Package 'Rcwl'

November 12, 2026

Title An R interface to the Common Workflow Language

Version 1.28.0

Description The Common Workflow Language (CWL) is an open standard for development of data analysis workflows that is portable and scalable across different tools and working environments. Rcwl provides a simple way to wrap command line tools and build CWL data analysis pipelines programmatically within R. It increases the ease of usage, development, and maintenance of CWL pipelines.

Depends R (>= 3.6), yaml, methods, S4Vectors

Imports utils, stats, BiocParallel, batchtools, DiagrammeR, shiny, R.utils, codetools, basilisk

License GPL-2 | file LICENSE

Encoding UTF-8

LazyData true

Suggests testthat, knitr, rmarkdown, BiocStyle

VignetteBuilder knitr

RoxygenNote 7.3.1

biocViews Software, WorkflowStep, ImmunoOncology

StagedInstall no

Config/Bioconductor/UnsupportedPlatforms windows

git_url https://git.bioconductor.org/packages/Rcwl

git_branch RELEASE_3_23

git_last_commit d4c58a5

git_last_commit_date 2026-04-28

Repository Bioconductor 3.23

Date/Publication 2026-11-11

Author Qiang Hu [aut, cre], Qian Liu [aut]

Maintainer Qiang Hu <qiang.hu@roswellpark.org>

Contents

Rcwl-package . . . 2
cwl-requirements . . . 3
cwlProcess . . . 7
cwlProcess-methods . . . 8
cwlShiny . . . 10
cwlStep . . . 11
cwlWorkflow . . . 12
cwlWorkflow-methods . . . 14
env_Rcwl . . . 15
InputArrayParam-class . . . 15
install_cwltool . . . 19
install_udocker . . . 19
meta . . . 20
plotCWL . . . 22
readCWL . . . 23
runCWL . . . 23
runCWLBatch . . . 25
runCWLBP . . . 25
stepInputs . . . 26
stepOutputs . . . 26
writeCWL . . . 27

Index **28**

`Rcwl-package` *Rcwl-package*

Description

An R package to wrap command line tools and build pipelines with Common Workflow Language (CWL). _PACKAGE

See Also

`cwlProcess`

`cwlWorkflow`

`cwlStep`

`runCWL`

cwl-requirements *CWL requirements functions*

Description

requireDocker: If a workflow component should be run in a Docker container, this function specifies how to fetch or build the image.

requireJS: Indicates that the workflow platform must support inline Javascript expressions. If this requirement is not present, the workflow platform must not perform expression interpolatation.

requireSoftware: A list of software packages that should be configured in the environment of the defined process.

SoftwarePackage from anaconda.

InitialWorkDirRequirement: Define a list of files and subdirectories that must be created by the workflow platform in the designated output directory prior to executing the command line tool.

: Dirent: Define a file or subdirectory that must be placed in the designated output directory prior to executing the command line tool. May be the result of executing an expression, such as building a configuration file from a template.

Create manifest for configure files.

requireShellScript: create shell script to work dir.

CondaTool: create dockerfile for tools.

requireNetwork: Whether a process requires network access.

Usage

```
requireDocker(
  docker = NULL,
  Load = NULL,
  File = NULL,
  Import = NULL,
  ImageId = NULL,
  OutputDir = NULL
)

requireJS(expressionLib = list())

requireSoftware(packages = list())

condaPackage(package, source = "bioconda", version = NULL)

requireInitialWorkDir(listing = list())

Dirent(entryname = character(), entry, writable = FALSE)

requireManifest(inputID, sep = "\\n")
```

```
requireSubworkflow()

requireScatter()

requireMultipleInput()

requireStepInputExpression()

requireEnvVar(envlist)

requireRscript(rscript)

requireResource(
  coresMin = NULL,
  coresMax = NULL,
  ramMin = NULL,
  ramMax = NULL,
  tmpdirMin = NULL,
  tmpdirMax = NULL,
  outdirMin = NULL,
  outdirMax = NULL
)

requireShellCommand()

requireShellScript(script)

ShellScript(shell = "bash", script = "script.sh")

CondaTool(tools)

requireNetwork(networkAccess = TRUE)
```

Arguments

docker	Character. Specify a Docker image to retrieve using docker pull.
Load	Character. Specify a HTTP URL from which to download a Docker image using docker load.
File	Character. Supply the contents of a Dockerfile which will be built using docker build.
Import	Character. Provide HTTP URL to download and gunzip a Docker images using ‘docker import.
ImageId	Character. The image id that will be used for docker run. May be a human-readable image name or the image identifier hash. May be skipped if dockerPull is specified, in which case the dockerPull image id must be used.
OutputDir	Character. Set the designated output directory to a specific location inside the Docker container.

