`openlane.steps` API

The Step Module

This modules includes various functions for importing and/or generating OpenLane configuration objects. Configuration objects are the primary input to a flow.

```
exception openlane.steps.StepError(*args, underlying_error: Exception |
    None = None, **kwargs)
```

Bases: `RuntimeError`

A `RuntimeError` that occurs when a Step fails to finish execution properly.

PARAMETERS:

underlying_error (*Optional[Exception]*)

```
exception openlane.steps.DeferredStepError(*args, underlying_error:
    Exception | None = None, **kwargs)
```

Bases: `StepError`

A variant of `StepError` where parent Flows are encouraged to continue execution of subsequent steps regardless and then finally flag the Error at the very end.

PARAMETERS:

underlying_error (*Optional[Exception]*)

```
exception openlane.steps.StepException(*args, underlying_error: Exception
    | None = None, **kwargs)
```

latest

Bases: `StepError`

A variant of `StepError` for unexpected failures or failures due to misconfiguration, such as:

- Invalid inputs
- Mis-use of class interfaces of the `Step`
- Other unexpected failures

PARAMETERS:

underlying_error (*Optional[Exception]*)

```
exception openlane.steps.StepNotFound(*args: object, id: str | None = None)
```

Bases: `NameError`

PARAMETERS:

- **args** (*object*)
- **id** (*Optional[str]*)

RETURN TYPE:

None

```
class openlane.steps.Step(config: Config | None = None, state_in: State | None | Future[State] = None, *, id: str | None = None, name: str | None = None, long_name: str | None = None, flow: Any | None = None, _config_quiet: bool = False, _no_revalidate_conf: bool = False, **kwargs)
```

Bases: `ABC`

An abstract base class for Step objects.

Steps encapsulate a subroutine that acts upon certain classes of formats in an input state and returns a new output state with updated design format paths and/or metrics.

latest

Warning: The initializer for Step is not thread-safe. Please use it on the main thread and then, if you're using a Flow object, use `start_step_async`, or if

you're not, you may use `start` in another thread. That part's fine.

PARAMETERS:

- **config** (*Optional[Config]*) –

  A configuration object.

  If running in interactive mode, you can set this to `None`, but it is otherwise required.

- **state_in** (*Future[State]*) –

  The state object this step will use as an input.

  The state may also be a `Future[State]`, in which case, the `start()` call will block until that Future is realized. This allows you to chain a number of asynchronous steps.

  See https://en.wikipedia.org/wiki/Futures_and_promises for a primer.

  If running in interactive mode, you can set this to `None`, where it will use the last generated state, but it is otherwise required.

- **step_dir** –

  A "scratch directory" for the step. Required.

  You may omit this argument as `None` if "flow" is specified.

- **id** (*str*) –

  A string ID for the Step. The convention is f"{a}.{b}", where the first is common between all Steps using the same tools.

  The ID should be in `UpperCamelCase`.

  While this is technically a class variable, instances allowed to change it per-instance to disambiguate when the same step is used multiple times in a flow.

  `Step` subclasses without the `id` class property declared are considered abstract and cannot be initialized or used in a `Flow`.

- **name** (*str*) –

  A short name for the Step, used in progress bars and the like.

  While this is technically an instance variable [illegible] subclass to override this variable and instances are only to change it to disambiguate when the same step is used multiple times in a flow.

- **long_name** (*str*) –

  A longer descriptive for the Step, used to delimit logs.

  While this is technically an instance variable, it is expected for every subclass to override this variable and instances are only to change it to disambiguate when the same step is used multiple times in a flow.

- **flow** (*Optional[Any]*) – Deprecated: the parent flow. Ignored if passed.
- **_config_quiet** (*bool*)
- **_no_revalidate_conf** (*bool*)

VARIABLES:

- **inputs** –

  A list of `openlane.state.DesignFormat` objects that are required for this step. These will be validated by the `start()` method.

  `Step` subclasses without the `inputs` class property declared are considered abstract and cannot be initialized or used in a `Flow`.

- **outputs** –

  A list of `openlane.state.DesignFormat` objects that may be emitted by this step. A step is not allowed to modify design formats not declared in `outputs`.

  `Step` subclasses without the `outputs` class property declared are considered abstract and cannot be initialized or used in a `Flow`.

