bjec Documentation

Release 0.3.dev1

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

A constructor function is similar to `__init__()` in Python. It is function which configures an object instance without being attached to the object's type.

It is suitable to alleviate the need of sub-classing: A constructor function is defined for each kind of :obj:`Job` instead of a sub-class with only a `__init__()` method. Furthermore, additional capabilities can be made available within the constructor function. This results in a well-defined interface for the user. For example, the list of dependencies is already populated. Constructor function can be defined for each kind of user-constructible object.

Similar to `self`, an object to be manipulated is passed as the first parameter. This is the *constructor object*. Constructor objects are not the final object to be configured, but instead a mutable container which holds all configurable values. For the user these details are not relevant and

The parameter is commonly named after the type. For example, `job` or `j` is used for :obj:`Job` constructor functions.

The lifecycle of object construction through a constructor function is as follows:

1. The constructor object is initialised. It may receive additional data such as the list of dependencies or be linked to an instance of the class which it configures. The constructor object types are implemented through the use of mix-ins with clear responsibilities.
2. The constructor function is called with the constructor object as its first parameter.
3. The final object, e.g. a :obj:`Job` instance, is finalised from the configuration stored in the constructor object.

These steps are performed by the constructor of the final object, e.g. the :obj:`Job` instance, during the :obj:`Runnable.run` call.

3.2 Configuration File

3.3 Fluid Builder

CHAPTER 4

API Reference

4.1 bjec package

4.1.1 Submodules

4.1.2 bjec.build module

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

Bases: *bjec.master.Dependency*, *bjec.master.Constructible*, *bjec.master.Artefactor*, *bjec.master.WrapperRun*, *bjec.master.Runnable*

class Constructor(*obj: Any*)

Bases: bjec.master.ResolveConstructor, bjec.master.Constructor, bjec.master.Constructor

builder(*builder*)

dependencies

source(*source*)

class bjec.build.**Builder**

Bases: object

build()

Must be implemented by inheriting classes.

last_built()

Must be implemented by inheriting classes.

class bjec.build.**ChangeInfo**(*status*, *last_changed*)

Bases: object

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
Bases: `enum.Enum`

An enumeration.

CHANGED = 2

UNCHANGED = 1

UNKNOWN = 0

class `bjec.build.`**GitRepo**(*url*, *branch='master'*)
Bases: *bjec.build.Source*

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*)
Bases: *bjec.build.Source*

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*)
Bases: *bjec.build.Builder*

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**
Bases: `object`

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*)

4.1.3 bjec.cli module

class bjec.cli.**RunArgs**
Bases: object

bjec.cli.**main**() → None

bjec.cli.**run**(*args: bjec.cli.RunArgs*) → None

4.1.4 bjec.collector module

class bjec.collector.**Collector**
Bases: typing.Generic, abc.ABC

Collects and processes per-parameter set results.

Collector provides the context manager interface. Each collector is a non-reentrent context manager. Any long-held resources will only be acquired upon entering the context manager, i.e. by opening an aggregation file. These resources will be released when exiting the context manager, i.e. closing all open files.

collect(*results: Iterable[Tuple[Mapping[str, Any], _T_contra]]*) → None
Collects and processes all elements within results.

This method **must** be called while the context manager is in the open state.

Collect may be called never, once or multiple times.

Parameters **results** – Iterable over tuples of the parameter set with the associated result.

class bjec.collector.**Concatenate**(*path: Union[str, bytes, os.PathLike, None] = None, before_all: Union[bjec.io.Writeable, str, bytes, None] = None, after_all: Union[bjec.io.Writeable, str, bytes, None] = None, before: Union[bjec.io.Writeable, str, bytes, bjec.params.ParamsEvaluable[typing.Union[bjec.io.Writeable, str, bytes]][Union[bjec.io.Writeable, str, bytes]], None] = None, after: Union[bjec.io.Writeable, str, bytes, bjec.params.ParamsEvaluable[typing.Union[bjec.io.Writeable, str, bytes]][Union[bjec.io.Writeable, str, bytes]], None] = None*)
Bases: *bjec.collector.Collector*

Concatenates file-like openables into a new file.

Parameters **path** – The file path to be opened as the aggregate file. If `None` a temporary file is created (which is not deleted).

collect(*results: Iterable[Tuple[Mapping[str, Any], bjec.io.ReadOpenable]]*) → None
Collects and processes all elements within `results`.

This method **must** be called while the context manager is in the open state.

Collect may be called never, once or multiple times.

Parameters **results** – Iterable over tuples of the parameter set with the associated result.

path

class bjec.collector.**Convert**(*f: Callable[[_T_contra], _S], collector: bjec.collector.Collector[~_S][_S]*)
Bases: *bjec.collector.Collector*, typing.Generic

collect(*results: Iterable[Tuple[Mapping[str, Any], _T_contra]]*) → None
Collects and processes all elements within `results`.

This method **must** be called while the context manager is in the open state.

Collect may be called never, once or multiple times.

Parameters **results** – Iterable over tuples of the parameter set with the associated result.

collector

class bjec.collector.**Demux**(*keys: Iterable[str], factory: Callable[[Mapping[str, Any]], bjec.collector.Collector[-_T_contra][_T_contra]]*)
Bases: *bjec.collector.Collector*, typing.Generic

Demux de-multiplexes results, by distributing to different Collectors.

Parameters

- **keys** – Keys in the parameter set which to consider during demuxing. For each distinct combination of values of these keys, a collector is maintained.
- **factory** – Function to call to create a new collector. A reduced parameter set is passed as the only argument, containing only those parameters specified in `keys`.

collect(*results: Iterable[Tuple[Mapping[str, Any], _T_contra]]*) → None
Collects and processes all elements within `results`.

This method **must** be called while the context manager is in the open state.

Collect may be called never, once or multiple times.

Parameters **results** – Iterable over tuples of the parameter set with the associated result.

keys

class bjec.collector.**Multi**(**collectors*)
Bases: *bjec.collector.Collector*, typing.Generic

collect(*results: Iterable[Tuple[Mapping[str, Any], _T_contra]]*) → None
Collects and processes all elements within `results`.

This method **must** be called while the context manager is in the open state.

Collect may be called never, once or multiple times.