expressionLib	optional list. Additional code fragments that will also be inserted before executing the expression code. Allows for function definitions that may be called from CWL expressions.
packages	The list of software to be configured.
package	The software name.
source	The source of software in anaconda. 'bioconda' is used by default.
version	The version of software.
listing	The list of files or subdirectories that must be placed in the designated output directory prior to executing the command line tool.
entryname	Character or Expression. The name of the file or subdirectory to create in the output directory. The name of the file or subdirectory to create in the output directory. If entry is a File or Directory, the entryname field overrides the value of basename of the File or Directory object. Optional.
entry	Charactor or expression. Required.
writable	Logical. If true, the file or directory must be writable by the tool. Changes to the file or directory must be isolated and not visible by any other CommandLineTool process. Default is FALSE (files and directories are read-only). Optional.
inputID	The input ID from corresponding 'InputParam'.
sep	The separator of the input files in the manifest config.
envlist	A list of environment variables.
rscript	An R script to run.
coresMin	Minimum reserved number of CPU cores (default is 1).
coresMax	Maximum reserved number of CPU cores.
ramMin	Minimum reserved RAM in mebibytes (2**20) (default is 256).
ramMax	Maximum reserved RAM in mebibytes (2**20)
tmpdirMin	Minimum reserved filesystem based storage for the designated temporary directory, in mebibytes (2**20) (default is 1024).
tmpdirMax	Maximum reserved filesystem based storage for the designated temporary directory, in mebibytes (2**20).
outdirMin	Minimum reserved filesystem based storage for the designated output directory, in mebibytes (2**20) (default is 1024).
outdirMax	Maximum reserved filesystem based storage for the designated output directory, in mebibytes (2**20).
script	script.sh
shell	Default shell.
tools	A character vector for tools to install by conda.
networkAccess	TRUE or FALSE.

Details

More details about 'requireDocker', see: https://www.commonwl.org/v1.0/CommandLineTool.html#DockerRequirement

More details about 'requireJS', see: https://www.commonwl.org/v1.0/CommandLineTool.html#InlineJavascriptRequirement

More details about 'requireSoftware', see: https://www.commonwl.org/v1.0/CommandLineTool.html#SoftwareRequirement

More details about 'requireSoftware', see: https://www.commonwl.org/v1.0/CommandLineTool.html#SoftwarePackage

More details about 'requireInitialWorkDir', See: https://www.commonwl.org/v1.0/CommandLineTool.html#InitialWorkDirR

More details about 'Dirent', See:https://www.commonwl.org/v1.0/CommandLineTool.html#Dirent

Value

requireDocker: A list of docker requirement to fetch or build the image.

requireJS: A list of inline Javascript requirement.

requireSoftware: A list of software requirements.

A list of software package.

requireInitialWorkDir: A list of initial work directory requirements.

Dirent: A list.

requireSubworkflow: A SubworkflowFeatureRequirement list.

rquireScatter: A ScatterFeatureRequirement list.

requireMultipleInput: A MultipleInputFeatureRequirement list.

requireStepInputExpression: A StepInputExpressionRequirement list.

requireEnvVar: A EnvVarRequirementlist.

A requirement list with Rscript as manifest entry.

ResourceRequirement: A ResourceRequirement list.

ShellCommandRequirement: A ShellCommandRequirement list.

requireShellScript: Initial directory with shell script.

baseCommand for shell script

CondaTool: Dockerfile

requireNetwork: a list of NetworkAccess requirement.

Examples

```
p1 <- InputParam(id = "ifiles", type = "File[]?", position = -1)
CAT <- cwlProcess(baseCommand = "cat",
     requirements = list(requireDocker("alpine"), requireManifest("ifiles"), requireJS()),
       arguments = list("ifiles"),
       inputs = InputParamList(p1))
```

cwlProcess *Parameters for CWL*

Description

The main CWL parameter class and constructor for command tools. More details: https://www.commonwl.org/v1.0/Comman

Usage

```
cwlProcess(
  cwlVersion = "v1.0",
  cwlClass = "CommandLineTool",
  baseCommand = character(),
  requirements = list(),
  hints = list(),
  arguments = list(),
  id = character(),
  label = character(),
  doc = character(),
  inputs = InputParamList(),
  outputs = OutputParamList(),
  stdout = character(),
  stdin = character(),
  expression = character(),
  extensions = list(),
  intent = list()
)
```

Arguments

cwlVersion	CWL version
cwlClass	"CommandLineTool"
baseCommand	Specifies the program or R function to execute
requirements	A list of requirements that apply to either the runtime environment or the workflow engine that must be met in order to execute this process.
hints	Any or a list for the workflow engine.
arguments	Command line bindings which are not directly associated with input parameters.
id	The unique identifier for this process object.
label	A short, human-readable label of this process object.
doc	A documentation string for this object, or an array of strings which should be concatenated.
inputs	A object of 'InputParamList'.
outputs	A object of 'OutputParamList'.

stdout	Capture the command's standard output stream to a file written to the designated output directory.
stdin	A path to a file whose contents must be piped into the command's standard input stream.
expression	Javascripts for ExpressionTool class.
extensions	A list of extensions and metadata
intent	An identifier for the type of computational operation, of this Process.

