bjec Documentation

Release 0.1

Seoester

May 14, 2023

Contents:

CHAPTER 1

Getting Started

1.1 Example

1.2 Command Line Interface

1.3 Library Use

CHAPTER 2

Runnables

2.1 Job

2.2 Build

CHAPTER 3

Concepts

3.1 Constructor

3.1.1 Constructor Function

3.1.2 Configuration Method

3.2 Configuration File

CHAPTER 4

API Reference

4.1 bjec.build

class bjec.build.**Build**(*constructor_func*, *depends=None*)

class Constructor(*obj*)

builder(*builder*)

dependencies

source(*source*)

class bjec.build.**Builder**

build()
Must be implemented by inheriting classes.

last_built()
Must be implemented by inheriting classes.

class bjec.build.**ChangeInfo**(*status*, *last_changed*)
Comprises information about the state of changes of a Source.

A ChangeInfo-like object is returned by `Source.scan()`.

status
Conveys any knowledge the Source has about whether changes have taken place. A Source may set `status` to `CHANGED`, when it changed its files directly, e.g. pulled from a remote source, etc. `UNCHANGED` may be set, when a version management system did not perform an update, `UNKNOWN` is the general case.

Type *ChangeInfo.Status*

last_changed
Date and time of the last change which took place in the Source. Generally only changes to a file's content are regarded as change.

Type datetime.datetime

class Status
An enumeration.

CHANGED = 2

UNCHANGED = 1

UNKNOWN = 0

class bjec.build.**GitRepo**(*url*, *branch='master'*)
docstring for GitRepo

Parameters

- **url** (*str*) – Remote URL of the repository
- **branch** (*str*) – Branch of the remote repository to use, default: "master"

Configuration Options:

- **repos_path: Path to local directory which repositories are** downloaded to, defaults to *default_repos_path*
- identity_file: Path to an (SSH) identity file for authentication
- **identity_content: Content of an (SSH) identity file for** authentication

default_repos_path = '~/bjec/repos'
See configuration option repos_path.

local_path()
Return the (base) path to the source on the local file system.

Must be implemented by inheriting classes.

Returns The absolute path to the Source's local base directory.

Return type str

scan()
Perform a scan over the source set and return change info.

Must be implemented by inheriting classes.

Returns An object adhering to the ChangeInfo documentation.

Return type *ChangeInfo*

class bjec.build.**Local**(*path*)
docstring for Local

local_path()
Return the (base) path to the source on the local file system.

Must be implemented by inheriting classes.

Returns The absolute path to the Source's local base directory.

Return type str

scan()
Perform a scan over the source set and return change info.

Must be implemented by inheriting classes.

Returns An object adhering to the ChangeInfo documentation.

Return type *ChangeInfo*

class bjec.build.**Make**(*path*, *target=None*, *creates=None*, *clean_first=False*, *clean_target=None*)
docstring for Make

Parameters

- **path** (*str*) – Path to the directory containing the Makefile
- **target** (*str or list of str, optional*) – make target(s) to execute
- **creates** (*str or list of str, optional*) – File path(s) created by make, may be absolute (starting with "/") or relative to *path*
- **clean_first** (*bool, optional*) – When True, call *clean()* before starting to build (*clean_target* must be given)
- **clean_target** (*str or list of str, optional*) – make target(s) to execute for cleaning

Configuration Options:

- environment: Map of environment variables passed to the make call

build()
Must be implemented by inheriting classes.

clean()

last_built()

Returns

The earliest mtime of any file in *creates*.

If *creates* is None, empty or None of the files exist, datetime.datetime.min (aware, i.e. with added tzinfo) is returned.

Return type datetime.datetime

result()

class bjec.build.**Source**

local_path()
Return the (base) path to the source on the local file system.

Must be implemented by inheriting classes.

Returns The absolute path to the Source's local base directory.

Return type str

scan()
Perform a scan over the source set and return change info.

Must be implemented by inheriting classes.

Returns An object adhering to the ChangeInfo documentation.