- **config_vars** – A list of configuration `openlane.config.Variable` objects to be used to alter the behavior of this Step.
- **output_processors** – A default set of `openlane.steps.OutputProcessor` classes for use with `run_subprocess()`.
- **state_out** –

  The last output state from running this step object, if it exists.

  If `start()` is called again, the reference is destroyed.

- **start_time** –

  The last starting time from running this step object, if it exists.

  If `start()` is called again, the reference is destroyed.

- **end_time** –

  The last ending time from running this step object, if it exists.

  If `start()` is called again, the reference is destroyed.

- **toolbox** –

  The last `Toolbox` used while running this step object, if it exists.

  If `start()` is called again, the reference is destroyed.

```
warn(msg: object, /, **kwargs)
```

latest

Logs to the OpenLane logger with the log level WARNING, appending

the step's ID as extra data.

PARAMETERS:

msg (*object*) – The message to log

`err(msg: object, /, **kwargs)`

Logs to the OpenLane logger with the log level ERROR, appending the step's ID as extra data.

PARAMETERS:

msg (*object*) – The message to log

`classmethod assert_concrete(action: str = 'initialized')`

Checks if the Step class in question is concrete, with abstract methods AND `NotImplemented` classes implemented and declared respectively.

Should be called before any `Step` subclass is used.

If the class is not concrete, a `NotImplementedError` is raised.

PARAMETERS:

action (*str*) – The action to be attempted, to be included in the `NotImplementedError` message.

`classmethod get_help_md(*, docstring_override: str = '', use_dropdown: bool = False)`

Renders Markdown help for this step to a string.

PARAMETERS:

- **docstring_override** (*str*)
- **use_dropdown** (*bool*)

`classmethod display_help()`

IPython-only. Displays Markdown help for a given step.

`layout_preview() → str | None`

RETURNS:

An HTML tag that could act as a preview for a specific stage or `None` if a preview is unavailable for this

RETURN TYPE:

str | None

display_result()

IPython-only. Displays the results of a given step.

classmethod load(config: str | PathLike | Config, state_in: str | State, pdk_root: str | None = None) → Step

Creates a step object, but instead of using a Flow or a global state, the config_path and input state are deserialized from JSON files.

Useful for re-running steps that have already run.

PARAMETERS:

- **config** (*str* | *PathLike* | *Config*) –

  (Path to) a **Step-filtered** configuration

  The step will not tolerate variables unrelated to this specific step.

- **state** – (Path to) a valid input state
- **pdk_root** (*str* | *None*) –

  The PDK root, which is needed for some utilities.

  If your utility doesn't require it, just keep the default value as-is.

- **state_in** (*str* | *State*)

RETURNS:

The created step object

RETURN TYPE:

Step

create_reproducible(target_dir: str, include_pdk: bool = True, flatten: bool = False)

Creates a folder that, given a specific version of OpenLane being installed, makes a portable reproducible of that step's execution.

..note

> Reproducibles are limited on Magic and Netgen, as their RC files form an indirect dependency on many *.maa* files or similar that cannot be enumerated by OpenLane.

latest

Reproducibles are automatically generated for failed steps, but this may be called manually on any step, too.

PARAMETERS:

- **target_dir** (*str*) – The directory in which to create the reproducible
- **include_pdk** (*bool*) – Include PDK files. If set to false, Path pointing to PDK files will be prefixed with `pdk_dir::` instead of being copied.
- **flatten** (*bool*) – Creates a reproducible with a flat (single-directory) file structure, except for the PDK which will maintain its internal folder structure (as it is sensitive to it.)

```
start(toolbox: Toolbox | None = None, step_dir: str | None = None,
    _no_rule: bool = False, **kwargs) → State
```

Begins execution on a step.

This method is final and should not be subclassed.

PARAMETERS:

- **toolbox** (*Toolbox* | *None*) –

  The flow's `Toolbox` object, required.

  If running in interactive mode, you may omit this argument as `None`, where a global toolbox will be used instead.

  If running inside a flow, you may also omit this argument as `None`, where the flow's toolbox will used to be instead.

- ****kwargs** – Passed on to subprocess execution: useful if you want to redirect stdin, stdout, etc.
- **step_dir** (*str* | *None*)
- **_no_rule** (*bool*)

RETURNS:

An altered State object.

