Common Workflow Language reference implementation

Release 3.1.20220502060230

Peter Amstutz and contributors

May 02, 2022

CONTENTS:

This is the reference implementation of the Common Workflow Language. 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.

CHAPTER

ONE

CWLTOOL COMMAND LINE OPTIONS

1.1 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 | --quiet | --debug]
               [--strict-memory-limit] [--strict-cpu-limit] [--timestamps]
               [--js-console] [--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]
```

(continues on next page)

(continued from previous page)

```
               [--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

[experimental] 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 20s.

--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

--quiet
: Only print warnings and errors.

--debug
: Print even more logging

--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

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

--podman

[experimental] 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.

CHAPTER

TWO

MODULES

2.1 Python version support policy

Cwltool will always support stable Python 3 releases with active branches.

For versions that are no longer supported by Python upstream, cwltool support also extends to the default Python version included in the following major Linux distributions:

- Debian (stable, oldstable)
- Ubuntu (LTS release standard support)
- Centos 7 (while in maintenance)

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 Feburary 2022, here are approximate cwltool support periods for across Python versions:

Python	cwltool end of support
2.7	ended January 2020
3.5	ended October 2020
3.6	June 2024 (Centos 7 EOL)
3.7	June 2023 (upstream EOL)
3.8	April 2025 (Ubuntu 20.04 EOL)
3.9	October 2025 (upstream EOL)
3.10	October 2026 (upstream EOL)

Default Python version of supported Linux distributions, for reference (as of Feburary 2022)

Python	Linux distros where it is the default version
3.6	Ubuntu 18.04, Centos 7
3.7	Debian 10
3.8	Ubuntu 20.04
3.9	Debian 11
3.10	None

2.2 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
```

(continues on next page)

(continued from previous page)

```
  requirements:
    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.