Return type *ChangeInfo*

bjec.build.**build**(*depends=None*, *master=None*)

4.2 bjec.cli

bjec.cli.**main**()

bjec.cli.**run**(*path*, *name*, *config=None*)

4.3 bjec.collector

class bjec.collector.**CSV**(*file_path=None*, *tempfile_class=<function TemporaryFile>*, *close_files=True*, *lock_class=<built-in function allocate_lock>*, *input_encoding='utf-8'*, *output_encoding='utf-8'*, *input_csv_args=None*, *output_csv_args=None*, *before_all=None*, *after_all=None*, *before_row=None*, *after_row=None*, *before=None*, *after=None*)

Collector concatenating CSV output (from file-like objects).

The Collector expects file-like objects, those are read as CSV files. Each row is appended to an output file.

Parameters

- **file_path** (*str*) – The file path to opened as the aggregate file. If `None` a temporary file will be created according to *tempfile_class*.
- **tempfile_class** (*class object or function*) – The class used to create a temporary file as the aggregate file. Only used when *file_path* is set to `None`. Please note that a function may be passed in, e.g. thus enabling use of *functools.partial* to set *max_size* for *tempfile.SpooledTemporaryFile*.
- **close_files** (*bool*) – If set to `True`, *add()* will attempt to close the output argument (by calling *close()* on it), ignoring any AttributeError (i.e. *close()* not defined).
- **lock_class** (*class object or function*) – The class used to create a lock object.
- **input_encoding** (*str, optional*) – Input encoding (for *output* passed into *add()*). Defaults to `"utf-8"`.
- **output_encoding** (*str, optional*) – Output encoding (for the aggregate file). Defaults to `"utf-8"`.
- **input_csv_args** (*dict, optional*) – kwargs passed to the *csv.reader()* call used to create a reader for CSV input (from *output* passed into *add()*).
- **output_csv_args** (*dict, optional*) – kwargs passed to the *csv.writer()* call used to create a writer for CSV output (to the aggregate file).
- **before_all** (*iterable of iterables, optional*) – Is inserted at the beginning of the output file. *before_all* is interpreted as rows, each item in one row is written as column content.
- **after_all** (*iterable of iterables, optional*) – Is appended to the end of the output file. *after_all* is interpreted as rows, each item in one row is written as column content.

- **before** (*iterable of iterables, optional*) – Is inserted before each item's data, which is handed to the Collector using *add()*. *before* is interpreted as rows, each item in one row is written as column content.
- **after** (*iterable of iterables, optional*) – Is appended to each item's data, which is handed to the Collector using *add()*. *after* is interpreted as rows, each item in one row is written as column content.
- **before_row** (*iterable, optional*) – Is inserted at the beginning of each row. Each item of *before_row* is written as column content.
- **after_row** (*iterable, optional*) – Is appended to each row. Each item of *after_row* is written as column content.

add(*params*, *output*)
Adds the output of a run to the collector.

Must be implemented by inheriting classes.

Inheriting classes can specify whether *add()* may be called after *aggregate()* has been called. Inheriting classes must ensure, that *add()* is thread-safe.

Parameters

- **params** (*dict*) – The parameters o the run.
- **output** (*any*) – Output of the run. What kind of object is passed in will depend on the Runner.

aggregate()
Aggregates and returns all the outputs collected.

Must be implemented by inheriting classes.

Inheriting classes can specify whether *aggregate* may be called multiple times. Inheriting classes may add optional parameters.

Returns Returns the aggregate of all outputs added to the Collector.

Return type any

class bjec.collector.**Collector**
docstring for Collector

add(*params*, *output*)
Adds the output of a run to the collector.

Must be implemented by inheriting classes.

Inheriting classes can specify whether *add()* may be called after *aggregate()* has been called. Inheriting classes must ensure, that *add()* is thread-safe.

Parameters

- **params** (*dict*) – The parameters o the run.
- **output** (*any*) – Output of the run. What kind of object is passed in will depend on the Runner.

aggregate()
Aggregates and returns all the outputs collected.

Must be implemented by inheriting classes.

Inheriting classes can specify whether *aggregate* may be called multiple times. Inheriting classes may add optional parameters.

Returns Returns the aggregate of all outputs added to the Collector.

Return type any

class bjec.collector.**Concatenate**(*file_path=None*, *tempfile_class=<function TemporaryFile>*, *close_files=True*, *lock_class=<built-in function allocate_lock>*, *before_all=None*, *after_all=None*, *before=None*, *after=None*)

Collector concatenating output (file-like objects) into a new file.