Parameters **results** – Iterable over tuples of the parameter set with the associated result.

collectors

class bjec.collector.**Noop**
Bases: *bjec.collector.Collector*, typing.Generic

collect(*results: Iterable[Tuple[Mapping[str, Any], _T_contra]]*) → None
Collects and processes all elements within `results`.

This method **must** be called while the context manager is in the open state.

Collect may be called never, once or multiple times.

Parameters **results** – Iterable over tuples of the parameter set with the associated result.

4.1.5 bjec.config module

class bjec.config.**Config**(*namespace: str = 'bjec'*)
Bases: object

namespace

read_yaml(*path: Union[str, bytes, os.PathLike]*) → None

user

class bjec.config.**ModuleConfig**(*config: bjec.config.Config, key_parts: Iterable[str]*)
Bases: object

get(*key: str*, *default: Optional[Any] = None*) → Optional[Any]

key_parts

4.1.6 bjec.csv module

class bjec.csv.**Collector**(*path: Union[str, bytes, os.PathLike, None] = None, before_all: Optional[Iterable[Iterable[Any]]] = None, after_all: Optional[Iterable[Iterable[Any]]] = None, before: Union[bjec.params.ParamsEvaluable[typing.Iterable[typing.Iterable[typing.Any]]][Iterable[Iterable[Union[Iterable[Union[Any, bjec.params.ParamsEvaluable[typing.Any][Any]]], Iterable[Any], bjec.params.ParamsEvaluable[typing.Iterable[typing.Any]][Iterable[Any]]]], None] = None, after: Union[bjec.params.ParamsEvaluable[typing.Iterable[typing.Iterable[typing.Iterable[Union[Iterable[Union[Any, bjec.params.ParamsEvaluable[typing.Any][Any]]], Iterable[Any], bjec.params.ParamsEvaluable[typing.Iterable[typing.Any]][Iterable[Any]]]], None] = None, before_row: Union[Iterable[Union[Any, bjec.params.ParamsEvaluable[typing.Any][Any]]], Iterable[Any], bjec.params.ParamsEvaluable[typing.Iterable[typing.Any]][Iterable[Any]]], None] = None, after_row: Union[Iterable[Union[Any, bjec.params.ParamsEvaluable[typing.Any][Any]]], Iterable[Any], bjec.params.ParamsEvaluable[typing.Iterable[typing.Any]][Iterable[Any]]], None] = None, manage_headers: bool = False, before_header_row: Optional[Iterable[Any]] = None, after_header_row: Optional[Iterable[Any]] = None, input_encoding: Optional[str] = None, input_errors: Optional[str] = None, input_csv_args: Optional[Mapping[str, Any]] = None, output_encoding: Optional[str] = None, output_errors: Optional[str] = None, output_csv_args: Optional[Mapping[str, Any]] = None*)
Bases: *bjec.collector.Collector*

Concatenates CSV from file-like read openables into an aggregate file.

Parameters

- **path** – The file path to be opened as the aggregate file. If `None` a a file in a system specific location for temporary files is created. This file is never deleted by the Collector but may be deleted by OS mechanisms. It should not be treated as permanent.
- **before_all** – Rows (of columns) added before any rows from input files. That means these are written at the very beginning of the aggregate file. If `manage_headers` is `True`, these rows are written **after** the header row into the aggregate file.
- **after_all** – Rows (of columns) added after all rows from input files. That means these are written at the very end of the aggregate file.
- **before** – Rows (of columns) added before any input rows for each input file. That means these are written at the beginning of input file specific rows in the aggregate file. Parameters of the input file may be used in `before`.
- **after** – Rows (of columns) added after all input rows for each input file. That means these are written at the end of input file specific rows in the aggregate file. Parameters of the input file may be used in `after`.
- **before_row** – Columns inserted before each input row. That means these are written at the beginning of each input row in the aggregate file. Parameters of the input file may be used in `before_row`.
- **after_row** – Columns inserted after each input row. That means these are written at the end of each input row in the aggregate file. Parameters of the input file may be used in `after_row`.
- **manage_headers** – If `True` the first row of each input file is treated as a header row and only actual data rows in input files are concatenated. The header row is written once at the very beginning of the aggregate file. For this to work, the headers of all input file must be identical. An exception is raised if inconsistent
- **before_header_row** – Columns added before any input headers. That means these are written at the front of the header row of the aggregate file. Only interpreted if `manage_headers` is `True`.
- **after_header_row** – Columns added after any input headers. That means these are written at the end of the header row of the aggregate file. Only interpreted if `manage_headers` is `True`.
- **input_encoding** – Encoding to use when reading input files. Passed as-is to the `TextIOWrapper` constructor.
- **input_error** – Error setting to use when reading input files. Passed as-is to the `TextIOWrapper` constructor.
- **input_csv_args** – Args passed to `csv.reader()` when constructing readers for input files. This may include the `dialect` key.
- **output_encoding** – Encoding to use when writing output files. Passed as-is to the `TextIOWrapper` constructor.
- **output_error** – Error setting to use when writing output files. Passed as-is to the `TextIOWrapper` constructor.
- **output_csv_args** – Args passed to `csv.writer()` when constructing the writer for output file. This may include the `dialect` key.

collect (*results: Iterable[Tuple[Mapping[str, Any], bjec.io.ReadOpenable]]*) → None

Collects and processes all elements within `results`.

This method **must** be called while the context manager is in the open state.

Collect may be called never, once or multiple times.

Parameters **results** – Iterable over tuples of the parameter set with the associated result.

path

4.1.7 bjec.generator module

class bjec.generator.**Chain**(**generators*)
Bases: *bjec.generator.Generator*

class bjec.generator.**FromIterable**(*it: Iterable[Mapping[str, Any]]*)
Bases: *bjec.generator.Generator*

class bjec.generator.**Generator**
Bases: abc.ABC

Produces parameter sets.

A top-level generator produces fully specified parameter sets. Each such parameter set results in an independent invocation or execution when processed by a Processor.

So-called higher-level generators take other generators on input and combine the produced parameter sets in specific ways.