2.2.1 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
```

2.3 API Reference

This page contains auto-generated API reference documentation[1].

2.3.1 cwltool

Reference implementation of the CWL standards.

Submodules

cwltool.__main__

Default entrypoint for the cwltool module.

[1] Created with sphinx-autoapi

cwltool.argparser

Command line argument parsing for cwltool.

Module Contents

Classes

FSAction	Information about how to convert command line strings to Python objects.
FSAppendAction	Information about how to convert command line strings to Python objects.
FileAction	Information about how to convert command line strings to Python objects.
DirectoryAction	Information about how to convert command line strings to Python objects.
FileAppendAction	Information about how to convert command line strings to Python objects.
DirectoryAppendAction	Information about how to convert command line strings to Python objects.

Functions

arg_parser()	
get_default_args()	Get default values of cwltool's command line options.
add_argument(toolparser, name, inptype, records, description = '', default = None, input_required = True, urljoin = urllib.parse.urljoin, base_uri = '')	
generate_parser(toolparser, tool, namemap, records, input_required = True, urljoin = urllib.parse.urljoin, base_uri = '')	

cwltool.argparser.**arg_parser**()

Return type argparse.ArgumentParser

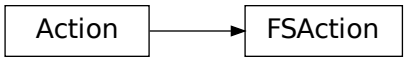
cwltool.argparser.**get_default_args**()

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



Information about how to convert command line strings to Python objects.

Action objects are used by an ArgumentParser to represent the information needed to parse a single argument from one or more strings from the command line. The keyword arguments to the Action constructor are also all attributes of Action instances.

Keyword Arguments:

- **option_strings – A list of command-line option strings which** should be associated with this action.
- dest – The name of the attribute to hold the created object(s)
- **nargs – The number of command-line arguments that should be** consumed. By default, one argument will be consumed and a single value will be produced. Other values include:
  - N (an integer) consumes N arguments (and produces a list)
  - '?' consumes zero or one arguments
  - '*' consumes zero or more arguments (and produces a list)
  - '+' consumes one or more arguments (and produces a list)

  Note that the difference between the default and nargs=1 is that with the default, a single value will be produced, while with nargs=1, a list containing a single value will be produced.
- **const – The value to be produced if the option is specified and the** option uses an action that takes no values.
- default – The value to be produced if the option is not specified.
- **type – A callable that accepts a single string argument, and** returns the converted value. The standard Python types str, int, float, and complex are useful examples of such callables. If None, str is used.
- **choices – A container of values that should be allowed. If not None,** after a command-line argument has been converted to the appropriate type, an exception will be raised if it is not a member of this collection.
- **required – True if the action must always be specified at the** command line. This is only meaningful for optional command-line arguments.
- help – The help string describing the argument.
- **metavar – The name to be used for the option's argument with the** help string. If None, the 'dest' value will be used as the name.

Parameters

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

- **base_uri** (*str*) –
- **kwargs** (*Any*) –

objclass :str

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

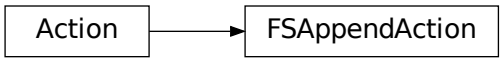
Parameters

- **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



Information about how to convert command line strings to Python objects.

Action objects are used by an ArgumentParser to represent the information needed to parse a single argument from one or more strings from the command line. The keyword arguments to the Action constructor are also all attributes of Action instances.

Keyword Arguments:

- **option_strings – A list of command-line option strings which** should be associated with this action.
- dest – The name of the attribute to hold the created object(s)
- **nargs – The number of command-line arguments that should be** consumed. By default, one argument will be consumed and a single value will be produced. Other values include:
  - N (an integer) consumes N arguments (and produces a list)
  - '?' consumes zero or one arguments
  - '*' consumes zero or more arguments (and produces a list)
  - '+' consumes one or more arguments (and produces a list)

  Note that the difference between the default and nargs=1 is that with the default, a single value will be produced, while with nargs=1, a list containing a single value will be produced.
- **const – The value to be produced if the option is specified and the** option uses an action that takes no values.
- default – The value to be produced if the option is not specified.

- **type – A callable that accepts a single string argument, and** returns the converted value. The standard Python types str, int, float, and complex are useful examples of such callables. If None, str is used.
- **choices – A container of values that should be allowed. If not None,** after a command-line argument has been converted to the appropriate type, an exception will be raised if it is not a member of this collection.
- **required – True if the action must always be specified at the** command line. This is only meaningful for optional command-line arguments.
- help – The help string describing the argument.
- **metavar – The name to be used for the option's argument with the** help string. If None, the 'dest' value will be used as the name.

Parameters

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

objclass :str

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

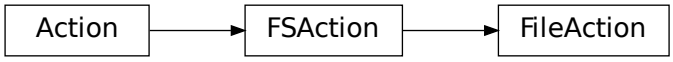
Parameters

- **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*



Information about how to convert command line strings to Python objects.

Action objects are used by an ArgumentParser to represent the information needed to parse a single argument from one or more strings from the command line. The keyword arguments to the Action constructor are also all attributes of Action instances.

Keyword Arguments:

- **option_strings – A list of command-line option strings which** should be associated with this action.
- dest – The name of the attribute to hold the created object(s)
- **nargs – The number of command-line arguments that should be** consumed. By default, one argument will be consumed and a single value will be produced. Other values include:
  - N (an integer) consumes N arguments (and produces a list)
  - '?' consumes zero or one arguments
  - '*' consumes zero or more arguments (and produces a list)
  - '+' consumes one or more arguments (and produces a list)

  Note that the difference between the default and nargs=1 is that with the default, a single value will be produced, while with nargs=1, a list containing a single value will be produced.
- **const – The value to be produced if the option is specified and the** option uses an action that takes no values.
- default – The value to be produced if the option is not specified.
- **type – A callable that accepts a single string argument, and** returns the converted value. The standard Python types str, int, float, and complex are useful examples of such callables. If None, str is used.
- **choices – A container of values that should be allowed. If not None,** after a command-line argument has been converted to the appropriate type, an exception will be raised if it is not a member of this collection.
- **required – True if the action must always be specified at the** command line. This is only meaningful for optional command-line arguments.
- help – The help string describing the argument.
- **metavar – The name to be used for the option's argument with the** help string. If None, the 'dest' value will be used as the name.

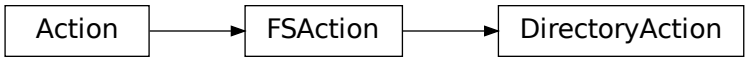
Parameters

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

objclass = File

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

Bases: *FSAction*



Information about how to convert command line strings to Python objects.

Action objects are used by an ArgumentParser to represent the information needed to parse a single argument from one or more strings from the command line. The keyword arguments to the Action constructor are also all attributes of Action instances.

Keyword Arguments:

- **option_strings – A list of command-line option strings which** should be associated with this action.
- dest – The name of the attribute to hold the created object(s)
- **nargs – The number of command-line arguments that should be** consumed. By default, one argument will be consumed and a single value will be produced. Other values include:
  - N (an integer) consumes N arguments (and produces a list)
  - '?' consumes zero or one arguments
  - '*' consumes zero or more arguments (and produces a list)
  - '+' consumes one or more arguments (and produces a list)

  Note that the difference between the default and nargs=1 is that with the default, a single value will be produced, while with nargs=1, a list containing a single value will be produced.
- **const – The value to be produced if the option is specified and the** option uses an action that takes no values.
- default – The value to be produced if the option is not specified.
- **type – A callable that accepts a single string argument, and** returns the converted value. The standard Python types str, int, float, and complex are useful examples of such callables. If None, str is used.
- **choices – A container of values that should be allowed. If not None,** after a command-line argument has been converted to the appropriate type, an exception will be raised if it is not a member of this collection.
- **required – True if the action must always be specified at the** command line. This is only meaningful for optional command-line arguments.
- help – The help string describing the argument.
- **metavar – The name to be used for the option's argument with the** help string. If None, the 'dest' value will be used as the name.

Parameters

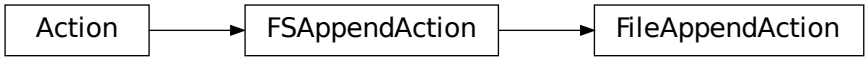
- **option_strings** (*List[str]*) –
- **dest** (*str*) –
- **nargs** (*Any*) –
- **urljoin** (*Callable[[str, str], str]*) –

- **base_uri** (*str*) –
- **kwargs** (*Any*) –

objclass = Directory

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

Bases: *FSAppendAction*



Information about how to convert command line strings to Python objects.

Action objects are used by an ArgumentParser to represent the information needed to parse a single argument from one or more strings from the command line. The keyword arguments to the Action constructor are also all attributes of Action instances.

Keyword Arguments:

- **option_strings – A list of command-line option strings which** should be associated with this action.
- dest – The name of the attribute to hold the created object(s)
- **nargs – The number of command-line arguments that should be** consumed. By default, one argument will be consumed and a single value will be produced. Other values include:
  - N (an integer) consumes N arguments (and produces a list)
  - '?' consumes zero or one arguments
  - '*' consumes zero or more arguments (and produces a list)
  - '+' consumes one or more arguments (and produces a list)

  Note that the difference between the default and nargs=1 is that with the default, a single value will be produced, while with nargs=1, a list containing a single value will be produced.
- **const – The value to be produced if the option is specified and the** option uses an action that takes no values.
- default – The value to be produced if the option is not specified.