Parameters

- **file_path** (*str*) – The file path to opened as the aggregate file. If `None` a temporary file will be created according to *tempfile_class*.
- **tempfile_class** (*class object or function*) – The class used to create a temporary file as the aggregate file. Only used when *file_path* is set to `None`. Please note that a function may be passed in, e.g. thus enabling use of *functools.partial* to set *max_size* for *tempfile.SpooledTemporaryFile*.
- **close_files** (*bool*) – If set to `True`, *add()* will attempt to close the output argument (by calling *close()* on it), ignoring any AttributeError (i.e. *close()* not defined).
- **lock_class** (*class object or function*) – The class used to create a lock object.

add(*params*, *output*)

Adds the output of a run to the collector.

Must be implemented by inheriting classes.

Inheriting classes can specify whether *add()* may be called after *aggregate()* has been called. Inheriting classes must ensure, that *add()* is thread-safe.

Parameters

- **params** (*dict*) – The parameters o the run.
- **output** (*any*) – Output of the run. What kind of object is passed in will depend on the Runner.

aggregate()

Returns the file object containing the aggregated output.

Returns The file object containing the aggregated output, the position in the file is reset to `0` before returning. The caller has the responsible to *close()* the returned file-like object.

Return type file-like object

class bjec.collector.**Demux**(*watch*, *factory*, *lock_class=<built-in function allocate_lock>*)

Demux de-multiplexes output, distributing it to different Collectors.

Parameters

- **watch** (*list of str*) – List of parameters to watch for: For each distinct combination of values in this list, a collector is maintained.
- **factory** (*function*) – Called to create a new collector. A dict of parameters is passed as the only argument, containing only those parameters specified in *watch*.
- **lock_class** (*class object or function*) – The class used to create a lock object.

add(*params*, *output*)

aggregate()

4.4 bjec.config

class bjec.config.**Config**(*namespace='bjec'*)
docstring for Config

read_yaml(*path*)

class bjec.config.**ModuleConfig**(*config*, *key_parts*)
docstring for ModuleConfig

get(*key*, *default=None*)

4.5 bjec.generator

class bjec.generator.**Chain**(**generators*)

class bjec.generator.**Combine**(**generators*)

class bjec.generator.**Generator**
Generator represents a generator for input parameters of tasks.

Every parameter set produced by the generator represents the input for a a task.

The `Generator` interface basically is a standard python iterable, i.e. the `__iter__` method has to be defined and return an iterator.

class bjec.generator.**List**(*iterable*)

class bjec.generator.**Product**(***params*)
docstring for Product

class bjec.generator.**Repeat**(*params*, *n*)

class bjec.generator.**RepeatG**(*generator*, *n*)

4.6 bjec.job

class bjec.job.**Job**(*constructor_func*, *depends=None*)

class Constructor(*obj*)

after(**after_func*)

collector(*collector*)

generator(*generator*)

processor(*processor*)

runner(*runner*)

run()
Must be implemented by inheriting classes.

bjec.job.**job**(*depends=None*, *master=None*)

4.7 bjec.master

class bjec.master.**Artefactor**
docstring for Artefactor

class Constructor

artefact(***kwargs*)

artefact(***kwargs*)

w_run()

class bjec.master.**Constructible**
docstring for Constructible

class Constructor(*obj*)

construct()

constructed()

constructor_func(*constructor_func*)

w_run()

class bjec.master.**Dependency**
docstring for Dependency

Dependency has two different Constructor variants: `SetUpConstructor` allows adding dependencies to the object, while `ResolveConstructor` makes resolved dependencies available with its *dependencies* attribute.

class ResolveConstructor

dependencies

class SetUpConstructor

depends(**args*)

depends(**args*)

fulfill()
Fulfills this dependency.

May be implemented by inheriting classes, but defaults to calling *self.run()*. In this case however, self.run() has to ensure *_fulfill_dependencies()* is run.

