes.

Module Contents

Classes

<i>DockerCommandLineJob</i>	Runs a <i>CommandLineJob</i> in a software container using the Docker engine.
<i>PodmanCommandLineJob</i>	Runs a <i>CommandLineJob</i> in a software container using the podman engine.

```
class cwltool.docker.DockerCommandLineJob(builder, joborder, make_path_mapper, requirements, hints,
                                         name)
```

Bases: *cwltool.job.ContainerCommandLineJob*



Runs a *CommandLineJob* in a software container using the Docker engine.

Parameters

- **builder** (*cwltool.builder.Builder*)
- **joborder** (*cwltool.utils.CWLObjectType*)
- **make_path_mapper** (*Callable[[List[cwltool.utils.CWLObjectType], str, cwltool.context.RuntimeContext, bool], cwltool.pathmapper.PathMapper]*)
- **requirements** (*List[cwltool.utils.CWLObjectType]*)
- **hints** (*List[cwltool.utils.CWLObjectType]*)
- **name** (*str*)

```
get_image(docker_requirement, pull_image, force_pull, tmp_outdir_prefix)
```

Retrieve the relevant Docker container image.

Returns

True upon success

Parameters

- **docker_requirement** (*Dict[str, str]*)
- **pull_image** (*bool*)
- **force_pull** (*bool*)
- **tmp_outdir_prefix** (*str*)

Return type`bool`**get_from_requirements**(*r*, *pull_image*, *force_pull*, *tmp_outdir_prefix*)**Parameters**

- **r** (*cwltool.utils.CWLObjectType*)
- **pull_image** (*bool*)
- **force_pull** (*bool*)
- **tmp_outdir_prefix** (*str*)

Return type`Optional[str]`**static append_volume**(*runtime*, *source*, *target*, *writable=False*, *skip_mkdirs=False*)

Add binding arguments to the runtime list.

Parameters

- **runtime** (*List[str]*)
- **source** (*str*)
- **target** (*str*)
- **writable** (*bool*)
- **skip_mkdirs** (*bool*)

Return type`None`**add_file_or_directory_volume**(*runtime*, *volume*, *host_outdir_tgt*)

Append volume a file/dir mapping to the runtime option list.

Parameters

- **runtime** (*List[str]*)
- **volume** (*cwltool.pathmapper.MapperEnt*)
- **host_outdir_tgt** (*Optional[str]*)

Return type`None`**add_writable_file_volume**(*runtime*, *volume*, *host_outdir_tgt*, *tmpdir_prefix*)

Append a writable file mapping to the runtime option list.

Parameters

- **runtime** (*List[str]*)
- **volume** (*cwltool.pathmapper.MapperEnt*)
- **host_outdir_tgt** (*Optional[str]*)
- **tmpdir_prefix** (*str*)

Return type`None`

add_writable_directory_volume(runtime, volume, host_outdir_tgt, tmpdir_prefix)

Append a writable directory mapping to the runtime option list.

Parameters

- **runtime** (*List[str]*)
- **volume** (*cwltool.pathmapper.MapperEnt*)
- **host_outdir_tgt** (*Optional[str]*)
- **tmpdir_prefix** (*str*)

Return type

None

create_runtime(env, runtimeContext)

Return the list of commands to run the selected container engine.

Parameters

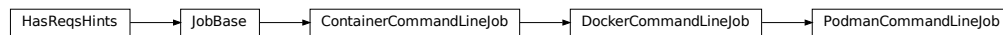
- **env** (*MutableMapping[str, str]*)
- **runtimeContext** (*cwltool.context.RuntimeContext*)

Return type

Tuple[List[str], Optional[str]]

class cwltool.docker.PodmanCommandLineJob(builder, joborder, make_path_mapper, requirements, hints, name)

Bases: *DockerCommandLineJob*



Runs a *CommandLineJob* in a software container using the podman engine.

Parameters

- **builder** (*cwltool.builder.Builder*)
- **joborder** (*cwltool.utils.CWLObjectType*)
- **make_path_mapper** (*Callable[[List[cwltool.utils.CWLObjectType], str, cwltool.context.RuntimeContext, bool], cwltool.pathmapper.PathMapper]*)
- **requirements** (*List[cwltool.utils.CWLObjectType]*)
- **hints** (*List[cwltool.utils.CWLObjectType]*)
- **name** (*str*)

`cwltool.docker_id`

Helper functions for docker.

Module Contents

Functions

<code>docker_vm_id()</code>	Return the User ID and Group ID of the default docker user inside the VM.
<code>check_output_and_strip(cmd)</code>	Pass a command list to <code>subprocess.check_output()</code> .
<code>docker_machine_name()</code>	Get the machine name of the active docker-machine machine.
<code>cmd_output_matches(check_cmd, expected_status)</code>	Run a command and compares output to expected.
<code>boot2docker_running()</code>	Check if boot2docker CLI reports that boot2docker vm is running.
<code>docker_machine_running()</code>	Ask docker-machine for the active machine and checks if its VM is running.
<code>cmd_output_to_int(cmd)</code>	Run the provided command and returns the integer value of the result.
<code>boot2docker_id()</code>	Get the UID and GID of the docker user inside a running boot2docker vm.
<code>docker_machine_id()</code>	Ask docker-machine for active machine and gets the UID of the docker user.

`cwltool.docker_id.docker_vm_id()`

Return the User ID and Group ID of the default docker user inside the VM.

When a host is using boot2docker or docker-machine to run docker with boot2docker.iso (As on Mac OS X), the UID that mounts the shared filesystem inside the VirtualBox VM is likely different than the user's UID on the host. :return: A tuple containing numeric User ID and Group ID of the docker account inside the boot2docker VM

Return type

Tuple[Optional[int], Optional[int]]

`cwltool.docker_id.check_output_and_strip(cmd)`

Pass a command list to `subprocess.check_output()`.

Returning None if an expected exception is raised :param cmd: The command to execute :return: Stripped string output of the command, or None if error

Parameters

cmd (List[str])

Return type

Optional[str]

`cwltool.docker_id.docker_machine_name()`

Get the machine name of the active docker-machine machine.

Returns

Name of the active machine or None if error

Return typeOptional[*str*]`cwltool.docker_id.cmd_output_matches(check_cmd, expected_status)`

Run a command and compares output to expected.

Parameters

- **check_cmd** (*List*[*str*]) – Command list to execute
- **expected_status** (*str*) – Expected output, e.g. “Running” or “poweroff”

Returns

Boolean value, indicating whether or not command result matched

Return type*bool*`cwltool.docker_id.boot2docker_running()`

Check if boot2docker CLI reports that boot2docker vm is running.

Returns

True if vm is running, False otherwise

Return type*bool*`cwltool.docker_id.docker_machine_running()`

Ask docker-machine for the active machine and checks if its VM is running.

Returns

True if vm is running, False otherwise

Return type*bool*`cwltool.docker_id.cmd_output_to_int(cmd)`

Run the provided command and returns the integer value of the result.

Parameters**cmd** (*List*[*str*]) – The command to run**Returns**

Integer value of result, or None if an error occurred

Return typeOptional[*int*]`cwltool.docker_id.boot2docker_id()`

Get the UID and GID of the docker user inside a running boot2docker vm.

Returns

Tuple (UID, GID), or (None, None) if error (e.g. boot2docker not present or stopped)

Return typeTuple[Optional[*int*], Optional[*int*]]`cwltool.docker_id.docker_machine_id()`

Ask docker-machine for active machine and gets the UID of the docker user.

inside the vm :return: tuple (UID, GID), or (None, None) if error (e.g. docker-machine not present or stopped)

Return typeTuple[Optional[*int*], Optional[*int*]]

`cwltool.env_to_stdout`

Python script that acts like (GNU coreutils) `env -0`.
When run as a script, it prints the the environment as `(VARIABLE=value0)*`.
Ideally we would just use `env -0`, because python (thanks to PEPs 538 and 540) will set zero to two environment variables to better handle Unicode-locale interactions, however BSD family implementations of `env` do not all support the `-0` flag so we supply this script that produces equivalent output.

Module Contents

Functions

<code>deserialize_env(data)</code>	Deserialize the output of <code>env -0</code> to dictionary.
<code>main()</code>	Print the null-separated environment to stdout.

`cwltool.env_to_stdout.deserialize_env(data)`
Deserialize the output of `env -0` to dictionary.

Parameters
`data` (`str`)

Return type
`Dict[str, str]`

`cwltool.env_to_stdout.main()`
Print the null-separated environment to stdout.

Return type
`None`

`cwltool.errors`

Module Contents

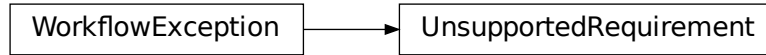
exception `cwltool.errors.WorkflowException`
Bases: `Exception`



Common base class for all non-exit exceptions.

exception `cwltool.errors.UnsupportedRequirement`

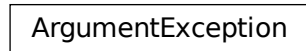
Bases: `WorkflowException`



Common base class for all non-exit exceptions.

exception `cwltool.errors.ArgumentException`

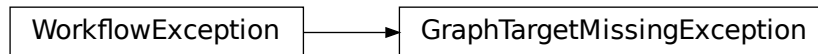
Bases: `Exception`



Mismatched command line arguments provided.

exception `cwltool.errors.GraphTargetMissingException`

Bases: `WorkflowException`



When a `$graph` is encountered and there is no target and no `main/#main`.

cwltool.executors

Single and multi-threaded executors.

Module Contents

Classes

<i>JobExecutor</i>	Abstract base job executor.
<i>SingleJobExecutor</i>	Default single-threaded CWL reference executor.
<i>MultiThreadedJobExecutor</i>	Experimental multi-threaded CWL executor.
<i>NoopJobExecutor</i>	Do nothing executor, for testing purposes only.

Attributes

<i>TMPDIR_LOCK</i>

`cwltool.executors.TMPDIR_LOCK`

class `cwltool.executors.JobExecutor`

Abstract base job executor.

__call__(*process*, *job_order_object*, *runtime_context*, *logger*=*_logger*)

Parameters

- **process** (`cwltool.process.Process`)
- **job_order_object** (`cwltool.utils.CWLObjectType`)
- **runtime_context** (`cwltool.context.RuntimeContext`)
- **logger** (`logging.Logger`)

Return type

Tuple[Optional[`cwltool.utils.CWLObjectType`], `str`]

output_callback(*out*, *process_status*)

Collect the final status and outputs.

Parameters

- **out** (Optional[`cwltool.utils.CWLObjectType`])
- **process_status** (`str`)

Return type

None

abstract run_jobs(*process*, *job_order_object*, *logger*, *runtime_context*)

Execute the jobs for the given Process.

Parameters

- **process** (`cwltool.process.Process`)
- **job_order_object** (`cwltool.utils.CWLObjectType`)
- **logger** (`logging.Logger`)
- **runtime_context** (`cwltool.context.RuntimeContext`)

Return type

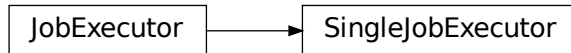
None

execute(*process*, *job_order_object*, *runtime_context*, *logger*=*_logger*)

Execute the process.

Parameters

- **process** (`cwltool.process.Process`)
- **job_order_object** (`cwltool.utils.CWLObjectType`)
- **runtime_context** (`cwltool.context.RuntimeContext`)
- **logger** (`logging.Logger`)

Return typeTuple[Union[Optional[`cwltool.utils.CWLObjectType`]], `str`]**class** `cwltool.executors.SingleJobExecutor`Bases: `JobExecutor`

Default single-threaded CWL reference executor.

run_jobs(*process*, *job_order_object*, *logger*, *runtime_context*)

Execute the jobs for the given Process.

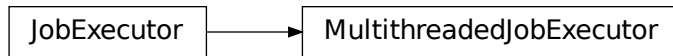
Parameters

- **process** (`cwltool.process.Process`)
- **job_order_object** (`cwltool.utils.CWLObjectType`)
- **logger** (`logging.Logger`)
- **runtime_context** (`cwltool.context.RuntimeContext`)

Return type

None

class `cwltool.executors.MultithreadedJobExecutor`Bases: `JobExecutor`



Experimental multi-threaded CWL executor.

Does simple resource accounting, will not start a job unless it has cores / ram available, but does not make any attempt to optimize usage.

select_resources(*request*, *runtime_context*)

Naïve check for available cpu cores and memory.

Parameters

- **request** (*Dict[str, Union[int, float]]*)
- **runtime_context** (*cwltool.context.RuntimeContext*)

Return type

Dict[str, Union[int, float]]

run_job(*job*, *runtime_context*)

Execute a single Job in a separate thread.

Parameters

- **job** (*Optional[cwltool.utils.JobsType]*)
- **runtime_context** (*cwltool.context.RuntimeContext*)

Return type

None

wait_for_next_completion(*runtime_context*)

Wait for jobs to finish.

Parameters

runtime_context (*cwltool.context.RuntimeContext*)

Return type

None

run_jobs(*process*, *job_order_object*, *logger*, *runtime_context*)

Execute the jobs for the given Process.

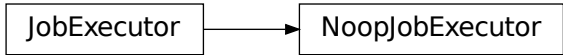
Parameters

- **process** (*cwltool.process.Process*)
- **job_order_object** (*cwltool.utils.CWLObjectType*)
- **logger** (*logging.Logger*)
- **runtime_context** (*cwltool.context.RuntimeContext*)

Return type

None


```
class cwltool.executors.NoopJobExecutor
  Bases: JobExecutor
```



Do nothing executor, for testing purposes only.

run_jobs(*process*, *job_order_object*, *logger*, *runtime_context*)

Execute the jobs for the given Process.

- Parameters
- **process** (`cwltool.process.Process`)
 - **job_order_object** (`cwltool.utils.CWLObjectType`)
 - **logger** (`logging.Logger`)
 - **runtime_context** (`cwltool.context.RuntimeContext`)

Return type

None

execute(*process*, *job_order_object*, *runtime_context*, *logger=None*)

Execute the process.

- Parameters
- **process** (`cwltool.process.Process`)
 - **job_order_object** (`cwltool.utils.CWLObjectType`)
 - **runtime_context** (`cwltool.context.RuntimeContext`)
 - **logger** (`Optional[logging.Logger]`)

Return type

Tuple[Optional[cwltool.utils.CWLObjectType], str]

`cwltool.factory`

Module Contents

Classes

<i>Callable</i>	Result of <code>:Factory.make()</code> .
<i>Factory</i>	Easy way to load a CWL document for execution.

exception `cwltool.factory.WorkflowStatus(out, status)`

Bases: `Exception`

WorkflowStatus

Common base class for all non-exit exceptions.

Parameters

- **out** (*Optional*[`cwltool.utils.CWLObjectType`])
- **status** (`str`)

class `cwltool.factory.Callable(t, factory)`

Result of `:Factory.make()`.

Parameters

- **t** (`cwltool.process.Process`)
- **factory** (`Factory`)

`__call__` (***kwargs*)

Parameters

kwargs (*Any*)

Return type

`Union`[`str`, *Optional*[`cwltool.utils.CWLObjectType`]]

class `cwltool.factory.Factory(executor=None, loading_context=None, runtime_context=None)`

Easy way to load a CWL document for execution.

Parameters

- **executor** (*Optional*[`cwltool.executors.JobExecutor`])
- **loading_context** (*Optional*[`cwltool.context.LoadingContext`])
- **runtime_context** (*Optional*[`cwltool.context.RuntimeContext`])

loading_context: `cwltool.context.LoadingContext`

runtime_context: `cwltool.context.RuntimeContext`

make(*cwl*)

Instantiate a CWL object from a CWL document.

Parameters

cwl (*Union*[`str`, *Dict*[`str`, *Any*]])

Return type

`Callable`

cwltool.flatten

Module Contents

Functions

```
flatten(thing[, ltypes])
```

cwltool.flatten.flatten(thing, ltypes=(list, tuple))

Parameters

- thing (Any)
- ltypes (Any)

Return type

List[Any]

cwltool.job

Module Contents

Classes

JobBase	Base class for get_requirement().
CommandLineJob	Base class for get_requirement().
ContainerCommandLineJob	Commandline job using containers.

Functions

```
relink_initialworkdir(pathmapper, host_outdir,  
...[, ...])  
neverquote(string[, pos, endpos])
```

Attributes

<code>CollectOutputsType</code>
<code>needs_shell_quoting_re</code>
<code>FORCE_SHELLED_POPEN</code>
<code>SHELL_COMMAND_TEMPLATE</code>
<code>CONTROL_CODE_RE</code>

`cwltool.job.CollectOutputsType`

`cwltool.job.needs_shell_quoting_re`

`cwltool.job.FORCE_SHELLED_POPEN`

`cwltool.job.SHELL_COMMAND_TEMPLATE` = Multiline-String

```
"""#!/bin/bash
python3 "run_job.py" "job.json"
"""
```

`cwltool.job.relink_initialworkdir(pathmapper, host_outdir, container_outdir, inplace_update=False)`

Parameters

- `pathmapper` (`cwltool.pathmapper.PathMapper`)
- `host_outdir` (`str`)
- `container_outdir` (`str`)
- `inplace_update` (`bool`)

Return type

None

`cwltool.job.neverquote(string, pos=0, endpos=0)`

Parameters

- `string` (`str`)
- `pos` (`int`)
- `endpos` (`int`)

Return type

Optional[Match[`str`]]

class `cwltool.job.JobBase(builder, joborder, make_path_mapper, requirements, hints, name)`

Bases: `cwltool.utils.HasReqsHints`



Base class for `get_requirement()`.

Parameters

- **builder** (`cwltool.builder.Builder`)
- **joborder** (`cwltool.utils.CWLObjectType`)
- **make_path_mapper** (`Callable[[List[cwltool.utils.CWLObjectType], str, cwltool.context.RuntimeContext, bool], cwltool.pathmapper.PathMapper]`)
- **requirements** (`List[cwltool.utils.CWLObjectType]`)
- **hints** (`List[cwltool.utils.CWLObjectType]`)
- **name** (`str`)

`__repr__()`

Represent this Job object.

Return type

`str`

abstract `run(runtimeContext, tmpdir_lock=None)`

Parameters

- **runtimeContext** (`cwltool.context.RuntimeContext`)
- **tmpdir_lock** (`Optional[threading.Lock]`)

Return type

`None`

prepare_environment (`runtimeContext, envVarReq`)

Set up environment variables.

Here we prepare the environment for the job, based on any preserved variables and *EnvVarRequirement*. Later, changes due to *MPIRequirement*, *Secrets*, or *SoftwareRequirement* are applied (in that order).

Parameters

- **runtimeContext** (`cwltool.context.RuntimeContext`)
- **envVarReq** (`Mapping[str, str]`)

Return type

`None`

process_monitor (`sproc`)

Watch a process, logging its max memory usage.