- **type – A callable that accepts a single string argument, and** returns the converted value. The standard Python types str, int, float, and complex are useful examples of such callables. If None, str is used.
- **choices – A container of values that should be allowed. If not None,** after a command-line argument has been converted to the appropriate type, an exception will be raised if it is not a member of this collection.
- **required – True if the action must always be specified at the** command line. This is only meaningful for optional command-line arguments.
- help – The help string describing the argument.

- **metavar – The name to be used for the option's argument with the** help string. If None, the 'dest' value will be used as the name.

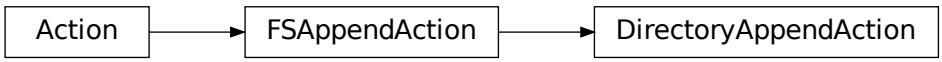
Parameters

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

objclass = File

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

Bases: *FSAppendAction*



Information about how to convert command line strings to Python objects.

Action objects are used by an ArgumentParser to represent the information needed to parse a single argument from one or more strings from the command line. The keyword arguments to the Action constructor are also all attributes of Action instances.

Keyword Arguments:

- **option_strings – A list of command-line option strings which** should be associated with this action.
- dest – The name of the attribute to hold the created object(s)
- **nargs – The number of command-line arguments that should be** consumed. By default, one argument will be consumed and a single value will be produced. Other values include:
  - N (an integer) consumes N arguments (and produces a list)
  - '?' consumes zero or one arguments
  - '*' consumes zero or more arguments (and produces a list)
  - '+' consumes one or more arguments (and produces a list)

  Note that the difference between the default and nargs=1 is that with the default, a single value will be produced, while with nargs=1, a list containing a single value will be produced.
- **const – The value to be produced if the option is specified and the** option uses an action that takes no values.
- default – The value to be produced if the option is not specified.

- **type – A callable that accepts a single string argument, and** returns the converted value. The standard Python types str, int, float, and complex are useful examples of such callables. If None, str is used.
- **choices – A container of values that should be allowed. If not None,** after a command-line argument has been converted to the appropriate type, an exception will be raised if it is not a member of this collection.
- **required – True if the action must always be specified at the** command line. This is only meaningful for optional command-line arguments.
- help – The help string describing the argument.
- **metavar – The name to be used for the option's argument with the** help string. If None, the 'dest' value will be used as the name.

Parameters

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

objclass = Directory

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=''*)

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

Module Contents

Classes

Builder	Base class for get_requirement().

Functions

content_limit_respected_read_bytes(f)	
content_limit_respected_read(f)	
substitute(value, replace)	
formatSubclassOf(fmt, cls, ontology, visited)	Determine if *fmt* is a subclass of *cls*.
check_format(actual_file, input_formats, ontology)	Confirm that the format present is valid for the allowed formats.

Attributes

INPUT_OBJ_VOCAB

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

cwltool.builder.**content_limit_respected_read_bytes**(*f*)

Parameters **f** (*IO[bytes]*) –

Return type bytes

cwltool.builder.**content_limit_respected_read**(*f*)

Parameters **f** (*IO[bytes]*) –

Return type str

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

Parameters

- **value** (*str*) –
- **replace** (*str*) –

Return type str

cwltool.builder.**formatSubclassOf**(*fmt*, *cls*, *ontology*, *visited*)

Determine if *fmt* is a subclass of *cls*.

Parameters

- **fmt** (*str*) –
- **cls** (*str*) –
- **ontology** (*Optional[rdflib.Graph]*) –
- **visited** (*Set[str]*) –

Return type bool

cwltool.builder.**check_format**(*actual_file*, *input_formats*, *ontology*)

Confirm that the format present is valid for the allowed formats.

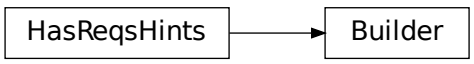
Parameters

- **actual_file** (*Union[cwltool.utils.CWLObjectType, List[cwltool.utils.CWLObjectType]]*) –
- **input_formats** (*Union[List[str], str]*) –
- **ontology** (*Optional[rdflib.Graph]*) –

Return type None

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*



Base class for get_requirement().

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** (*typing_extensions.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** (*str*) –
- **outdir** (*str*) –
- **tmpdir** (*str*) –
- **stagedir** (*str*) –
- **cwlVersion** (*str*) –
- **container_engine** (*str*) –

build_job_script(*self*, *commands*)

Parameters **commands** (*List[str]*) –

Return type Optional[str]

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

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(*self*, *value*)

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

Return type str

generate_arg(*self*, *binding*)

Parameters **binding** (*cwltool.utils.CWLObjectType*) –

Return type List[str]

do_eval(*self*, *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

check_types(srctype, sinktype, linkMerge, valueFrom)	Check if the source and sink types are correct.
merge_flatten_type(src)	Return the merge flattened type of the source type.
can_assign_src_to_sink(src, sink, strict = False)	Check for identical type specifications, ignoring extra keys like inputBinding.
missing_subset(fullset, subset)	
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.
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.
circular_dependency_checker(step_inputs)	Check if a workflow has circular dependency.
get_dependency_tree(step_inputs)	Get the dependency tree in the form of adjacency list.
processDFS(adjacency, traversal_path, processed, cycles)	Perform depth first search.
get_step_id(field_id)	Extract step id from either input or output fields.
is_conditional_step(param_to_step, parm_id)	

Attributes

SrcSink

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

Check if the source and sink types are correct.

Acceptable types are "pass", "warning", or "exception".

Parameters

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

Return type str

cwltool.checker.**merge_flatten_type**(*src*)

Return the merge flattened type of the source type.

Parameters **src** (*cwltool.utils.SinkType*) –

Return type cwltool.utils.CWLOutputType

cwltool.checker.**can_assign_src_to_sink**(*src*, *sink*, *strict=False*)

Check for identical type specifications, ignoring extra keys like inputBinding.

src: admissible source types sink: admissible sink types

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

Parameters

- **src** (*cwltool.utils.SinkType*) –
- **sink** (*Optional[cwltool.utils.SinkType]*) –
- **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.

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.

sourceField is either "source" or "outputSource"

Parameters

- **src_dict** (*Dict[str, cwltool.utils.CWLObjectType]*) –
- **sinks** (*MutableSequence[cwltool.utils.CWLObjectType]*) –
- **sourceField** (*str*) –
- **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.

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.command_line_tool

Implementation of CommandLineTool.

Module Contents

Classes

PathCheckingMode	What characters are allowed in path names.
ExpressionJob	Job for ExpressionTools.
ExpressionTool	Base class for get_requirement().
AbstractOperation	Base class for get_requirement().
CallbackJob	Callback Job class, used by CommandLine.job().
CommandLineTool	Base class for get_requirement().

Functions

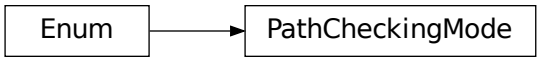
remove_path(f)	
revmap_file(builder, outdir, f)	Remap a file from internal path to external path.
check_adjust(accept_re, builder, file_o)	Map files to assigned path inside a container.
check_valid_locations(fs_access, ob)	

Attributes

OutputPortsType

class cwltool.command_line_tool.**PathCheckingMode**

Bases: enum.Enum



What characters are allowed in path names.

We have the strict (default) mode and the relaxed mode.

STRICT

RELAXED

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

Job for ExpressionTools.

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

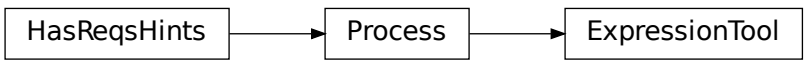
Parameters

- **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*



Base class for get_requirement().

Parameters

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

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

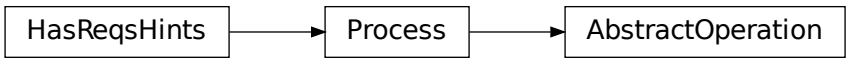
Parameters

- **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*



Base class for get_requirement().

Parameters

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

job(*self*, *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 tho 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 CommandLine.job().

Parameters

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

run(*self*, *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

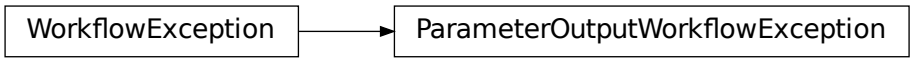
- **fs_access** (cwltool.stdfsaccess.StdFsAccess) –
- **ob** (*cwltool.utils.CWLObjectType*) –

Return type None

cwltool.command_line_tool.**OutputPortsType**

exception cwltool.command_line_tool.**ParameterOutputWorkflowException**(*msg*, *port*, ***kwargs*)

Bases: *cwltool.errors.WorkflowException*



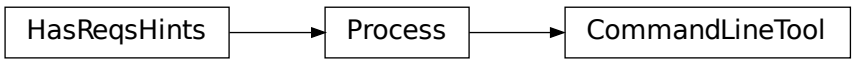
Common base class for all non-exit exceptions.

Parameters

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

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

Bases: *cwltool.process.Process*



Base class for get_requirement().

Parameters

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

make_job_runner(*self*, *runtimeContext*)

Parameters **runtimeContext** (cwltool.context.RuntimeContext) –

Return type typing_extensions.Type[*cwltool.job.JobBase*]

make_path_mapper(*self*, *reffiles*, *stagedir*, *runtimeContext*, *separateDirs*)

Parameters

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

Return type *cwltool.pathmapper.PathMapper*

updatePathmap(*self*, *outdir*, *pathmap*, *fn*)

Parameters

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

Return type None

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

Parameters

- **job_order** (*cwltool.utils.CWLObjectType*) –

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

Return type Generator[Union[*cwltool.job.JobBase*, *CallbackJob*], None, None]

collect_output_ports(*self*, *ports*, *builder*, *outdir*, *rcode*, *compute_checksum=True*, *jobname=''*, *readers=None*)

Parameters

- **ports** (*Union[ruamel.yaml.comments.CommentedSeq, 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(*self*, *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

ContextBase	Shared kwargs based initilizer for {Runtime,Loading}Context.
LoadingContext	Shared kwargs based initilizer for {Runtime,Loading}Context.
RuntimeContext	Shared kwargs based initilizer for {Runtime,Loading}Context.

Functions

make_tool_notimpl(toolpath_object, loadingContext)	
log_handler(outdir, base_path_logs, stdout_path, stderr_path)	Move logs from log location to final output.
set_log_dir(outdir, log_dir, subdir_name)	Default handler for setting the log directory.
getdefault(val, default)	

Attributes

default_make_tool

class cwltool.context.**ContextBase**(*kwargs=None*)

Shared kwargs based initilizer for {Runtime,Loading}Context.

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

cwltool.context.**make_tool_notimpl**(*toolpath_object*, *loadingContext*)

Parameters

- **toolpath_object** (*ruamel.yaml.comments.CommentedMap*) –