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

class bjec.generator.**Literal**(***params*)
Bases: *bjec.generator.Generator*

class bjec.generator.**Matrix**(***params*)
Bases: *bjec.generator.Generator*

class bjec.generator.**Product**(**generators*)
Bases: *bjec.generator.Generator*

class bjec.generator.**Repeat**(*generator: bjec.generator.Generator*, *n: int*)
Bases: *bjec.generator.Generator*

4.1.8 bjec.htcondor module

4.1.9 bjec.io module

class bjec.io.**ReadOpenable**(**args*, ***kwargs*)
Bases: typing_extensions.Protocol

open_bytes() → io.BufferedIOBase

open_text(*encoding: Optional[str] = None*, *errors: Optional[str] = None*, *newline: Optional[str] = None*) → io.TextIOBase

class bjec.io.**ReadOpenableFromPath**(*path: Union[str, bytes, os.PathLike]*)
Bases: *bjec.io.ReadOpenable*

open_bytes() → io.BufferedIOBase

open_text(*encoding: Optional[str] = None*, *errors: Optional[str] = None*, *newline: Optional[str] = None*) → io.TextIOBase

path

class bjec.io.**ReadOpenableWrapBinaryIO**(*b: BinaryIO*)
Bases: *bjec.io.ReadOpenable*

open_bytes() → io.BufferedIOBase

open_text(*encoding: Optional[str] = None*, *errors: Optional[str] = None*, *newline: Optional[str] = None*) → io.TextIOBase

class bjec.io.**WriteOpenable**(**args*, ***kwargs*)
Bases: typing_extensions.Protocol

open_bytes() → io.BufferedIOBase

open_text(*encoding: Optional[str] = None*, *errors: Optional[str] = None*, *newline: Optional[str] = None*) → io.TextIOBase

class bjec.io.**WriteOpenableFromPath**(*path: Union[str, bytes, os.PathLike]*)
Bases: *bjec.io.WriteOpenable*

open_bytes() → io.BufferedIOBase

open_text(*encoding: Optional[str] = None*, *errors: Optional[str] = None*, *newline: Optional[str] = None*) → io.TextIOBase

path

class bjec.io.**WriteOpenableWrapBinaryIO**(*b: BinaryIO*)
Bases: *bjec.io.WriteOpenable*

open_bytes() → io.BufferedIOBase

open_text(*encoding: Optional[str] = None*, *errors: Optional[str] = None*, *newline: Optional[str] = None*) → io.TextIOBase

class bjec.io.**Writeable**(**args*, ***kwargs*)
Bases: typing_extensions.Protocol

write_to(*w: bjec.io.WriteOpenable*) → None

class bjec.io.**WriteableFromBytes**(*content: bytes*)
Bases: *bjec.io.Writeable*

content

write_to(*w: bjec.io.WriteOpenable*) → None

class bjec.io.**WriteableFromPath**(*path: Union[str, bytes, os.PathLike]*)
Bases: *bjec.io.Writeable*

class Parameterised(*path: Union[str, bytes, os.PathLike, bjec.params.ParamsEvaluable[typing.Union[str, bytes, os.PathLike]][Union[str, bytes, os.PathLike]]]*)
Bases: object

evaluate_with_params(*params: Mapping[str, Any]*) → bjec.io.WriteableFromPath

path

write_to(*w: bjec.io.WriteOpenable*) → None

class bjec.io.**WriteableFromStr**(*content: str*, *encoding: Optional[str] = None*, *errors: Optional[str] = None*, *newline: Optional[str] = None*)
Bases: *bjec.io.Writeable*

content

encoding

errors

newline

write_to(*w: bjec.io.WriteOpenable*) → None

class bjec.io.**WriteableWrapFunc**(*func: Callable[[bjec.io.WriteOpenable], None]*)
Bases: *bjec.io.Writeable*

write_to(*w: bjec.io.WriteOpenable*) → None

bjec.io.**ensure_writeable**(*source: Union[bjec.io.Writeable, str, bytes]*) → bjec.io.Writeable

bjec.io.**resolve_abs_path**(*path: Union[str, bytes, os.PathLike, bjec.params.ParamsEvaluable[typing.Union[str, bytes, os.PathLike]][Union[str, bytes, os.PathLike]]], params: Mapping[str, Any]*) → Union[str, bytes]

bjec.io.**resolve_path**(*path: Union[str, bytes, os.PathLike, bjec.params.ParamsEvaluable[typing.Union[str, bytes, os.PathLike]][Union[str, bytes, os.PathLike]]], params: Mapping[str, Any]*) → Union[str, bytes]

bjec.io.**resolve_writable**(*source: Union[bjec.io.Writeable, str, bytes, bjec.params.ParamsEvaluable[typing.Union[bjec.io.Writeable, str, bytes]][Union[bjec.io.Writeable, str, bytes]]], params: Mapping[str, Any]*) → bjec.io.Writeable

4.1.10 bjec.job module

class bjec.job.**Job**(*constructor_func: Callable[[Job.Constructor], None], depends: List[Union[str, Callable[[...], None]]] = []*)
Bases: *bjec.master.Dependency*, *bjec.master.Constructible*, *bjec.master.Artefactor*, *bjec.master.WrapperRun*, *bjec.master.Runnable*

class Constructor(*obj: Any*)
Bases: bjec.master.ResolveConstructor, bjec.master.Constructor, bjec.master.Constructor

after(**after_funcs*) → None

collector

generator

processor

runnable

collector

generator

processor

run() → None
Must be implemented by inheriting classes.

runnable

bjec.job.**job**(*depends: List[Union[str, Callable[[...], None]]] = [], master: Optional[bjec.master.Master] = None*) → Callable[[Callable[[bjec.job.Job.Constructor], None]], Callable[[], None]]

4.1.11 bjec.json module

class bjec.json.**Writeable**(*value: Union[Any, bjec.params.ParamsEvaluable[typing.Any][Any], Iterable[Union[Any, bjec.params.ParamsEvaluable[typing.Any][Any]]], Iterable[Any], bjec.params.ParamsEvaluable[typing.Iterable[typing.Any]][Iterable[Any]], Mapping[Union[str, bjec.params.ParamsEvaluable[str][str]], Union[Any, bjec.params.ParamsEvaluable[typing.Any][Any]]], Mapping[str, Any], bjec.params.ParamsEvaluable[typing.Mapping[str, typing.Any]][Mapping[str, Any]]]*)

Bases: object

evaluate_with_params(*params: Mapping[str, Any]*) → bjec.io.WriteableWrapFunc

4.1.12 bjec.master module

class bjec.master.**Artefactor**

Bases: object

docstring for Artefactor

class Constructor

Bases: object

add_artefacts(***kwargs*) → None

add_artefacts(***kwargs*) → None

artefacts

w_run() → None

class bjec.master.**Constructible**

Bases: object

docstring for Constructible

class Constructor(*obj: Any*)