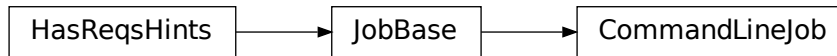
Parameters

sproc (`subprocess.Popen[str]`)

Return type

None

```
class cwltool.job.CommandLineJob(builder, joborder, make_path_mapper, requirements, hints, name)
```

Bases: [JobBase](#)Base class for `get_requirement()`.**Parameters**

- **builder** (`cwltool.builder.Builder`)
- **joborder** (`cwltool.utils.CWLObjectType`)
- **make_path_mapper** (`Callable[[List[cwltool.utils.CWLObjectType], str, cwltool.context.RuntimeContext, bool], cwltool.pathmapper.PathMapper]`)
- **requirements** (`List[cwltool.utils.CWLObjectType]`)
- **hints** (`List[cwltool.utils.CWLObjectType]`)
- **name** (`str`)

```
run(runtimeContext, tmpdir_lock=None)
```

Parameters

- **runtimeContext** (`cwltool.context.RuntimeContext`)
- **tmpdir_lock** (`Optional[threading.Lock]`)

Return type

None

```
cwltool.job.CONTROL_CODE_RE = '\\x1b\\[[0-9;]*[a-zA-Z]'
```

```
class cwltool.job.ContainerCommandLineJob(builder, joborder, make_path_mapper, requirements, hints, name)
```

Bases: [JobBase](#)

Commandline job using containers.

Parameters

- **builder** (`cwltool.builder.Builder`)
- **joborder** (`cwltool.utils.CWLObjectType`)
- **make_path_mapper** (`(Callable[[List[cwltool.utils.CWLObjectType], str, cwltool.context.RuntimeContext, bool], cwltool.pathmapper.PathMapper])`)
- **requirements** (`List[cwltool.utils.CWLObjectType]`)
- **hints** (`List[cwltool.utils.CWLObjectType]`)
- **name** (`str`)

CONTAINER_TMPDIR: `str = '/tmp'`

abstract get_from_requirements(`r, pull_image, force_pull, tmp_outdir_prefix`)

Parameters

- **r** (`cwltool.utils.CWLObjectType`)
- **pull_image** (`bool`)
- **force_pull** (`bool`)
- **tmp_outdir_prefix** (`str`)

Return type

`Optional[str]`

abstract create_runtime(`env, runtime_context`)

Return the list of commands to run the selected container engine.

Parameters

- **env** (`MutableMapping[str, str]`)
- **runtime_context** (`cwltool.context.RuntimeContext`)

Return type

`Tuple[List[str], Optional[str]]`

abstract static append_volume(`runtime, source, target, writable=False`)

Add binding arguments to the runtime list.

Parameters

- **runtime** (`List[str]`)
- **source** (`str`)
- **target** (`str`)
- **writable** (`bool`)

Return type

`None`

abstract add_file_or_directory_volume(`runtime, volume, host_outdir_tgt`)

Append volume a file/dir mapping to the runtime option list.

Parameters

- **runtime** (`List[str]`)
- **volume** (`cwltool.pathmapper.MapperEnt`)

- **host_outdir_tgt** (*Optional* [*str*])

Return type

None

abstract add_writable_file_volume(*runtime*, *volume*, *host_outdir_tgt*, *tmpdir_prefix*)

Append a writable file mapping to the runtime option list.

Parameters

- **runtime** (*List* [*str*])
- **volume** (*cwltool.pathmapper.MapperEnt*)
- **host_outdir_tgt** (*Optional* [*str*])
- **tmpdir_prefix** (*str*)

Return type

None

abstract add_writable_directory_volume(*runtime*, *volume*, *host_outdir_tgt*, *tmpdir_prefix*)

Append a writable directory mapping to the runtime option list.

Parameters

- **runtime** (*List* [*str*])
- **volume** (*cwltool.pathmapper.MapperEnt*)
- **host_outdir_tgt** (*Optional* [*str*])
- **tmpdir_prefix** (*str*)

Return type

None

create_file_and_add_volume(*runtime*, *volume*, *host_outdir_tgt*, *secret_store*, *tmpdir_prefix*)

Create the file and add a mapping.

Parameters

- **runtime** (*List* [*str*])
- **volume** (*cwltool.pathmapper.MapperEnt*)
- **host_outdir_tgt** (*Optional* [*str*])
- **secret_store** (*Optional* [*cwltool.secrets.SecretStore*])
- **tmpdir_prefix** (*str*)

Return type

str

add_volumes(*pathmapper*, *runtime*, *tmpdir_prefix*, *secret_store=None*, *any_path_okay=False*)

Append volume mappings to the runtime option list.

Parameters

- **pathmapper** (*cwltool.pathmapper.PathMapper*)
- **runtime** (*List* [*str*])
- **tmpdir_prefix** (*str*)
- **secret_store** (*Optional* [*cwltool.secrets.SecretStore*])

- **any_path_okay** (*bool*)

Return type

None

run(*runtimeContext, tmpdir_lock=None*)

Parameters

- **runtimeContext** (*cwltool.context.RuntimeContext*)
- **tmpdir_lock** (*Optional[threading.Lock]*)

Return type

None

docker_monitor(*cidfile, tmpdir_prefix, cleanup_cidfile, docker_exe, process*)

Record memory usage of the running Docker container.

Parameters

- **cidfile** (*str*)
- **tmpdir_prefix** (*str*)
- **cleanup_cidfile** (*bool*)
- **docker_exe** (*str*)
- **process** (*subprocess.Popen[str]*)

Return type

None

cwltool.load_tool

Loads a CWL document.

Module Contents

Functions

<code>default_loader</code>	([<code>fetcher_constructor</code> , <code>enable_dev</code> , ...])	
<code>resolve_tool_uri</code>	(<code>argsworkflow</code> [, <code>resolver</code> , ...])	
<code>fetch_document</code>	(<code>argsworkflow</code> [, <code>loadingContext</code>])	Retrieve a CWL document.
<code>update_index</code>	(<code>document_loader</code> , <code>pr</code>)	
<code>fast_parser</code>	(<code>workflowobj</code> , <code>fileuri</code> , <code>uri</code> , <code>loadingContext</code> , ...)	
<code>resolve_and_validate_document</code>	(<code>loadingContext</code> , ...[, ...])	Validate a CWL document.
<code>make_tool</code>	(<code>uri</code> , <code>loadingContext</code>)	Make a Python CWL object.
<code>load_tool</code>	(<code>argsworkflow</code> [, <code>loadingContext</code>])	
<code>resolve_overrides</code>	(<code>ov</code> , <code>ov_uri</code> , <code>baseurl</code>)	
<code>load_overrides</code>	(<code>ov</code> , <code>base_url</code>)	
<code>recursive_resolve_and_validate_document</code>	(...[, ...])	Validate a CWL document, checking that a tool object can be built.

Attributes

<code>docloaderctx</code>
<code>jobloader_id_name</code>
<code>jobloaderctx</code>
<code>overrides_ctx</code>

`cwltool.load_tool.docloaderctx`: `schema_salad.utils.ContextType`

`cwltool.load_tool.jobloader_id_name` = `'__id'`

`cwltool.load_tool.jobloaderctx`: `schema_salad.utils.ContextType`

`cwltool.load_tool.overrides_ctx`: `schema_salad.utils.ContextType`

`cwltool.load_tool.default_loader`(`fetcher_constructor=None`, `enable_dev=False`, `doc_cache=True`)

Parameters

- `fetcher_constructor` (*Optional*[`schema_salad.utils.FetcherCallableType`])
- `enable_dev` (*bool*)

- **doc_cache** (*bool*)

Return type

`schema_salad.ref_resolver.Loader`

`cwltool.load_tool.resolve_tool_uri`(*argsworkflow*, *resolver=None*, *fetcher_constructor=None*, *document_loader=None*)

Parameters

- **argsworkflow** (*str*)
- **resolver** (*Optional[cwltool.utils.ResolverType]*)
- **fetcher_constructor** (*Optional[schema_salad.utils.FetcherCallableType]*)
- **document_loader** (*Optional[schema_salad.ref_resolver.Loader]*)

Return type

`Tuple[str, str]`

`cwltool.load_tool.fetch_document`(*argsworkflow*, *loadingContext=None*)

Retrieve a CWL document.

Parameters

- **argsworkflow** (*Union[str, cwltool.utils.CWLObjectType]*)
- **loadingContext** (*Optional[cwltool.context.LoadingContext]*)

Return type

`Tuple[cwltool.context.LoadingContext, ruamel.yaml.comments.CommentedList, str]`

`cwltool.load_tool.update_index`(*document_loader*, *pr*)

Parameters

- **document_loader** (*schema_salad.ref_resolver.Loader*)
- **pr** (*ruamel.yaml.comments.CommentedList*)

Return type

`None`

`cwltool.load_tool.fast_parser`(*workflowobj*, *fileuri*, *uri*, *loadingContext*, *fetcher*)

Parameters

- **workflowobj** (*Union[ruamel.yaml.comments.CommentedList, ruamel.yaml.comments.CommentedList, None]*)
- **fileuri** (*Optional[str]*)
- **uri** (*str*)
- **loadingContext** (*cwltool.context.LoadingContext*)
- **fetcher** (*schema_salad.fetcher.Fetcher*)

Return type

`Tuple[Union[ruamel.yaml.comments.CommentedList, ruamel.yaml.comments.CommentedList], ruamel.yaml.comments.CommentedList]`

```
cwltool.load_tool.resolve_and_validate_document(loadingContext, workflowobj, uri,  
                                                preprocess_only=False)
```

Validate a CWL document.

Parameters

- **loadingContext** (`cwltool.context.LoadingContext`)
- **workflowobj** (`Union[ruamel.yaml.comments.CommentMap, ruamel.yaml.comments.CommentSeq]`)
- **uri** (`str`)
- **preprocess_only** (`bool`)

Return type

`Tuple[cwltool.context.LoadingContext, str]`

```
cwltool.load_tool.make_tool(uri, loadingContext)
```

Make a Python CWL object.

Parameters

- **uri** (`Union[str, ruamel.yaml.comments.CommentMap, ruamel.yaml.comments.CommentSeq]`)
- **loadingContext** (`cwltool.context.LoadingContext`)

Return type

`cwltool.process.Process`

```
cwltool.load_tool.load_tool(argsworkflow, loadingContext=None)
```

Parameters

- **argsworkflow** (`Union[str, cwltool.utils.CWLObjectType]`)
- **loadingContext** (`Optional[cwltool.context.LoadingContext]`)

Return type

`cwltool.process.Process`

```
cwltool.load_tool.resolve_overrides(ov, ov_uri, baseurl)
```

Parameters

- **ov** (`schema_salad.utils.IdxResultType`)
- **ov_uri** (`str`)
- **baseurl** (`str`)

Return type

`List[cwltool.utils.CWLObjectType]`

```
cwltool.load_tool.load_overrides(ov, base_url)
```

Parameters

- **ov** (`str`)
- **base_url** (`str`)

Return type

`List[cwltool.utils.CWLObjectType]`

`cwltool.load_tool.recursive_resolve_and_validate_document`(*loadingContext*, *workflowobj*, *uri*,
preprocess_only=False)

Validate a CWL document, checking that a tool object can be built.

Parameters

- **loadingContext** (`cwltool.context.LoadingContext`)
- **workflowobj** (`Union[ruamel.yaml.comments.CommentedList, ruamel.yaml.comments.CommentedSeq]`)
- **uri** (*str*)
- **preprocess_only** (*bool*)

Return type

Tuple[`cwltool.context.LoadingContext`, *str*, `cwltool.process.Process`]

`cwltool.loghandler`

Shared logger for cwltool.

Module Contents

Functions

```
configure_logging(stderr_handler, no_warnings, Configure logging.  
quiet, ...)
```

Attributes

```
defaultStreamHandler
```

`cwltool.loghandler.defaultStreamHandler`

`cwltool.loghandler.configure_logging`(*stderr_handler*, *no_warnings*, *quiet*, *debug*, *enable_color*,
timestamps, *base_logger=_logger*)

Configure logging.

Parameters

- **stderr_handler** (`logging.Handler`)
- **no_warnings** (*bool*)
- **quiet** (*bool*)
- **debug** (*bool*)
- **enable_color** (*bool*)
- **timestamps** (*bool*)

- `base_logger` (*logging.Logger*)

Return type
None

`cwltool.main`

Entry point for cwltool.

Module Contents

Classes

<i>ProvLogFormatter</i>	Enforce ISO8601 with both T and Z.
-------------------------	------------------------------------

Functions

<code>append_word_to_default_user_agent(word)</code>	Append the specified word to the requests http user agent string if it's not already there.
<code>generate_example_input(inptype, default)</code>	Convert a single input schema into an example.
<code>realize_input_schema(input_types, schema_defs)</code>	Replace references to named typed with the actual types.
<code>generate_input_template(tool)</code>	Generate an example input object for the given CWL process.
<code>load_job_order(args, stdin, fetcher_constructor, ...)</code>	
<code>init_job_order(job_order_object, args, process, ..., ...)</code>	
<code>make_relative(base, obj)</code>	Relativize the location URI of a File or Directory object.
<code>printdeps(obj, document_loader, stdout, relative_deps, uri)</code>	Print a JSON representation of the dependencies of the CWL document.
<code>prov_deps(obj, document_loader, uri[, basedir])</code>	
<code>find_deps(obj, document_loader, uri[, basedir, nestedirs])</code>	Find the dependencies of the CWL document.
<code>print_pack>LoadingContext, uri)</code>	Return a CWL serialization of the CWL document in JSON.
<code>supported_cwl_versions(enable_dev)</code>	
<code>setup_schema(args, custom_schema_callback)</code>	
<code>setup_provenance(args, runtimeContext[, args])</code>	
<code>setup_loadingContext>LoadingContext, runtime-Context, args)</code>	Prepare a LoadingContext from the given arguments.
<code>make_template(tool, target)</code>	Make a template CWL input object for the give Process.
<code>inherit_reqshints(tool, parent)</code>	Copy down requirements and hints from ancestors of a given process.
<code>choose_target(args, tool, loading_context)</code>	Walk the Workflow, extract the subset matches all the args.targets.
<code>choose_step(args, tool, loading_context)</code>	Walk the given Workflow and extract just args.single_step.
<code>choose_process(args, tool, loadingContext)</code>	Walk the given Workflow and extract just args.single_process.
<code>check_working_directories(runtimeContext)</code>	Make any needed working directories.
<code>print_targets(tool, stdout, loading_context[, prefix])</code>	Recursively find targets for --subgraph and friends.
<code>main([argsl, args, job_order_object, stdin, stdout, ...])</code>	
<code>find_default_container(builder[, default_container, ...])</code>	Find a container.
<code>windows_check()</code>	See if we are running on MS Windows and warn about the lack of support.
<code>run(*args, **kwargs)</code>	Run cwltool.

Attributes

<code><i>docker_exe</i></code>
<code><i>ProvOut</i></code>

`cwltool.main.docker_exe: str`

`cwltool.main.append_word_to_default_user_agent(word)`

Append the specified word to the requests http user agent string if it's not already there.

Parameters

word (*str*)

Return type

None

`cwltool.main.generate_example_input(inptype, default)`

Convert a single input schema into an example.

Parameters

- **inptype** (*Optional[cwltool.utils.CWLObjectType]*)
- **default** (*Optional[cwltool.utils.CWLObjectType]*)

Return type

Tuple[Any, str]

`cwltool.main.realize_input_schema(input_types, schema_defs)`

Replace references to named typed with the actual types.

Parameters

- **input_types** (*MutableSequence[Union[str, cwltool.utils.CWLObjectType]]*)
- **schema_defs** (*MutableMapping[str, cwltool.utils.CWLObjectType]*)

Return type

MutableSequence[Union[str, cwltool.utils.CWLObjectType]]

`cwltool.main.generate_input_template(tool)`

Generate an example input object for the given CWL process.

Parameters

tool (*cwltool.process.Process*)

Return type

cwltool.utils.CWLObjectType

`cwltool.main.load_job_order(args, stdin, fetcher_constructor, overrides_list, tool_file_uri)`

Parameters

- **args** (*argparse.Namespace*)
- **stdin** (*IO[Any]*)
- **fetcher_constructor** (*Optional[schema_salad.utils.FetcherCallableType]*)
- **overrides_list** (*List[cwltool.utils.CWLObjectType]*)

- **tool_file_uri** (*str*)

Return type

Tuple[Optional[cwltool.utils.CWLObjectType], str, schema_salad.ref_resolver.Loader]

`cwltool.main.init_job_order(job_order_object, args, process, loader, stdout, print_input_deps=False, relative_deps='primary', make_fs_access=StdFsAccess, input_basedir="", secret_store=None, input_required=True, runtime_context=None)`

Parameters

- **job_order_object** (Optional[cwltool.utils.CWLObjectType])
- **args** (*argparse.Namespace*)
- **process** (cwltool.process.Process)
- **loader** (schema_salad.ref_resolver.Loader)
- **stdout** (IO[*str*])
- **print_input_deps** (*bool*)
- **relative_deps** (*str*)
- **make_fs_access** (Callable[[*str*], cwltool.stdfsaccess.StdFsAccess])
- **input_basedir** (*str*)
- **secret_store** (Optional[cwltool.secrets.SecretStore])
- **input_required** (*bool*)
- **runtime_context** (Optional[cwltool.context.RuntimeContext])

Return type

cwltool.utils.CWLObjectType

`cwltool.main.make_relative(base, obj)`

Relativize the location URI of a File or Directory object.

Parameters

- **base** (*str*)
- **obj** (cwltool.utils.CWLObjectType)

Return type

None

`cwltool.main.printdeps(obj, document_loader, stdout, relative_deps, uri, basedir=None, nestdirs=True)`

Print a JSON representation of the dependencies of the CWL document.

Parameters

- **obj** (cwltool.utils.CWLObjectType)
- **document_loader** (schema_salad.ref_resolver.Loader)
- **stdout** (IO[*str*])
- **relative_deps** (*str*)
- **uri** (*str*)
- **basedir** (Optional[*str*])
- **nestdirs** (*bool*)

Return type

None

`cwltool.main.prov_deps(obj, document_loader, uri, basedir=None)`**Parameters**

- `obj` (`cwltool.utils.CWLObjectType`)
- `document_loader` (`schema_salad.ref_resolver.Loader`)
- `uri` (`str`)
- `basedir` (`Optional[str]`)

Return type`cwltool.utils.CWLObjectType``cwltool.main.find_deps(obj, document_loader, uri, basedir=None, nestdirs=True)`

Find the dependencies of the CWL document.

Parameters

- `obj` (`cwltool.utils.CWLObjectType`)
- `document_loader` (`schema_salad.ref_resolver.Loader`)
- `uri` (`str`)
- `basedir` (`Optional[str]`)
- `nestdirs` (`bool`)

Return type`cwltool.utils.CWLObjectType``cwltool.main.print_pack>LoadingContext, uri`

Return a CWL serialization of the CWL document in JSON.

Parameters

- `loadingContext` (`cwltool.context>LoadingContext`)
- `uri` (`str`)

Return type`str``cwltool.main.supported_cwl_versions(enable_dev)`**Parameters**`enable_dev` (`bool`)**Return type**`List[str]``cwltool.main.setup_schema(args, custom_schema_callback)`**Parameters**

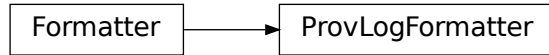
- `args` (`argparse.Namespace`)
- `custom_schema_callback` (`Optional[Callable[[], None]]`)

Return type

None

```
class cwltool.main.ProvLogFormatter
```

```
    Bases: logging.Formatter
```



Enforce ISO8601 with both T and Z.

```
formatTime(record, datefmt=None)
```

Override the default formatTime to include the timezone.