- **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*)

Default handler for setting the log directory.

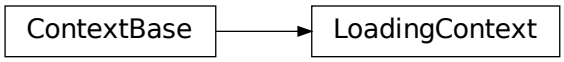
Parameters

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

Return type str

class cwltool.context.**LoadingContext**(*kwargs=None*)

Bases: *ContextBase*



Shared kwargs based initilizer for {Runtime,Loading}Context.

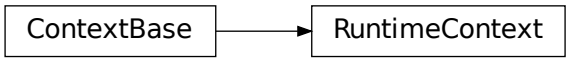
Parameters **kwargs** (*Optional[Dict[str, Any]]*) –

copy(*self*)

Return type *LoadingContext*

class cwltool.context.**RuntimeContext**(*kwargs=None*)

Bases: *ContextBase*



Shared kwargs based initilizer for {Runtime,Loading}Context.

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

get_outdir(*self*)

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

Return type str

get_tmpdir(*self*)

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

Return type str

get_stagedir(*self*)

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

Return type str

create_tmpdir(*self*)

Create a temporary directory that respects self.tmpdir_prefix.

Return type str

create_outdir(*self*)

Create a temporary directory that respects self.tmp_outdir_prefix.

Return type str

copy(*self*)

Return type *RuntimeContext*

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

Parameters

- **val** (*Any*) –
- **default** (*Any*) –

Return type Any

cwltool.cuda

Module Contents

Functions

cuda_version_and_device_count()
cuda_check(cuda_req, requestCount)

cwltool.cuda.**cuda_version_and_device_count**()

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** (*Union[TextIO, codecs.StreamWriter]*) –

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(*self*)

Get the dot graph object.

Return type pydot.Graph

dot(*self*)

Get the graph as graphviz.

Return type str

cwltool.docker

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

Module Contents

Classes

DockerCommandLineJob	Runs a CommandLineJob in a software container using the Docker 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[Ellipsis, cwltool.pathmapper.PathMapper]*) –
- **requirements** (*List[cwltool.utils.CWLObjectType]*) –
- **hints** (*List[cwltool.utils.CWLObjectType]*) –
- **name** (*str*) –

static **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(*self*, *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*)

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(*self*, *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(*self*, *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(*self*, *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(*self*, *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]]

cwltool.docker_id

Helper functions for docker.

Module Contents

Functions

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

deserialize_env(data)	Deserialize the output of *env -0* to dictionary.
main()	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.

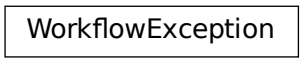
Return type None

cwltool.errors

Module Contents

exception cwltool.errors.**WorkflowException**

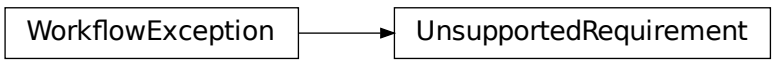
Bases: Exception



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

JobExecutor	Abstract base job executor.
SingleJobExecutor	Default single-threaded CWL reference executor.
MultithreadedJobExecutor	Experimental multi-threaded CWL executor.
NoopJobExecutor	Do nothing executor, for testing purposes only.

Attributes

TMPDIR_LOCK

cwltool.executors.**TMPDIR_LOCK**

class cwltool.executors.**JobExecutor**

Abstract base job executor.

__call__(*self*, *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(*self*, *out*, *process_status*)

Collect the final status and outputs.

Parameters

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

Return type None

abstract run_jobs(*self*, *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(*self*, *process*, *job_order_object*, *runtime_context*, *logger=_logger*)

Execute the process.

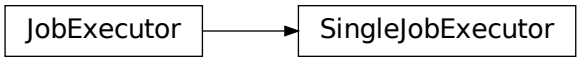
Parameters

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

Return type Tuple[Union[Optional[cwltool.utils.CWLObjectType]], str]

class cwltool.executors.**SingleJobExecutor**

Bases: *JobExecutor*



Default single-threaded CWL reference executor.

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

Execute the jobs for the given Process.

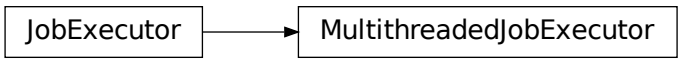
Parameters

- **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(*self*, *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(*self*, *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(*self*, *runtime_context*)

Wait for jobs to finish.

Parameters **self** (cwltool.context.RuntimeContext) –

Return type None

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

Execute the jobs for the given Process.

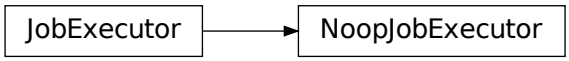
Parameters

- **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(*self*, *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(*self*, *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.expression

Parse CWL expressions.

Module Contents

Functions

jshead(engine_config, rootvars)	
scanner(scan)	
next_seg(parsed_string, remaining_string, current_value)	
evaluator(ex, jslib, obj, timeout, fullJS = False, force_docker_pull = False, debug = False, js_console = False, container_engine = 'docker')	
interpolate(scan, rootvars, timeout = default_timeout, fullJS = False, jslib = '', force_docker_pull = False, debug = False, js_console = False, strip_whitespace = True, escaping_behavior = 2, convert_to_expression = False, container_engine = 'docker')	Interpolate and evaluate.
needs_parsing(snippet)	
do_eval(ex, jobinput, requirements, outdir, tmpdir, resources, context = None, timeout = default_timeout, force_docker_pull = False, debug = False, js_console = False, strip_whitespace = True, cwlVersion = '', container_engine = 'docker')	

Attributes

seg_symbol
seg_single
seg_double
seg_index
segments
segment_re
param_str
param_re

cwltool.expression.**jshead**(*engine_config*, *rootvars*)

Parameters

- **engine_config** (*List[str]*) –
- **rootvars** (*cwltool.utils.CWLObjectType*) –

Return type str

cwltool.expression.**seg_symbol** = \w+

cwltool.expression.**seg_single** = \['([^']|\\')+'\]

cwltool.expression.**seg_double** = \["([^"]|\\")+"\]

cwltool.expression.**seg_index** = \[[0-9]+\]

cwltool.expression.**segments**

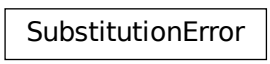
cwltool.expression.**segment_re**

cwltool.expression.**param_str**

cwltool.expression.**param_re**

exception cwltool.expression.**SubstitutionError**

Bases: Exception



Common base class for all non-exit exceptions.

cwltool.expression.**scanner**(*scan*)

Parameters **scan** (*str*) –

Return type Optional[Tuple[int, int]]

cwltool.expression.**next_seg**(*parsed_string*, *remaining_string*, *current_value*)

Parameters

- **parsed_string** (*str*) –
- **remaining_string** (*str*) –
- **current_value** (*cwltool.utils.CWLOutputType*) –

Return type cwltool.utils.CWLOutputType

cwltool.expression.**evaluator**(*ex*, *jslib*, *obj*, *timeout*, *fullJS=False*, *force_docker_pull=False*, *debug=False*, *js_console=False*, *container_engine='docker'*)

Parameters

- **ex** (*str*) –
- **jslib** (*str*) –
- **obj** (*cwltool.utils.CWLObjectType*) –
- **timeout** (*float*) –
- **fullJS** (*bool*) –
- **force_docker_pull** (*bool*) –
- **debug** (*bool*) –
- **js_console** (*bool*) –
- **container_engine** (*str*) –

Return type Optional[cwltool.utils.CWLOutputType]

cwltool.expression.**interpolate**(*scan*, *rootvars*, *timeout=default_timeout*, *fullJS=False*, *jslib=''*, *force_docker_pull=False*, *debug=False*, *js_console=False*, *strip_whitespace=True*, *escaping_behavior=2*, *convert_to_expression=False*, *container_engine='docker'*)

Interpolate and evaluate.

Note: only call with convert_to_expression=True on CWL Expressions in $() form that need interpolation.

Parameters

- **scan** (*str*) –
- **rootvars** (*cwltool.utils.CWLObjectType*) –
- **timeout** (*float*) –
- **fullJS** (*bool*) –
- **jslib** (*str*) –
- **force_docker_pull** (*bool*) –
- **debug** (*bool*) –
- **js_console** (*bool*) –

- **strip_whitespace** (*bool*) –
- **escaping_behavior** (*int*) –
- **convert_to_expression** (*bool*) –
- **container_engine** (*str*) –

Return type Optional[cwltool.utils.CWLOutputType]

cwltool.expression.**needs_parsing**(*snippet*)

Parameters **snippet** (*Any*) –

Return type bool

cwltool.expression.**do_eval**(*ex*, *jobinput*, *requirements*, *outdir*, *tmpdir*, *resources*, *context=None*, *timeout=default_timeout*, *force_docker_pull=False*, *debug=False*, *js_console=False*, *strip_whitespace=True*, *cwlVersion=''*, *container_engine='docker'*)

Parameters

- **ex** (*Optional[cwltool.utils.CWLOutputType]*) –
- **jobinput** (*cwltool.utils.CWLObjectType*) –
- **requirements** (*List[cwltool.utils.CWLObjectType]*) –
- **outdir** (*Optional[str]*) –
- **tmpdir** (*Optional[str]*) –
- **resources** (*Dict[str, Union[float, int]]*) –
- **context** (*Optional[cwltool.utils.CWLOutputType]*) –
- **timeout** (*float*) –
- **force_docker_pull** (*bool*) –
- **debug** (*bool*) –
- **js_console** (*bool*) –
- **strip_whitespace** (*bool*) –
- **cwlVersion** (*str*) –
- **container_engine** (*str*) –

Return type Optional[cwltool.utils.CWLOutputType]

cwltool.factory

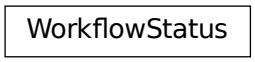
Module Contents

Classes

Callable	Result of Factory.make().
Factory	Easy way to load a CWL document for execution.

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

Bases: Exception



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__(*self*, ***kwargs*)

Parameters **self** (*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(*self*, *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 = (list, tuple))	

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, container_outdir, inplace_update = False)	
neverquote(string, pos = 0, endpos = 0)	

Attributes

needs_shell_quoting_re
FORCE_SHELLED_POPEN
SHELL_COMMAND_TEMPLATE
CollectOutputsType
CONTROL_CODE_RE

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

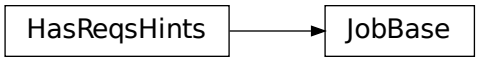
- **string** (*str*) –
- **pos** (*int*) –
- **endpos** (*int*) –

Return type Optional[Match[str]]

cwltool.job.**CollectOutputsType**

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[*Ellipsis*, cwltool.pathmapper.PathMapper]) –
- **requirements** (*List[cwltool.utils.CWLObjectType]*) –
- **hints** (*List[cwltool.utils.CWLObjectType]*) –
- **name** (*str*) –

__repr__(*self*)

Represent this Job object.

Return type str

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

Parameters

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

Return type None

prepare_environment(*self*, *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

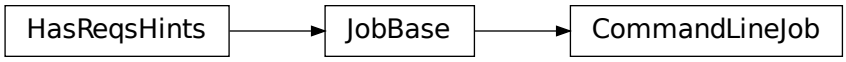
process_monitor(*self*, *sproc*)

Parameters **self** (*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[*Ellipsis*, cwltool.pathmapper.PathMapper]) –
- **requirements** (*List[cwltool.utils.CWLObjectType]*) –
- **hints** (*List[cwltool.utils.CWLObjectType]*) –
- **name** (*str*) –

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

Parameters

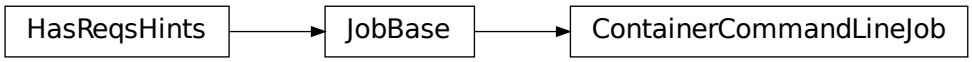
- **runtimeContext** (cwltool.context.RuntimeContext) –
- **tmpdir_lock** (*Optional[threading.Lock]*) –

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[*Ellipsis*, cwltool.pathmapper.PathMapper]) –
- **requirements** (*List[cwltool.utils.CWLObjectType]*) –
- **hints** (*List[cwltool.utils.CWLObjectType]*) –

- **name** (*str*) –

CONTAINER_TMPDIR :str = /tmp

abstract get_from_requirements(*self*, *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(*self*, *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(*self*, *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(*self*, *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(*self*, *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(*self*, *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(*self*, *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(*self*, *runtimeContext*, *tmpdir_lock=None*)