Details

https://www.commonwl.org/v1.0/CommandLineTool.html

Value

A 'cwlProcess' class object.

Examples

```
input1 <- InputParam(id = "sth")
echo <- cwlProcess(baseCommand = "echo", inputs = InputParamList(input1))
```

---

cwlProcess-methods *cwlProcess methods*

Description

Some useful methods for 'cwlProcess' objects.

'$': Extract input values for 'cwlProcess' object. (Can auto-complete the input names using tab)

'$<-': Set input values for 'cwlProcess' object by name.

outputs: The outputs of a 'cwlProcess' object.

stdOut: stdout of 'cwlProcess' object.

extensions: Extensions and metadata of 'cwlProcess' object.

short: The function to show a short summary of 'cwlProcess' or 'cwlWorkflow' object.

Usage

```
cwlVersion(cwl)

cwlVersion(cwl) <- value

cwlClass(cwl)

cwlClass(cwl) <- value

baseCommand(cwl)
```

```
baseCommand(cwl) <- value

arguments(cwl, step = NULL)

arguments(cwl, step = NULL) <- value

hints(cwl)

hints(cwl) <- value

requirements(cwl, step = NULL)

requirements(cwl, step = NULL) <- value

inputs(cwl)

## S4 method for signature 'cwlProcess'
x$name

## S4 replacement method for signature 'cwlProcess'
x$name <- value

outputs(cwl)

stdOut(cwl)

stdOut(cwl) <- value

extensions(cwl)

extensions(cwl) <- value

short(cwl)
```

Arguments

cwl	A ‘cwlProcess‘ (or ‘cwlWorkflow‘) object.
value	To assign a list of ‘requirements‘ value.
step	To specifiy a step ID when ‘cwl‘ is a workflow.
x	A ‘cwlProcess‘ object.
name	One of input list.

Value

cwlVersion: cwl document version

cwlClass: CWL class of ‘cwlProcess‘ or ‘cwlWorkflow‘ object.

baseCommand: base command for the ‘cwlProcess‘ object.

arguments: CWL arguments.

hints: CWL hints.

requirements: CWL requirments.

inputs: A list of ‘InputParam‘.

‘$‘: the ‘InputParam‘ value for ‘cwlProcess‘ object.

outputs: A list of ‘OutputParam‘.

stdOut: CWL stdout.

extensions: A list of extensions or metadata.

short: A short summary of an object of ‘cwlProcess‘ or ‘cwlWorkflow‘.

Examples

```
ip <- InputParam(id = "sth")
echo <- cwlProcess(baseCommand = "echo", inputs = InputParamList(ip))
cwlVersion(echo)
cwlClass(echo)
baseCommand(echo)
hints(echo)
requirements(echo)
inputs(echo)
outputs(echo)
stdOut(echo)
extensions(echo)

s1 <- cwlWorkflow()
runs(s1)
s1
short(s1)
```

---

cwlShiny *cwlShiny*

Description

Function to generate shiny app automaticlly for a ‘cwlProcess‘ object.

Usage

```
cwlShiny(cwl, inputList = list(), upload = FALSE, ...)
```

Arguments

cwl	A cwlProcess object.
inputList	a list of choices for the inputs of cwl object. The name of the list must match the inputs of the cwl object.
upload	Whether to upload file. If FALSE, the upload field will be text input (file path) instead of file input.
...	More options for 'runCWL'.

Value

A shiny webapp.

Examples

```
input1 <- InputParam(id = "sth")
echo <- cwlProcess(baseCommand = "echo", inputs = InputParamList(input1))
echoApp <- cwlShiny(echo)
```

---

cwlStep *cwlStep function*

Description

Constructor function for 'cwlStep' object.

Usage

```
cwlStep(
  id,
  run = cwlProcess(),
  In = list(),
  Out = list(),
  scatter = character(),
  scatterMethod = character(),
  label = character(),
  doc = character(),
  requirements = list(),
  hints = list(),
  when = character()
)
```

Arguments

id	A user-defined unique identifier for this workflow step.
run	A 'cwlProcess' object for command line tool, or path to a CWL file.
In	A list of input parameters which will be constructed into 'stepInParamList'.
Out	A list of outputs.
scatter	character or a list. The inputs to be scattered.
scatterMethod	required if scatter is an array of more than one element. It can be one of "dotproduct", "nested_crossproduct" and "flat_crossproduct".
label	A short, human-readable label of this object.
doc	A documentation string for this object, or an array of strings which should be concatenated.
requirements	Requirements that apply to either the runtime environment or the workflow engine.
hints	Hints applying to either the runtime environment or the workflow engine.
when	If defined, only run the step when the expression evaluates to true. If false the step is skipped.