Bases: object

construct() → None

constructed

constructor_func

w_run() → None

class bjec.master.**Dependency**

Bases: *bjec.master.Registerable*, object

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

Bases: object

dependencies

class SetUpConstructor

Bases: object

depends (**args*) → None

depends (**args*) → None

fulfill () → None
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 () → bool

w_run () → None

class bjec.master.**Master**
Bases: object

register (*obj: bjec.master.Registerable, func: Callable[[...], None], aliases: Union[str, Sequence[str], None] = None*) → None

class bjec.master.**Registerable**
Bases: object

registered_with (*master: bjec.master.Master*) → None

class bjec.master.**Runnable**
Bases: abc.ABC

run () → None
Must be implemented by inheriting classes.

class bjec.master.**WrapperRun**
Bases: object

docstring for WrapperRun

run () → None

w_run () → None

4.1.13 bjec.params module

class bjec.params.**Call** (*func: Callable[[...], _T], *args, **kwargs*)
Bases: bjec.params._IdentityMixIn, bjec.params._WithMixIn, typing.Generic

Calls a function with ParamsEvaluable arguments.

Call can also be used to instantiate objects, as this happens in the same way a function is called.

Example

```
Call(Concatenate, file_path=Join("out.", P("n"), ".data"), close_files=True)
```

Parameters

- **func** – Function to be called.
- ***args** – Variable arguments passed to the class constructor. May contain `ParamsEvaluable` elements.
- ****kwargs** – Keyword arguments passed to the class constructor. May contain `ParamsEvaluable` values.

evaluate_with_params(*params: Mapping[str, Any]*) → _T

class bjec.params.**Dict**

Bases: `bjec.params._WithMixIn`, `typing.Generic`

Utility to construct complex dictionaries depending on parameters.

Example

```
{'--mu': P('mu')} + Dict.Conditional(lambda p: 'sigma' in p, {'--sigma': P('sigma
↪')}) + {P('extra_key'): P('extra_value')}
```

class Conditional(*condition: Callable[[Mapping[str, Any]], bool], m: Union[Mapping[Union[_T_inner, bjec.params.ParamsEvaluable[~_T_inner][_T_inner]], Union[_S_inner, bjec.params.ParamsEvaluable[~_S_inner][_S_inner]]], Mapping[_T_inner, _S_inner], bjec.params.ParamsEvaluable[typing.Mapping[~_T_inner, ~_S_inner]][Mapping[_T_inner, _S_inner]]]*)

Bases: `bjec.params._Part`

evaluate_with_params(*params: Mapping[str, Any]*) → Mapping[_T_inner, _S_inner]

class Literal(*m: Union[Mapping[Union[_T_inner, bjec.params.ParamsEvaluable[~_T_inner][_T_inner]], Union[_S_inner, bjec.params.ParamsEvaluable[~_S_inner][_S_inner]]], Mapping[_T_inner, _S_inner], bjec.params.ParamsEvaluable[typing.Mapping[~_T_inner, ~_S_inner]][Mapping[_T_inner, _S_inner]]]*)

Bases: `bjec.params._Part`

evaluate_with_params(*params: Mapping[str, Any]*) → Mapping[_T_inner, _S_inner]

class Pairs(*it: Union[Iterable[Union[Tuple[_T_inner, _S_inner], bjec.params.ParamsEvaluable[typing.Tuple[~_T_inner, ~_S_inner]][Tuple[_T_inner, _S_inner]]]], Iterable[Tuple[_T_inner, _S_inner]], bjec.params.ParamsEvaluable[typing.Iterable[typing.Tuple[~_T_inner, ~_S_inner]]][Iterable[Tuple[_T_inner, _S_inner]]]]*)

Bases: `bjec.params._Part`

evaluate_with_params(*params: Mapping[str, Any]*) → Mapping[_T_inner, _S_inner]

evaluate_with_params(*params: Mapping[str, Any]*) → Dict[_T, _S]

class bjec.params.**Join**(**args*, *sep: Optional[_T_sb] = None*)

Bases: `bjec.params._IdentityMixIn`, `bjec.params._WithMixIn`, `typing.Generic`

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** – Elements to join, may be instances of ParamsEvaluable classes.
- **sep** – 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 `''` (str).

evaluate_with_params (*params: Mapping[str, Any]*) → _T_sb

class bjec.params.**Lambda** (*func: Callable[[Mapping[str, Any]], _T]*)

Bases: bjec.params._IdentityMixIn, bjec.params._WithMixIn, typing.Generic

Calls a function with the params dict as the only argument.

Convenient way to compute values based on parameters using a lambda expression.

Example

```
Lambda(lambda p: p['alpha'] / p['beta'])
```

Parameters **func** – Function to be called on evaluation. The params dict is passed as the only argument.

evaluate_with_params (*params: Mapping[str, Any]*) → _T

class bjec.params.**List**

Bases: bjec.params._WithMixIn, typing.Generic

Utility to construct complex lists depending on parameters.

Example

```
['--mu', P('mu')] + List.Conditional(lambda p: 'sigma' in p, ['--sigma', P('sigma
↪')]) + ['-']
```

class Conditional (*condition: Callable[[Mapping[str, Any]], bool], it: Union[Iterable[Union[_T_inner, bjec.params.ParamsEvaluable[~_T_inner][_T_inner]]], Iterable[_T_inner], bjec.params.ParamsEvaluable[typing.Iterable[~_T_inner]][Iterable[_T_inner]]]*)

Bases: bjec.params._Part

evaluate_with_params (*params: Mapping[str, Any]*) → Iterable[_T_inner]

class Literal (*it: Union[Iterable[Union[_T_inner, bjec.params.ParamsEvaluable[~_T_inner][_T_inner]]], Iterable[_T_inner], bjec.params.ParamsEvaluable[typing.Iterable[~_T_inner]][Iterable[_T_inner]]]*)

Bases: bjec.params._Part

evaluate_with_params (*params: Mapping[str, Any]*) → Iterable[_T_inner]

evaluate_with_params (*params: Mapping[str, Any]*) → List[_T]

class bjec.params.**P**(*key: str*)

Bases: bjec.params._IdentityMixIn, bjec.params._WithMixIn, typing.Generic

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

```
Environment.Fluid().set(CPUS=P('n_cpus'))
```

Parameters **key** – Parameter (key of the parameter in the *params*) dict.

evaluate_with_params(*params: Mapping[str, Any]*) → _T

exception bjec.params.**ParamUnavailable**

Bases: KeyError

classmethod **wrap_params**(*params: Mapping[str, Any]*) → bjec.params._CustomKeyErrorMapping

Returns wrapped params raising ParamUnavailable on key miss.

class bjec.params.**ParamsEvaluable**(**args*, ***kwargs*)

Bases: typing_extensions.Protocol

evaluate_with_params(*params: Mapping[str, Any]*) → _T_co

class bjec.params.**Path**

Bases: bjec.params._WithMixIn

Utility to construct complex paths depending on parameters with pathlib.