Parameters

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

Return type None

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

Record memory usage of the running Docker container.

Parameters

- **cidfile** (*str*) –

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

Return type None

cwltool.load_tool

Loads a CWL document.

Module Contents

Functions

default_loader(fetcher_constructor = None, enable_dev = False, doc_cache = True)	
resolve_tool_uri(argsworkflow, resolver = None, fetcher_constructor = None, document_loader = None)	
fetch_document(argsworkflow, loadingContext = None)	Retrieve a CWL document.
resolve_and_validate_document(loadingContext, workflowobj, uri, preprocess_only = False, skip_schemas = None)	Validate a CWL document.
make_tool(uri, loadingContext)	Make a Python CWL object.
load_tool(argsworkflow, loadingContext = None)	
resolve_overrides(ov, ov_uri, baseurl)	
load_overrides(ov, base_url)	
recursive_resolve_and_validate_document(loadingContext, workflowobj, uri, preprocess_only = False, skip_schemas = None)	Validate a CWL document, checking that a tool object can be built.

Attributes

jobloaderctx	
overrides_ctx	

cwltool.load_tool.**jobloaderctx :ContextType**

cwltool.load_tool.**overrides_ctx :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.CommentedMap, str]

cwltool.load_tool.**resolve_and_validate_document**(*loadingContext*, *workflowobj*, *uri*, *preprocess_only=False*, *skip_schemas=None*)

Validate a CWL document.

Parameters

- **loadingContext** (cwltool.context.LoadingContext) –
- **workflowobj** (*Union[ruamel.yaml.comments.CommentedMap, ruamel.yaml.comments.CommentedSeq]*) –
- **uri** (*str*) –
- **preprocess_only** (*bool*) –
- **skip_schemas** (*Optional[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.CommentedMap, ruamel.yaml.comments.CommentedSeq]*) –
- **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*, *skip_schemas=None*)

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

Parameters

- **loadingContext** (*cwltool.context.LoadingContext*) –
- **workflowobj** (*Union[ruamel.yaml.comments.CommentedMap, ruamel.yaml.comments.CommentedSeq]*) –
- **uri** (*str*) –
- **preprocess_only** (*bool*) –
- **skip_schemas** (*Optional[bool]*) –

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

cwltool.loghandler

Shared logger for cwltool.

Module Contents

Functions

configure_logging(stderr_handler, quiet, debug, enable_color, timestamps, base_logger = _logger)	Configure logging.

Attributes

defaultStreamHandler

cwltool.loghandler.**defaultStreamHandler**

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

Configure logging.

Parameters

- **stderr_handler** (*logging.Handler*) –
- **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

ProvLogFormatter	Enforce ISO8601 with both T and Z.

Functions

generate_example_input(inptype, default)	Convert a single input schema into an example.
realize_input_schema(input_types, schema_defs)	Replace references to named typed with the actual types.
generate_input_template(tool)	Generate an example input object for the given CWL process.
load_job_order(args, stdin, fetcher_constructor, overrides_list, tool_file_uri)	
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)	
make_relative(base, obj)	Relativize the location URI of a File or Directory object.
printdeps(obj, document_loader, stdout, relative_deps, uri, basedir = None, nestdirs = True)	Print a JSON representation of the dependencies of the CWL document.
prov_deps(obj, document_loader, uri, basedir = None)	
find_deps(obj, document_loader, uri, basedir = None, nestdirs = True)	Find the dependencies of the CWL document.
print_pack(loadingContext, uri)	Return a CWL serialization of the CWL document in JSON.
supported_cwl_versions(enable_dev)	
setup_schema(args, custom_schema_callback)	
setup_provenance(args, argsl, runtimeContext)	
setup_loadingContext(loadingContext, runtimeContext, args)	Prepare a LoadingContext from the given arguments.
make_template(tool)	Make a template CWL input object for the give Process.
inherit_reqshints(tool, parent)	Copy down requirements and hints from ancestors of a given process.
choose_target(args, tool, loading_context)	Walk the Workflow, extract the subset matches all the args.targets.
choose_step(args, tool, loading_context)	Walk the given Workflow and extract just args.single_step.
choose_process(args, tool, loadingContext)	Walk the given Workflow and extract just args.single_process.
check_working_directories(runtimeContext)	Make any needed working directories.
print_targets(tool, stdout, loading_context, prefix = '')	Recursively find targets for --subgraph and friends.
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)	
find_default_container(builder, default_container = None, use_biocontainers = None)	Find a container.
windows_check()	See if we are running on MS Windows and warn about the lack of support.
run(*args, **kwargs)	Run cwltool.

Attributes

ProvOut

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

Convert a single input schema into an example.

Parameters

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

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** (*Union[TextIO, codecs.StreamWriter]*) –
- **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** (*Union[TextIO, codecs.StreamWriter]*) –
- **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*)

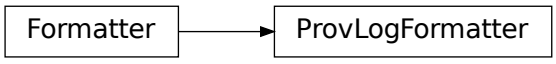
Parameters

- **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(*self*, *record*, *datefmt=None*)

Return the creation time of the specified LogRecord as formatted text.

This method should be called from format() by a formatter which wants to make use of a formatted time. This method can be overridden in formatters to provide for any specific requirement, but the basic behaviour is as follows: if datefmt (a string) is specified, it is used with time.strftime() to format the creation time

of the record. Otherwise, an ISO8601-like (or RFC 3339-like) format is used. The resulting string is returned. This function uses a user-configurable function to convert the creation time to a tuple. By default, time.localtime() is used; to change this for a particular formatter instance, set the 'converter' attribute to a function with the same signature as time.localtime() or time.gmtime(). To change it for all formatters, for example if you want all logging times to be shown in GMT, set the 'converter' attribute in the Formatter class.

Parameters

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

Return type str

cwltool.main.**ProvOut**

cwltool.main.**setup_provenance**(*args*, *argsl*, *runtimeContext*)

Parameters

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

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*)

Make a template CWL input object for the give Process.

Parameters **tool** (cwltool.process.Process) –

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 type Optional[*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 type Optional[*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 type Optional[*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** (*Union[TextIO, codecs.StreamWriter]*) –
- **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[Union[TextIO, codecs.StreamWriter]]*) –
- **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*)

Find a container.

Parameters

- **builder** (cwltool.utils.HasReqsHints) –
- **default_container** (*Optional[str]*) –
- **use_biocontainers** (*Optional[bool]*) –

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 None

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(*cls*, *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

- **cls** (*Type[MpiConfigT]*) –
- **config_file_name** (*str*) –

Return type MpiConfigT

pass_through_env_vars(*self*, *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(*self*, *env*)

Set some variables to the value configured.

Parameters **env** (*MutableMapping[str, str]*) –

Return type None

cwltool.mutation

Module Contents

Classes

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

Attributes

MutationState

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(*self*, *stepname*, *obj*)

Parameters

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

Return type None

release_reader(*self*, *stepname*, *obj*)

Parameters

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

Return type None

register_mutation(*self*, *stepname*, *obj*)

Parameters

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

Return type None

set_generation(*self*, *obj*)

Parameters **obj** (*cwltool.utils.CWLObjectType*) –

Return type None

unset_generation(*self*, *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 = None, loader = None)	

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

MapperEnt

cwltool.pathmapper.**MapperEnt**

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 File and 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(*self*, *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(*self*, *obj*, *stagedir*, *basedir*, *copy=False*, *staged=False*)

Parameters

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

- **stagedir** (*str*) –
- **basedir** (*str*) –
- **copy** (*bool*) –
- **staged** (*bool*) –

Return type None

setup(*self*, *referenced_files*, *basedir*)

Parameters

- **referenced_files** (*List[cwltool.utils.CWLObjectType]*) –
- **basedir** (*str*) –

Return type None

mapper(*self*, *src*)

Parameters **src** (*str*) –

Return type MapperEnt

files(*self*)

Return type List[str]

items(*self*)

Return type List[Tuple[str, MapperEnt]]

reversemap(*self*, *target*)

Find the (source, resolved_path) for the given target, if any.

Parameters **target** (*str*) –

Return type Optional[Tuple[str, str]]

update(*self*, *key*, *resolved*, *target*, *ctype*, *stage*)

Parameters

- **key** (*str*) –
- **resolved** (*str*) –
- **target** (*str*) –
- **ctype** (*str*) –
- **stage** (*bool*) –

Return type MapperEnt

__contains__(*self*, *key*)

Test for the presence of the given relative path in this mapper.

Parameters **key** (*str*) –

Return type bool

__iter__(*self*)

Get iterator for the maps.

Return type Iterator[MapperEnt]

cwltool.process

Classes and methods relevant for all CWL Process types.

Module Contents

Classes

LogAsDebugFilter	Filter instances are used to perform arbitrary filtering of LogRecords.
Process	Base class for get_requirement().

Functions

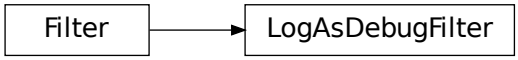
use_standard_schema(version)	
use_custom_schema(version, name, text)	
get_schema(version)	
shortname(inputid)	
checkRequirements(rec, supported_process_requirements)	
stage_files(pathmapper, stage_func = None, ignore_writable = False, symlink = True, secret_store = None, fix_conflicts = False)	Link or copy files to their targets. Create them as needed.
relocateOutputs(outputObj, destination_path, source_directories, action, fs_access, compute_checksum = True, path_mapper = PathMapper)	
cleanIntermediate(output_dirs)	
add_sizes(fsaccess, obj)	
fill_in_defaults(inputs, job, fsaccess)	
avroize_type(field_type, name_prefix = '')	Add missing information to a type so that CWL types are valid.
get_overrides(overrides, toolid)	
var_spool_cwl_detector(obj, item = None, obj_key = None)	Detect any textual reference to /var/spool/cwl.
eval_resource(builder, resource_req)	
uniquename(stem, names = None)	
nestdir(base, deps)	
mergedirs(listing)	
scandeps(base, doc, reffields, urlfields, loadref, urljoin = urllib.parse.urljoin, nestdirs = True)	Given a CWL document or input object, search for dependencies
compute_checksums(fs_access, fileobj)	

Attributes

supportedProcessRequirements
cwl_files
salad_files
SCHEMA_CACHE
SCHEMA_FILE
SCHEMA_DIR
SCHEMA_ANY
custom_schemas
FILE_COUNT_WARNING
CWL_IANA

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(*self*, *record*)

Determine if the specified record is to be logged.