Should the object only be run once, the following can be inserted at the beginning of this method's implementation (or *self.run()*):

```
if self.fulfilled():
        return
```

fulfilled()

w_run()

class bjec.master.**Master**

register(*obj*, *func*, *secondary=None*)

class bjec.master.**Registerable**

registered_with(*master*)

class bjec.master.**Runnable**

run()
Must be implemented by inheriting classes.

class bjec.master.**WrapperRun**
docstring for WrapperRun

run()

w_run()

4.8 bjec.params

class bjec.params.**Factory**(*cls*, **args*, ***kwargs*)
Factory for objects with ParamsEvaluable arguments.

Example

```
Factory(Concatenate, file_path=Join("out.", P("n"), ".data"))
```

Parameters

- **cls** (*class object*) –
- ***args** (*arbitrary, ParamsEvaluable*) – Variable arguments passed to the class constructor. May contain ParamsEvaluable elements.
- ****kwargs** (*arbitrary, ParamsEvaluable*) – Keyword arguments passed to the class constructor. May contain ParamsEvaluable values.

evaluate(*params*)

class bjec.params.**Function**(*func*)
Wrapper for functions.

Example

```
Function(lambda p: p["alpha"] / p["beta"])
```

Parameters **func** (*function*) – Function to be called on evaluation. The parameters are passed as the only argument.

evaluate(*params*)

class bjec.params.**Join**(**args*, *sep=''*)

String / Bytes Join for lists containing ParamsEvaluable objects.

The type of output is determined by the type of the *sep* argument.

If the output should be a `str`, `str(.)` will be called on each list element (in **args*). If the output should be of type `bytes`, the user has to ensure that each of the list elements are of `bytes` type and that `ParamsEvaluable(.)` returns a `bytes` object.

Example

```
Join("out.", P("n"), ".csv")
```

Parameters

- ***args** (*object supporting str(), ParamsEvaluable or bytes*) – Elements to join, may be instances of ParamsEvaluable classes. If the output type is `str`, `str()` is applied to every element before joining.
- **sep** (*str or bytes, optional*) – Separator used to join elements of **args*. Must have the type of the output, i.e. if the output should be of a `bytes` type, *sep* must be as well. Defaults to `""`.

evaluate(*params*)

class bjec.params.**P**(*key*, *f=None*)

Wrapper to allow intuitive parameter inclusion.

P instances represent a 'future' parameter value, every instance contains the *key* of the parameter in the *params* dict. Each instance evaluates to the corresponding parameter's value.

Other modules may accept *P* objects or lists containing *P* objects. These are then evaluated for every parameter set.

Example

```
ProcessArgs("--offset", P("offset"))
```

Parameters

- **key** (*str*) – Parameter (key of the parameter in the *params*) dict.
- **f** (*None or function, optional*) – If not None, *f* is applied to the value of `params[key]` and the result is returned.

evaluate(*params*)

classmethod evaluate_list(*l*, *params*)

class bjec.params.**ParamsEvaluable**

evaluate(*params*)

bjec.params.**evaluate**(*obj*, *params*)

4.9 bjec.processor

class bjec.processor.**Inline**

process()
Process all parameter sets produced by the generator.

Must be implemented by inheriting classes.

class bjec.processor.**Processor**
docstring for Processor

A `Processor` is responsible for the task execution pipeline, that is fetching parameter sets from a `Generator`, handing them to a `Runner` and passing the `Runner`'s output to a `Collector`. Meanwhile the `Processor` has to manage its `Runners'` lifecycle.

collector(*collector*)

generator(*generator*)

process()
Process all parameter sets produced by the generator.

Must be implemented by inheriting classes.

runner_factory(*runner_factory*)

class bjec.processor.**Threading**(*n*)
docstring for Threading

Parameters **n** (*int*) – Number of threads to be run. If <= 0, the configuration option of the same name is used instead.

Configuration Options:

- **n: Number of threads to run, it is used when *n* passed to the** constructor is <= 0. Defaults to 1.

process()
Process all parameter sets produced by the generator.

Must be implemented by inheriting classes.

4.10 bjec.runner

class bjec.runner.**InputMethod**

class Wrapper(*obj*, *params*, *args*, *kwargs*)

wrapper(*params*, *args*, *kwargs*)

class bjec.runner.**OutputMethod**

class Wrapper(*obj*, *params*, *args*, *kwargs*)

output()
Returns the output of the subprocess.

Must be implemented by inheriting classes.