Example

```
Path.Literal(base_dir) / P('scenario') / Path.Conditional(lambda p: 'sub_scenario
↪' in p, P('sub_scenario')) / Path.Format('{case}.csv')
```

class **Conditional**(*condition: Callable[[Mapping[str, Any]], bool]*, *s: Union[os.PathLike, str, bjec.params.ParamsEvaluable[typing.Union[os.PathLike, str]][Union[os.PathLike, str]]]*)

Bases: bjec.params._Part

evaluate_with_params(*params: Mapping[str, Any]*) → pathlib.PurePath

class **Format**(*format_str: str*, ***resolvables*)

Bases: bjec.params._Part

Expands a format string with the params dict on evaluation.

Example

```
Path.Format('{case}.csv')
```

Parameters

- **format_str** – Path which is expanded with the params dict values using `str.format()`.
- ****resolvables** – *Resolvable's which are resolved and made available as ''{name}'* (the argument's name) during `str.format()` evaluation.

evaluate_with_params (*params: Mapping[str, Any]*) → pathlib.PurePath

class Literal (*s: Union[os.PathLike, str, bjec.params.ParamsEvaluable[typing.Union[os.PathLike, str]][Union[os.PathLike, str]]]*)

Bases: `bjec.params._Part`

evaluate_with_params (*params: Mapping[str, Any]*) → pathlib.PurePath

evaluate_with_params (*params: Mapping[str, Any]*) → pathlib.PurePath

class bjec.params.**String**

Bases: `bjec.params._WithMixIn`

Utility to construct complex strings depending on parameters.

Example

```
P('lambda') + String.Format('.{mu}.') + String.Conditional(lambda p: 'sigma' in p,
↪ P('sigma'))
```

class Conditional (*condition: Callable[[Mapping[str, Any]], bool], s: Union[str, bjec.params.ParamsEvaluable[str][str]]*)

Bases: `bjec.params._Part`

evaluate_with_params (*params: Mapping[str, Any]*) → str

class Format (*format_str: str*, ***resolvables*)

Bases: `bjec.params._Part`

Expands a format string with the params dict on evaluation.

Example

```
String.Format('--nprocs={n}')
```

Parameters

- **format_str** – String which is expanded with the params dict values using `str.format()`.
- ****resolvables** – *Resolvable's which are resolved and made available as ''{name}'* (the argument's name) during `str.format()` evaluation.

evaluate_with_params (*params: Mapping[str, Any]*) → str

class Literal (*s: Union[str, bjec.params.ParamsEvaluable[str][str]]*)

Bases: `bjec.params._Part`

evaluate_with_params (*params: Mapping[str, Any]*) → str

evaluate_with_params (*params: Mapping[str, Any]*) → str

bjec.params.**ensure_multi_iterable**(*it: Union[Iterable[Union[_T, bjec.params.ParamsEvaluable[~_T][_T]]], Iterable[_T], bjec.params.ParamsEvaluable[typing.Iterable[~_T]][Iterable[_T]]]*) → Union[Iterable[Union[_T, bjec.params.ParamsEvaluable[~_T][_T]]], Iterable[_T], bjec.params.ParamsEvaluable[typing.Iterable[~_T]][Iterable[_T]]]

Returns a multi-iterable variant of it.

An iterator is a valid iterable but can only be iterated once. This function creates a semantic copy of `it` which can be iterated many times.

If `it` fulfills the `ParamsEvaluable` protocol, it is assumed that multi-iteration is supported and `it` is returned as is. If `it` fulfills the `Sequence` ABC, multi-iteration is supported and `it` is returned as is. Otherwise, `it` is read into a list which is then returned.

bjec.params.**resolve**(*obj: Union[_T, bjec.params.ParamsEvaluable[~_T][_T]], params: Mapping[str, Any]*) → _T

bjec.params.**resolve_dict**(*m: Union[Mapping[Union[_T, bjec.params.ParamsEvaluable[~_T][_T]], Union[_S, bjec.params.ParamsEvaluable[~_S][_S]]], Mapping[_T, _S], bjec.params.ParamsEvaluable[typing.Mapping[~_T, ~_S]][Mapping[_T, _S]]], params: Mapping[str, Any]*) → Dict[_T, _S]

bjec.params.**resolve_iterable**(*it: Union[Iterable[Union[_T, bjec.params.ParamsEvaluable[~_T][_T]]], Iterable[_T], bjec.params.ParamsEvaluable[typing.Iterable[~_T]][Iterable[_T]]], params: Mapping[str, Any]*) → Iterable[_T]

bjec.params.**resolve_list**(*it: Union[Iterable[Union[_T, bjec.params.ParamsEvaluable[~_T][_T]]], Iterable[_T], bjec.params.ParamsEvaluable[typing.Iterable[~_T]][Iterable[_T]]], params: Mapping[str, Any]*) → List[_T]

bjec.params.**resolve_mapping**(*m: Union[Mapping[Union[_T, bjec.params.ParamsEvaluable[~_T][_T]], Union[_S, bjec.params.ParamsEvaluable[~_S][_S]]], Mapping[_T, _S], bjec.params.ParamsEvaluable[typing.Mapping[~_T, ~_S]][Mapping[_T, _S]]], params: Mapping[str, Any]*) → Mapping[_T, _S]

bjec.params.**transform**(*obj: Union[_T, bjec.params.ParamsEvaluable[~_T][_T]], transform_func: Callable[[_T], _S]*) → Union[_S, bjec.params.ParamsEvaluable[~_S][_S]]