Details

For more details: https://www.commonwl.org/v1.0/Workflow.html#WorkflowStep

Value

An object of class 'cwlStep'.

See Also

cwlWorkflow

Examples

```
s1 <- cwlStep(id = "s1")
```

---

cwlWorkflow *cwlWorkflow function*

Description

The constructor function for 'cwlWorkflow' object, which connects multiple command line steps into a workflow.

steps: Function to extract and assign workflow step slots.

Usage

```
cwlWorkflow(
  cwlVersion = "v1.0",
  cwlClass = "Workflow",
  requirements = list(),
  id = character(),
  label = character(),
  doc = character(),
  intent = list(),
  hints = list(),
  arguments = list(),
  extensions = list(),
  inputs = InputParamList(),
  outputs = OutputParamList(),
  steps = cwlStepList()
)

## S4 method for signature 'cwlWorkflow,cwlStep'
e1 + e2

steps(cwl)

steps(cwl) <- value
```

Arguments

cwlVersion	CWL version
cwlClass	"Workflow".
requirements	Requirements that apply to either the runtime environment or the workflow engine.
id	A user-defined unique identifier for this workflow.
label	A short, human-readable label of this object.
doc	A documentation string for this object.
intent	An identifier for the type of computational operation, of this Process.
hints	Any or a list for the workflow engine.
arguments	Command line bindings which are not directly associated with input parameters.
extensions	A list of extensions and metadata.
inputs	An object of ‘InputParamList‘.
outputs	An object of ‘OutputParamList‘.
steps	A list of ‘cwlStepList‘.
e1	A ‘cwlWorkflow‘ object.
e2	A ‘cwlStep‘ object.
cwl	A ‘cwlWorkflow‘ object.
value	A list of ‘cwlSteps‘ to assign.

Value

cwlWorkflow: An object of class ‘cwlWorkflow‘.

steps: A list of ‘cwlStep‘ objects.

See Also

stepInParamList

Examples

```
input1 <- InputParam(id = "sth")
echo1 <- cwlProcess(baseCommand = "echo",
                    inputs = InputParamList(input1))
input2 <- InputParam(id = "sthout", type = "File")
echo2 <- cwlProcess(baseCommand = "echo",
                    inputs = InputParamList(input2),
                    stdout = "out.txt")
i1 <- InputParam(id = "sth")
o1 <- OutputParam(id = "out", type = "File", outputSource = "echo2/output")
wf <- cwlWorkflow(inputs = InputParamList(i1),
                   outputs = OutputParamList(o1))
s1 <- cwlStep(id = "echo1", run = echo1, In = list(sth = "sth"))
s2 <- cwlStep(id = "echo2", run = echo2, In = list(sthout = "echo1/output"))
wf <- wf + s1 + s2
```

---

cwlWorkflow-methods *cwlWorkflow methods*

Description

runs: The function to access all runs of a ‘cwlWorkflow‘ object.

Usage

```
runs(object)
```

Arguments

object	A ‘cwlWorkflow‘ object.

Value

‘cwlProcess‘ objects or paths of CWL file.

Examples

```
s1 <- cwlWorkflow()
runs(s1)
s1
short(s1)
```

env_Rcwl *Rcwl conda environment*

Description

Rcwl conda envrionment to install ‘cwltool‘ by basilisk.

Usage

```
env_Rcwl
```

Format

An object of class `BasiliskEnvironment` of length 1.

InputArrayParam-class *All classes defined in the package of ‘Rcwl‘ and the class constructor functions.*

Description

InputArrayParam: Parameters for array inputs. To specify an array parameter, the array definition is nested under the type field with 'type: array' and items defining the valid data types that may appear in the array.

InputParam: parameter for a command line tool.

InputParamList: A list of ‘InputParam‘ objects.

OutputArrayParam: Parameters for array outputs.

OutputParam: An output parameter for a Command Line Tool.

OutputParamList: A list of ‘InputParam‘ objects.

stepInParam: The input parameter of a workflow step.

stepInParamList: A list of ‘stepInParam‘ objects.

cwlStepList: A list of ‘cwlStep‘ objects.