Is the specified record to be logged? Returns 0 for no, nonzero for yes. 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[Loader, Union[Names, SchemaParseException], CWLObjectType, Loader]]**

cwltool.process.**SCHEMA_FILE :Optional[CWLObjectType]**

cwltool.process.**SCHEMA_DIR :Optional[CWLObjectType]**

cwltool.process.**SCHEMA_ANY :Optional[CWLObjectType]**

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 Tuple[schema_salad.ref_resolver.Loader, Union[schema_salad.avro.schema.Names, schema_salad.avro.schema.SchemaParseException], cwltool.utils.CWLObjectType, schema_salad.ref_resolver.Loader]

cwltool.process.**shortname**(*inputid*)

Parameters **inputid** (*str*) –

Return type str

cwltool.process.**checkRequirements**(*rec*, *supported_process_requirements*)

Parameters

- **rec** (*Union[MutableSequence[cwltool.utils.CWLObjectType], cwltool.utils.CWLObjectType, cwltool.utils.CWLOutputType, None]*) –
- **supported_process_requirements** (*Iterable[str]*) –

Return type None

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.

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*)

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*)

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

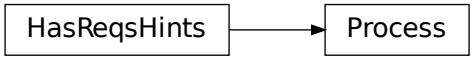
- **builder** (cwltool.builder.Builder) –
- **resource_req** (*Union[str, int, float]*) –

Return type Optional[Union[str, int, float]]

cwltool.process.**FILE_COUNT_WARNING = 5000**

class cwltool.process.**Process**(*toolpath_object*, *loadingContext*)

Bases: *cwltool.utils.HasReqsHints*



Base class for get_requirement().

Parameters

- **toolpath_object** (*ruamel.yaml.comments.CommentedMap*) –

- **loadingContext** (cwltool.context.LoadingContext) –

evalResources(*self*, *builder*, *runtimeContext*)

Parameters

- **builder** (cwltool.builder.Builder) –
- **runtimeContext** (cwltool.context.RuntimeContext) –

Return type Dict[str, Union[int, float]]

validate_hints(*self*, *avsc_names*, *hints*, *strict*)

Parameters

- **avsc_names** (*schema_salad.avro.schema.Names*) –
- **hints** (*List[cwltool.utils.CWLObjectType]*) –
- **strict** (*bool*) –

Return type None

visit(*self*, *op*)

Parameters **op** (*Callable[[ruamel.yaml.comments.CommentedMap], None]*) –

Return type None

abstract job(*self*, *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__(*self*)

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*)

Given a CWL document or input object, search for dependencies (references to external files) of 'doc' and return them as a list of File or Directory objects.

The 'base' is the base URL for relative references.

Looks for objects with 'class: File' or 'class: Directory' and adds them to the list of dependencies.

Anything in 'urlfields' is also added as a File dependency.

Anything in 'reffields' (such as 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.

If "nestdirs" is 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.

Parameters

- **base** (*str*) –
- **doc** (*Union[cwltool.utils.CWLObjectType, MutableSequence[cwltool.utils.CWLObjectType]]*) –
- **reffields** (*Set[str]*) –
- **urlfields** (*Set[str]*) –
- **loadref** (*Callable[[str, str], Union[ruamel.yaml.comments.CommentedMap, ruamel.yaml.comments.CommentedSeq, str, None]]*) –
- **urljoin** (*Callable[[str, str], str]*) –
- **nestdirs** (*bool*) –

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

ProcessGeneratorJob	Result of ProcessGenerator.job().
ProcessGenerator	Base class for get_requirement().

class cwltool.procgenerator.**ProcessGeneratorJob**(*procgenerator*)

Result of ProcessGenerator.job().

Parameters **procgenerator** (ProcessGenerator) –

receive_output(*self*, *jobout*, *processStatus*)

Parameters

- **jobout** (*Optional[cwltool.utils.CWLObjectType]*) –
- **processStatus** (*str*) –

Return type None

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

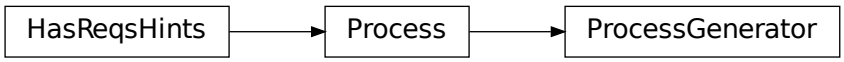
Parameters

- **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*



Base class for get_requirement().

Parameters

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

job(*self*, *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(*self*, *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.provenance

Stores Research Object including provenance.

Module Contents

Classes

WritableBagFile	Writes files in research object.
ResearchObject	CWLProv Research Object.

Functions

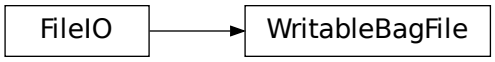
checksum_copy(src_file, dst_file = None, hasher = Hasher, buffersize = 1024 * 1024)	Compute checksums while copying a file.

Attributes

Annotation
Aggregate
AuthoredBy

class cwltool.provenance.**WritableBagFile**(*research_object*, *rel_path*)

Bases: io.FileIO



Writes files in research object.

Parameters

- **research_object** (ResearchObject) –
- **rel_path** (*str*) –

write(*self*, *b*)

Write some content to the Bag.

Parameters **b** (*Any*) –

Return type int

close(*self*)

Close the file.

A closed file cannot be used for further I/O operations. close() may be called more than once without error.

Return type None

seekable(*self*)

True if file supports random-access.

Return type bool

readable(*self*)

True if file was opened in a read mode.

Return type bool

truncate(*self*, *size=None*)

Truncate the file to at most size bytes and return the truncated size.

Size defaults to the current file position, as returned by tell(). The current file position is changed to the value of size.

Parameters **size** (*Optional[int]*) –

Return type int

cwltool.provenance.**Annotation**

cwltool.provenance.**Aggregate**

cwltool.provenance.**AuthoredBy**

class cwltool.provenance.**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(*self*)

Raise ValueError if this RO is closed.

Return type None

__str__(*self*)

Represent this RO as a string.

Return type str

open_log_file_for_activity(*self*, *uuid_uri*)

Parameters **uuid_uri** (*str*) –

Return type Union[io.TextIOWrapper, *WritableBagFile*]

user_provenance(*self*, *document*)

Add the user provenance.

Parameters **document** (*prov.model.ProvDocument*) –

Return type None

write_bag_file(*self*, *path*, *encoding=ENCODING*)

Write the bag file into our research object.

Parameters

- **path** (*str*) –
- **encoding** (*Optional[str]*) –

Return type Union[io.TextIOWrapper, *WritableBagFile*]

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

Add tag files to our research object.

Parameters

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

Return type None

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

Parameters

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

Return type Aggregate

add_annotation(*self*, *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(*self*, *prov_dep*)

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

Parameters **prov_dep** (*cwltool.utils.CWLObjectType*) –

Return type None

packed_workflow(*self*, *packed*)

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

Parameters **packed** (*str*) –

Return type None

has_data_file(*self*, *sha1hash*)

Confirm the presence of the given file in the RO.

Parameters **sha1hash** (*str*) –

Return type bool

add_data_file(*self*, *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(*self*, *rel_path*, *checksums*)

Add files to the research object manifest.

Parameters

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

Return type None

create_job(*self*, *builder_job*, *is_output=False*)

Generate the new job object with RO specific relative paths.

Parameters

- **builder_job** (*cwltool.utils.CWLObjectType*) –
- **is_output** (*bool*) –

Return type cwltool.utils.CWLObjectType

close(*self*, *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 **save_to** (*Optional[str]*) –

Return type None

cwltool.provenance.**checksum_copy**(*src_file*, *dst_file=None*, *hasher=Hasher*, *buffersize=1024 * 1024*)

Compute checksums while copying a file.

Parameters

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

Return type str

cwltool.provenance_constants

Module Contents

cwltool.provenance_constants.**__citation__ = https://doi.org/10.5281/zenodo.1208477**

cwltool.provenance_constants.**CWLPROV_VERSION = https://w3id.org/cwl/prov/0.6.0**

cwltool.provenance_constants.**METADATA = metadata**

cwltool.provenance_constants.**DATA = data**

cwltool.provenance_constants.**WORKFLOW = workflow**

cwltool.provenance_constants.**SNAPSHOT = snapshot**

cwltool.provenance_constants.**MAIN**

cwltool.provenance_constants.**PROVENANCE**

cwltool.provenance_constants.**LOGS**

cwltool.provenance_constants.**WFDESC**

cwltool.provenance_constants.**WFPROV**

cwltool.provenance_constants.**WF4EVER**

cwltool.provenance_constants.**RO**

cwltool.provenance_constants.**ORE**

cwltool.provenance_constants.**FOAF**

cwltool.provenance_constants.**SCHEMA**

cwltool.provenance_constants.**CWLPROV**

cwltool.provenance_constants.**ORCID**

cwltool.provenance_constants.**UUID**

cwltool.provenance_constants.**ENCODING = UTF-8**

cwltool.provenance_constants.**TEXT_PLAIN**

cwltool.provenance_constants.**Hasher**

cwltool.provenance_constants.**SHA1 = sha1**

cwltool.provenance_constants.**SHA256 = sha256**

cwltool.provenance_constants.**SHA512 = sha512**

cwltool.provenance_constants.**USER_UUID**

cwltool.provenance_constants.**ACCOUNT_UUID**

cwltool.provenance_profile

Module Contents

Classes

ProvenanceProfile	Provenance profile.

Functions

copy_job_order(job, job_order_object)	Create copy of job object for provenance.

cwltool.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.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.provenance.ResearchObject) –
- **full_name** (*str*) –
- **host_provenance** (*bool*) –
- **user_provenance** (*bool*) –
- **orcid** (*str*) –
- **fsaccess** (cwltool.stdfsaccess.StdFsAccess) –
- **run_uuid** (*Optional[uuid.UUID]*) –

__str__(*self*)

Represent this Provenvance profile as a string.

Return type str

generate_prov_doc(*self*)

Add basic namespaces.

Return type Tuple[str, prov.model.ProvDocument]

evaluate(*self*, *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.provenance.ResearchObject) –

Return type None

record_process_start(*self*, *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(*self*, *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(*self*, *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(*self*, *value*)

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

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

declare_directory(*self*, *value*)

Register any nested files/directories.

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

Return type prov.model.ProvEntity

declare_string(*self*, *value*)

Save as string in UTF-8.

Parameters **value** (*str*) –

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

declare_artefact(*self*, *value*)

Create data artefact entities for all file objects.

Parameters **value** (*Any*) –

Return type prov.model.ProvEntity

used_artefacts(*self*, *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(*self*, *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(*self*, *job*)

Create prospective prov recording as wfdesc prov:Plan.

Parameters **job** (*cwltool.utils.JobsType*) –

Return type None

activity_has_provenance(*self*, *activity*, *prov_ids*)

Add http://www.w3.org/TR/prov-aq/ relations to nested PROV files.

Parameters

- **self** (*str*) –
- **activity** (*List[prov.identifier.Identifier]*) –

Return type None

finalize_prov_profile(*self*, *name*)

Transfer the provenance related files to the RO.

Parameters **self** (*Optional[str]*) –

Return type List[prov.identifier.Identifier]

cwltool.resolver

Resolves references to CWL documents from local or remote places.

Module Contents

Functions

resolve_local(document_loader, uri)

tool_resolver(document_loader, uri)

resolve_ga4gh_tool(document_loader, uri)

Attributes

ga4gh_tool_registries

GA4GH_TRS_FILES

GA4GH_TRS_PRIMARY_DESCRIPTOR

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

handle_software_environment(cwl_env, script)	Update the provided environment dict by running the script.