RETURN TYPE:

State

```
abstract run(state_in: State, **kwargs) → Tuple[Dict[DesignFormat,
    Path | List[Path] | Dict[str, Path | List[Path]]], State,
    Dict[str, Any]]
```

latest

protected

The "core" of a step.

This step is considered per-object private, i.e., if a Step's run is called anywhere outside of the same object's `start()`, its behavior is undefined.

PARAMETERS:

- **state_in** (*State*) –

  The input state.

  Note that `self.state_in` is stored as a future and would need to be resolved before use first otherwise.

  For reference, `start()` is responsible for resolving it for `.run()`.

- ****kwargs** – Passed on to subprocess execution: useful if you want to redirect stdin, stdout, etc.

RETURN TYPE:

Tuple[*Dict*[*DesignFormat*, *Path* | *List*[*Path*] | *Dict*[str, *Path* | *List*[*Path*]] | None], *Dict*[str, *Any*]]

```
get_log_path() → str
```

protected

RETURNS:

the default value for `run_subprocess()`'s "log_to" parameter.

Override it to change the default log path.

RETURN TYPE:

str

```
run_subprocess(cmd: ~typing.Sequence[str | ~os.PathLike], log_to: str
    | ~os.PathLike | None = None, silent: bool = False, report_dir:
    str | ~os.PathLike | None = None, env: ~typing.Dict[str,
    ~typing.Any] | None = None, *, check: bool = True,
    output_processing:
    ~typing.Sequence[~typing.Type[~openlane.steps.step.OutputProcess
    or]] | None = None, _popen_callable: ~typing.Callable[[...],
    ~psutil.Popen] = <class 'psutil.Popen':
    Any]
```

latest

protected

A helper function for `Step` objects to run subprocesses.

The output from the subprocess is processed line-by-line by instances

of output processor classes.

PARAMETERS:

- **cmd** (*Sequence[str | PathLike]*) – A list of variables, representing a program and its arguments, similar to how you would use it in a shell.
- **log_to** (*str | PathLike | None*) – An optional override for the log path from `get_log_path()`. Useful for if you run multiple subprocesses within one step.
- **silent** (*bool*) – If specified, the subprocess does not print anything to the terminal. Useful when running multiple processes simultaneously.
- **report_dir** (*str | PathLike | None*) – An optional override for where reports by output processors
- **check** (*bool*) – Whether to raise `subprocess.CalledProcessError` in the event of a non-zero exit code. Set to `False` if you'd like to do further processing on the output(s).
- **output_processing** (*Sequence[Type[OutputProcessor]] | None*) – An override for the class's list of `openlane.steps.OutputProcessor` classes.
- ****kwargs** – Passed on to subprocess execution: useful if you want to redirect stdin, stdout, etc.
- **env** (*Dict[str, Any] | None*)
- **_popen_callable** (*Callable[[...], Popen]*)

RETURNS:

A dictionary of output processor results.

These key/value pairs are included in all cases: * `returncode`: Exit code for the subprocess * `log_path`: The resolved log path for the subprocess

The other key value pairs depend on the `key` class variables and `openlane.steps.OutputProcessor.result()` methods of the output processors.

RAISES:

subprocess.CalledProcessError – If the process has a non-zero exit, and `check` is True, this exception will be raised.

RETURN TYPE:

Dict[str, *Any*]

`extract_env(kwargs) → Tuple[dict, Dict[str, str]]`

protected

An assisting function: Given a `kwargs` object, it does the following:

> - **If the kwargs object has an "env" variable, it separates it into**
> its own variable.
> - **If the kwargs object has no "env" variable, a new "env" dictionary**
> is created based on the current environment.

PARAMETERS:

kwargs – A Python keyword arguments object.

RETURNS (KWARGS, ENV):

A kwargs without an `env` object, and an isolated `env` object.

RETURN TYPE:

Tuple[dict, *Dict*[str, str]]

`classmethod with_id(id: str) → Type[Step]`

Syntactic sugar for creating a subclass of a step with a different ID.

Useful in flows, where you want different IDs for different instance of the same step.

PARAMETERS:

id (*str*)

RETURN TYPE:

Type[*Step*]

`class StepFactory`

Bases: `object`

latest

A factory singleton for Steps, allowing steps types to be registered and then retrieved by name.