4.1.14 bjec.process module

class bjec.process.**Environment**(*variables: Iterable[Tuple[Union[str, bjec.params.ParamsEvaluable[str][str]], Union[str, bjec.params.ParamsEvaluable[str][str]]]]*)

Bases: `object`

class Fluid

Bases: `object`

build() → bjec.process.Environment

from_environment(*environment: bjec.process.Environment*) → bjec.process.Environment.Fluid

inherit(*blacklist: Iterable[str] = {}, whitelist: Iterable[str] = {}*) → bjec.process.Environment.Fluid

set(***variables*) → bjec.process.Environment.Fluid

Use __add__ if the keys must be ParamsEvaluable or the variables are available as an iterable.

unset(**variables*) → bjec.process.Environment.Fluid

unset_from_iterable(*variables: Iterable[Union[str, bjec.params.ParamsEvaluable[str][str]]]*) → bjec.process.Environment.Fluid

evaluate_with_params(*params: Mapping[str, Any]*) → Dict[str, str]

class bjec.process.**FileAccessor**(*name: str, open_path: Union[str, bytes], path: Union[str, bytes, None] = None*)

Bases: object

Represents a file accessible for reading.

name

open_bytes() → io.BufferedIOBase

open_path

open_text(*encoding: Optional[str] = None*, *errors: Optional[str] = None*, *newline: Optional[str] = None*) → io.TextIOBase

path

class bjec.process.**Process**

Bases: object

Process template which may contain parameter reference.

Implementer refers to a component which interprets a Process instance and executes a program accordingly for each ParamSet. It then constructs a Result which is treated as the result for each ParamSet.

Implementers should receive the information about a process execution through *Process.with_params()*. All fields are resolved and simplified as far as possible through property accessors in *Process.WithParams*.

Regarding all file related methods of the Fluid builder: Further configuration options may be made available as part of an implementer's configuration. This might include details such as the directory for temporary files, the temporary file class to use, buffering details, network transfer options, ...

Lifecycle: Construction using Fluid. Passing to processor / runner. Deferred result passing to following stages (linked to process instance? this could perform checking, e.g. is stdout available?). Process execution for each ParamSet. Construction and return of a Result instance. Finish of result processing (causes cleanup) after the following stage is done with the Result instance.

class FailureMode(*interpret_exit_code: Union[Callable[[int], bool], NoneType] = None, interpret_stderr: Union[Callable[[ForwardRef('FileAccessor')], bool], NoneType] = None, interpret_stdout: Union[Callable[[ForwardRef('FileAccessor')], bool], NoneType] = None*)

Bases: object

interpret_exit_code = None

interpret_stderr = None

interpret_stdout = None

class Fluid

Bases: object

add_input_file(*name: str, source: Union[bjec.io.Writeable, str, bytes, bjec.params.ParamsEvaluable[typing.Union[bjec.io.Writeable, str, bytes]][Union[bjec.io.Writeable, str, bytes]]], path: Union[str, bytes, os.PathLike, bjec.params.ParamsEvaluable[typing.Union[str, bytes, os.PathLike]][Union[str, bytes, os.PathLike]], None] = None, must_not_exist: bool = True, create_parents: bool = False, mode: int = 438, cleanup_after_finish: bool = False*) → bjec.process.Process.Fluid

Adds an input file to the Process.

Parameters

- **name** – Name through which the file is available for referencing. The file's path is available as `P('__file_NAME')` during evaluation of all `ParamsEvaluable` constructs of the `Process`. If an input file with this name already exists, its configuration is overwritten. The same name must not be used for an input file and an output file, *`Process.validate()`* will raise if this is the case.
- **source** – Source of the file's content. Use `WriteableFromPath` to refer to a file in the file system.
- **path** – If not `None` the input file is made available at this path when the result is yielded. Otherwise the implementer may use a temporary file.
- **must_not_exist** – If `True` the execution is considered failed if the file already exists before the process is started. This is evaluated before the process is started and before the file is created from `source`. Only considered if `path` is not `None`, as otherwise the implementer manages the file.
- **create_parents** – If `True` all parent directories of the file are created if non-existent. Directories are created with the default mode, disregarding the `mode` parameter. Only considered if `path` is not `None`.
- **mode** – Mode bits of the file, see `os.open()` for details. Only considered when `path` is not `None` and `source` is not `None`.
- **cleanup_after_finish** – If `True` the input file is deleted when the *finish* lifetime stage is reached. Only considered when `path` is not `None`.

add_output_file(*name: str, path: Union[str, bytes, os.PathLike, bjec.params.ParamsEvaluable[typing.Union[str, bytes, os.PathLike]][Union[str, bytes, os.PathLike]], None] = None, must_not_exist: bool = True, create: bool = True, create_parents: bool = False, mode: int = 438, cleanup_after_finish: bool = False*) → bjec.process.Process.Fluid

Adds an output file to the Process.

Parameters

- **name** – Name through which the file is available for referencing. The file's path is available as `P('__file_NAME')` during evaluation of all `ParamsEvaluable` constructs of the `Process`. If an output file with this name already exists, its configuration is overwritten. The same name must not be used for an input file and an output file, *`Process.validate()`* will raise if this is the case.
- **path** – If not `None` the output file is made available at this path when the result is yielded. Otherwise the implementer may use a temporary file.
- **must_not_exist** – If `True` the execution is considered failed if the file already exists before the process is started. This is evaluated before the process is started and before the file is created via `create`. Only considered if `path` is not `None`, as otherwise the implementer manages the file. If `False` it is considered a failure if the process did not create the file.
- **create** – If `True` the file is ensured to be present before the process is started. If `False` the file is ensured to not be present, meaning any file at the path will be deleted.
- **create_parents** – If `True` all parent directories of the file are created if non-existent. Directories are created with the default mode, disregarding the `mode` parameter. The directories are created before the process is started. Only considered if `path` is not `None`.
- **mode** – Mode bits of the file, see `os.open()` for details. Only considered when `path` is not `None`. If `create` is `True`, the bits are set before the process is started, otherwise after the process has finished.
- **cleanup_after_finish** – If `True` the output file is deleted when the *finish* lifetime stage is reached. Only considered when `path` is not `None`.

args(**args*) → bjec.process.Process.Fluid

Sets the argument list with which the process is started.