Parameters

- **record** (*logging.LogRecord*)
- **datefmt** (*Optional[str]*)

Return type

str

```
cwltool.main.ProvOut
```

```
cwltool.main.setup_provenance(args, runtimeContext, argsl=None)
```

Parameters

- **args** (*argparse.Namespace*)
- **runtimeContext** (*cwltool.context.RuntimeContext*)
- **argsl** (*Optional[List[str]]*)

Return type

Tuple[ProvOut, logging.StreamHandler[ProvOut]]

```
cwltool.main.setup_loadingContext/loadingContext, runtimeContext, args)
```

Prepare a LoadingContext from the given arguments.

Parameters

- **loadingContext** (*Optional[cwltool.context.LoadingContext]*)
- **runtimeContext** (*cwltool.context.RuntimeContext*)
- **args** (*argparse.Namespace*)

Return type

cwltool.context.LoadingContext

```
cwltool.main.make_template(tool, target)
```

Make a template CWL input object for the give Process.

Parameters

- **tool** (*cwltool.process.Process*)
- **target** (*IO[str]*)

Return type

None

`cwltool.main.inherit_reqshints(tool, parent)`

Copy down requirements and hints from ancestors of a given process.

Parameters

- **tool** (`cwltool.process.Process`)
- **parent** (`cwltool.process.Process`)

Return type

None

`cwltool.main.choose_target(args, tool, loading_context)`

Walk the Workflow, extract the subset matches all the args.targets.

Parameters

- **args** (`argparse.Namespace`)
- **tool** (`cwltool.process.Process`)
- **loading_context** (`cwltool.context.LoadingContext`)

Return typeOptional[`cwltool.process.Process`]`cwltool.main.choose_step(args, tool, loading_context)`

Walk the given Workflow and extract just args.single_step.

Parameters

- **args** (`argparse.Namespace`)
- **tool** (`cwltool.process.Process`)
- **loading_context** (`cwltool.context.LoadingContext`)

Return typeOptional[`cwltool.process.Process`]`cwltool.main.choose_process(args, tool, loadingContext)`

Walk the given Workflow and extract just args.single_process.

Parameters

- **args** (`argparse.Namespace`)
- **tool** (`cwltool.process.Process`)
- **loadingContext** (`cwltool.context.LoadingContext`)

Return typeOptional[`cwltool.process.Process`]`cwltool.main.check_working_directories(runtimeContext)`

Make any needed working directories.

Parameters**runtimeContext** (`cwltool.context.RuntimeContext`)**Return type**Optional[`int`]

`cwltool.main.print_targets(tool, stdout, loading_context, prefix="")`

Recursively find targets for –subgraph and friends.

Parameters

- **tool** (`cwltool.process.Process`)
- **stdout** (`IO[str]`)
- **loading_context** (`cwltool.context.LoadingContext`)
- **prefix** (`str`)

Return type

None

`cwltool.main.main(argsl=None, args=None, job_order_object=None, stdin=sys.stdin, stdout=None, stderr=sys.stderr, versionfunc=versionstring, logger_handler=None, custom_schema_callback=None, executor=None, loadingContext=None, runtimeContext=None, input_required=True)`

Parameters

- **argsl** (`Optional[List[str]]`)
- **args** (`Optional[argparse.Namespace]`)
- **job_order_object** (`Optional[cwltool.utils.CWLObjectType]`)
- **stdin** (`IO[Any]`)
- **stdout** (`Optional[IO[str]]`)
- **stderr** (`IO[Any]`)
- **versionfunc** (`Callable[[], str]`)
- **logger_handler** (`Optional[logging.Handler]`)
- **custom_schema_callback** (`Optional[Callable[[], None]]`)
- **executor** (`Optional[cwltool.executors.JobExecutor]`)
- **loadingContext** (`Optional[cwltool.context.LoadingContext]`)
- **runtimeContext** (`Optional[cwltool.context.RuntimeContext]`)
- **input_required** (`bool`)

Return type

`int`

`cwltool.main.find_default_container(builder, default_container=None, use_biocontainers=None, container_image_cache_path=None)`

Find a container.

Parameters

- **builder** (`cwltool.utils.HasReqHints`)
- **default_container** (`Optional[str]`)
- **use_biocontainers** (`Optional[bool]`)
- **container_image_cache_path** (`Optional[str]`)

Return type

`Optional[str]`

`cwltool.main.windows_check()`

See if we are running on MS Windows and warn about the lack of support.

Return type

None

`cwltool.main.run(*args, **kwargs)`

Run cwltool.

Parameters

- **args** (*Any*)
- **kwargs** (*Any*)

Return type

`int`

`cwltool.mpi`

Experimental support for MPI.

Module Contents

Classes

MpiConfig

Attributes

MpiConfigT

MPIRequirementName

`cwltool.mpi.MpiConfigT`

`cwltool.mpi.MPIRequirementName = 'http://commonwl.org/cwltool#MPIRequirement'`

`class cwltool.mpi.MpiConfig(runner='mpirun', nproc_flag='-n', default_nproc=1, extra_flags=None, env_pass=None, env_pass_regex=None, env_set=None)`

Parameters

- **runner** (*str*)
- **nproc_flag** (*str*)
- **default_nproc** (*Union[int, str]*)
- **extra_flags** (*Optional[List[str]]*)

- `env_pass` (*Optional*[*List*[*str*]])
- `env_pass_regex` (*Optional*[*List*[*str*]])
- `env_set` (*Optional*[*Mapping*[*str*, *str*]])

classmethod `load`(*config_file_name*)

Create the `MpiConfig` object from the contents of a YAML file.

The file must contain exactly one object, whose attributes must be in the list allowed in the class initialiser (all are optional).

Parameters

`config_file_name` (*str*)

Return type

`MpiConfigT`

pass_through_env_vars(*env*)

Take the configured list of environment variables and pass them to the executed process.

Parameters

`env` (*MutableMapping*[*str*, *str*])

Return type

`None`

set_env_vars(*env*)

Set some variables to the value configured.

Parameters

`env` (*MutableMapping*[*str*, *str*])

Return type

`None`

`cwltool.mutation`

Module Contents

Classes

<i>MutationManager</i>	Lock manager for checking correctness of in-place update of files.
------------------------	--

Attributes

<i>MutationState</i>

`cwltool.mutation.MutationState`

class `cwltool.mutation.MutationManager`

Lock manager for checking correctness of in-place update of files.

Used to validate that in-place file updates happen sequentially, and that a file which is registered for in-place update cannot be read or updated by any other steps.

register_reader(*stepname*, *obj*)

Parameters

- **stepname** (*str*)
- **obj** (`cwltool.utils.CWLObjectType`)

Return type

None

release_reader(*stepname*, *obj*)

Parameters

- **stepname** (*str*)
- **obj** (`cwltool.utils.CWLObjectType`)

Return type

None

register_mutation(*stepname*, *obj*)

Parameters

- **stepname** (*str*)
- **obj** (`cwltool.utils.CWLObjectType`)

Return type

None

set_generation(*obj*)

Parameters

- **obj** (`cwltool.utils.CWLObjectType`)

Return type

None

unset_generation(*obj*)

Parameters

- **obj** (`cwltool.utils.CWLObjectType`)

Return type

None

cwltool.pack

Reformat a CWL document and all its references to be a single stream.

Module Contents

Functions

find_run(d, loadref, runs)

find_ids(d, ids)

replace_refs(d, rewrite, stem, newstem)

import_embed(d, seen)

pack(loadingContext, uri[, rewrite_out, loader])

Attributes

LoadRefType

cwltool.pack.**LoadRefType**

cwltool.pack.**find_run**(d, loadref, runs)

Parameters

- **d** (*Union*[cwltool.utils.CWLObjectType, schema_salad.utils.ResolveType])
- **loadref** (*LoadRefType*)
- **runs** (*Set*[*str*])

Return type

None

cwltool.pack.**find_ids**(d, ids)

Parameters

- **d** (*Union*[cwltool.utils.CWLObjectType, cwltool.utils.CWLOutputType, MutableSequence[cwltool.utils.CWLObjectType], None])
- **ids** (*Set*[*str*])

Return type

None

`cwltool.pack.replace_refs(d, rewrite, stem, newstem)`

Parameters

- **d** (*Any*)
- **rewrite** (*Dict[str, str]*)
- **stem** (*str*)
- **newstem** (*str*)

Return type

None

`cwltool.pack.import_embed(d, seen)`

Parameters

- **d** (*Union[MutableSequence[cwltool.utils.CWLObjectType], cwltool.utils.CWLObjectType, cwltool.utils.CWLOutputType]*)
- **seen** (*Set[str]*)

Return type

None

`cwltool.pack.pack(loadingContext, uri, rewrite_out=None, loader=None)`

Parameters

- **loadingContext** (*cwltool.context.LoadingContext*)
- **uri** (*str*)
- **rewrite_out** (*Optional[Dict[str, str]]*)
- **loader** (*Optional[schema_salad.ref_resolver.Loader]*)

Return type

cwltool.utils.CWLObjectType

`cwltool.pathmapper`

Module Contents

Classes

PathMapper

Mapping of files from relative path provided in the file to a tuple.

Attributes

<i>MapperEnt</i>	Mapper entries.
------------------	-----------------

`cwltool.pathmapper.MapperEnt`

Mapper entries.

`cwltool.pathmapper.resolved: str`

The “real” path on the local file system (after resolving relative paths and traversing symlinks)

`cwltool.pathmapper.target: str`

The path on the target file system (under stagedir)

`cwltool.pathmapper.type: str`

The object type. One of “File”, “Directory”, “CreateFile”, “WritableFile”, or “CreateWritableFile”.

`cwltool.pathmapper.staged: bool`

If the File has been staged yet

class `cwltool.pathmapper.PathMapper`(*referenced_files, basedir, stagedir, separateDirs=True*)

Mapping of files from relative path provided in the file to a tuple.

(absolute local path, absolute container path)

The tao of PathMapper:

The initializer takes a list of `class: File` and `class: Directory` objects, a base directory (for resolving relative references) and a staging directory (where the files are mapped to).

The purpose of the setup method is to determine where each File or Directory should be placed on the target file system (relative to stagedir).

If `separatedirs=True`, unrelated files will be isolated in their own directories under stagedir. If `separatedirs=False`, files and directories will all be placed in stagedir (with the possibility for name collisions...)

The path map maps the “location” of the input Files and Directory objects to a tuple (resolved, target, type). The “resolved” field is the “real” path on the local file system (after resolving relative paths and traversing symlinks). The “target” is the path on the target file system (under stagedir). The type is the object type (one of File, Directory, CreateFile, WritableFile, CreateWritableFile).

The latter three (CreateFile, WritableFile, CreateWritableFile) are used by InitialWorkDirRequirement to indicate files that are generated on the fly (CreateFile and CreateWritableFile, in this case “resolved” holds the file contents instead of the path because they file doesn’t exist) or copied into the output directory so they can be opened for update (“r+” or “a”) (WritableFile and CreateWritableFile).

Parameters

- **referenced_files** (*List[cwltool.utils.CWLObjectType]*)
- **basedir** (*str*)
- **stagedir** (*str*)
- **separateDirs** (*bool*)

visitlisting(*listing, stagedir, basedir, copy=False, staged=False*)

Parameters

- **listing** (*List[cwltool.utils.CWLObjectType]*)

- **stagedir** (*str*)
- **basedir** (*str*)
- **copy** (*bool*)
- **staged** (*bool*)

Return type

None

visit(*obj*, *stagedir*, *basedir*, *copy=False*, *staged=False*)**Parameters**

- **obj** (*cwltool.utils.CWLObjectType*)
- **stagedir** (*str*)
- **basedir** (*str*)
- **copy** (*bool*)
- **staged** (*bool*)

Return type

None

setup(*referenced_files*, *basedir*)**Parameters**

- **referenced_files** (*List[cwltool.utils.CWLObjectType]*)
- **basedir** (*str*)

Return type

None

mapper(*src*)**Parameters****src** (*str*)**Return type**

MapperEnt

files()

Return a dictionary keys view of locations.

Return typeKeysView[*str*]**items**()

Return a dictionary items view.

Return typeItemsView[*str*, MapperEnt]**items_exclude_children**()

Return a dictionary items view minus any entries which are children of other entries.

Return typeItemsView[*str*, MapperEnt]

reversemap(*target*)
Find the (source, resolved_path) for the given target, if any.

Parameters
target (*str*)
Return type
Optional[Tuple[*str*, *str*]]

update(*key, resolved, target, ctype, stage*)
Update an existine entry.

Parameters
• **key** (*str*)
• **resolved** (*str*)
• **target** (*str*)
• **ctype** (*str*)
• **stage** (*bool*)

Return type
MapperEnt

__contains__(*key*)
Test for the presence of the given relative path in this mapper.

Parameters
key (*str*)
Return type
bool

__iter__()
Get iterator for the maps.

Return type
Iterator[MapperEnt]

cwltool.process

Classes and methods relevant for all CWL Process types.

Module Contents

Classes

<i>LogAsDebugFilter</i>	Filter instances are used to perform arbitrary filtering of LogRecords.
<i>Process</i>	Abstract CWL Process.

Functions

<i>use_standard_schema</i> (version)	
<i>use_custom_schema</i> (version, name, text)	
<i>get_schema</i> (version)	
<i>shortname</i> (inputid)	
<i>stage_files</i> (pathmapper[, stage_func, ig- nore_writable, ...])	Link or copy files to their targets. Create them as needed.
<i>relocateOutputs</i> (outputObj, destination_path, ...[, ...])	
<i>cleanIntermediate</i> (output_dirs)	
<i>add_sizes</i> (fsaccess, obj)	
<i>fill_in_defaults</i> (inputs, job, fsaccess)	For each missing input in the input object, copy over the default.
<i>avroize_type</i> (field_type[, name_prefix])	Add missing information to a type so that CWL types are valid.
<i>get_overrides</i> (overrides, toolid)	Combine overrides for the target tool ID.
<i>var_spool_cwl_detector</i> (obj[, item, obj_key])	Detect any textual reference to /var/spool/cwl.
<i>eval_resource</i> (builder, resource_req)	
<i>uniquename</i> (stem[, names])	
<i>nestdir</i> (base, deps)	
<i>mergedirs</i> (listing)	
<i>scandeps</i> (base, doc, reffields, urlfields, loadref[, ...])	Search for external files references in a CWL document or input object.
<i>compute_checksums</i> (fs_access, fileobj)	

Attributes

<i>supportedProcessRequirements</i>
<i>cwl_files</i>
<i>salad_files</i>
<i>SCHEMA_CACHE</i>
<i>SCHEMA_FILE</i>
<i>SCHEMA_DIR</i>
<i>SCHEMA_ANY</i>
<i>custom_schemas</i>
<i>FILE_COUNT_WARNING</i>
<i>CWL_IANA</i>

```
class cwltool.process.LogAsDebugFilter(name, parent)
    Bases: logging.Filter
```



Filter instances are used to perform arbitrary filtering of LogRecords.

Loggers and Handlers can optionally use Filter instances to filter records as desired. The base filter class only allows events which are below a certain point in the logger hierarchy. For example, a filter initialized with “A.B” will allow events logged by loggers “A.B”, “A.B.C”, “A.B.C.D”, “A.B.D” etc. but not “A.BB”, “B.A.B” etc. If initialized with the empty string, all events are passed.

Parameters

- **name** (*str*)
- **parent** (*logging.Logger*)

filter(*record*)

Determine if the specified record is to be logged.

Returns True if the record should be logged, or False otherwise. If deemed appropriate, the record may be modified in-place.

Parameters

- record** (*logging.LogRecord*)

Return type`bool``cwltool.process.supportedProcessRequirements``cwltool.process.cwl_files = ('Workflow.yml', 'CommandLineTool.yml',
'CommonWorkflowLanguage.yml', 'Process.yml',...``cwltool.process.salad_files = ('metaschema.yml', 'metaschema_base.yml', 'salad.md',
'field_name.yml', 'import_include.md',...``cwltool.process.SCHEMA_CACHE: Dict[str, Tuple[schema_salad.ref_resolver.Loader,
schema_salad.avro.schema.Names | schema_salad.avro.schema.SchemaParseException,
cwltool.utils.CWLObjectType, schema_salad.ref_resolver.Loader]]``cwltool.process.SCHEMA_FILE: cwltool.utils.CWLObjectType | None``cwltool.process.SCHEMA_DIR: cwltool.utils.CWLObjectType | None``cwltool.process.SCHEMA_ANY: cwltool.utils.CWLObjectType | None``cwltool.process.custom_schemas: Dict[str, Tuple[str, str]]``cwltool.process.use_standard_schema(version)`**Parameters**`version (str)`**Return type**`None``cwltool.process.use_custom_schema(version, name, text)`**Parameters**

- `version (str)`
- `name (str)`
- `text (str)`

Return type`None``cwltool.process.get_schema(version)`**Parameters**`version (str)`**Return type**

<code>Tuple[schema_salad.ref_resolver.Loader, schema_salad.avro.schema.SchemaParseException, schema_salad.ref_resolver.Loader]</code>	<code>Union[schema_salad.avro.schema.Names, cwltool.utils.CWLObjectType, cwltool.utils.Loader]</code>
---	---

`cwltool.process.shortname(inputid)`**Parameters**`inputid (str)`**Return type**`str`

`cwltool.process.stage_files(pathmapper, stage_func=None, ignore_writable=False, symlink=True, secret_store=None, fix_conflicts=False)`

Link or copy files to their targets. Create them as needed.

Raises

`WorkflowException` – if there is a file staging conflict

Parameters

- `pathmapper` (`cwltool.pathmapper.PathMapper`)
- `stage_func` (`Optional[Callable[[str, str], None]]`)
- `ignore_writable` (`bool`)
- `symlink` (`bool`)
- `secret_store` (`Optional[cwltool.secrets.SecretStore]`)
- `fix_conflicts` (`bool`)

Return type

None

`cwltool.process.relocateOutputs(outputObj, destination_path, source_directories, action, fs_access, compute_checksum=True, path_mapper=PathMapper)`

Parameters

- `outputObj` (`cwltool.utils.CWLObjectType`)
- `destination_path` (`str`)
- `source_directories` (`Set[str]`)
- `action` (`str`)
- `fs_access` (`cwltool.stdfsaccess.StdFsAccess`)
- `compute_checksum` (`bool`)
- `path_mapper` (`Type[cwltool.pathmapper.PathMapper]`)

Return type

`cwltool.utils.CWLObjectType`

`cwltool.process.cleanIntermediate(output_dirs)`

Parameters

`output_dirs` (`Iterable[str]`)

Return type

None

`cwltool.process.add_sizes(fsaccess, obj)`

Parameters

- `fsaccess` (`cwltool.stdfsaccess.StdFsAccess`)
- `obj` (`cwltool.utils.CWLObjectType`)

Return type

None

`cwltool.process.fill_in_defaults(inputs, job, fsaccess)`

For each missing input in the input object, copy over the default.

Raises

`WorkflowException` – if a required input parameter is missing

Parameters

- **inputs** (`List[cwltool.utils.CWLObjectType]`)
- **job** (`cwltool.utils.CWLObjectType`)
- **fsaccess** (`cwltool.stdfsaccess.StdFsAccess`)

Return type

None

`cwltool.process.avroize_type(field_type, name_prefix="")`

Add missing information to a type so that CWL types are valid.