Will be called by SubprocessRunner after the `subprocess.run()` call has finished.

wrapper(*params*, *args*, *kwargs*)

class bjec.runner.**ProcessArgs**(**args*)
docstring for ProcessArgs

Parameters ***args** (*str, ParamsEvaluable*) – Arguments to execute the subprocess with. Supports ParamsEvaluable arguments.

class Wrapper(*obj*, *params*, *args*, *kwargs*)

class bjec.runner.**Runner**
docstring for Runner

classmethod factory(**args*, ***kwargs*)
Creates a factory for properly set-up `Runner` objects.

Here, a factory is a function taking no parameters and returning a new instance of a `Runner` (subclass).

May be implemented by inheriting classes. The default implementation will create a new object of the current class with the exact same parameters as passed into the `factory` method.

Returns Calling this function will return a new `Runner` instance with the parameters passed into the `factory` method.

Return type function

run(*params*)
run is called to have the Runner execute a task.

Must be implemented by inheriting classes.

The parameter params consists of the task's parameters, its type will depend on the Runner's configuration. run() must return only after processing completed. Its return type will be depend on the Runner's configuration.

start()
start is called before the Runner is used for the first time.

May be implemented by inheriting classes (but must be defined).

If starting is an asynchronous process, start() must return after this process completed.

stop()
stop is called when the Runner is no longer needed.

May be implemented by inheriting classes (but must be defined).

If stopping is an asynchronous process, stop() must return after this process completed.

class bjec.runner.**Stdout**(*spool=0*, *named=False*, *stdout=True*, *stderr=False*)
docstring for Stdout

Parameters

- **spool** (*int, default 0*) – If spool is greater 0, the stdout will be stored in a spooled file in memory until its size exceeds `spool`.
- **named** (*bool, default False*) – If True, the output file will be located on the file system with its path in the its `name` attribute. Implies `spool = 0` if set to True.
- **stdout** (*bool, default True*) – If True, the stdout of the subprocess will be included in the output file.

- **stderr** (*bool, default False*) – If True, the stderr of the subprocess will be included in the output file

class Wrapper(**args*, ***kwargs*)

output()
Returns the output of the subprocess.

Must be implemented by inheriting classes.

Will be called by SubprocessRunner after the `subprocess.run()` call has finished.

class bjec.runner.**SubprocessRunner**(**args*, *input=None*, *output=None*, ***kwargs*)
docstring for SubprocessRunner

class Wrapper(*obj*, *params*, *args*, *kwargs*)
Wrapper is a helper class used by both input and output methods.

Wrapper is used as a context manager, the `subprocess.run()` is wrapped in it. Input and output methods can therefore use its `__enter__` methods to modify the `args` and `kwargs` passed to the `subprocess.run()` call.

For details also check out the `InputMethod` and `OutputMethod` classes as well as the concrete implementations of the both.

obj
Arbitrary object, meant to contain the instance of of the Wrapper's method class, thus enabling access to its members.
Type object

params
The parameter set serving as the input of the current run / task.
Type dict

args
The args list passed to `subprocess.run()`. May be modified.
Type list

kwargs
The kwargs list passed to `subprocess.run()`. May be modified.
Type dict

run(*params*)
run is called to have the Runner execute a task.

Must be implemented by inheriting classes.

The parameter params consists of the task's parameters, its type will depend on the Runner's configuration. run() must return only after processing completed. Its return type will be depend on the Runner's configuration.

4.11 bjec.utils

bjec.utils.**listify**(*obj*, *none_empty=False*)
listify turns *obj* into an iterable.

Returns *obj* is simply returned, if it already is an iterable. Otherwise - or if it a string - it is wrapped in a list. If *none_empty* is set to `True`, an empty list is returned, if *obj* is `None`.

bjec.utils.**max_datetime = datetime.datetime(9999, 12, 31, 23, 59, 59, 999999, tzinfo=datet**
Maximum representable datetime with timezone (“aware”) set to UTC.

bjec.utils.**min_datetime = datetime.datetime(1, 1, 1, 0, 0, tzinfo=datetime.timezone.utc)**
Minimum representable datetime with timezone (“aware”) set to UTC.

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Python Module Index

b

Index

E

F

G

I

J

K

L

M

O

P

R

S

T

U

W