Paths of input and output files are available during evaluation as `P('__file_NAME')`.

args_from_iterable(*args: Union[Iterable[Union[str, bjec.params.ParamsEvaluable[str][str]]], Iterable[str], bjec.params.ParamsEvaluable[typing.Iterable[str]][Iterable[str]]]*) → bjec.process.Process.Fluid

Sets the argument list with which the process is started.

Paths of input and output files are available during evaluation as `P('__file_NAME')`.

build() → bjec.process.Process

capture_stderr(*capture: bool = True, path: Union[str, bytes, os.PathLike, bjec.params.ParamsEvaluable[typing.Union[str, bytes, os.PathLike]][Union[str, bytes, os.PathLike]], None] = None, must_not_exist: bool = True, create_parents: bool = False, mode: int = 438, cleanup_after_finish: bool = False*) → bjec.process.Process.Fluid

Configure whether and how stderr is captured.

Parameters

- **capture** – If `True` stderr is captured and made available in `Result` instances. Subsequent calls may disable capturing by setting `False`.
- **path** – If not `None` the stderr is made available at this path. Otherwise the implementer may use a temporary file or store content in-memory.
- **must_not_exist** – If `True` the execution is considered failed if the file already exists before the process is started. This is evaluated before the process is started. Only considered if `path` is not `None`, as otherwise the implementer manages the file.
- **create_parents** – If `True` all parent directories of the file are created if non-existent. Directories are created with the default mode, disregarding the `mode` parameter. Only considered if `path` is not `None`.
- **mode** – Mode bits of the file, see `os.open()` for details. Only considered when `path` is not `None`.
- **cleanup_after_finish** – If `True` the stderr file is deleted when the *finish* lifetime stage is reached. Only considered when `path` is not `None`.

Raises `ValueError` – If the combination of arguments is not valid.

capture_stdout(*capture: bool = True, path: Union[str, bytes, os.PathLike, bjec.params.ParamsEvaluable[typing.Union[str, bytes, os.PathLike]][Union[str, bytes, os.PathLike]], None] = None, must_not_exist: bool = True, create_parents: bool = False, mode: int = 438, cleanup_after_finish: bool = False*) → bjec.process.Process.Fluid

Configure whether and how stdout is captured.

Parameters

- **capture** – If `True` stdout is captured and made available in `Result` instances. Subsequent calls may disable capturing by setting `False`.
- **path** – If not `None` the stdout is made available at this path. Otherwise the implementer may use a temporary file or store content in-memory.
- **must_not_exist** – If `True` the execution is considered failed if the file already exists before the process is started. This is evaluated before the process is started. Only considered if `path` is not `None`, as otherwise the implementer manages the file.
- **create_parents** – If `True` all parent directories of the file are created if non-existent. Directories are created with the default mode, disregarding the `mode` parameter. Only considered if `path` is not `None`.
- **mode** – Mode bits of the file, see `os.open()` for details. Only considered when `path` is not `None`.
- **cleanup_after_finish** – If `True` the stdout file is deleted when the *finish* lifetime stage is reached. Only considered when `path` is not `None`.

Raises `ValueError` – If the combination of arguments is not valid.

cmd(*cmd: Union[str, bjec.params.ParamsEvaluable[str][str]]*) → bjec.process.Process.Fluid

Sets the command to be executed.

The command has to be set, a process cannot execute without setting this. If unset, *Process.validate()* will raise.

How the command is resolved to a path is up to the implementer.

connect_stdin(*source: Union[bjec.io.Writeable, str, bytes, bjec.params.ParamsEvaluable[typing.Union[bjec.io.Writeable, str, bytes]][Union[bjec.io.Writeable, str, bytes]], None] = None, path: Union[str, bytes, os.PathLike, bjec.params.ParamsEvaluable[typing.Union[str, bytes, os.PathLike]][Union[str, bytes, os.PathLike]], None] = None, must_not_exist: bool = True, create_parents: bool = False, mode: int = 438, cleanup_after_finish: bool = False*) → bjec.process.Process.Fluid

Configures a file to connect to the process's stdin.

Parameters

- **source** – Source of the file's content. Use `WriteableFromPath` to refer to a file in the file system. The value `None` is the same as specifying an empty file.
- **path** – If not `None` the file is made available at this path when the result is yielded. Otherwise the implementer may use a temporary file.
- **must_not_exist** – If `True` the execution is considered failed if the file already exists before the process is started. This is evaluated before the process is started and before the file is created from `source`. Only considered if `path` is not `None`, as otherwise the implementer manages the file.
- **create_parents** – If `True` all parent directories of the file are created if non-existent. Directories are created with the default mode, disregarding the `mode` parameter. Only considered if `path` is not `None`.
- **mode** – Mode bits of the file, see `os.open()` for details. Only considered when `path` is not `None` and `source` is not `None`.
- **cleanup_after_finish** – If `True` the file is deleted when the *finish* lifetime stage is reached. Only considered when `path` is not `None`.

environment(*environment: Union[Mapping[Union[str, bjec.params.ParamsEvaluable[str][str]], Union[str, bjec.params.ParamsEvaluable[str][str]]], Mapping[str, str], bjec.params.ParamsEvaluable[typing.Mapping[str, str]][Mapping[str, str]]]*) → bjec.process.Process.Fluid

Sets the environment variables of the process.

The recommended way of constructing an environment is the *Environment* type. It can be built through a fluid interface via *Environment.Fluid*.

Paths of input and output files are available during evaluation as `P('__file_NAME')`.

failure_mode(*interpret_exit_code: Optional[Callable[[int], bool]] = None, interpret_stderr: Optional[Callable[[FileAccessor], bool]] = None, interpret_stdout: Optional[Callable[[FileAccessor], bool]] = None*) → bjec.process.Process.Fluid

Configure when a process execution is considered to be failed.

The default behaviour is to consider any execution returning a non-0 exit code as failed. If any argument is passed, this behaviour is disabled.

If *any* predicate evaluates to `True`, the execution is considered a failure.

Parameters

- **interpret_exit_code** – Predicate function to interpret the exit code. Return `True` if the exit code is considered a failure.