Parameters

- **field_type** (`Union[cwltool.utils.CWLObjectType, MutableSequence[Any], cwltool.utils.CWLOutputType, None]`)
- **name_prefix** (`str`)

Return type

`Union[cwltool.utils.CWLObjectType, MutableSequence[Any], cwltool.utils.CWLOutputType, None]`

`cwltool.process.get_overrides(overrides, toolid)`

Combine overrides for the target tool ID.

Parameters

- **overrides** (`MutableSequence[cwltool.utils.CWLObjectType]`)
- **toolid** (`str`)

Return type

`cwltool.utils.CWLObjectType`

`cwltool.process.var_spool_cwl_detector(obj, item=None, obj_key=None)`

Detect any textual reference to /var/spool/cwl.

Parameters

- **obj** (`cwltool.utils.CWLOutputType`)
- **item** (`Optional[Any]`)
- **obj_key** (`Optional[Any]`)

Return type

`bool`

`cwltool.process.eval_resource(builder, resource_req)`

Parameters

- **builder** (`cwltool.builder.Builder`)
- **resource_req** (`Union[str, int, float]`)

Return type

Optional[Union[str, int, float]]

`cwltool.process.FILE_COUNT_WARNING = 5000`**class** `cwltool.process.Process(toolpath_object, loadingContext)`Bases: `cwltool.utils.HasReqsHints`

Abstract CWL Process.

Parameters

- **toolpath_object** (`ruamel.yaml.comments.CommentedList`)
- **loadingContext** (`cwltool.context.LoadingContext`)

evalResources(*builder, runtimeContext*)**Parameters**

- **builder** (`cwltool.builder.Builder`)
- **runtimeContext** (`cwltool.context.RuntimeContext`)

Return type

Dict[str, Union[int, float]]

checkRequirements(*rec, supported_process_requirements*)

Check the presence of unsupported requirements.

Parameters

- **rec** (`Union[MutableSequence[cwltool.utils.CWLObjectType], cwltool.utils.CWLObjectType, cwltool.utils.CWLObjectType, None]`)
- **supported_process_requirements** (`Iterable[str]`)

Return type

None

validate_hints(*avsc_names, hints, strict*)

Process the hints field.

Parameters

- **avsc_names** (`schema_salad.avro.schema.Names`)
- **hints** (`List[cwltool.utils.CWLObjectType]`)
- **strict** (`bool`)

Return type

None

visit(*op*)

Parameters

op ([Callable](#)[[[ruamel.yaml.comments.CommentedMap](#)], *None*])

Return type

None

abstract job(*job_order, output_callbacks, runtimeContext*)

Parameters

- **job_order** ([cwltool.utils.CWLObjectType](#))
- **output_callbacks** ([Optional](#)[[cwltool.utils.OutputCallbackType](#)])
- **runtimeContext** ([cwltool.context.RuntimeContext](#))

Return type

[cwltool.utils.JobsGeneratorType](#)

__str__()

Return the id of this CWL process.

Return type

[str](#)

[cwltool.process.uniquename](#)(*stem, names=None*)

Parameters

- **stem** ([str](#))
- **names** ([Optional](#)[[Set](#)[[str](#)]])

Return type

[str](#)

[cwltool.process.nestdir](#)(*base, deps*)

Parameters

- **base** ([str](#))
- **deps** ([cwltool.utils.CWLObjectType](#))

Return type

[cwltool.utils.CWLObjectType](#)

[cwltool.process.mergedirs](#)(*listing*)

Parameters

listing ([MutableSequence](#)[[cwltool.utils.CWLObjectType](#)])

Return type

[MutableSequence](#)[[cwltool.utils.CWLObjectType](#)]

[cwltool.process.CWL_IANA](#) = '<https://www.iana.org/assignments/media-types/application/cwl>'

[cwltool.process.scandeps](#)(*base, doc, reffields, urlfields, loadref, urljoin=urllib.parse.urljoin, nestdirs=True*)

Search for external files references in a CWL document or input object.

Looks for objects with 'class: File' or 'class: Directory' and adds them to the list of dependencies.

Parameters

- **base** (*str*) – the base URL for relative references.
- **doc** (*Union[cwltool.utils.CWLObjectType, MutableSequence[cwltool.utils.CWLObjectType]]*) – a CWL document or input object
- **urlfields** (*Set[str]*) – added as a File dependency
- **reffields** (*Set[str]*) – field name like a workflow step ‘run’; will be added as a dependency and also loaded (using the ‘loadref’ function) and recursively scanned for dependencies. Those dependencies will be added as secondary files to the primary file.
- **nestdirs** (*bool*) – if true, create intermediate directory objects when a file is located in a subdirectory under the starting directory. This is so that if the dependencies are materialized, they will produce the same relative file system locations.
- **loadref** (*Callable[[str, str], Union[ruamel.yaml.comments.CommentedException, ruamel.yaml.comments.CommentedException, str, None]]*)
- **urljoin** (*Callable[[str, str], str]*)

Returns

A list of File or Directory dependencies

Return type

MutableSequence[cwltool.utils.CWLObjectType]

`cwltool.process.compute_checksums(fs_access, fileobj)`

Parameters

- **fs_access** (*cwltool.stdfsaccess.StdFsAccess*)
- **fileobj** (*cwltool.utils.CWLObjectType*)

Return type

None

`cwltool.procgenerator`

Module Contents**Classes**

<i>ProcessGeneratorJob</i>	Result of ProcessGenerator.job().
<i>ProcessGenerator</i>	Abstract CWL Process.

class `cwltool.procgenerator.ProcessGeneratorJob(procgenerator)`

Result of ProcessGenerator.job().

Parameters

procgenerator (*ProcessGenerator*)

receive_output (*jobout, processStatus*)

Process the results.

Parameters

- **jobout** (*Optional[cwltool.utils.CWLObjectType]*)

- **processStatus** (*str*)

Return type

None

job(*job_order*, *output_callbacks*, *runtimeContext*)

Parameters

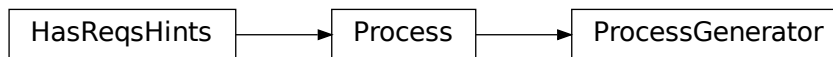
- **job_order** (*cwltool.utils.CWLObjectType*)
- **output_callbacks** (*Optional[cwltool.utils.OutputCallbackType]*)
- **runtimeContext** (*cwltool.context.RuntimeContext*)

Return type

cwltool.utils.JobsGeneratorType

class *cwltool.procgenerator.ProcessGenerator*(*toolpath_object*, *loadingContext*)

Bases: *cwltool.process.Process*



Abstract CWL Process.

Parameters

- **toolpath_object** (*ruamel.yaml.comments.CommentedList*)
- **loadingContext** (*cwltool.context.LoadingContext*)

job(*job_order*, *output_callbacks*, *runtimeContext*)

Parameters

- **job_order** (*cwltool.utils.CWLObjectType*)
- **output_callbacks** (*Optional[cwltool.utils.OutputCallbackType]*)
- **runtimeContext** (*cwltool.context.RuntimeContext*)

Return type

cwltool.utils.JobsGeneratorType

result(*job_order*, *jobout*, *runtimeContext*)

Parameters

- **job_order** (*cwltool.utils.CWLObjectType*)
- **jobout** (*cwltool.utils.CWLObjectType*)
- **runtimeContext** (*cwltool.context.RuntimeContext*)

Return type

Tuple[cwltool.process.Process, cwltool.utils.CWLObjectType]

cwltool.resolver

Resolves references to CWL documents from local or remote places.

Module Contents

Functions

<code>resolve_local(document_loader, uri)</code>
<code>tool_resolver(document_loader, uri)</code>
<code>resolve_ga4gh_tool(document_loader, uri)</code>

Attributes

<code>ga4gh_tool_registries</code>
<code>GA4GH_TRS_FILES</code>
<code>GA4GH_TRS_PRIMARY_DESCRIPTOR</code>

`cwltool.resolver.resolve_local(document_loader, uri)`

Parameters

- **document_loader** (*Optional*[`schema_salad.ref_resolver.Loader`])
- **uri** (*str*)

Return type

Optional[*str*]

`cwltool.resolver.tool_resolver(document_loader, uri)`

Parameters

- **document_loader** (`schema_salad.ref_resolver.Loader`)
- **uri** (*str*)

Return type

Optional[*str*]

`cwltool.resolver.ga4gh_tool_registries = ['https://dockstore.org/api']`

`cwltool.resolver.GA4GH_TRS_FILES = '{0}/api/ga4gh/v2/tools/{1}/versions/{2}/CWL/files'`

`cwltool.resolver.GA4GH_TRS_PRIMARY_DESCRIPTOR = '{0}/api/ga4gh/v2/tools/{1}/versions/{2}/plain-CWL/descriptor/{3}'`

`cwltool.resolver.resolve_ga4gh_tool(document_loader, uri)`

Parameters

- **document_loader** (*schema_salad.ref_resolver.Loader*)
- **uri** (*str*)

Return type

Optional[*str*]

`cwltool.run_job`

Only used when there is a job script or `CWLTOOL_FORCE_SHELL_POPEN=1`.

Module Contents

Functions

<code>handle_software_environment(cwl_env, script)</code>	Update the provided environment dict by running the script.
<code>main(argv)</code>	Read in the configuration JSON and execute the commands.

`cwltool.run_job.handle_software_environment(cwl_env, script)`

Update the provided environment dict by running the script.

Parameters

- **cwl_env** (*Dict[str, str]*)
- **script** (*str*)

Return type

Dict[str, str]

`cwltool.run_job.main(argv)`

Read in the configuration JSON and execute the commands.

The first argument is the path to the JSON dictionary file containing keys:

“commands”: an array of strings that represents the command line to run “cwd”: A string specifying which directory to run in “env”: a dictionary of strings containing the environment variables to set “stdin_path”: a string (or a null) giving the path that should be piped to STDIN “stdout_path”: a string (or a null) giving the path that should receive the STDOUT “stderr_path”: a string (or a null) giving the path that should receive the STDERR

The second argument is optional, it specifies a shell script to execute prior,

and the environment variables it sets will be combined with the environment variables from the “env” key in the JSON dictionary from the first argument.

Parameters

argv (*List[str]*)

Return type

int

cwltool.secrets

Minimal in memory storage of secrets.

Module Contents

Classes

<i>SecretStore</i>

Minimal implementation of a secret storage.

class cwltool.secrets.SecretStore

Minimal implementation of a secret storage.

add(*value*)

Add the given value to the store.

Returns a placeholder value to use until the real value is needed.

Parameters

value (*Optional*[cwltool.utils.CWLOutputType])

Return type

Optional[cwltool.utils.CWLOutputType]

store(*secrets*, *job*)

Sanitize the job object of any of the given secrets.

Parameters

- **secrets** (*List*[*str*])
- **job** (cwltool.utils.CWLObjectType)

Return type

None

has_secret(*value*)

Test if the provided document has any of our secrets.

Parameters

value (cwltool.utils.CWLOutputType)

Return type

bool

retrieve(*value*)

Replace placeholders with their corresponding secrets.

Parameters

value (cwltool.utils.CWLOutputType)

Return type

cwltool.utils.CWLOutputType

`cwltool.singularity`

Support for executing Docker format containers using Singularity {2,3}.x or Apptainer 1.x.

Module Contents

Classes

<code>SingularityCommandLineJob</code>	Commandline job using containers.
--	-----------------------------------

Functions

<code>get_version()</code>	Parse the output of 'singularity --version' to determine the flavor and version.
<code>is_apptainer_1_or_newer()</code>	Check if apptainer singularity distribution is version 1.0 or higher.
<code>is_version_2_6()</code>	Check if this singularity version is exactly version 2.6.
<code>is_version_3_or_newer()</code>	Check if this version is singularity version 3 or newer or equivalent.
<code>is_version_3_1_or_newer()</code>	Check if this version is singularity version 3.1 or newer or equivalent.
<code>is_version_3_4_or_newer()</code>	Detect if Singularity v3.4+ is available.
<code>is_version_3_9_or_newer()</code>	Detect if Singularity v3.9+ is available.

`cwltool.singularity.get_version()`

Parse the output of 'singularity --version' to determine the flavor and version.

Both pieces of information will be cached.

Returns

A tuple containing: - A tuple with major and minor version numbers as integer. - A string with the name of the singularity flavor.

Return type

`Tuple[List[int], str]`

`cwltool.singularity.is_apptainer_1_or_newer()`

Check if apptainer singularity distribution is version 1.0 or higher.

Apptainer v1.0.0 is compatible with SingularityCE 3.9.5. See: <https://github.com/apptainer/apptainer/releases>

Return type

`bool`

`cwltool.singularity.is_version_2_6()`

Check if this singularity version is exactly version 2.6.

Also returns False if the flavor is not singularity or singularity-ce.

Return type

`bool`

`cwltool.singularity.is_version_3_or_newer()`

Check if this version is singularity version 3 or newer or equivalent.

Return type

`bool`

`cwltool.singularity.is_version_3_1_or_newer()`

Check if this version is singularity version 3.1 or newer or equivalent.

Return type

`bool`

`cwltool.singularity.is_version_3_4_or_newer()`

Detect if Singularity v3.4+ is available.

Return type

`bool`

`cwltool.singularity.is_version_3_9_or_newer()`

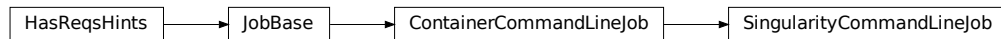
Detect if Singularity v3.9+ is available.

Return type

`bool`

class `cwltool.singularity.SingularityCommandLineJob`(*builder, joborder, make_path_mapper, requirements, hints, name*)

Bases: `cwltool.job.ContainerCommandLineJob`



Commandline job using containers.

Parameters

- **builder** (`cwltool.builder.Builder`)
- **joborder** (`cwltool.utils.CWLObjectType`)
- **make_path_mapper** (`Callable[[List[cwltool.utils.CWLObjectType], str, cwltool.context.RuntimeContext, bool], cwltool.pathmapper.PathMapper]`)
- **requirements** (`List[cwltool.utils.CWLObjectType]`)
- **hints** (`List[cwltool.utils.CWLObjectType]`)
- **name** (`str`)

static `get_image`(*dockerRequirement, pull_image, tmp_outdir_prefix, force_pull=False*)

Acquire the software container image in the specified dockerRequirement.

Uses Singularity and returns the success as a bool. Updates the provided dockerRequirement with the specific dockerImageId to the full path of the local image, if found. Likewise the dockerRequirement['dockerPull'] is updated to a docker:// URI if needed.

Parameters

- `dockerRequirement` (*Dict[str, str]*)
- `pull_image` (*bool*)
- `tmp_outdir_prefix` (*str*)
- `force_pull` (*bool*)

Return type

bool

get_from_requirements(*r, pull_image, force_pull, tmp_outdir_prefix*)

Return the filename of the Singularity image.

(e.g. hello-world-latest.{img,sif}).

Parameters

- `r` (*cwltool.utils.CWLObjectType*)
- `pull_image` (*bool*)
- `force_pull` (*bool*)
- `tmp_outdir_prefix` (*str*)

Return type

Optional[str]

static append_volume(*runtime, source, target, writable=False*)

Add binding arguments to the runtime list.

Parameters

- `runtime` (*List[str]*)
- `source` (*str*)
- `target` (*str*)
- `writable` (*bool*)

Return type

None

add_file_or_directory_volume(*runtime, volume, host_outdir_tgt*)

Append volume a file/dir mapping to the runtime option list.

Parameters

- `runtime` (*List[str]*)
- `volume` (*cwltool.pathmapper.MapperEnt*)
- `host_outdir_tgt` (*Optional[str]*)

Return type

None

add_writable_file_volume(*runtime, volume, host_outdir_tgt, tmpdir_prefix*)

Append a writable file mapping to the runtime option list.

Parameters

- `runtime` (*List[str]*)
- `volume` (*cwltool.pathmapper.MapperEnt*)

- `host_outdir_tgt` (*Optional*[*str*])
- `tmpdir_prefix` (*str*)

Return type

None

add_writable_directory_volume(*runtime*, *volume*, *host_outdir_tgt*, *tmpdir_prefix*)

Append a writable directory mapping to the runtime option list.

Parameters

- `runtime` (*List*[*str*])
- `volume` (*cwltool.pathmapper.MapperEnt*)
- `host_outdir_tgt` (*Optional*[*str*])
- `tmpdir_prefix` (*str*)

Return type

None

create_runtime(*env*, *runtime_context*)

Return the Singularity runtime list of commands and options.

Parameters

- `env` (*MutableMapping*[*str*, *str*])
- `runtime_context` (*cwltool.context.RuntimeContext*)

Return typeTuple[List[*str*], *Optional*[*str*]]**cwltool.singularity_utils**

Support for executing Docker format containers using Singularity {2,3}.x or Apptainer 1.x.

Module Contents**Functions***singularity_supports_userns*()

Confirm if the version of Singularity install supports the --userns flag.

`cwltool.singularity_utils.singularity_supports_userns()`

Confirm if the version of Singularity install supports the --userns flag.

Return type*bool*

cwltool.software_requirements

This module handles resolution of SoftwareRequirement hints.

This is accomplished mainly by adapting cwltool internals to galaxy-tool-util’s concept of “dependencies”. Despite the name, galaxy-tool-util is a light weight library that can be used to map SoftwareRequirements in all sorts of ways - Homebrew, Conda, custom scripts, environment modules. We’d be happy to find ways to adapt new packages managers and such as well.

Module Contents

Classes

<i>DependenciesConfiguration</i>	Dependency configuration class, for RuntimeContext.job_script_provider.
----------------------------------	---

Functions

<i>get_dependencies</i> (builder)
<i>get_container_from_software_requirements</i> (...[, ...])
<i>ensure_galaxy_lib_available</i> ()

Attributes

<i>ToolRequirement</i>
<i>SOFTWARE_REQUIREMENTS_ENABLED</i>
<i>COMMAND_WITH_DEPENDENCIES_TEMPLATE</i>

`cwltool.software_requirements.ToolRequirement`

`cwltool.software_requirements.SOFTWARE_REQUIREMENTS_ENABLED`

`cwltool.software_requirements.COMMAND_WITH_DEPENDENCIES_TEMPLATE`

class `cwltool.software_requirements.DependenciesConfiguration`(args)

Dependency configuration class, for RuntimeContext.job_script_provider.

Parameters

args (*argparse.Namespace*)

build_job_script(*builder*, *command*)

Parameters

- **builder** (`cwltool.builder.Builder`)
- **command** (`List[str]`)

Return type

`str`

`cwltool.software_requirements.get_dependencies(builder)`

Parameters

builder (`cwltool.utils.HasReqHints`)

Return type

`galaxy.tool_util.deps.requirements.ToolRequirements`

`cwltool.software_requirements.get_container_from_software_requirements(use_biocontainers,
builder, con-
tainer_image_cache_path='.')`

Parameters

- **use_biocontainers** (`bool`)
- **builder** (`cwltool.utils.HasReqHints`)
- **container_image_cache_path** (`Optional[str]`)

Return type

`Optional[str]`

`cwltool.software_requirements.ensure_galaxy_lib_available()`

Return type

`None`

`cwltool.stdfsaccess`

Abstracted IO access.

Module Contents

Classes

<code>StdFsAccess</code>

Local filesystem implementation.