Usage

```
InputArrayParam(
  label = "",
  doc = character(),
  name = character(),
  type = "array",
  items = character(),
  prefix = "",
  separate = TRUE,
```

```
  itemSeparator = character(),
  valueFrom = character()
)

InputParam(
  id,
  label = "",
  type = "string",
  doc = character(),
  secondaryFiles = character(),
  streamable = logical(),
  format = character(),
  loadListing = character(),
  loadContents = logical(),
  position = 0L,
  prefix = "",
  separate = TRUE,
  itemSeparator = character(),
  valueFrom = character(),
  shellQuote = logical(),
  default = character(),
  value = character()
)

InputParamList(...)

OutputArrayParam(
  label = character(),
  doc = character(),
  name = character(),
  type = "array",
  items = character()
)

OutputParam(
  id = "output",
  label = character(),
  doc = character(),
  type = "stdout",
  format = character(),
  secondaryFiles = character(),
  streamable = logical(),
  glob = character(),
  loadContents = logical(),
  loadListing = character(),
  outputEval = character(),
  outputSource = character(),
  linkMerge = character(),
```

```
  pickValue = character()
)

OutputParamList(out = OutputParam(), ...)

stepInParam(
  id,
  source = character(),
  linkMerge = character(),
  pickValue = character(),
  loadContents = logical(),
  loadListing = character(),
  default = character(),
  valueFrom = character()
)

stepInParamList(...)

cwlStepList(...)
```

Arguments

label	A short, human-readable label of this object.
doc	A documentation string for this object, or an array of strings which should be concatenated.
name	The identifier for this type.
type	Specify valid types of data that may be assigned to this parameter.
items	Defines the type of the array elements.
prefix	Command line prefix to add before the value.
separate	If true (default), then the prefix and value must be added as separate command line arguments; if false, prefix and value must be concatenated into a single command line argument.
itemSeparator	Join the array elements into a single string with the elements separated by by itemSeparator.
valueFrom	value from string or expression.
id	A unique identifier for this workflow input parameter.
secondaryFiles	Provides a pattern or expression specifying files or directories. Only valid when type: File or is an array of items: File.
streamable	A value of true indicates that the file is read or written sequentially without seeking. Only valid when type: File or is an array of items: File.
format	Only valid when type: File or is an array of items: File. This is the file format that will be assigned to the output File object.
loadListing	Only valid when type: Directory or is an array of items: Directory. "no_listing", "shallow_listing" or "deep_listing".
loadContents	Only valid when type: File or is an array of items: File.

position	The position for this parameter.
shellQuote	If ShellCommandRequirement is in the requirements for the current command, this controls whether the value is quoted on the command line (default is true).
default	The default value for this parameter to use if either there is no source field, or the value produced by the source is null.
value	Assigned value for this parameter
...	A list of 'cwlStep' objects.
glob	Pattern to find files relative to the output directory.
outputEval	Evaluate an expression to generate the output value.
outputSource	Specifies one or more workflow parameters that supply the value of to the output parameter.
linkMerge	The method to use to merge multiple inbound links into a single array.
pickValue	The method to use to choose non-null elements among multiple sources. "first_non_null", "the_only_non_null", or "all_non_null".
out	The default stdout parameter.
source	Specifies one or more workflow parameters that will provide input to the underlying step parameter.

Details

More details of 'InputArrayParam', see: https://www.commonwl.org/v1.0/CommandLineTool.html#CommandInputArraySc

More details for 'InputParam', see: https://www.commonwl.org/v1.0/CommandLineTool.html#CommandInputParameter

More details for 'OutputArrayParam', see: https://www.commonwl.org/v1.0/CommandLineTool.html#CommandOutputArra

More details for 'OutputParam', see: https://www.commonwl.org/v1.0/CommandLineTool.html#CommandOutputParameter

More details for 'stepInParam', see: https://www.commonwl.org/v1.0/Workflow.html#WorkflowStepInput

Value

InputArrayParam: An object of class 'InputArrayParam'.

An object of class 'InputParam'.

InputParamList: An object of class 'InputParamList'.

An object of class 'OutputArrayParam'.

OutputParam: An object of class 'OutputParam'.

OutputParamList: An object of class 'OutputParamList'.

stepInParam: An object of class 'stepInParam'.

An object of class 'stepInParamList'.

cwlStepList: An object of class 'cwlStepList'.

Examples

```
InputArrayParam(items = "string", prefix="-B=", separate = FALSE)
input1 <- InputParam(id = "sth")
InputParamList(input1)
OutputParam(id = "b", type = OutputArrayParam(items = "File"), glob = "*.txt")
o1 <- OutputParam(id = "file", type = "File", glob = "*.txt")
o1

o1 <- OutputParam(id = "file", type = "File", glob = "*.txt")
OutputParamList(o1)
s1 <- stepInParam(id = "s1")

s1 <- stepInParam(id = "s1")
stepInParamList(s1)
s1 <- cwlStep(id = "s1")
cwlStepList(s1)
```

`install_cwltool` *install cwltool*

Description

To download and install cwltool using basilisk

Usage

```
install_cwltool()
```

`install_udocker` *install udocker*

Description

To download and install udocker for python3.

Usage

```
install_udocker(version = "1.3.4")
```

Arguments

`version`	The version of udocker.

meta *addMeta Add or change meta information for a cwl recipe.*

Description

addMeta Add or change meta information for a cwl recipe.