main(argv)	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 specifes 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.sandboxjs

Evaluate CWL Javascript Expressions in a sandbox.

Module Contents

Functions

check_js_threshold_version(working_alias)	Check if the nodeJS engine version on the system with the allowed minimum version.
new_js_proc(js_text, force_docker_pull = False, container_engine = 'docker')	Return a subprocess ready to submit javascript to.
exec_js_process(js_text, timeout = default_timeout, js_console = False, context = None, force_docker_pull = False, container_engine = 'docker')	
code_fragment_to_js(jscript, jslib = '')	
execjs(js, jslib, timeout, force_docker_pull = False, debug = False, js_console = False, container_engine = 'docker')	

Attributes

localdata
default_timeout
have_node_slim
minimum_node_version_str
PROCESS_FINISHED_STR

exception cwltool.sandboxjs.**JavascriptException**

Bases: Exception

Common base class for all non-exit exceptions.

cwltool.sandboxjs.**localdata**

cwltool.sandboxjs.**default_timeout = 20**

cwltool.sandboxjs.**have_node_slim = False**

cwltool.sandboxjs.**minimum_node_version_str = 0.10.26**

cwltool.sandboxjs.**check_js_threshold_version**(*working_alias*)

Check if the nodeJS engine version on the system with the allowed minimum version.

https://github.com/nodejs/node/blob/master/CHANGELOG.md#nodejs-changelog

Parameters **working_alias** (*str*) –

Return type bool

cwltool.sandboxjs.**new_js_proc**(*js_text*, *force_docker_pull=False*, *container_engine='docker'*)

Return a subprocess ready to submit javascript to.

Parameters

- **js_text** (*str*) –
- **force_docker_pull** (*bool*) –
- **container_engine** (*str*) –

Return type subprocess.Popen[str]

cwltool.sandboxjs.**PROCESS_FINISHED_STR = r1cepzbhUTxtykz5XTC4**

cwltool.sandboxjs.**exec_js_process**(*js_text*, *timeout=default_timeout*, *js_console=False*, *context=None*, *force_docker_pull=False*, *container_engine='docker'*)

Parameters

- **js_text** (*str*) –
- **timeout** (*float*) –
- **js_console** (*bool*) –
- **context** (*Optional[str]*) –
- **force_docker_pull** (*bool*) –
- **container_engine** (*str*) –

Return type Tuple[int, str, str]

cwltool.sandboxjs.**code_fragment_to_js**(*jscript*, *jslib=''*)

Parameters

- **jscript** (*str*) –
- **jslib** (*str*) –

Return type str

cwltool.sandboxjs.**execjs**(*js*, *jslib*, *timeout*, *force_docker_pull=False*, *debug=False*, *js_console=False*, *container_engine='docker'*)

Parameters

- **js** (*str*) –
- **jslib** (*str*) –
- **timeout** (*float*) –
- **force_docker_pull** (*bool*) –
- **debug** (*bool*) –
- **js_console** (*bool*) –
- **container_engine** (*str*) –

Return type cwltool.utils.CWLOutputType

cwltool.secrets

Minimal in memory storage of secrets.

Module Contents

Classes

SecretStore	Minimal implementation of a secret storage.

class cwltool.secrets.**SecretStore**

Minimal implementation of a secret storage.

add(*self*, *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(*self*, *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(*self*, *value*)

Test if the provided document has any of our secrets.

Parameters **value** (*cwltool.utils.CWLOutputType*) –

Return type bool

retrieve(*self*, *value*)

Replace placeholders with their corresponding secrets.

Parameters **value** (*cwltool.utils.CWLOutputType*) –

Return type cwltool.utils.CWLOutputType

cwltool.singularity

Support for executing Docker containers using the Singularity 2.x engine.

Module Contents

Classes

SingularityCommandLineJob	Commandline job using containers.

Functions

get_version()	
is_version_2_6()	
is_version_3_or_newer()	
is_version_3_1_or_newer()	
is_version_3_4_or_newer()	Detect if Singularity v3.4+ is available.

cwltool.singularity.**get_version**()

Return type str

cwltool.singularity.**is_version_2_6**()

Return type bool

cwltool.singularity.**is_version_3_or_newer**()

Return type bool

cwltool.singularity.**is_version_3_1_or_newer**()

Return type bool

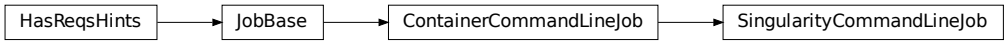
cwltool.singularity.**is_version_3_4_or_newer**()

Detect if Singularity v3.4+ 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[*Ellipsis*, cwltool.pathmapper.PathMapper]) –
- **requirements** (*List[cwltool.utils.CWLObjectType]*) –
- **hints** (*List[cwltool.utils.CWLObjectType]*) –
- **name** (*str*) –

static get_image(*dockerRequirement*, *pull_image*, *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*) –
- **force_pull** (*bool*) –

Return type bool

get_from_requirements(*self*, *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(*self*, *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(*self*, *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(*self*, *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(*self*, *env*, *runtime_context*)

Return the Singularity runtime list of commands and options.

Parameters

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

Return type Tuple[List[str], Optional[str]]

cwltool.singularity_utils

Support for executing Docker containers using the Singularity 2.x engine.

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

DependenciesConfiguration	Dependency configuration class, for RuntimeContext.job_script_provider.

Functions

get_dependencies(builder)	
get_container_from_software_requirements(use_biocontainers, builder)	
ensure_galaxy_lib_available()	

Attributes

ToolRequirement	
SOFTWARE_REQUIREMENTS_ENABLED	
COMMAND_WITH_DEPENDENCIES_TEMPLATE	

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(*self*, *builder*, *command*)

Parameters

- **builder** (cwltool.builder.Builder) –

- **command** (*List[str]*) –

Return type str

cwltool.software_requirements.**get_dependencies**(*builder*)

Parameters **builder** (cwltool.utils.HasReqsHints) –

Return type galaxy.tool_util.deps.requirements.ToolRequirements

cwltool.software_requirements.**get_container_from_software_requirements**(*use_biocontainers*, *builder*)

Parameters

- **use_biocontainers** (*bool*) –
- **builder** (cwltool.utils.HasReqsHints) –

Return type Optional[str]

cwltool.software_requirements.**ensure_galaxy_lib_available**()

Return type None

cwltool.stdfsaccess

Abstracted IO access.

Module Contents

Classes

StdFsAccess	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(*self*, *pattern*)

Parameters **pattern** (*str*) –

Return type List[str]

open(*self*, *fn*, *mode*)

Parameters

- **fn** (*str*) –
- **mode** (*str*) –

Return type IO[Any]

exists(*self*, *fn*)

Parameters **fn** (*str*) –

Return type bool

size(*self*, *fn*)

Parameters **fn** (*str*) –

Return type int

isfile(*self*, *fn*)

Parameters **fn** (*str*) –

Return type bool

isdir(*self*, *fn*)

Parameters **fn** (*str*) –

Return type bool

listdir(*self*, *fn*)

Parameters **fn** (*str*) –

Return type List[str]

join(*self*, *path*, **paths*)

Parameters

- **self** (*str*) –
- **path** (*str*) –

Return type str

realpath(*self*, *path*)

Parameters **path** (*str*) –

Return type str

cwltool.subgraph

Module Contents

Functions

subgraph_visit(current, nodes, visited, direction)	
declare_node(nodes, nodeid, tp)	
find_step(steps, stepid, loading_context)	Find the step (raw dictionary and WorkflowStep) for a given step id.
get_subgraph(roots, tool, loading_context)	Extract the subgraph for the given roots.
get_step(tool, step_id, loading_context)	Extract a single WorkflowStep for the given step_id.
get_process(tool, step_id, loading_context)	Find the underlying Process for a given Workflow step id.

Attributes

Node
UP
DOWN
INPUT
OUTPUT
STEP

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.CommentedMap

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.CommentedMap

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

TaskQueue	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(*self*, *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(*self*)

Drain the queue.

Return type None

join(*self*)

Wait for all threads to complete.

Return type None

cwltool.udocker

Enables Docker software containers via the udocker runtime.

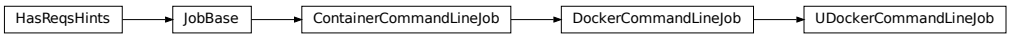
Module Contents

Classes

UDockerCommandLineJob	Runs a CommandLineJob 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[*Ellipsis*, cwltool.pathmapper.PathMapper]) –
- **requirements** (*List[cwltool.utils.CWLObjectType]*) –
- **hints** (*List[cwltool.utils.CWLObjectType]*) –
- **name** (*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

cwltool.update

Module Contents

Functions

v1_1to1_2(doc, loader, baseuri)	Public updater for v1.1 to v1.2.
v1_0to1_1(doc, loader, baseuri)	Public updater for v1.0 to v1.1.
v1_1_0dev1to1_1(doc, loader, baseuri)	Public updater for v1.1.0-dev1 to v1.1.
v1_2_0dev1todev2(doc, loader, baseuri)	Public updater for v1.2.0-dev1 to v1.2.0-dev2.
v1_2_0dev2todev3(doc, loader, baseuri)	Public updater for v1.2.0-dev2 to v1.2.0-dev3.
v1_2_0dev3todev4(doc, loader, baseuri)	Public updater for v1.2.0-dev3 to v1.2.0-dev4.
v1_2_0dev4todev5(doc, loader, baseuri)	Public updater for v1.2.0-dev4 to v1.2.0-dev5.
v1_2_0dev5to1_2(doc, loader, baseuri)	Public updater for v1.2.0-dev5 to v1.2.
identity(doc, loader, baseuri)	Do-nothing, CWL document upgrade function.
checkversion(doc, metadata, enable_dev)	Check the validity of the version of the give CWL document.
update(doc, loader, baseuri, enable_dev, metadata, update_to = None)	Update a CWL document to 'update_to' (if provided) or INTERNAL_VERSION.

Attributes

ORDERED_VERSIONS
UPDATES
DEVUPDATES
ALLUPDATES
INTERNAL_VERSION
ORIGINAL_CWLVERSION

cwltool.update.**v1_1to1_2**(*doc*, *loader*, *baseuri*)

Public updater for v1.1 to v1.2.

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_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.CommentedMap*) –
- **loader** (*schema_salad.ref_resolver.Loader*) –

- **baseuri** (*str*) –

Return type Tuple[ruamel.yaml.comments.CommentedMap, 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.CommentedMap*) –
- **loader** (*schema_salad.ref_resolver.Loader*) –
- **baseuri** (*str*) –

Return type Tuple[ruamel.yaml.comments.CommentedMap, 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, Optional[Callable[[CommentedMap, Loader, str], Tuple[CommentedMap, str]]]]**

cwltool.update.**DEVUPDATES :Dict[str, Optional[Callable[[CommentedMap, Loader, str], Tuple[CommentedMap, str]]]]**

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.CommentedMap*) –
- **loader** (*schema_salad.ref_resolver.Loader*) –
- **baseuri** (*str*) –

Return type Tuple[ruamel.yaml.comments.CommentedMap, 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.

Module Contents

Classes

HasReqsHints	Base class for get_requirement().

Functions

versionstring()	Version of CWLtool used to execute the workflow.
aslist(thing)	Wrap any non-MutableSequence/list in a list.
copytree_with_merge(src, dst)	
cmp_like_py2(dict1, dict2)	Compare in the same manner as Python2.
bytes2str_in_dicts(inp)	Convert any present byte string to unicode string, in-place.
visit_class(rec, cls, op)	Apply a function to with "class" in cls.
visit_field(rec, field, op)	Apply a function to mapping with 'field'.
random_outdir()	Return the random directory name chosen to use for tool / workflow output.
shared_file_lock(fd)	
upgrade_lock(fd)	
adjustFileObjs(rec, op)	Apply an update function to each File object in the object *rec*.
adjustDirObjs(rec, op)	Apply an update function to each Directory object in the object *rec*.
dedup(listing)	
get_listing(fs_access, rec, recursive = True)	
trim_listing(obj)	Remove 'listing' field from Directory objects that are file references.
downloadHttpFile(httpurl)	
ensure_writable(path, include_root = False)	Ensure that 'path' is writable.
ensure_non_writable(path)	
normalizeFilesDirs(job)	
posix_path(local_path)	
local_path(posix_path)	
create_tmp_dir(tmpdir_prefix)	Create a temporary directory that respects the given tmpdir_prefix.

Attributes

CONTENT_LIMIT
DEFAULT_TMP_PREFIX
processes_to_kill
CWLOutputAtomType
CWLOutputType
CWLObjectType
JobsType
JobsGeneratorType