Functions

```
abspath(src, basedir)
```

```
cwltool.stdfsaccess.abspath(src, basedir)
```

Parameters

- **src** (*str*)
- **basedir** (*str*)

Return type

str

```
class cwltool.stdfsaccess.StdFsAccess(basedir)
```

Local filesystem implementation.

Parameters

basedir (*str*)

```
glob(pattern)
```

Parameters

pattern (*str*)

Return type

List[*str*]

```
open(fn, mode)
```

Parameters

- **fn** (*str*)
- **mode** (*str*)

Return type

IO[*Any*]

```
exists(fn)
```

Parameters

fn (*str*)

Return type

bool

```
size(fn)
```

Parameters

fn (*str*)

Return type

int

```
isfile(fn)
```

Parameters

fn (*str*)

Return type
`bool`

`isdir(fn)`

Parameters
`fn (str)`

Return type
`bool`

`listdir(fn)`

Parameters
`fn (str)`

Return type
`List[str]`

`join(path, *paths)`

- Parameters**
- `path (str)`
 - `paths (str)`

Return type
`str`

`realpath(path)`

Parameters
`path (str)`

Return type
`str`

`cwltool.subgraph`

Module Contents

Functions

<code>subgraph_visit(current, nodes, visited, direction)</code>	
<code>declare_node(nodes, nodeid, tp)</code>	
<code>find_step(steps, stepid, loading_context)</code>	Find the step (raw dictionary and WorkflowStep) for a given step id.
<code>get_subgraph(roots, tool, loading_context)</code>	Extract the subgraph for the given roots.
<code>get_step(tool, step_id, loading_context)</code>	Extract a single WorkflowStep for the given step_id.
<code>get_process(tool, step_id, loading_context)</code>	Find the underlying Process for a given Workflow step id.

Attributes

<i>Node</i>
<i>UP</i>
<i>DOWN</i>
<i>INPUT</i>
<i>OUTPUT</i>
<i>STEP</i>

`cwltool.subgraph.Node`

`cwltool.subgraph.UP = 'up'`

`cwltool.subgraph.DOWN = 'down'`

`cwltool.subgraph.INPUT = 'input'`

`cwltool.subgraph.OUTPUT = 'output'`

`cwltool.subgraph.STEP = 'step'`

`cwltool.subgraph.subgraph_visit(current, nodes, visited, direction)`

Parameters

- **current** (*str*)
- **nodes** (*MutableMapping[str, Node]*)
- **visited** (*Set[str]*)
- **direction** (*str*)

Return type

None

`cwltool.subgraph.declare_node(nodes, nodeid, tp)`

Parameters

- **nodes** (*Dict[str, Node]*)
- **nodeid** (*str*)
- **tp** (*Optional[str]*)

Return type

Node

`cwltool.subgraph.find_step(steps, stepid, loading_context)`

Find the step (raw dictionary and WorkflowStep) for a given step id.

Parameters

- **steps** (*List[cwltool.workflow.WorkflowStep]*)

- **stepid** (*str*)
- **loading_context** (`cwltool.context.LoadingContext`)

Return type

Tuple[Optional[`cwltool.utils.CWLObjectType`], Optional[`cwltool.workflow.WorkflowStep`]]

`cwltool.subgraph.get_subgraph`(*roots*, *tool*, *loading_context*)

Extract the subgraph for the given roots.

Parameters

- **roots** (*MutableSequence*[*str*])
- **tool** (`cwltool.workflow.Workflow`)
- **loading_context** (`cwltool.context.LoadingContext`)

Return type

`ruamel.yaml.comments.CommentedList`

`cwltool.subgraph.get_step`(*tool*, *step_id*, *loading_context*)

Extract a single `WorkflowStep` for the given *step_id*.

Parameters

- **tool** (`cwltool.workflow.Workflow`)
- **step_id** (*str*)
- **loading_context** (`cwltool.context.LoadingContext`)

Return type

`ruamel.yaml.comments.CommentedList`

`cwltool.subgraph.get_process`(*tool*, *step_id*, *loading_context*)

Find the underlying `Process` for a given `Workflow` step id.

Parameters

- **tool** (`cwltool.workflow.Workflow`)
- **step_id** (*str*)
- **loading_context** (`cwltool.context.LoadingContext`)

Return type

Tuple[Any, `cwltool.workflow.WorkflowStep`]

`cwltool.task_queue`

`TaskQueue`.

Module Contents

Classes

<i>TaskQueue</i>	A TaskQueue class.
------------------	--------------------

class `cwltool.task_queue.TaskQueue(lock, thread_count)`

A TaskQueue class.

Uses a first-in, first-out queue of tasks executed on a fixed number of threads.

New tasks enter the queue and are started in the order received, as worker threads become available.

If `thread_count == 0` then tasks will be synchronously executed when `add()` is called (this makes the actual task queue behavior a no-op, but may be a useful configuration knob).

The `thread_count` is also used as the maximum size of the queue.

The threads are created during TaskQueue initialization. Call `join()` when you're done with the TaskQueue and want the threads to stop.

Parameters

- **lock** (*threading.Lock*)
- **thread_count** (*int*)

in_flight: `int = 0`

The number of tasks in the queue.

add(*task, unlock=None, check_done=None*)

Add your task to the queue.

The optional `unlock` will be released prior to attempting to add the task to the queue.

If the optional “`check_done`” *threading.Event*’s flag is set, then we will skip adding this task to the queue.

If the TaskQueue was created with `thread_count == 0` then your task will be synchronously executed.

Parameters

- **task** (*Callable[[], None]*)
- **unlock** (*Optional[threading.Condition]*)
- **check_done** (*Optional[threading.Event]*)

Return type

None

drain()

Drain the queue.

Return type

None

join()

Wait for all threads to complete.

Return type

None

`cwltool.udocker`

Enables Docker software containers via the udocker runtime.

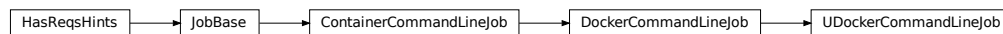
Module Contents

Classes

<code>UDockerCommandLineJob</code>	Runs a <code>CommandLineJob</code> in a software container using the udocker engine.
------------------------------------	--

```
class cwltool.udocker.UDockerCommandLineJob(builder, joborder, make_path_mapper, requirements, hints,
                                             name)
```

Bases: `cwltool.docker.DockerCommandLineJob`



Runs a `CommandLineJob` in a software container using the udocker engine.

Parameters

- **builder** (`cwltool.builder.Builder`)
- **joborder** (`cwltool.utils.CWLObjectType`)
- **make_path_mapper** (`Callable[[List[cwltool.utils.CWLObjectType], str, cwltool.context.RuntimeContext, bool], cwltool.pathmapper.PathMapper]`)
- **requirements** (`List[cwltool.utils.CWLObjectType]`)
- **hints** (`List[cwltool.utils.CWLObjectType]`)
- **name** (`str`)

```
static append_volume(runtime, source, target, writable=False, skip_mkdirs=False)
```

Add binding arguments to the runtime list.

Parameters

- **runtime** (`List[str]`)
- **source** (`str`)
- **target** (`str`)
- **writable** (`bool`)
- **skip_mkdirs** (`bool`)

Return type

None

cwltool.update**Module Contents****Functions**

<code>v1_1to1_2(doc, loader, baseuri)</code>	Public updater for v1.1 to v1.2.
<code>v1_0to1_1(doc, loader, baseuri)</code>	Public updater for v1.0 to v1.1.
<code>v1_1_0dev1to1_1(doc, loader, baseuri)</code>	Public updater for v1.1.0-dev1 to v1.1.
<code>v1_2_0dev1to1_2(doc, loader, baseuri)</code>	Public updater for v1.2.0-dev1 to v1.2.0-dev2.
<code>v1_2_0dev2to1_2(doc, loader, baseuri)</code>	Public updater for v1.2.0-dev2 to v1.2.0-dev3.
<code>v1_2_0dev3to1_2(doc, loader, baseuri)</code>	Public updater for v1.2.0-dev3 to v1.2.0-dev4.
<code>v1_2_0dev4to1_2(doc, loader, baseuri)</code>	Public updater for v1.2.0-dev4 to v1.2.0-dev5.
<code>v1_2_0dev5to1_2(doc, loader, baseuri)</code>	Public updater for v1.2.0-dev5 to v1.2.
<code>identity(doc, loader, baseuri)</code>	Do-nothing, CWL document upgrade function.
<code>checkversion(doc, metadata, enable_dev)</code>	Check the validity of the version of the give CWL document.
<code>update(doc, loader, baseuri, enable_dev, metadata[, ...])</code>	Update a CWL document to 'update_to' (if provided) or INTERNAL_VERSION.

Attributes

<code>ORDERED_VERSIONS</code>
<code>UPDATES</code>
<code>DEVUPDATES</code>
<code>ALLUPDATES</code>
<code>INTERNAL_VERSION</code>
<code>ORIGINAL_CWLVERSION</code>

`cwltool.update.v1_1to1_2(doc, loader, baseuri)`

Public updater for v1.1 to v1.2.

Parameters

- **doc** (`ruamel.yaml.comments.CommentMap`)
- **loader** (`schema_salad.ref_resolver.Loader`)
- **baseuri** (`str`)

Return type

`Tuple[ruamel.yaml.comments.CommentMap, str]`

`cwltool.update.v1_0to1_1(doc, loader, baseuri)`

Public updater for v1.0 to v1.1.

Parameters

- **doc** (*ruamel.yaml.comments.CommentedMap*)
- **loader** (*schema_salad.ref_resolver.Loader*)
- **baseuri** (*str*)

Return type

Tuple[ruamel.yaml.comments.CommentedMap, str]

`cwltool.update.v1_1_0dev1to1_1(doc, loader, baseuri)`

Public updater for v1.1.0-dev1 to v1.1.

Parameters

- **doc** (*ruamel.yaml.comments.CommentedMap*)
- **loader** (*schema_salad.ref_resolver.Loader*)
- **baseuri** (*str*)

Return type

Tuple[ruamel.yaml.comments.CommentedMap, str]

`cwltool.update.v1_2_0dev1todev2(doc, loader, baseuri)`

Public updater for v1.2.0-dev1 to v1.2.0-dev2.

Parameters

- **doc** (*ruamel.yaml.comments.CommentedMap*)
- **loader** (*schema_salad.ref_resolver.Loader*)
- **baseuri** (*str*)

Return type

Tuple[ruamel.yaml.comments.CommentedMap, str]

`cwltool.update.v1_2_0dev2todev3(doc, loader, baseuri)`

Public updater for v1.2.0-dev2 to v1.2.0-dev3.

Parameters

- **doc** (*ruamel.yaml.comments.CommentedMap*)
- **loader** (*schema_salad.ref_resolver.Loader*)
- **baseuri** (*str*)

Return type

Tuple[ruamel.yaml.comments.CommentedMap, str]

`cwltool.update.v1_2_0dev3todev4(doc, loader, baseuri)`

Public updater for v1.2.0-dev3 to v1.2.0-dev4.

Parameters

- **doc** (*ruamel.yaml.comments.CommentedMap*)
- **loader** (*schema_salad.ref_resolver.Loader*)
- **baseuri** (*str*)

Return type

Tuple[ruamel.yaml.comments.CommentedMap, str]

`cwltool.update.v1_2_0dev4todev5(doc, loader, baseuri)`

Public updater for v1.2.0-dev4 to v1.2.0-dev5.

Parameters

- **doc** (`ruamel.yaml.comments.CommentedList`)
- **loader** (`schema_salad.ref_resolver.Loader`)
- **baseuri** (`str`)

Return type

`Tuple[ruamel.yaml.comments.CommentedList, str]`

`cwltool.update.v1_2_0dev5to1_2(doc, loader, baseuri)`

Public updater for v1.2.0-dev5 to v1.2.

Parameters

- **doc** (`ruamel.yaml.comments.CommentedList`)
- **loader** (`schema_salad.ref_resolver.Loader`)
- **baseuri** (`str`)

Return type

`Tuple[ruamel.yaml.comments.CommentedList, str]`

`cwltool.update.ORDERED_VERSIONS = ['v1.0', 'v1.1.0-dev1', 'v1.1', 'v1.2.0-dev1', 'v1.2.0-dev2', 'v1.2.0-dev3', 'v1.2.0-dev4', ...]`

`cwltool.update.UPDATES: Dict[str, Callable[[ruamel.yaml.comments.CommentedList, schema_salad.ref_resolver.Loader, str], Tuple[ruamel.yaml.comments.CommentedList, str]] | None]`

`cwltool.update.DEVUPDATES: Dict[str, Callable[[ruamel.yaml.comments.CommentedList, schema_salad.ref_resolver.Loader, str], Tuple[ruamel.yaml.comments.CommentedList, str]] | None]`

`cwltool.update.ALLUPDATES`

`cwltool.update.INTERNAL_VERSION = 'v1.2'`

`cwltool.update.ORIGINAL_CWLVERSION = 'http://commonwl.org/cwltool#original_cwlVersion'`

`cwltool.update.identity(doc, loader, baseuri)`

Do-nothing, CWL document upgrade function.

Parameters

- **doc** (`ruamel.yaml.comments.CommentedList`)
- **loader** (`schema_salad.ref_resolver.Loader`)
- **baseuri** (`str`)

Return type

`Tuple[ruamel.yaml.comments.CommentedList, str]`

`cwltool.update.checkversion(doc, metadata, enable_dev)`

Check the validity of the version of the give CWL document.

Returns the document and the validated version string.

Parameters

- **doc** (*Union[ruamel.yaml.comments.CommentedSeq, ruamel.yaml.comments.CommentedMap]*)
- **metadata** (*ruamel.yaml.comments.CommentedMap*)
- **enable_dev** (*bool*)

Return type

Tuple[ruamel.yaml.comments.CommentedMap, str]

`cwltool.update.update(doc, loader, baseuri, enable_dev, metadata, update_to=None)`

Update a CWL document to 'update_to' (if provided) or INTERNAL_VERSION.

Parameters

- **doc** (*Union[ruamel.yaml.comments.CommentedSeq, ruamel.yaml.comments.CommentedMap]*)
- **loader** (*schema_salad.ref_resolver.Loader*)
- **baseuri** (*str*)
- **enable_dev** (*bool*)
- **metadata** (*ruamel.yaml.comments.CommentedMap*)
- **update_to** (*Optional[str]*)

Return type

ruamel.yaml.comments.CommentedMap

cwltool.utils

Shared functions and other definitions.

cwltool.validate_js**Module Contents****Classes**

SuppressLog

Filter instances are used to perform arbitrary filtering of LogRecords.

Functions

<code>is_expression(tool, schema)</code>	Test a field/schema combo to see if it is a CWL Expression.
<code>get_expressions(tool, schema[, source_line])</code>	
<code>jshint_js(js_text[, globals, options, ...])</code>	
<code>print_js_hint_messages(js_hint_messages, source_line)</code>	Log the message from JSHint, using the line number.
<code>validate_js_expressions(tool, schema[, ...])</code>	

Attributes

JSHintJSReturn

`cwltool.validate_js.is_expression(tool, schema)`
Test a field/schema combo to see if it is a CWL Expression.

Parameters

- **tool** (*Any*)
- **schema** (*Optional*[`schema_salad.avro.schema.Schema`])

Return type

`bool`

class `cwltool.validate_js.SuppressLog(name)`

Bases: `logging.Filter`



Filter instances are used to perform arbitrary filtering of LogRecords.

Loggers and Handlers can optionally use Filter instances to filter records as desired. The base filter class only allows events which are below a certain point in the logger hierarchy. For example, a filter initialized with “A.B” will allow events logged by loggers “A.B”, “A.B.C”, “A.B.C.D”, “A.B.D” etc. but not “A.BB”, “B.A.B” etc. If initialized with the empty string, all events are passed.

Parameters

name (*str*)

filter(*record*)

Never accept a record.

Parameters

record (*logging.LogRecord*)

Return type

bool

`cwltool.validate_js.get_expressions(tool, schema, source_line=None)`

Parameters

- **tool** (*Union[ruamel.yaml.comments.CommentedList, str, ruamel.yaml.comments.CommentedSeq]*)
- **schema** (*Optional[Union[schema_salad.avro.schema.Schema, schema_salad.avro.schema.ArraySchema]]*)
- **source_line** (*Optional[schema_salad.sourceline.SourceLine]*)

Return type

List[Tuple[str, Optional[schema_salad.sourceline.SourceLine]]]

`cwltool.validate_js.JSHintJSReturn`

`cwltool.validate_js.jshint_js(js_text, globals=None, options=None, container_engine='docker', eval_timeout=60)`

Parameters

- **js_text** (*str*)
- **globals** (*Optional[List[str]]*)
- **options** (*Optional[Dict[str, Union[List[str], str, int]]]*)
- **container_engine** (*str*)
- **eval_timeout** (*float*)

Return type

JSHintJSReturn

`cwltool.validate_js.print_js_hint_messages(js_hint_messages, source_line)`

Log the message from JSHint, using the line number.

Parameters

- **js_hint_messages** (*List[str]*)
- **source_line** (*Optional[schema_salad.sourceline.SourceLine]*)

Return type

None

`cwltool.validate_js.validate_js_expressions(tool, schema, jshint_options=None, container_engine='docker', eval_timeout=60)`

Parameters

- **tool** (*ruamel.yaml.comments.CommentedList*)
- **schema** (*schema_salad.avro.schema.Schema*)
- **jshint_options** (*Optional[Dict[str, Union[List[str], str, int]]]*)

- `container_engine` (*str*)
- `eval_timeout` (*float*)

Return type
None

`cwltool.workflow`

Module Contents

Classes

<i>Workflow</i>	Abstract CWL Process.
<i>WorkflowStep</i>	Abstract CWL Process.

Functions

<i>default_make_tool</i> (toolpath_object, loadingCon- text)	Instantiate the given CWL Process.
<i>used_by_step</i> (step, shortinputid)	

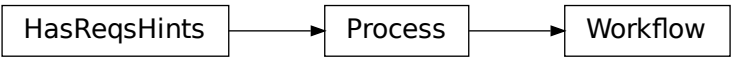
`cwltool.workflow.default_make_tool(toolpath_object, loadingContext)`
Instantiate the given CWL Process.

Parameters

- `toolpath_object` (*ruamel.yaml.comments.CommentedList*)
- `loadingContext` (*cwltool.context.LoadingContext*)

Return type
cwltool.process.Process

`class cwltool.workflow.Workflow(toolpath_object, loadingContext)`
Bases: *cwltool.process.Process*



Abstract CWL Process.