Usage

```
meta(cwl)

meta(cwl) <- value

addMeta(
  cwl,
  label = character(),
  doc = character(),
  inputLabels = character(),
  inputDocs = character(),
  outputLabels = character(),
  outputDocs = character(),
  stepLabels = character(),
  stepDocs = character(),
  extensions = list()
)
```

Arguments

cwl	'cwlProcess' object for data or tool recipe. 'cwlWorkflow' object for a pipeline recipe.
value	A list of meta information to add to 'cwl'.
label	Character string specifying a label for the recipe. E.g., "bwa align", "gencode annotation".
doc	Character string describing the recipe. E.g, "Align reads to reference genome".
inputLabels	Vector of character string, specifying labels for each input parameter.
inputDocs	Vector of character string as descriptions for each input parameter.
outputLabels	Vector of character string, specifying labels for each output parameter.
outputDocs	Vector of character string as descriptions for each output parameter.
stepLabels	Vector of character string, specifying labels for each step. Use only if 'cwl' is a 'cwlWorkflow' object.
stepDocs	Vector of character string as description for each step. Use only if 'cwl' is a 'cwlWorkflow' object.

`extensions` A list of character strings. Can be used to add meta information about the recipe. Generally, add fields of information that does not require execution as part of the recipe evaluation. for information about "author", "url", "date", "example", use the exact names as list names as shown in examples, so that they can be correctly passed into corresponding fields in markdown file when using ‘meta2md‘. Other information can be added as a list element with arbitrary names.

Value

‘meta()‘: return a list of all available meta information for the ‘cwl‘ object.

‘addMeta()‘: ‘cwlProcess‘ or ‘cwlWorkflow‘ object, with added meta information, which can be returned using ‘meta(cwl)‘. Meta information can be converted into markdown file with ‘meta2md‘ function.

Examples

```
## Not run:
library(RcwlPipelines)
cwlSearch(c("bwa", "align"))
bwaAlign <- RcwlPipelines::cwlLoad("pl_bwaAlign")
bwaAlign <- addMeta(
  cwl = bwaAlign,
  label = "align",
  doc = "align reads to reference genome",
  inputLabels = c("threads", "readgroup", "reference", "read1", "read2"),
  inputDocs = c("number of threads", "read groups",
                "reference genome", "read pair1", "read pair2"),
  outputLabels = c("Bam", "Idx"),
  outputDocs = c("outputbam file", "index file"),
  stepLabels = c(bwa = "bwa"),
  stepDocs = c(bwa = "bwa alignment"))
cat(meta2md(bwaAlign))

## End(Not run)

## Not run:
rcp <- ReUseData::recipeLoad("gencode_annotation")
meta(rcp)
rcp1 <- addMeta(
  cwl = rcp,
  label = "",
  doc = "An empty description line",
  inputLabels = c("input label1", "input label2"),
  inputDocs = c("input description 1", "input description 2"),
  outputLabels = c("output label1"),
  outputDocs = c("output description 1"),
  extensions = list(
    author = "recipe author's name",
    url = "http://ftp.ebi.ac.uk/pub/databases/gencode/",
    date = as.character(Sys.Date()),
    example = "An example"))
meta(rcp1)
```

```
cat(meta2md(rcp1))

## End(Not run)
```

---

plotCWL *plotCWL*

Description

Function to plot cwlWorkflow object.

Usage

```
plotCWL(cwl, output = "graph", layout = "tree", ...)
```

Arguments

cwl	A cwlWorkflow object to plot
output	A string specifying the output type. An option inherits from ‘render_graph‘ and can also be "mermaid".
layout	Layout from ‘render_graph‘.
...	other parameters from ‘mermaid‘ or ‘render_graph‘ function

Value

A workflow plot.

Examples

```
input1 <- InputParam(id = "sth")
echo1 <- cwlProcess(baseCommand = "echo",
                    inputs = InputParamList(input1))
input2 <- InputParam(id = "sthout", type = "File")
echo2 <- cwlProcess(baseCommand = "echo",
                    inputs = InputParamList(input2),
                    stdout = "out.txt")
i1 <- InputParam(id = "sth")
o1 <- OutputParam(id = "out", type = "File", outputSource = "echo2/output")
wf <- cwlWorkflow(inputs = InputParamList(i1),
                   outputs = OutputParamList(o1))
s1 <- cwlStep(id = "echo1", run = echo1, In = list(sth = "sth"))
s2 <- cwlStep(id = "echo2", run = echo2, In = list(sthout = "echo1/output"))
wf <- wf + s1 + s2
plotCWL(wf)
```

readCWL *Read CWL Function to read CWL command or workflow files.*

Description

Read CWL Function to read CWL command or workflow files.

Usage

```
readCWL(cwlfile)
```

Arguments

cwlfile	The cwl file to read.

Value

A object of class ‘cwlProcess‘ or ‘cwlWorkflow‘.