OutputCallbackType
ResolverType
DestinationsType
ScatterDestinationsType
ScatterOutputCallbackType
SinkType
DirectoryType
JSONAtomType
JSONType
WorkflowStateItem
ParametersType
StepType
fcntl
msvcrt

cwltool.utils.**CONTENT_LIMIT**

cwltool.utils.**DEFAULT_TMP_PREFIX**

cwltool.utils.**processes_to_kill :Deque[subprocess.Popen[str]]**

cwltool.utils.**CWLOutputAtomType**

cwltool.utils.**CWLOutputType**

cwltool.utils.**CWLObjectType**

cwltool.utils.**JobsType**

cwltool.utils.**JobsGeneratorType**

cwltool.utils.**OutputCallbackType**

cwltool.utils.**ResolverType**

cwltool.utils.**DestinationsType**

cwltool.utils.**ScatterDestinationsType**

cwltool.utils.**ScatterOutputCallbackType**

cwltool.utils.**SinkType**

cwltool.utils.**DirectoryType**

cwltool.utils.**JSONAtomType**

cwltool.utils.**JSONType**

cwltool.utils.**WorkflowStateItem**

cwltool.utils.**ParametersType**

cwltool.utils.**StepType**

cwltool.utils.**versionstring**()

Version of CWLtool used to execute the workflow.

Return type str

cwltool.utils.**aslist**(*thing*)

Wrap any non-MutableSequence/list in a list.

Parameters **thing** (*Any*) –

Return type MutableSequence[Any]

cwltool.utils.**copytree_with_merge**(*src*, *dst*)

Parameters

- **src** (*str*) –
- **dst** (*str*) –

Return type None

cwltool.utils.**cmp_like_py2**(*dict1*, *dict2*)

Compare in the same manner as Python2.

Comparison function to be used in sorting as python3 doesn't allow sorting of different types like str() and int(). This function re-creates sorting nature in py2 of heterogeneous list of *int* and *str*

Parameters

- **dict1** (*Dict[str, Any]*) –
- **dict2** (*Dict[str, Any]*) –

Return type int

cwltool.utils.**bytes2str_in_dicts**(*inp*)

Convert any present byte string to unicode string, inplace.

input is a dict of nested dicts and lists

Parameters **inp** (*Union[MutableMapping[str, Any], MutableSequence[Any], Any]*) –

Return type Union[str, MutableSequence[Any], MutableMapping[str, Any]]

cwltool.utils.**visit_class**(*rec*, *cls*, *op*)

Apply a function to with "class" in cls.

Parameters

- **rec** (*Any*) –
- **cls** (*Iterable[Any]*) –
- **op** (*Callable[Ellipsis, Any]*) –

Return type None

cwltool.utils.**visit_field**(*rec*, *field*, *op*)

Apply a function to mapping with 'field'.

Parameters

- **rec** (*Any*) –
- **field** (*str*) –
- **op** (*Callable[Ellipsis, Any]*) –

Return type None

cwltool.utils.**random_outdir**()

Return the random directory name chosen to use for tool / workflow output.

Return type str

cwltool.utils.**fcntl :Optional[ModuleType]**

cwltool.utils.**msvcrt :Optional[ModuleType]**

cwltool.utils.**shared_file_lock**(*fd*)

Parameters **fd** (*IO[Any]*) –

Return type None

cwltool.utils.**upgrade_lock**(*fd*)

Parameters **fd** (*IO[Any]*) –

Return type None

cwltool.utils.**adjustFileObjs**(*rec*, *op*)

Apply an update function to each File object in the object *rec*.

Parameters

- **rec** (*Any*) –
- **op** (*Union[Callable[Ellipsis, Any], functools.partial[Any]]*) –

Return type None

cwltool.utils.**adjustDirObjs**(*rec*, *op*)

Apply an update function to each Directory object in the object *rec*.

Parameters

- **rec** (*Any*) –
- **op** (*Union[Callable[Ellipsis, Any], functools.partial[Any]]*) –

Return type None

cwltool.utils.**dedup**(*listing*)

Parameters **listing** (*List[CWLObjectType]*) –

Return type List[CWLObjectType]

cwltool.utils.**get_listing**(*fs_access*, *rec*, *recursive=True*)

Parameters

- **fs_access** (cwltool.stdfsaccess.StdFsAccess) –
- **rec** (*CWLObjectType*) –
- **recursive** (*bool*) –

Return type None

cwltool.utils.**trim_listing**(*obj*)

Remove 'listing' field from Directory objects that are file references.

It redundant and potentially expensive to pass fully enumerated Directory objects around if not explicitly needed, so delete the 'listing' field when it is safe to do so.

Parameters **obj** (*Dict[str, Any]*) –

Return type None

cwltool.utils.**downloadHttpFile**(*httpurl*)

Parameters **httpurl** (*str*) –

Return type str

cwltool.utils.**ensure_writable**(*path*, *include_root=False*)

Ensure that 'path' is writable.

If 'path' is a directory, then all files and directories under 'path' are made writable, recursively. If 'path' is a file or if 'include_root' is *True*, then 'path' itself is made writable.

Parameters

- **path** (*str*) –
- **include_root** (*bool*) –

Return type None

cwltool.utils.**ensure_non_writable**(*path*)

Parameters **path** (*str*) –

Return type None

cwltool.utils.**normalizeFilesDirs**(*job*)

Parameters **job** (*Optional[Union[MutableSequence[MutableMapping[str, Any]], MutableMapping[str, Any], DirectoryType]]*) –

Return type None

cwltool.utils.**posix_path**(*local_path*)

Parameters **local_path** (*str*) –

Return type str

cwltool.utils.**local_path**(*posix_path*)

Parameters **posix_path** (*str*) –

Return type str

cwltool.utils.**create_tmp_dir**(*tmpdir_prefix*)

Create a temporary directory that respects the given tmpdir_prefix.

Parameters **tmpdir_prefix** (*str*) –

Return type str

class cwltool.utils.**HasReqsHints**

Base class for get_requirement().

get_requirement(*self*, *feature*)

Parameters **feature** (*str*) –

Return type Tuple[Optional[CWLObjectType], Optional[bool]]

cwltool.validate_js

Module Contents

Classes

SuppressLog	Filter instances are used to perform arbitrary filtering of LogRecords.

Functions

is_expression(tool, schema)	
get_expressions(tool, schema, source_line = None)	
jshint_js(js_text, globals = None, options = None, container_engine = 'docker')	
print_js_hint_messages(js_hint_messages, source_line)	
validate_js_expressions(tool, schema, jshint_options = None, container_engine = 'docker')	

Attributes

JSHintJSReturn

cwltool.validate_js.**is_expression**(*tool*, *schema*)

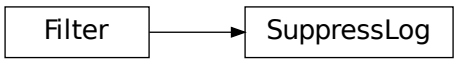
Parameters

- **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 **self** (*str*) –

filter(*self*, *record*)

Determine if the specified record is to be logged.

Is the specified record to be logged? Returns 0 for no, nonzero for yes. If deemed appropriate, the record may be modified in-place.

Parameters **self** (*logging.LogRecord*) –

Return type bool

cwltool.validate_js.**get_expressions**(*tool*, *schema*, *source_line=None*)

Parameters

- **tool** (*Union[ruamel.yaml.comments.CommentedMap, 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'*)

Parameters

- **js_text** (*str*) –
- **globals** (*Optional[List[str]]*) –
- **options** (*Optional[Dict[str, Union[List[str], str, int]]]*) –
- **container_engine** (*str*) –

Return type JSHintJSReturn

cwltool.validate_js.**print_js_hint_messages**(*js_hint_messages*, *source_line*)

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'*)

Parameters

- **tool** (*ruamel.yaml.comments.CommentedMap*) –
- **schema** (*schema_salad.avro.schema.Schema*) –
- **jshint_options** (*Optional[Dict[str, Union[List[str], str, int]]]*) –
- **container_engine** (*str*) –

Return type None

cwltool.workflow

Module Contents

Classes

Workflow	Base class for get_requirement().
WorkflowStep	Base class for get_requirement().

Functions

default_make_tool(toolpath_object, loadingContext)	
used_by_step(step, shortinputid)	

Attributes

default_make_tool(toolpath_object, loadingContext)	

cwltool.workflow.**default_make_tool**(*toolpath_object*, *loadingContext*)

Parameters

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

Return type *cwltool.process.Process*

cwltool.workflow.**default_make_tool**

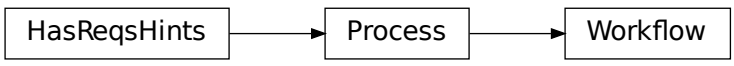
Parameters

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

Return type *cwltool.process.Process*

class cwltool.workflow.**Workflow**(*toolpath_object*, *loadingContext*)

Bases: *cwltool.process.Process*



Base class for get_requirement().

Parameters

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

make_workflow_step(*self*, *toolpath_object*, *pos*, *loadingContext*, *parentworkflowProv=None*)

Parameters

- **toolpath_object** (*ruamel.yaml.comments.CommentedMap*) –
- **pos** (*int*) –
- **loadingContext** (cwltool.context.LoadingContext) –
- **parentworkflowProv** (*Optional[*cwltool.provenance_profile.ProvenanceProfile*]*) –

Return type *WorkflowStep*

job(*self*, *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(*self*, *op*)

Parameters **op** (*Callable[[ruamel.yaml.comments.CommentedMap], None]*) –

Return type None

cwltool.workflow.**used_by_step**(*step*, *shortinputid*)

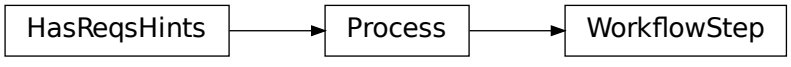
Parameters

- **step** (*cwltool.utils.StepType*) –
- **shortinputid** (*str*) –

Return type bool

class cwltool.workflow.**WorkflowStep**(*toolpath_object*, *pos*, *loadingContext*, *parentworkflowProv=None*)

Bases: *cwltool.process.Process*



Base class for get_requirement().

Parameters

- **toolpath_object** (*ruamel.yaml.comments.CommentedMap*) –
- **pos** (*int*) –
- **loadingContext** (cwltool.context.LoadingContext) –
- **parentworkflowProv** (*Optional[*cwltool.provenance_profile.ProvenanceProfile*]*) –

receive_output(*self*, *output_callback*, *jobout*, *processStatus*)

Parameters

- **output_callback** (*cwltool.utils.OutputCallbackType*) –
- **jobout** (*cwltool.utils.CWLObjectType*) –
- **processStatus** (*str*) –

Return type None

job(*self*, *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(*self*, *op*)

Parameters **op** (*Callable[[ruamel.yaml.comments.CommentedMap], None]*) –

Return type None

cwltool.workflow_job

Module Contents

Classes

WorkflowJobStep	Generated for each step in Workflow.steps().
ReceiveScatterOutput	Produced by the scatter generators.
WorkflowJob	Generates steps from the Workflow.

Functions

parallel_steps(steps, rc, runtimeContext)	
nested_crossproduct_scatter(process, joborder, scatter_keys, output_callback, runtimeContext)	
crossproduct_size(joborder, scatter_keys)	
flat_crossproduct_scatter(process, joborder, scatter_keys, output_callback, runtimeContext)	
dotproduct_scatter(process, joborder, scatter_keys, output_callback, runtimeContext)	
match_types(sinktype, src, iid, inputobj, linkMerge, valueFrom)	
object_from_state(state, params, frag_only, supportsMultipleInput, sourceField, incomplete = False)	

class cwltool.workflow_job.**WorkflowJobStep**(*step*)

Generated for each step in Workflow.steps().

Parameters **step** (cwltool.workflow.WorkflowStep) –

job(*self*, *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(*self*, *index*, *jobout*, *processStatus*)

Parameters

- **index** (*int*) –
- **jobout** (*cwltool.utils.CWLObjectType*) –
- **processStatus** (*str*) –

Return type None

setTotal(*self*, *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*)

Parameters

- **joborder** (*cwltool.utils.CWLObjectType*) –
- **scatter_keys** (*MutableSequence[str]*) –

Return type int

cwltool.workflow_job.**flat_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.**dotproduct_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.**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(*self*, *final_output_callback*)

Parameters **final_output_callback** (*cwltool.utils.OutputCallbackType*) –

Return type None

receive_output(*self*, *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(*self*, *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(*self*, *runtimeContext*, *tmpdir_lock=None*)

Log the start of each workflow.

Parameters

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

Return type None

job(*self*, *joborder*, *output_callback*, *runtimeContext*)

Parameters

- **joborder** (*cwltool.utils.CWLObjectType*) –
- **output_callback** (*Optional[cwltool.utils.OutputCallbackType]*) –
- **runtimeContext** (cwltool.context.RuntimeContext) –

Return type cwltool.utils.JobsGeneratorType

Package Contents

cwltool.**__author__ = pamstutz@veritasgenetics.com**

cwltool.**CWL_CONTENT_TYPES = ['text/plain', 'application/json', 'text/vnd.yaml', 'text/yaml', 'text/x-yaml',...**

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

C

INDEX

Symbols

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

R

S

T

U

V

W