Parameters

- `toolpath_object` (*ruamel.yaml.comments.CommentedList*)

- **loadingContext** (`cwltool.context.LoadingContext`)

make_workflow_step(*toolpath_object*, *pos*, *loadingContext*, *parentworkflowProv=None*)

Parameters

- **toolpath_object** (`ruamel.yaml.comments.CommentedList`)
- **pos** (`int`)
- **loadingContext** (`cwltool.context.LoadingContext`)
- **parentworkflowProv** (`Optional[cwltool.cwlprov.provenance_profile.ProvenanceProfile]`)

Return type

`WorkflowStep`

job(*job_order*, *output_callbacks*, *runtimeContext*)

Parameters

- **job_order** (`cwltool.utils.CWLObjectType`)
- **output_callbacks** (`Optional[cwltool.utils.OutputCallbackType]`)
- **runtimeContext** (`cwltool.context.RuntimeContext`)

Return type

`cwltool.utils.JobsGeneratorType`

visit(*op*)

Parameters

op (`Callable[[ruamel.yaml.comments.CommentedList], None]`)

Return type

`None`

`cwltool.workflow.used_by_step`(*step*, *shortinputid*)

Parameters

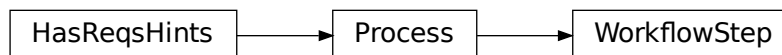
- **step** (`cwltool.utils.StepType`)
- **shortinputid** (`str`)

Return type

`bool`

class `cwltool.workflow.WorkflowStep`(*toolpath_object*, *pos*, *loadingContext*, *parentworkflowProv=None*)

Bases: `cwltool.process.Process`



Abstract CWL Process.

Parameters

- **toolpath_object** (*ruamel.yaml.comments.CommentedMap*)
- **pos** (*int*)
- **loadingContext** (*cwltool.context.LoadingContext*)
- **parentworkflowProv** (*Optional[cwltool.cwlprov.provenance_profile.ProvenanceProfile]*)

checkRequirements(*rec, supported_process_requirements*)

Check the presence of unsupported requirements.

Parameters

- **rec** (*Union[MutableSequence[cwltool.utils.CWLObjectType], cwltool.utils.CWLObjectType, cwltool.utils.CWLOutputType, None]*)
- **supported_process_requirements** (*Iterable[str]*)

Return type

None

receive_output(*output_callback, jobout, processStatus*)

Parameters

- **output_callback** (*cwltool.utils.OutputCallbackType*)
- **jobout** (*cwltool.utils.CWLObjectType*)
- **processStatus** (*str*)

Return type

None

job(*job_order, output_callbacks, runtimeContext*)

Initialize sub-workflow as a step in the parent profile.

Parameters

- **job_order** (*cwltool.utils.CWLObjectType*)
- **output_callbacks** (*Optional[cwltool.utils.OutputCallbackType]*)
- **runtimeContext** (*cwltool.context.RuntimeContext*)

Return type

cwltool.utils.JobsGeneratorType

visit(*op*)

Parameters

op (*Callable[[ruamel.yaml.comments.CommentedMap], None]*)

Return type

None

`cwltool.workflow_job`

Module Contents

Classes

<i>WorkflowJobStep</i>	Generated for each step in <code>Workflow.steps()</code> .
<i>ReceiveScatterOutput</i>	Produced by the scatter generators.
<i>WorkflowJob</i>	Generates steps from the Workflow.
<i>WorkflowJobLoopStep</i>	Generated for each step in <code>Workflow.steps()</code> containing a Loop requirement.

Functions

<i>parallel_steps</i> (steps, rc, runtimeContext)	
<i>nested_crossproduct_scatter</i> (process, joborder, ...)	
<i>crossproduct_size</i> (joborder, scatter_keys)	Compute the size of a cross product.
<i>flat_crossproduct_scatter</i> (process, joborder, ...)	
<i>dotproduct_scatter</i> (process, joborder, scatter_keys, ...)	
<i>match_types</i> (sinktype, src, iid, inputobj, linkMerge, ...)	
<i>object_from_state</i> (state, params, frag_only, ..., ...)	

class `cwltool.workflow_job.WorkflowJobStep`(step)

Generated for each step in `Workflow.steps()`.

Parameters

step (`cwltool.workflow.WorkflowStep`)

job(joborder, output_callback, runtimeContext)

Parameters

- **joborder** (`cwltool.utils.CWLObjectType`)
- **output_callback** (`Optional[cwltool.utils.OutputCallbackType]`)
- **runtimeContext** (`cwltool.context.RuntimeContext`)

Return type

`cwltool.utils.JobsGeneratorType`

class `cwltool.workflow_job.ReceiveScatterOutput`(output_callback, dest, total)

Produced by the scatter generators.

Parameters

- **output_callback** (`cwltool.utils.ScatterOutputCallbackType`)

- **dest** (*cwltool.utils.ScatterDestinationsType*)
- **total** (*int*)

receive_scatter_output(*index, jobout, processStatus*)

Record the results of a scatter operation.

Parameters

- **index** (*int*)
- **jobout** (*cwltool.utils.CWLObjectType*)
- **processStatus** (*str*)

Return type

None

setTotal(*total, steps*)

Set the total number of expected outputs along with the steps.

This is necessary to finish the setup.

Parameters

- **total** (*int*)
- **steps** (*List[Optional[cwltool.utils.JobsGeneratorType]]*)

Return type

None

cwltool.workflow_job.parallel_steps(*steps, rc, runtimeContext*)

Parameters

- **steps** (*List[Optional[cwltool.utils.JobsGeneratorType]]*)
- **rc** (*ReceiveScatterOutput*)
- **runtimeContext** (*cwltool.context.RuntimeContext*)

Return type

cwltool.utils.JobsGeneratorType

cwltool.workflow_job.nested_crossproduct_scatter(*process, joborder, scatter_keys, output_callback, runtimeContext*)

Parameters

- **process** (*WorkflowJobStep*)
- **joborder** (*cwltool.utils.CWLObjectType*)
- **scatter_keys** (*MutableSequence[str]*)
- **output_callback** (*cwltool.utils.ScatterOutputCallbackType*)
- **runtimeContext** (*cwltool.context.RuntimeContext*)

Return type

cwltool.utils.JobsGeneratorType

cwltool.workflow_job.crossproduct_size(*joborder, scatter_keys*)

Compute the size of a cross product.

Parameters

- **jobborder** (*cwltool.utils.CWLObjectType*)
- **scatter_keys** (*MutableSequence[[str](#)]*)

Return type

[int](#)

`cwltool.workflow_job.flat_crossproduct_scatter`(*process, jobborder, scatter_keys, output_callback, runtimeContext*)

Parameters

- **process** (*WorkflowJobStep*)
- **jobborder** (*cwltool.utils.CWLObjectType*)
- **scatter_keys** (*MutableSequence[[str](#)]*)
- **output_callback** (*cwltool.utils.ScatterOutputCallbackType*)
- **runtimeContext** (*cwltool.context.RuntimeContext*)

Return type

cwltool.utils.JobsGeneratorType

`cwltool.workflow_job.dotproduct_scatter`(*process, jobborder, scatter_keys, output_callback, runtimeContext*)

Parameters

- **process** (*WorkflowJobStep*)
- **jobborder** (*cwltool.utils.CWLObjectType*)
- **scatter_keys** (*MutableSequence[[str](#)]*)
- **output_callback** (*cwltool.utils.ScatterOutputCallbackType*)
- **runtimeContext** (*cwltool.context.RuntimeContext*)

Return type

cwltool.utils.JobsGeneratorType

`cwltool.workflow_job.match_types`(*sinktype, src, iid, inputobj, linkMerge, valueFrom*)

Parameters

- **sinktype** (*Optional[cwltool.utils.SinkType]*)
- **src** (*cwltool.utils.WorkflowStateItem*)
- **iid** ([str](#))
- **inputobj** (*cwltool.utils.CWLObjectType*)
- **linkMerge** (*Optional[[str](#)]*)
- **valueFrom** (*Optional[[str](#)]*)

Return type

[bool](#)

`cwltool.workflow_job.object_from_state`(*state, params, frag_only, supportsMultipleInput, sourceField, incomplete=False*)

Parameters

- **state** (*Dict[[str](#), Optional[cwltool.utils.WorkflowStateItem]]*)

- **params** (*cwltool.utils.ParametersType*)
- **frag_only** (*bool*)
- **supportsMultipleInput** (*bool*)
- **sourceField** (*str*)
- **incomplete** (*bool*)

Return type

Optional[cwltool.utils.CWLObjectType]

class *cwltool.workflow_job.WorkflowJob*(*workflow, runtimeContext*)

Generates steps from the Workflow.

Parameters

- **workflow** (*cwltool.workflow.Workflow*)
- **runtimeContext** (*cwltool.context.RuntimeContext*)

do_output_callback(*final_output_callback*)

Parameters

final_output_callback (*cwltool.utils.OutputCallbackType*)

Return type

None

receive_output(*step, outputparms, final_output_callback, jobout, processStatus*)

Parameters

- **step** (*WorkflowJobStep*)
- **outputparms** (*List[cwltool.utils.CWLObjectType]*)
- **final_output_callback** (*cwltool.utils.OutputCallbackType*)
- **jobout** (*cwltool.utils.CWLObjectType*)
- **processStatus** (*str*)

Return type

None

try_make_job(*step, final_output_callback, runtimeContext*)

Parameters

- **step** (*WorkflowJobStep*)
- **final_output_callback** (*Optional[cwltool.utils.OutputCallbackType]*)
- **runtimeContext** (*cwltool.context.RuntimeContext*)

Return type

cwltool.utils.JobsGeneratorType

run(*runtimeContext, tmpdir_lock=None*)

Log the start of each workflow.

Parameters

- **runtimeContext** (*cwltool.context.RuntimeContext*)
- **tmpdir_lock** (*Optional[threading.Lock]*)

Return type

None

job(*joborder*, *output_callback*, *runtimeContext*)**Parameters**

- **joborder** (*cwltool.utils.CWLObjectType*)
- **output_callback** (*Optional[cwltool.utils.OutputCallbackType]*)
- **runtimeContext** (*cwltool.context.RuntimeContext*)

Return type*cwltool.utils.JobsGeneratorType***class** *cwltool.workflow_job.WorkflowJobLoopStep*(*step*, *container_engine*)Generated for each step in *Workflow.steps()* containing a Loop requirement.**Parameters**

- **step** (*WorkflowJobStep*)
- **container_engine** (*str*)

job(*joborder*, *output_callback*, *runtimeContext*)Generate a *WorkflowJobStep* job until the *loopWhen* condition evaluates to False.**Parameters**

- **joborder** (*cwltool.utils.CWLObjectType*)
- **output_callback** (*cwltool.utils.OutputCallbackType*)
- **runtimeContext** (*cwltool.context.RuntimeContext*)

Return type*cwltool.utils.JobsGeneratorType***loop_callback**(*runtimeContext*, *jobout*, *processStatus*)Update the *joborder* object with output values from the last iteration.**Parameters**

- **runtimeContext** (*cwltool.context.RuntimeContext*)
- **jobout** (*cwltool.utils.CWLObjectType*)
- **processStatus** (*str*)

Return type

None

Package Contents*cwltool.__author__* = 'pamstutz@veritasgenetics.com'*cwltool.CWL_CONTENT_TYPES* = ['text/plain', 'application/json', 'text/vnd.yaml', 'text/yaml', 'text/x-yaml', ...]

3.5 Indices and tables

- [genindex](#)
- [modindex](#)
- [search](#)

PYTHON MODULE INDEX

C

- cwltool, 38
- cwltool.__main__, 49
- cwltool.argparser, 49
- cwltool.builder, 55
- cwltool.checker, 59
- cwltool.command_line_tool, 62
- cwltool.context, 69
- cwltool.cuda, 72
- cwltool.cwlprov, 38
- cwltool.cwlprov.provenance_constants, 39
- cwltool.cwlprov.provenance_profile, 40
- cwltool.cwlprov.ro, 43
- cwltool.cwlprov.writablebagfile, 45
- cwltool.cwlrdf, 72
- cwltool.cwlviewer, 74
- cwltool.docker, 75
- cwltool.docker_id, 78
- cwltool.env_to_stdout, 80
- cwltool.errors, 80
- cwltool.executors, 82
- cwltool.factory, 85
- cwltool.flatten, 87
- cwltool.job, 87
- cwltool.load_tool, 93
- cwltool.loghandler, 97
- cwltool.main, 98
- cwltool.mpi, 106
- cwltool.mutation, 107
- cwltool.pack, 109
- cwltool.pathmapper, 110
- cwltool.process, 113
- cwltool.procgenerator, 121
- cwltool.resolver, 123
- cwltool.run_job, 124
- cwltool.secrets, 125
- cwltool.singularity, 126
- cwltool.singularity_utils, 129
- cwltool.software_requirements, 130
- cwltool.stdfsaccess, 131
- cwltool.subgraph, 133
- cwltool.task_queue, 135
- cwltool.udocker, 137
- cwltool.update, 138
- cwltool.utils, 141
- cwltool.validate_js, 141
- cwltool.workflow, 144
- cwltool.workflow_job, 147

Symbols

- `__author__` (in module `cwltool`), 151
- `__call__()` (`cwltool.argparser.AppendAction` method), 54
- `__call__()` (`cwltool.argparser.FSAction` method), 51
- `__call__()` (`cwltool.argparser.FSAppendAction` method), 51
- `__call__()` (`cwltool.executors.JobExecutor` method), 82
- `__call__()` (`cwltool.factory.Callable` method), 86
- `__citation__` (in module `cwltool.cwlprov.provenance_constants`), 39
- `__contains__()` (`cwltool.pathmapper.PathMapper` method), 113
- `__iter__()` (`cwltool.pathmapper.PathMapper` method), 113
- `__repr__()` (`cwltool.job.JobBase` method), 89
- `__str__()` (`cwltool.cwlprov.provenance_profile.ProvenanceProfile` method), 40
- `__str__()` (`cwltool.cwlprov.ro.ResearchObject` method), 43
- `__str__()` (`cwltool.process.Process` method), 120
- `--add-ga4gh-tool-registry`
cwltool command line option, 23
- `--basedir`
cwltool command line option, 19
- `--cachedir`
cwltool command line option, 20
- `--cidfile-dir`
cwltool command line option, 19
- `--cidfile-prefix`
cwltool command line option, 20
- `--compute-checksum`
cwltool command line option, 23
- `--copy-outputs`
cwltool command line option, 20
- `--custom-net`
cwltool command line option, 23
- `--debug`
cwltool command line option, 22
- `--default-container`
cwltool command line option, 23
- `--disable-color`
cwltool command line option, 23
- `--disable-ga4gh-tool-registry`
cwltool command line option, 23
- `--disable-host-provenance`
cwltool command line option, 20
- `--disable-js-validation`
cwltool command line option, 22
- `--disable-pull`
cwltool command line option, 20
- `--disable-user-provenance`
cwltool command line option, 20
- `--doc-cache`
cwltool command line option, 21
- `--enable-color`
cwltool command line option, 23
- `--enable-dev`
cwltool command line option, 23
- `--enable-ext`
cwltool command line option, 23
- `--enable-ga4gh-tool-registry`
cwltool command line option, 23
- `--enable-host-provenance`
cwltool command line option, 20
- `--enable-pull`
cwltool command line option, 20
- `--enable-user-provenance`
cwltool command line option, 20
- `--eval-timeout`
cwltool command line option, 20
- `--force-docker-pull`
cwltool command line option, 23
- `--full-name`
cwltool command line option, 21
- `--help`
cwltool command line option, 19
- `--js-console`
cwltool command line option, 22
- `--js-hint-options-file`
cwltool command line option, 22
- `--leave-container`
cwltool command line option, 19
- `--leave-outputs`

cwltool command line option, 20	cwltool command line option, 21
--leave-tmpdir	--print-supported-versions
cwltool command line option, 20	cwltool command line option, 21
--log-dir	--print-targets
cwltool command line option, 19	cwltool command line option, 21
--make-template	--provenance
cwltool command line option, 21	cwltool command line option, 20
--move-outputs	--quiet
cwltool command line option, 20	cwltool command line option, 22
--mpi-config-file	--rdf-serializer
cwltool command line option, 24	cwltool command line option, 20
--no-compute-checksum	--relative-deps
cwltool command line option, 23	cwltool command line option, 22
--no-container	--relax-path-checks
cwltool command line option, 22	cwltool command line option, 23
--no-doc-cache	--rm-container
cwltool command line option, 21	cwltool command line option, 19
--no-match-user	--rm-tmpdir
cwltool command line option, 23	cwltool command line option, 20
--no-read-only	--single-process
cwltool command line option, 23	cwltool command line option, 24
--no-warnings	--single-step
cwltool command line option, 22	cwltool command line option, 23
--non-strict	--singularity
cwltool command line option, 21	cwltool command line option, 22
--on-error	--skip-schemas
cwltool command line option, 23	cwltool command line option, 21
--orcid	--strict
cwltool command line option, 21	cwltool command line option, 21
--outdir	--strict-cpu-limit
cwltool command line option, 19	cwltool command line option, 22
--overrides	--strict-memory-limit
cwltool command line option, 23	cwltool command line option, 22
--pack	--target
cwltool command line option, 21	cwltool command line option, 23
--parallel	--timestamps
cwltool command line option, 19	cwltool command line option, 22
--podman	--tmp-outdir-prefix
cwltool command line option, 22	cwltool command line option, 20
--preserve-entire-environment	--tmpdir-prefix
cwltool command line option, 19	cwltool command line option, 20
--preserve-environment	--tool-help
cwltool command line option, 19	cwltool command line option, 22
--print-deps	--udocker
cwltool command line option, 21	cwltool command line option, 22
--print-dot	--user-space-docker-cmd
cwltool command line option, 21	cwltool command line option, 22
--print-input-deps	--validate
cwltool command line option, 21	cwltool command line option, 21
--print-pre	--verbose
cwltool command line option, 21	cwltool command line option, 22
--print-rdf	--version
cwltool command line option, 21	cwltool command line option, 21
--print-subgraph	--write-summary

- cwltool command line option, 22
- h cwltool command line option, 19
- t cwltool command line option, 23
- w cwltool command line option, 22
- A**
 - abspath() (in module cwltool.stdfsaccess), 132
 - AbstractOperation (class in cwltool.command_line_tool), 65
 - ACCOUNT_UUID (in module cwltool.cwlprov.provenance_constants), 39
 - activity_has_provenance() (cwltool.cwlprov.provenance_profile.ProvenanceProfile method), 43
 - add() (cwltool.secrets.SecretStore method), 125
 - add() (cwltool.task_queue.TaskQueue method), 136
 - add_annotation() (cwltool.cwlprov.ro.ResearchObject method), 44
 - add_argument() (in module cwltool.argparser), 54
 - add_data_file() (cwltool.cwlprov.ro.ResearchObject method), 45
 - add_file_or_directory_volume() (cwltool.docker.DockerCommandLineJob method), 76
 - add_file_or_directory_volume() (cwltool.job.ContainerCommandLineJob method), 91
 - add_file_or_directory_volume() (cwltool.singularity.SingularityCommandLineJob method), 128
 - add_sizes() (in module cwltool.process), 117
 - add_tagfile() (cwltool.cwlprov.ro.ResearchObject method), 44
 - add_to_manifest() (cwltool.cwlprov.ro.ResearchObject method), 45
 - add_uri() (cwltool.cwlprov.ro.ResearchObject method), 44
 - add_volumes() (cwltool.job.ContainerCommandLineJob method), 92
 - add_writable_directory_volume() (cwltool.docker.DockerCommandLineJob method), 76
 - add_writable_directory_volume() (cwltool.job.ContainerCommandLineJob method), 92
 - add_writable_directory_volume() (cwltool.singularity.SingularityCommandLineJob method), 129
 - add_writable_file_volume() (cwltool.docker.DockerCommandLineJob method), 76
 - add_writable_file_volume() (cwltool.job.ContainerCommandLineJob method), 92
 - add_writable_file_volume() (cwltool.singularity.SingularityCommandLineJob method), 128
 - Aggregate (class in cwltool.cwlprov), 48
 - ALLUPDATES (in module cwltool.update), 140
 - Annotation (in module cwltool.cwlprov), 48
 - append_volume() (cwltool.docker.DockerCommandLineJob static method), 76
 - append_volume() (cwltool.job.ContainerCommandLineJob static method), 91
 - append_volume() (cwltool.singularity.SingularityCommandLineJob static method), 128
 - append_volume() (cwltool.udocker.UDockerCommandLineJob static method), 137
 - append_word_to_default_user_agent() (in module cwltool.main), 100
 - AppendAction (class in cwltool.argparser), 54
 - arg_parser() (in module cwltool.argparser), 50
 - ArgumentException, 81
 - AuthoredBy (class in cwltool.cwlprov), 48
 - avroize_type() (in module cwltool.process), 118
- B**
 - bind_input() (cwltool.builder.Builder method), 57
 - boot2docker_id() (in module cwltool.docker_id), 79
 - boot2docker_running() (in module cwltool.docker_id), 79
 - build_job_script() (cwltool.builder.Builder method), 57
 - build_job_script() (cwltool.software_requirements.DependenciesConfiguration method), 130
 - Builder (class in cwltool.builder), 56
 - bundledAs (cwltool.cwlprov.Aggregate attribute), 48
- C**
 - Callable (class in cwltool.factory), 86
 - CallbackJob (class in cwltool.command_line_tool), 66
 - can_assign_src_to_sink() (in module cwltool.checker), 60
 - check_adjust() (in module cwltool.command_line_tool), 66
 - check_all_types() (in module cwltool.checker), 60
 - check_output_and_strip() (in module cwltool.docker_id), 78
 - check_types() (in module cwltool.checker), 59