Examples

```
input1 <- InputParam(id = "sth")
echo <- cwlProcess(baseCommand = "echo",
                   inputs = InputParamList(input1))
tf <- writeCWL(echo)
readCWL(tf[1])
```

runCWL *run cwlProcess*

Description

Execute a cwlProcess object with assigned inputs.

Usage

```
runCWL(
  cwl,
  outdir = ".",
  cwlRunner = "cwltool",
  cachedir = NULL,
  cwlTemp = NULL,
  cwlArgs = character(),
  stdout = TRUE,
  stderr = TRUE,
  showLog = FALSE,
```

```
  docker = TRUE,
  conda = FALSE,
  yml_prefix = deparse(substitute(cwl)),
  yml_outdir = tempfile(),
  ...
)
```

Arguments

cwl	A ‘cwlProcess‘ or ‘cwlWorkflow‘ object.
outdir	Output directory, default is current working directory.
cwlRunner	The path to the ‘cwltool‘ or ‘cwl-runner‘. If not exists, the cwltool package will be installed by ‘reticulate‘.
cachedir	Directory to cache intermediate workflow outputs to avoid recomputing steps.
cwlTemp	File path to keep intermediate files. If a directory path is given, the intermediate files will be kept in the directory. Default is NULL to remove all intermediate files.
cwlArgs	The arguments for ‘cwltool‘ or ‘cwl-runner‘. For example, "–debug" can work with ‘cwltool‘ to show debug information.
stdout	standard output from ‘system2‘.
stderr	standard error from ‘system2‘. By setting it to "", the detailed running logs will return directly.
showLog	Whether to show log details to standard out. i.e. stderr = "".
docker	Whether to use docker, or "sigularity" if use Singularity runtime to run container.
conda	Whether to install packages using conda if ‘SoftwareRequirement‘ is defined.
yml_prefix	The prefix of ‘.cwl‘ and ‘.yml‘ files that are to be internally executed.
yml_outdir	The output directory for the ‘.cwl‘ and ‘.yml‘ files.
...	The other options from ‘writeCWL‘ and ‘system2‘.

Value

A list of outputs from tools and logs from cwltool.

Examples

```
input1 <- InputParam(id = "sth")
echo <- cwlProcess(baseCommand = "echo",
                inputs = InputParamList(input1))
echo$sth <- "Hello World!"
## res <- runCWL(echo)
```

runCWLBatch *run CWL with batchtools*

Description

run CWL with batchtools

Usage

```
runCWLBatch(
  cwl,
  outdir = getwd(),
  inputList,
  paramList = list(),
  BPPARAM = BatchtoolsParam(workers = lengths(inputList)[1]),
  ...
)
```

Arguments

cwl	A 'cwlProcess' or 'cwlWorkflow' object.
outdir	Directory to output results
inputList	An input list to run in parallel. The list names must be in the inputs of cwl. Jobs will be submitted in parallel for each element in the list. The output directory of each job will be made using the name of each element under the 'outdir'.
paramList	A parameter list for the cwl. The list names must be in the inputs of cwl.
BPPARAM	The options for 'BiocParallelParam'.
...	The options from runCWL.

Value

Results from computing nodes and logs from cwltool.

runCWLBP *run CWL with BiocParallel*

Description

Submit one CWL object with assigned values with BiocParallel.

Usage

```
runCWLBP(cwl, outdir, BPPARAM, ...)
```

Arguments

cwl	cwl A ‘cwlProcess‘ or ‘cwlWorkflow‘ object.
outdir	Directory for output results
BPPARAM	The options for ‘BiocParallelParam‘.
...	The other options from runCWL.

Value

Results from computing nodes and logs from cwltool.

---

stepInputs *stepInputs*

Description

prepare inputs for workflow from ‘cwlStep‘ objects

Usage

```
stepInputs(stepList)
```

Arguments

stepList	a list of ‘cwlStep‘ objects.

Value

InputParamList.

---

stepOutputs *stepOutputs*

Description

prepare outputs for workflow from ‘cwlStep‘ objects

Usage

```
stepOutputs(stepList)
```

Arguments

stepList	a list of ‘cwlStep‘ objects.

Value

OutputParamList.

writeCWL *Write CWL*

Description

write ‘cwlProcess‘ to cwl and yml.

Usage

```
writeCWL(
  cwl,
  prefix = deparse(substitute(cwl)),
  outdir = tempfile(),
  docker = TRUE,
  libPaths = TRUE,
  ...
)
```

Arguments

cwl	A ‘cwlProcess‘ or ‘cwlWorkflow‘ object.
prefix	The prefix of ‘.cwl‘ and ‘.yml‘ files to be generated.
outdir	The output directory for the ‘.cwl‘ and ‘.yml‘ files.
docker	Whether to use docker.
libPaths	Whether to add local R libaray paths to R script.
...	Other options from ‘yaml::write_yaml‘.