- **interpret_stderr** – Predicate function to interpret the stderr stream. This only works if stderr capturing is configured via *capture_stderr()*, otherwise *Process.validate()* will raise. Return `True` if the exit code is considered a failure.
- **interpret_stdout** – Predicate function to interpret the stdout stream. This only works if stdout capturing is configured via *capture_stdout()*, otherwise *Process.validate()* will raise. Return `True` if the exit code is considered a failure.

remove_input_file(*name: str*) → bjec.process.Process.Fluid
Removes an input file from the Process by name.

remove_output_file(*name: str*) → bjec.process.Process.Fluid
Removes an output file from the Process by name.

working_directory(*dir: Union[str, bjec.params.ParamsEvaluable[str][str], None]*) → bjec.process.Process.Fluid
Sets the working directory of the process.

If unset, implementers may execute in any directory.

class InputFile(*name: str*, *source: bjec.io.Writeable*, *path: Union[str, bytes, NoneType] = None*, *must_not_exist: bool = True*, *create_parents: bool = False*, *mode: int = 438*, *cleanup_after_finish: bool = False*)
Bases: `object`

cleanup_after_finish = False

create_parents = False

mode = 438

must_not_exist = True

path = None

class OutputFile(*name: str*, *path: Union[str, bytes, NoneType] = None*, *must_not_exist: bool = True*, *create: bool = True*, *create_parents: bool = False*, *mode: int = 438*, *cleanup_after_finish: bool = False*)
Bases: `object`

cleanup_after_finish = False

create = True

create_parents = False

mode = 438

must_not_exist = True

path = None

class Stdin(*source: Union[bjec.io.Writeable, NoneType] = None*, *path: Union[str, bytes, NoneType] = None*, *must_not_exist: bool = True*, *create_parents: bool = False*, *mode: int = 438*, *cleanup_after_finish: bool = False*)
Bases: `object`

cleanup_after_finish = False

connected

create_parents = False

mode = 438

must_not_exist = True

path = None

source = None

class Stdout(*capture: bool = False*, *path: Union[str*, *bytes*, *NoneType] = None*, *must_not_exist: bool = True*, *create_parents: bool = False*, *mode: int = 438*, *cleanup_after_finish: bool = False*)

Bases: object

capture = False

cleanup_after_finish = False

create_parents = False

mode = 438

must_not_exist = True

path = None

class WithParams(*process: bjec.process.Process, params: Mapping[str, Any]*)

Bases: object

args

cmd

environment

failure_mode

input_files

output_files

stderr

stdin

stdout

working_directory

validate() → None

Raises if this instance is not complete or inconsistent.

with_params(*params: Mapping[str, Any]*) → bjec.process.Process.WithParams

class bjec.process.**Result**(*exit_code: int*, *stdin: Optional[bjec.process.FileAccessor] = None*, *stdout: Optional[bjec.process.FileAccessor] = None*, *stderr: Optional[bjec.process.FileAccessor] = None*, *input_files: Optional[Dict[str*, *bjec.process.FileAccessor]] = None*, *output_files: Optional[Dict[str*, *bjec.process.FileAccessor]] = None*)

Bases: object

exit_code

input_file(*name: str*) → bjec.process.FileAccessor

output_file(*name: str*) → bjec.process.FileAccessor

stderr

stdin

stdout

4.1.15 bjec.processor module

class bjec.processor.**Processor**
Bases: typing.Generic, abc.ABC

docstring for Processor

process(*runnable: Any, params_it: Iterable[Mapping[str, Any]]*) → Iterator[Tuple[Mapping[str, Any], _T_co]]
Process all parameter sets in the iterable according to a runnable.

Must be implemented by inheriting classes.

4.1.16 bjec.subprocessor module

class bjec.subprocessor.**FileDescriptor**(*name: str, open_path: Union[str, bytes], process_path: Union[str, bytes], temporary: bool, cleanup: bool, path: Union[str, bytes, NoneType] = None*)
Bases: object

accessor() → bjec.process.FileAccessor

path = None

class bjec.subprocessor.**Subprocessor**(*max_processes: int = 0*)
Bases: *bjec.processor.Processor*

Subprocessor runs Process executions concurrently using threads.

Parameters **max_processes** – Maximum number of processes to be run concurrently. If <= 0, the configuration option of the same name is used instead. 1 is used if the configuration option is not set.

Configuration Options:

- **max_processes: Maximum number of processes to be run** concurrently. The option is used when max_processes passed to the constructor is <= 0.

max_processes

process(*runnable: Any, params_it: Iterable[Mapping[str, Any]]*) → Iterator[Tuple[Mapping[str, Any], bjec.process.Result]]
Process all parameter sets in the iterable according to a runnable.

Must be implemented by inheriting classes.

bjec.subprocessor.**prepare_input_file**(*spec: Union[bjec.process.Process.InputFile, bjec.process.Process.Stdin], exit_handlers: bjec.utils.HandlersCollector, cleanup_handlers: bjec.utils.HandlersCollector, name: str = '', temp_dir: Optional[str] = None*) → bjec.subprocessor.FileDescriptor

bjec.subprocessor.**prepare_output_file**(*spec: Union[bjec.process.Process.OutputFile, bjec.process.Process.Stdout], exit_handlers: bjec.utils.HandlersCollector, cleanup_handlers: bjec.utils.HandlersCollector, name: str = '', temp_dir: Optional[str] = None*) → bjec.subprocessor.FileDescriptor

4.1.17 bjec.utils module

class bjec.utils.**CallbackOnException**(*f: Callable[[...], None], *args, **kwargs*)
Bases: object

Context manager calling a function on exit only if an exception occurred.

class bjec.utils.**HandlersCollector**(**args*, ***kwargs*)
Bases: typing_extensions.Protocol

callback(*callback: Callable[[...], Any], *args, **kwargs*) → Callable[[...], Any]

class bjec.utils.**HandlersList**
Bases: object

callback(*callback: Callable[[...], Any], *args, **kwargs*) → Callable[[...], Any]

clear() → None

bjec.utils.**consume**(*it: Iterable[Any], n: Optional[int] = None*) → None
Advance the iterable it n-steps ahead. If n is None, consume entirely.

Copied from: https://docs.python.org/3.7/library/itertools.html#itertools-recipes

bjec.utils.**listify**(*obj: Union[_T, Sequence[_T], None], none_empty: bool = False*) → List[_T]
Turns obj into a list. Returns [obj] if it.

Returns obj is simply returned, if it already is a list. 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.

4.1.18 Module contents

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Python Module Index

b

Index

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W