`check_valid_locations()` (in module `cwl-tool.command_line_tool`), 66
`check_working_directories()` (in module `cwl-tool.main`), 104
`checkRequirements()` (`cwltool.process.Process` method), 119
`checkRequirements()` (`cwltool.workflow.WorkflowStep` method), 146
`checksum_copy()` (in module `cwltool.cwlprov`), 49
`checkversion()` (in module `cwltool.update`), 140
`choose_process()` (in module `cwltool.main`), 104
`choose_step()` (in module `cwltool.main`), 104
`choose_target()` (in module `cwltool.main`), 104
`circular_dependency_checker()` (in module `cwl-tool.checker`), 61
`cleanIntermediate()` (in module `cwltool.process`), 117
`close()` (`cwltool.cwlprov.writablebagfile.WritableBagFile` method), 46
`close_ro()` (in module `cwltool.cwlprov.writablebagfile`), 47
`cmd_output_matches()` (in module `cwltool.docker_id`), 79
`cmd_output_to_int()` (in module `cwltool.docker_id`), 79
`collect_output()` (`cwl-tool.command_line_tool.CommandLineTool` method), 68
`collect_output_ports()` (`cwl-tool.command_line_tool.CommandLineTool` method), 68
`CollectOutputsType` (in module `cwltool.job`), 88
`COMMAND_WITH_DEPENDENCIES_TEMPLATE` (in module `cwltool.software_requirements`), 130
`CommandLineJob` (class in `cwltool.job`), 90
`CommandLineTool` (class in `cwl-tool.command_line_tool`), 67
`compute_checksums()` (in module `cwltool.process`), 121
`configure_logging()` (in module `cwltool.loghandler`), 97
`conformsTo` (`cwltool.cwlprov.Aggregate` attribute), 48
`CONTAINER_TMPDIR` (`cwl-tool.job.ContainerCommandLineJob` attribute), 91
`ContainerCommandLineJob` (class in `cwltool.job`), 90
`content_limit_respected_read()` (in module `cwl-tool.builder`), 56
`content_limit_respected_read_bytes()` (in module `cwltool.builder`), 56
`ContextBase` (class in `cwltool.context`), 69
`CONTROL_CODE_RE` (in module `cwltool.job`), 90
`copy()` (`cwltool.context.LoadingContext` method), 70
`copy()` (`cwltool.context.RuntimeContext` method), 71
`copy_job_order()` (in module `cwl-tool.cwlprov.provenance_profile`), 40
`create_file_and_add_volume()` (`cwl-tool.job.ContainerCommandLineJob` method), 92
`create_job()` (in module `cwl-tool.cwlprov.writablebagfile`), 47
`create_outdir()` (`cwltool.context.RuntimeContext` method), 71
`create_runtime()` (`cwl-tool.docker.DockerCommandLineJob` method), 77
`create_runtime()` (`cwl-tool.job.ContainerCommandLineJob` method), 91
`create_runtime()` (`cwl-tool.singularity.SingularityCommandLineJob` method), 129
`create_tmpdir()` (`cwltool.context.RuntimeContext` method), 71
`createdBy` (`cwltool.cwlprov.Aggregate` attribute), 48
`createdOn` (`cwltool.cwlprov.Aggregate` attribute), 48
`crossproduct_size()` (in module `cwl-tool.workflow_job`), 148
`cuda_check()` (in module `cwltool.cuda`), 72
`cuda_version_and_device_count()` (in module `cwl-tool.cuda`), 72
`custom_schemas` (in module `cwltool.process`), 116
`CWL_CONTENT_TYPES` (in module `cwltool`), 151
`cwl_document`
 `cwltool` command line option, 19
`cwl_files` (in module `cwltool.process`), 116
`CWL_IANA` (in module `cwltool.process`), 120
`CWLPROV` (in module `cwl-tool.cwlprov.provenance_constants`), 39
`CWLPROV_VERSION` (in module `cwl-tool.cwlprov.provenance_constants`), 39
`cwltool`
 module, 38
`cwltool` command line option
 `--add-ga4gh-tool-registry`, 23
 `--basedir`, 19
 `--cachedir`, 20
 `--cidfile-dir`, 19
 `--cidfile-prefix`, 20
 `--compute-checksum`, 23
 `--copy-outputs`, 20
 `--custom-net`, 23
 `--debug`, 22
 `--default-container`, 23
 `--disable-color`, 23
 `--disable-ga4gh-tool-registry`, 23
 `--disable-host-provenance`, 20
 `--disable-js-validation`, 22

```

--disable-pull, 20
--disable-user-provenance, 20
--doc-cache, 21
--enable-color, 23
--enable-dev, 23
--enable-ext, 23
--enable-ga4gh-tool-registry, 23
--enable-host-provenance, 20
--enable-pull, 20
--enable-user-provenance, 20
--eval-timeout, 20
--force-docker-pull, 23
--full-name, 21
--help, 19
--js-console, 22
--js-hint-options-file, 22
--leave-container, 19
--leave-outputs, 20
--leave-tmpdir, 20
--log-dir, 19
--make-template, 21
--move-outputs, 20
--mpi-config-file, 24
--no-compute-checksum, 23
--no-container, 22
--no-doc-cache, 21
--no-match-user, 23
--no-read-only, 23
--no-warnings, 22
--non-strict, 21
--on-error, 23
--orcid, 21
--outdir, 19
--overrides, 23
--pack, 21
--parallel, 19
--podman, 22
--preserve-entire-environment, 19
--preserve-environment, 19
--print-deps, 21
--print-dot, 21
--print-input-deps, 21
--print-pre, 21
--print-rdf, 21
--print-subgraph, 21
--print-supported-versions, 21
--print-targets, 21
--provenance, 20
--quiet, 22
--rdf-serializer, 20
--relative-deps, 22
--relax-path-checks, 23
--rm-container, 19
--rm-tmpdir, 20
--single-process, 24
--single-step, 23
--singularity, 22
--skip-schemas, 21
--strict, 21
--strict-cpu-limit, 22
--strict-memory-limit, 22
--target, 23
--timestamps, 22
--tmp-outdir-prefix, 20
--tmpdir-prefix, 20
--tool-help, 22
--udocker, 22
--user-space-docker-cmd, 22
--validate, 21
--verbose, 22
--version, 21
--write-summary, 22
-h, 19
-t, 23
-w, 22
cwl_document, 19
inputs_object, 19
cwltool.__main__
  module, 49
cwltool.argparser
  module, 49
cwltool.builder
  module, 55
cwltool.checker
  module, 59
cwltool.command_line_tool
  module, 62
cwltool.context
  module, 69
cwltool.cuda
  module, 72
cwltool.cwlprov
  module, 38
cwltool.cwlprov.provenance_constants
  module, 39
cwltool.cwlprov.provenance_profile
  module, 40
cwltool.cwlprov.ro
  module, 43
cwltool.cwlprov.writablebagfile
  module, 45
cwltool.cwlrdf
  module, 72
cwltool.cwlviewer
  module, 74
cwltool.docker
  module, 75
cwltool.docker_id

```

module, 78
cwltool.env_to_stdout
 module, 80
cwltool.errors
 module, 80
cwltool.executors
 module, 82
cwltool.factory
 module, 85
cwltool.flatten
 module, 87
cwltool.job
 module, 87
cwltool.load_tool
 module, 93
cwltool.loghandler
 module, 97
cwltool.main
 module, 98
cwltool.mpi
 module, 106
cwltool.mutation
 module, 107
cwltool.pack
 module, 109
cwltool.pathmapper
 module, 110
cwltool.process
 module, 113
cwltool.procgenerator
 module, 121
cwltool.resolver
 module, 123
cwltool.run_job
 module, 124
cwltool.secrets
 module, 125
cwltool.singularity
 module, 126
cwltool.singularity_utils
 module, 129
cwltool.software_requirements
 module, 130
cwltool.stdfsaccess
 module, 131
cwltool.subgraph
 module, 133
cwltool.task_queue
 module, 135
cwltool.udocker
 module, 137
cwltool.update
 module, 138
cwltool.utils

module, 141
cwltool.validate_js
 module, 141
cwltool.workflow
 module, 144
cwltool.workflow_job
 module, 147
CWLViewer (*class in cwltool.cwlviewer*), 74

D

DATA (*in module cwltool.cwlprov.provenance_constants*), 39
declare_artefact() (*cwltool.cwlprov.provenance_profile.ProvenanceProfile method*), 42
declare_directory() (*cwltool.cwlprov.provenance_profile.ProvenanceProfile method*), 42
declare_file() (*cwltool.cwlprov.provenance_profile.ProvenanceProfile method*), 41
declare_node() (*in module cwltool.subgraph*), 134
declare_string() (*cwltool.cwlprov.provenance_profile.ProvenanceProfile method*), 42
default_loader() (*in module cwltool.load_tool*), 94
default_make_tool (*in module cwltool.context*), 70
default_make_tool() (*in module cwltool.workflow*), 144
defaultStreamHandler (*in module cwltool.loghandler*), 97
DependenciesConfiguration (*class in cwltool.software_requirements*), 130
deserialize_env() (*in module cwltool.env_to_stdout*), 80
DEVUPDATES (*in module cwltool.update*), 140
DirectoryAction (*class in cwltool.argparser*), 52
DirectoryAppendAction (*class in cwltool.argparser*), 53
do_eval() (*cwltool.builder.Builder method*), 58
do_output_callback() (*cwltool.workflow_job.WorkflowJob method*), 150
docker_exe (*in module cwltool.main*), 100
docker_machine_id() (*in module cwltool.docker_id*), 79
docker_machine_name() (*in module cwltool.docker_id*), 78
docker_machine_running() (*in module cwltool.docker_id*), 79
docker_monitor() (*cwltool.job.ContainerCommandLineJob method*), 93
docker_vm_id() (*in module cwltool.docker_id*), 78

DockerCommandLineJob (class in *cwltool.docker*), 75
 docloaderctx (in module *cwltool.load_tool*), 94
 dot() (*cwltool.cwlviewer.CWLViewer* method), 74
 dot_with_parameters() (in module *cwltool.cwlrdf*), 73
 dot_without_parameters() (in module *cwltool.cwlrdf*), 73
 dotproduct_scatter() (in module *cwltool.workflow_job*), 149
 DOWN (in module *cwltool.subgraph*), 134
 drain() (*cwltool.task_queue.TaskQueue* method), 136

E

ENCODING (in module *cwltool.cwlprov.provenance_constants*), 39
 ensure_galaxy_lib_available() (in module *cwltool.software_requirements*), 131
 eval_resource() (in module *cwltool.process*), 118
 evalResources() (*cwltool.process.Process* method), 119
 evaluate() (*cwltool.cwlprov.provenance_profile.ProvenanceProfile* method), 41
 execute() (*cwltool.executors.JobExecutor* method), 83
 execute() (*cwltool.executors.NoopJobExecutor* method), 85
 exists() (*cwltool.stdfsaccess.StdFsAccess* method), 132
 ExpressionJob (class in *cwltool.command_line_tool*), 64
 ExpressionTool (class in *cwltool.command_line_tool*), 64

F

Factory (class in *cwltool.factory*), 86
 fast_parser() (in module *cwltool.load_tool*), 95
 fetch_document() (in module *cwltool.load_tool*), 95
 FILE_COUNT_WARNING (in module *cwltool.process*), 119
 FileAction (class in *cwltool.argparser*), 52
 FileAppendAction (class in *cwltool.argparser*), 53
 files() (*cwltool.pathmapper.PathMapper* method), 112
 fill_in_defaults() (in module *cwltool.process*), 117
 filter() (*cwltool.process.LogAsDebugFilter* method), 115
 filter() (*cwltool.validate_js.SuppressLog* method), 142
 finalize_prov_profile() (*cwltool.cwlprov.provenance_profile.ProvenanceProfile* method), 43
 find_default_container() (in module *cwltool.main*), 105
 find_deps() (in module *cwltool.main*), 102
 find_ids() (in module *cwltool.pack*), 109
 find_run() (in module *cwltool.pack*), 109
 find_step() (in module *cwltool.subgraph*), 134

flat_crossproduct_scatter() (in module *cwltool.workflow_job*), 149
 flatten() (in module *cwltool.flatten*), 87
 FOAF (in module *cwltool.cwlprov.provenance_constants*), 39
 FORCE_SHELLED_POPEN (in module *cwltool.job*), 88
 formatTime() (*cwltool.main.ProvLogFormatter* method), 103
 FSAction (class in *cwltool.argparser*), 50
 FSAppendAction (class in *cwltool.argparser*), 51

G

ga4gh_tool_registries (in module *cwltool.resolver*), 123
 GA4GH_TRS_FILES (in module *cwltool.resolver*), 123
 GA4GH_TRS_PRIMARY_DESCRIPTOR (in module *cwltool.resolver*), 123
 gather() (in module *cwltool.cwlrdf*), 73
 generate_arg() (*cwltool.builder.Builder* method), 58
 generate_example_input() (in module *cwltool.main*),
 generate_input_template() (in module *cwltool.main*), 100
 generate_output_prov() (*cwltool.cwlprov.provenance_profile.ProvenanceProfile* method), 42
 generate_parser() (in module *cwltool.argparser*), 55
 generate_prov_doc() (*cwltool.cwlprov.provenance_profile.ProvenanceProfile* method), 40
 generate_snapshot() (*cwltool.cwlprov.ro.ResearchObject* method), 44
 get_container_from_software_requirements() (in module *cwltool.software_requirements*), 131
 get_default_args() (in module *cwltool.argparser*), 50
 get_dependencies() (in module *cwltool.software_requirements*), 131
 get_dependency_tree() (in module *cwltool.checker*), 61
 get_dot_graph() (*cwltool.cwlviewer.CWLViewer* method), 74
 get_expressions() (in module *cwltool.validate_js*), 143
 get_from_requirements() (*cwltool.docker.DockerCommandLineJob* method), 76
 get_from_requirements() (*cwltool.job.ContainerCommandLineJob* method), 91
 get_from_requirements() (*cwltool.singularity.SingularityCommandLineJob* method), 128

- `get_image()` (*cwltool.docker.DockerCommandLineJob method*), 75
- `get_image()` (*cwltool.singularity.SingularityCommandLineJob static method*), 127
- `get_outdir()` (*cwltool.context.RuntimeContext method*), 71
- `get_overrides()` (*in module cwltool.process*), 118
- `get_process()` (*in module cwltool.subgraph*), 135
- `get_schema()` (*in module cwltool.process*), 116
- `get_stagedir()` (*cwltool.context.RuntimeContext method*), 71
- `get_step()` (*in module cwltool.subgraph*), 135
- `get_step_id()` (*in module cwltool.checker*), 61
- `get_subgraph()` (*in module cwltool.subgraph*), 135
- `get_tmpdir()` (*cwltool.context.RuntimeContext method*), 71
- `get_version()` (*in module cwltool.singularity*), 126
- `getdefault()` (*in module cwltool.context*), 71
- `glob()` (*cwltool.stdfsaccess.StdFsAccess method*), 132
- `GraphTargetMissingException`, 81
- ## H
- `handle_software_environment()` (*in module cwltool.run_job*), 124
- `has_data_file()` (*cwltool.cwlprov.ro.ResearchObject method*), 44
- `has_secret()` (*cwltool.secrets.SecretStore method*), 125
- `Hasher` (*in module cwltool.cwlprov.provenance_constants*), 39
- ## I
- `identity()` (*in module cwltool.update*), 140
- `import_embed()` (*in module cwltool.pack*), 110
- `in_flight` (*cwltool.task_queue.TaskQueue attribute*), 136
- `inherit_reqshints()` (*in module cwltool.main*), 104
- `init_job_order()` (*in module cwltool.main*), 101
- `INPUT` (*in module cwltool.subgraph*), 134
- `INPUT_OBJ_VOCAB` (*in module cwltool.builder*), 56
- `inputs_object`
cwltool command line option, 19
- `INTERNAL_VERSION` (*in module cwltool.update*), 140
- `is_all_output_method_loop_step()` (*in module cwltool.checker*), 62
- `is_apptainer_1_or_newer()` (*in module cwltool.singularity*), 126
- `is_conditional_step()` (*in module cwltool.checker*), 62
- `is_expression()` (*in module cwltool.validate_js*), 142
- `is_version_2_6()` (*in module cwltool.singularity*), 126
- `is_version_3_1_or_newer()` (*in module cwltool.singularity*), 127
- `is_version_3_4_or_newer()` (*in module cwltool.singularity*), 127
- `is_version_3_9_or_newer()` (*in module cwltool.singularity*), 127
- `is_version_3_or_newer()` (*in module cwltool.singularity*), 126
- `isdir()` (*cwltool.stdfsaccess.StdFsAccess method*), 133
- `isfile()` (*cwltool.stdfsaccess.StdFsAccess method*), 132
- `items()` (*cwltool.pathmapper.PathMapper method*), 112
- `items_exclude_children()` (*cwltool.pathmapper.PathMapper method*), 112
- ## J
- `job()` (*cwltool.command_line_tool.AbstractOperation method*), 65
- `job()` (*cwltool.command_line_tool.CommandLineTool method*), 68
- `job()` (*cwltool.command_line_tool.ExpressionTool method*), 65
- `job()` (*cwltool.process.Process method*), 120
- `job()` (*cwltool.procgenerator.ProcessGenerator method*), 122
- `job()` (*cwltool.procgenerator.ProcessGeneratorJob method*), 122
- `job()` (*cwltool.workflow.Workflow method*), 145
- `job()` (*cwltool.workflow.WorkflowStep method*), 146
- `job()` (*cwltool.workflow_job.WorkflowJob method*), 151
- `job()` (*cwltool.workflow_job.WorkflowJobLoopStep method*), 151
- `job()` (*cwltool.workflow_job.WorkflowJobStep method*), 147
- `JobBase` (*class in cwltool.job*), 88
- `JobExecutor` (*class in cwltool.executors*), 82
- `jobloader_id_name` (*in module cwltool.load_tool*), 94
- `jobloaderctx` (*in module cwltool.load_tool*), 94
- `join()` (*cwltool.stdfsaccess.StdFsAccess method*), 133
- `join()` (*cwltool.task_queue.TaskQueue method*), 136
- `jshint_js()` (*in module cwltool.validate_js*), 143
- `JSHintJSReturn` (*in module cwltool.validate_js*), 143
- ## L
- `lastpart()` (*in module cwltool.cwlrdf*), 73
- `listdir()` (*cwltool.stdfsaccess.StdFsAccess method*), 133
- `load()` (*cwltool.mpi.MpiConfig class method*), 107
- `load_job_order()` (*in module cwltool.main*), 100
- `load_overrides()` (*in module cwltool.load_tool*), 96
- `load_tool()` (*in module cwltool.load_tool*), 96
- `loading_context` (*cwltool.factory.Factory attribute*), 86
- `LoadingContext` (*class in cwltool.context*), 70
- `LoadRefType` (*in module cwltool.pack*), 109
- `log_handler()` (*in module cwltool.context*), 70
- `LogAsDebugFilter` (*class in cwltool.process*), 115
- `LOGS` (*in module cwltool.cwlprov.provenance_constants*), 39