Value

A CWL file and A YML file.

Examples

```
input1 <- InputParam(id = "sth")
echo <- cwlProcess(baseCommand = "echo",
                   inputs = InputParamList(input1))
writeCWL(echo)
```

Index

∗ **datasets**
    env_Rcwl, 15
+,cwlWorkflow,cwlStep-method
        *(*cwlWorkflow*)*, 12
$,cwlProcess-method
        *(*cwlProcess-methods*)*, 8
$<-,cwlProcess-method
        *(*cwlProcess-methods*)*, 8

addMeta *(*meta*)*, 20
arguments *(*cwlProcess-methods*)*, 8
arguments<- *(*cwlProcess-methods*)*, 8

baseCommand *(*cwlProcess-methods*)*, 8
baseCommand<- *(*cwlProcess-methods*)*, 8

condaPackage *(*cwl-requirements*)*, 3
CondaTool *(*cwl-requirements*)*, 3
cwl-requirements, 3
cwlClass *(*cwlProcess-methods*)*, 8
cwlClass<- *(*cwlProcess-methods*)*, 8
cwlProcess, *2*, 7
cwlProcess-class
        *(*InputArrayParam-class*)*, 15
cwlProcess-methods, 8
cwlShiny, 10
cwlStep, *2*, 11
cwlStep-class *(*InputArrayParam-class*)*, 15
cwlStepList *(*InputArrayParam-class*)*, 15
cwlStepList-class
        *(*InputArrayParam-class*)*, 15
cwlVersion *(*cwlProcess-methods*)*, 8
cwlVersion<- *(*cwlProcess-methods*)*, 8
cwlWorkflow, *2*, *12*, 12
cwlWorkflow-class
        *(*InputArrayParam-class*)*, 15
cwlWorkflow-methods, 14

Dirent *(*cwl-requirements*)*, 3

env_Rcwl, 15
extensions *(*cwlProcess-methods*)*, 8
extensions<- *(*cwlProcess-methods*)*, 8

hints *(*cwlProcess-methods*)*, 8
hints<- *(*cwlProcess-methods*)*, 8

InputArrayParam
        *(*InputArrayParam-class*)*, 15
InputArrayParam-class, 15
InputParam *(*InputArrayParam-class*)*, 15
InputParam-class
        *(*InputArrayParam-class*)*, 15
InputParamList *(*InputArrayParam-class*)*, 15
InputParamList-class
        *(*InputArrayParam-class*)*, 15
inputs *(*cwlProcess-methods*)*, 8
install_cwltool, 19
install_udocker, 19

meta, 20
meta<- *(*meta*)*, 20

OutputArrayParam
        *(*InputArrayParam-class*)*, 15
OutputArrayParam-class
        *(*InputArrayParam-class*)*, 15
OutputParam *(*InputArrayParam-class*)*, 15
OutputParam-class
        *(*InputArrayParam-class*)*, 15
OutputParamList
        *(*InputArrayParam-class*)*, 15
OutputParamList-class
        *(*InputArrayParam-class*)*, 15
outputs *(*cwlProcess-methods*)*, 8

plotCWL, 22

Rcwl *(*Rcwl-package*)*, 2
Rcwl,Rcwl-package *(*Rcwl-package*)*, 2

Rcwl-package, 2
readCWL, 23
requireDocker (cwl-requirements), 3
requireEnvVar (cwl-requirements), 3
requireInitialWorkDir
        (cwl-requirements), 3
requireJS (cwl-requirements), 3
requireManifest (cwl-requirements), 3
requirements (cwlProcess-methods), 8
requirements<- (cwlProcess-methods), 8
requireMultipleInput
        (cwl-requirements), 3
requireNetwork (cwl-requirements), 3
requireResource (cwl-requirements), 3
requireRscript (cwl-requirements), 3
requireScatter (cwl-requirements), 3
requireShellCommand (cwl-requirements),
        3
requireShellScript (cwl-requirements), 3
requireSoftware (cwl-requirements), 3
requireStepInputExpression
        (cwl-requirements), 3
requireSubworkflow (cwl-requirements), 3
runCWL, *2*, 23
runCWLBatch, 25
runCWLBP, 25
runs (cwlWorkflow-methods), 14

ShellScript (cwl-requirements), 3
short (cwlProcess-methods), 8
stdOut (cwlProcess-methods), 8
stdOut<- (cwlProcess-methods), 8
stepInParam (InputArrayParam-class), 15
stepInParam-class
        (InputArrayParam-class), 15
stepInParamList, *14*
stepInParamList
        (InputArrayParam-class), 15
stepInParamList-class
        (InputArrayParam-class), 15
stepInputs, 26
stepOutputs, 26
steps (cwlWorkflow), 12
steps<- (cwlWorkflow), 12

writeCWL, 27