`loop_callback()` (*cwltool.workflow_job.WorkflowJobLoopStep method*), 151
`loop_checker()` (*in module cwltool.checker*), 62

M

`MAIN` (*in module cwltool.cwlprov.provenance_constants*), 39
`main()` (*in module cwltool.env_to_stdout*), 80
`main()` (*in module cwltool.main*), 105
`main()` (*in module cwltool.run_job*), 124
`make()` (*cwltool.factory.Factory method*), 86
`make_job_runner()` (*cwltool.command_line_tool.CommandLineTool method*), 67
`make_path_mapper()` (*cwltool.command_line_tool.CommandLineTool static method*), 67
`make_relative()` (*in module cwltool.main*), 101
`make_template()` (*in module cwltool.main*), 103
`make_tool()` (*in module cwltool.load_tool*), 96
`make_tool_notimpl()` (*in module cwltool.context*), 69
`make_workflow_step()` (*cwltool.workflow.Workflow method*), 145
`mapper()` (*cwltool.pathmapper.PathMapper method*), 112
`MapperEnt` (*in module cwltool.pathmapper*), 111
`match_types()` (*in module cwltool.workflow_job*), 149
`mediatype` (*cwltool.cwlprov.Aggregate attribute*), 48
`merge_flatten_type()` (*in module cwltool.checker*), 60
`mergedirs()` (*in module cwltool.process*), 120
`METADATA` (*in module cwltool.cwlprov.provenance_constants*), 39
`missing_subset()` (*in module cwltool.checker*), 60
`module`
 `cwltool`, 38
 `cwltool.__main__`, 49
 `cwltool.argparser`, 49
 `cwltool.builder`, 55
 `cwltool.checker`, 59
 `cwltool.command_line_tool`, 62
 `cwltool.context`, 69
 `cwltool.cuda`, 72
 `cwltool.cwlprov`, 38
 `cwltool.cwlprov.provenance_constants`, 39
 `cwltool.cwlprov.provenance_profile`, 40
 `cwltool.cwlprov.ro`, 43
 `cwltool.cwlprov.writablebagfile`, 45
 `cwltool.cwlrdf`, 72
 `cwltool.cwlviewer`, 74
 `cwltool.docker`, 75
 `cwltool.docker_id`, 78
 `cwltool.env_to_stdout`, 80

`cwltool.errors`, 80
 `cwltool.executors`, 82
 `cwltool.factory`, 85
 `cwltool.flatten`, 87
 `cwltool.job`, 87
 `cwltool.load_tool`, 93
 `cwltool.loghandler`, 97
 `cwltool.main`, 98
 `cwltool.mpi`, 106
 `cwltool.mutation`, 107
 `cwltool.pack`, 109
 `cwltool.pathmapper`, 110
 `cwltool.process`, 113
 `cwltool.procgenerator`, 121
 `cwltool.resolver`, 123
 `cwltool.run_job`, 124
 `cwltool.secrets`, 125
 `cwltool.singularity`, 126
 `cwltool.singularity_utils`, 129
 `cwltool.software_requirements`, 130
 `cwltool.stdfsaccess`, 131
 `cwltool.subgraph`, 133
 `cwltool.task_queue`, 135
 `cwltool.udocker`, 137
 `cwltool.update`, 138
 `cwltool.utils`, 141
 `cwltool.validate_js`, 141
 `cwltool.workflow`, 144
 `cwltool.workflow_job`, 147
`MpiConfig` (*class in cwltool.mpi*), 106
`MpiConfigT` (*in module cwltool.mpi*), 106
`MPIRequirementName` (*in module cwltool.mpi*), 106
`MultithreadedJobExecutor` (*class in cwltool.executors*), 83
`MutationManager` (*class in cwltool.mutation*), 107
`MutationState` (*in module cwltool.mutation*), 107

N

`name` (*cwltool.cwlprov.AuthoredBy attribute*), 49
`needs_shell_quoting_re` (*in module cwltool.job*), 88
`nestdir()` (*in module cwltool.process*), 120
`nested_crossproduct_scatter()` (*in module cwltool.workflow_job*), 148
`neverquote()` (*in module cwltool.job*), 88
`Node` (*in module cwltool.subgraph*), 134
`NoopJobExecutor` (*class in cwltool.executors*), 84

O

`objclass` (*cwltool.argparser.DirectoryAction attribute*), 53
`objclass` (*cwltool.argparser.DirectoryAppendAction attribute*), 54
`objclass` (*cwltool.argparser.FileAction attribute*), 52

- objclass (*cwltool.argparser.FileAppendAction* attribute), 53
 objclass (*cwltool.argparser.FSAction* attribute), 51
 objclass (*cwltool.argparser.FSAppendAction* attribute), 51
 object_from_state() (in module *cwltool.workflow_job*), 149
 open() (*cwltool.stdfsaccess.StdFsAccess* method), 132
 open_log_file_for_activity() (in module *cwltool.cwlprov.writablebagfile*), 47
 orcid (*cwltool.cwlprov.AuthoredBy* attribute), 49
 ORCID (in module *cwltool.cwlprov.provenance_constants*), 39
 ORDERED_VERSIONS (in module *cwltool.update*), 140
 ORE (in module *cwltool.cwlprov.provenance_constants*), 39
 ORIGINAL_CWLVERSION (in module *cwltool.update*), 140
 outdir (*cwltool.context.RuntimeContext* attribute), 71
 OUTPUT (in module *cwltool.subgraph*), 134
 output_callback() (*cwltool.executors.JobExecutor* method), 82
 OutputPortsType (in module *cwltool.command_line_tool*), 66
 overrides_ctx (in module *cwltool.load_tool*), 94
- ## P
- pack() (in module *cwltool.pack*), 110
 packed_workflow() (in module *cwltool.cwlprov.writablebagfile*), 47
 parallel_steps() (in module *cwltool.workflow_job*), 148
 ParameterOutputWorkflowException, 66
 pass_through_env_vars() (*cwltool.mpi.MpiConfig* method), 107
 PathCheckingMode (class in *cwltool.command_line_tool*), 63
 PathMapper (class in *cwltool.pathmapper*), 111
 PodmanCommandLineJob (class in *cwltool.docker*), 77
 prepare_environment() (*cwltool.job.JobBase* method), 89
 print_js_hint_messages() (in module *cwltool.validate_js*), 143
 print_pack() (in module *cwltool.main*), 102
 print_targets() (in module *cwltool.main*), 104
 printdeps() (in module *cwltool.main*), 101
 printdot() (in module *cwltool.cwlrdf*), 74
 printtrdf() (in module *cwltool.cwlrdf*), 73
 Process (class in *cwltool.process*), 119
 process_monitor() (*cwltool.job.JobBase* method), 89
 processDFS() (in module *cwltool.checker*), 61
 ProcessGenerator (class in *cwltool.procgenerator*), 122
 ProcessGeneratorJob (class in *cwltool.procgenerator*), 121
 prospective_prov() (*cwltool.cwlprov.provenance_profile.ProvenanceProfile* method), 42
 prov_deps() (in module *cwltool.main*), 102
 PROVENANCE (in module *cwltool.cwlprov.provenance_constants*), 39
 ProvenanceProfile (class in *cwltool.cwlprov.provenance_profile*), 40
 ProvLogFormatter (class in *cwltool.main*), 102
 ProvOut (in module *cwltool.main*), 103
- ## R
- readable() (*cwltool.cwlprov.writablebagfile.WritableBagFile* method), 46
 realize_input_schema() (in module *cwltool.main*), 100
 realpath() (*cwltool.stdfsaccess.StdFsAccess* method), 133
 receive_output() (*cwltool.procgenerator.ProcessGeneratorJob* method), 121
 receive_output() (*cwltool.workflow.WorkflowStep* method), 146
 receive_output() (*cwltool.workflow_job.WorkflowJob* method), 150
 receive_scatter_output() (*cwltool.workflow_job.ReceiveScatterOutput* method), 148
 ReceiveScatterOutput (class in *cwltool.workflow_job*), 147
 record_process_end() (*cwltool.cwlprov.provenance_profile.ProvenanceProfile* method), 41
 record_process_start() (*cwltool.cwlprov.provenance_profile.ProvenanceProfile* method), 41
 recursive_resolve_and_validate_document() (in module *cwltool.load_tool*), 96
 register_mutation() (*cwltool.mutation.MutationManager* method), 108
 register_reader() (*cwltool.mutation.MutationManager* method), 108
 RELAXED (*cwltool.command_line_tool.PathCheckingMode* attribute), 64
 release_reader() (*cwltool.mutation.MutationManager* method), 108
 relink_initialworkdir() (in module *cwltool.job*), 88
 relocateOutputs() (in module *cwltool.process*), 117
 remove_path() (in module *cwltool.command_line_tool*), 65
 replace_refs() (in module *cwltool.pack*), 109

- ResearchObject (class in *cwltool.cwlprov.ro*), 43
 resolve_and_validate_document() (in module *cwltool.load_tool*), 95
 resolve_ga4gh_tool() (in module *cwltool.resolver*), 123
 resolve_local() (in module *cwltool.resolver*), 123
 resolve_overrides() (in module *cwltool.load_tool*), 96
 resolve_tool_uri() (in module *cwltool.load_tool*), 95
 resolved (in module *cwltool.pathmapper*), 111
 result() (*cwltool.procgenerator.ProcessGenerator* method), 122
 retrieve() (*cwltool.secrets.SecretStore* method), 125
 reversemap() (*cwltool.pathmapper.PathMapper* method), 112
 revmap_file() (in module *cwltool.command_line_tool*), 65
 RO (in module *cwltool.cwlprov.provenance_constants*), 39
 run() (*cwltool.command_line_tool.CallbackJob* method), 66
 run() (*cwltool.command_line_tool.ExpressionJob* method), 64
 run() (*cwltool.job.CommandLineJob* method), 90
 run() (*cwltool.job.ContainerCommandLineJob* method), 93
 run() (*cwltool.job.JobBase* method), 89
 run() (*cwltool.workflow_job.WorkflowJob* method), 150
 run() (in module *cwltool.main*), 106
 run_job() (*cwltool.executors.MultithreadedJobExecutor* method), 84
 run_jobs() (*cwltool.executors.JobExecutor* method), 82
 run_jobs() (*cwltool.executors.MultithreadedJobExecutor* method), 84
 run_jobs() (*cwltool.executors.NoopJobExecutor* method), 85
 run_jobs() (*cwltool.executors.SingleJobExecutor* method), 83
 runtime_context (*cwltool.factory.Factory* attribute), 86
 RuntimeContext (class in *cwltool.context*), 70
- ## S
- salad_files (in module *cwltool.process*), 116
 scandeps() (in module *cwltool.process*), 120
 SCHEMA (in module *cwltool.cwlprov.provenance_constants*), 39
 SCHEMA_ANY (in module *cwltool.process*), 116
 SCHEMA_CACHE (in module *cwltool.process*), 116
 SCHEMA_DIR (in module *cwltool.process*), 116
 SCHEMA_FILE (in module *cwltool.process*), 116
 SecretStore (class in *cwltool.secrets*), 125
 seekable() (*cwltool.cwlprov.writablebagfile.WritableBagFile* method), 46
 select_resources() (*cwltool.executors.MultithreadedJobExecutor* method), 84
 self_check() (*cwltool.cwlprov.ro.ResearchObject* method), 43
 set_env_vars() (*cwltool.mpi.MpiConfig* method), 107
 set_generation() (*cwltool.mutation.MutationManager* method), 108
 set_log_dir() (in module *cwltool.context*), 70
 setTotal() (*cwltool.workflow_job.ReceiveScatterOutput* method), 148
 setup() (*cwltool.pathmapper.PathMapper* method), 112
 setup_loadingContext() (in module *cwltool.main*), 103
 setup_provenance() (in module *cwltool.main*), 103
 setup_schema() (in module *cwltool.main*), 102
 SHA1 (in module *cwltool.cwlprov.provenance_constants*), 39
 SHA256 (in module *cwltool.cwlprov.provenance_constants*), 39
 SHA512 (in module *cwltool.cwlprov.provenance_constants*), 39
 SHELL_COMMAND_TEMPLATE (in module *cwltool.job*), 88
 shortname() (in module *cwltool.process*), 116
 SingleJobExecutor (class in *cwltool.executors*), 83
 singularity_supports_userns() (in module *cwltool.singularity_utils*), 129
 SingularityCommandLineJob (class in *cwltool.singularity*), 127
 size() (*cwltool.stdfsaccess.StdFsAccess* method), 132
 SNAPSHOT (in module *cwltool.cwlprov.provenance_constants*), 39
 SOFTWARE_REQUIREMENTS_ENABLED (in module *cwltool.software_requirements*), 130
 SrcSink (in module *cwltool.checker*), 60
 stage_files() (in module *cwltool.process*), 116
 staged (in module *cwltool.pathmapper*), 111
 stagedir (*cwltool.context.RuntimeContext* attribute), 71
 start_process() (*cwltool.cwlprov.provenance_profile.ProvenanceProfile* method), 41
 static_checker() (in module *cwltool.checker*), 60
 StdFsAccess (class in *cwltool.stdfsaccess*), 132
 STEP (in module *cwltool.subgraph*), 134
 store() (*cwltool.secrets.SecretStore* method), 125
 STRICT (*cwltool.command_line_tool.PathCheckingMode* attribute), 63
 subgraph_visit() (in module *cwltool.subgraph*), 134
 substitute() (in module *cwltool.builder*), 56
 supported_cwl_versions() (in module *cwltool.main*), 102
 supportedProcessRequirements (in module *cwltool.process*), 116

SuppressLog (class in *cwltool.validate_js*), 142

T

target (in module *cwltool.pathmapper*), 111

TaskQueue (class in *cwltool.task_queue*), 136

TEXT_PLAIN (in module *cwltool.cwlprov.provenance_constants*), 39

tmp_outdir_prefix (*cwltool.context.RuntimeContext* attribute), 71

tmpdir (*cwltool.context.RuntimeContext* attribute), 71

TMPDIR_LOCK (in module *cwltool.executors*), 82

tmpdir_prefix (*cwltool.context.RuntimeContext* attribute), 71

tool_resolver() (in module *cwltool.resolver*), 123

ToolRequirement (in module *cwltool.software_requirements*), 130

tostr() (*cwltool.builder.Builder* method), 58

truncate() (*cwltool.cwlprov.writablebagfile.WritableBagFile* method), 46

try_make_job() (*cwltool.workflow_job.WorkflowJob* method), 150

type (in module *cwltool.pathmapper*), 111

U

UDockerCommandLineJob (class in *cwltool.udocker*), 137

unique_name() (in module *cwltool.process*), 120

unset_generation() (*cwltool.mutation.MutationManager* method), 108

UnsupportedRequirement, 80

UP (in module *cwltool.subgraph*), 134

update() (*cwltool.pathmapper.PathMapper* method), 113

update() (in module *cwltool.update*), 141

update_index() (in module *cwltool.load_tool*), 95

updatePathmap() (*cwltool.command_line_tool.CommandLineTool* method), 68

UPDATES (in module *cwltool.update*), 140

uri (*cwltool.cwlprov.Aggregate* attribute), 48

uri (*cwltool.cwlprov.AuthoredBy* attribute), 49

use_custom_schema() (in module *cwltool.process*), 116

use_standard_schema() (in module *cwltool.process*), 116

used_artefacts() (*cwltool.cwlprov.provenance_profile.ProvenanceProfile* method), 42

used_by_step() (in module *cwltool.workflow*), 145

user_provenance() (*cwltool.cwlprov.ro.ResearchObject* method), 44

USER_UUID (in module *cwltool.cwlprov.provenance_constants*), 39

UUID (in module *cwltool.cwlprov.provenance_constants*), 39

V

v1_0to1_1() (in module *cwltool.update*), 138

v1_1_0dev1to1_1() (in module *cwltool.update*), 139

v1_1to1_2() (in module *cwltool.update*), 138

v1_2_0dev1to2dev2() (in module *cwltool.update*), 139

v1_2_0dev2to2dev3() (in module *cwltool.update*), 139

v1_2_0dev3to2dev4() (in module *cwltool.update*), 139

v1_2_0dev4to2dev5() (in module *cwltool.update*), 139

v1_2_0dev5to1_2() (in module *cwltool.update*), 140

validate_hints() (*cwltool.process.Process* method), 119

validate_js_expressions() (in module *cwltool.validate_js*), 143

var_spool_cwl_detector() (in module *cwltool.process*), 118

visit() (*cwltool.pathmapper.PathMapper* method), 112

visit() (*cwltool.process.Process* method), 119

visit() (*cwltool.workflow.Workflow* method), 145

visit() (*cwltool.workflow.WorkflowStep* method), 146

visitlisting() (*cwltool.pathmapper.PathMapper* method), 111

W

wait_for_next_completion() (*cwltool.executors.MultithreadedJobExecutor* method), 84

WF4EVER (in module *cwltool.cwlprov.provenance_constants*), 39

WFDESC (in module *cwltool.cwlprov.provenance_constants*), 39

WFPROV (in module *cwltool.cwlprov.provenance_constants*), 39

windows_check() (in module *cwltool.main*), 106

Workflow (class in *cwltool.workflow*), 144

WORKFLOW (in module *cwltool.cwlprov.provenance_constants*), 39

WorkflowException, 80

WorkflowJob (class in *cwltool.workflow_job*), 150

WorkflowJobLoopStep (class in *cwltool.workflow_job*), 151

WorkflowJobStep (class in *cwltool.workflow_job*), 147

WorkflowStatus, 85

WorkflowStep (class in *cwltool.workflow*), 145

WritableBagFile (class in *cwltool.cwlprov.writablebagfile*), 45

write() (*cwltool.cwlprov.writablebagfile.WritableBagFile* method), 46

write_bag_file() (in module *cwltool.cwlprov.writablebagfile*), 46