See https://en.wikipedia.org/wiki/Factory_(object-oriented_programming) for a primer.

classmethod register() → Callable[[Type[Step]], Type[Step]]

Adds a step type to the registry using its `Step.id` attribute.

RETURN TYPE:

Callable[[*Type*[*Step*]], *Type*[*Step*]]

classmethod get(name: str) → Type[Step] | None

Retrieves a Step type from the registry using a lookup string.

PARAMETERS:

name (*str*) – The registered name of the Step. Case-insensitive.

RETURN TYPE:

Type[*Step*] | None

classmethod list() → List[str]

RETURNS:

A list of IDs of all registered names.

RETURN TYPE:

List[str]

factory

alias of `StepFactory`

class openlane.steps.OutputProcessor(step: Step, report_dir: str, silent: bool)

Bases: `ABC`, `Generic[VT]`

latest

An abstract base class that processes terminal output from `openlane.steps.Step.run_subprocess()` and append a resultant key/value pair to its returned dictionary.

PARAMETERS:

- **step** (*Step*) – The step object instantiating this output processor
- **report_dir** (*str*) – The report directory for this instantiation of `run_subprocess`.
- **silent** (*bool*) – Whether the `run_subprocess` was called with `silent` or not.

VARIABLES:

key – The fixed key to be added to the return value of `run_subprocess`. Must be implemented by subclasses.

```
abstract process_line(line: str) → bool
```

Fires when a line is received by `openlane.steps.Step.run_subprocess()`. Subclasses may do any arbitrary processing here.

PARAMETERS:

line (*str*) – The line emitted by the subprocess

RETURNS:

`True` if the line is "consumed", i.e. other output processors are skipped. `False` if the line is to be passed on to later output processors.

RETURN TYPE:

bool

```
abstract result() → VT
```

RETURNS:

The result of all previous `process_line` calls.

RETURN TYPE:

VT

```
class openlane.steps.DefaultOutputProcessor(*args, **kwargs)
```

Bases: `OutputProcessor`[`Dict`[`str`, `Any`]]

An output processor that makes a number of special functions accessible to subprocesses by simply printing keywords in the terminal, such as:

- `%OL_CREATE_REPORT <file>`: Starts redirecting all output from standard output to a report file inside the step directory, with the name <file>.
- `%OL_END_REPORT`: Stops redirection behavior.
- `%OL_METRIC <name> <value>`: Adds a string metric with the name <name> and the value <value> to this function's returned object.
- `%OL_METRIC_F <name> <value>`: Adds a floating-point metric with the name <name> and the value <value> to this function's returned object.
- `%OL_METRIC_I <name> <value>`: Adds an integer metric with the name <name> and the value <value> to this function's returned object.

Otherwise, the line is simply printed to the logger.

```
process_line(line: str) → bool
```

Always returns `True`, so `DefaultOutputProcessor` should always be at the end of your list.

PARAMETERS:

line (*str*)

RETURN TYPE:

bool

```
result() → Dict[str, Any]
```

A dictionary of all generated metrics.

RETURN TYPE:

Dict[str, *Any*]

```
class openlane.steps.TclStep(config: Config | None = None, state_in: State | None | Future[State] = None, *, id: str | None = None, name: str | None = None, long_name: str | None = None, flow: Any | None = None, _config_quiet: bool = False, _no_revalidate_conf: bool = False, **kwargs)
```

Bases: `Step`

A subclass of `Step` that primarily deals with run as Yosys, OpenROAD and Magic.

A TclStep Step should ideally correspond to running one Tcl script with such a utility.

VARIABLES:

reproducibles_allowed – Whether this class can generate reproducibles.

PARAMETERS:

- **config** (*Optional[Config]*)
- **state_in** (*Future[State]*)
- **id** (*str*)
- **name** (*str*)
- **long_name** (*str*)
- **flow** (*Optional[Any]*)
- **_config_quiet** (*bool*)
- **_no_revalidate_conf** (*bool*)

```
static value_to_tcl(value: Any) → str
```

Converts an arbitrary Python value to Tcl as follows:

- If the value is an instance of a dataclass, it is serialized as a JSON object.
- If the value is a list, it is joined using `TclUtils.join()`.
- **If the value is a dict, the keys and values are escaped recursively using:**

  joined using `TclUtils.join()`.

- If the value is an Enum, its name is returned.
- If the value is Boolean, "1" is returned for True and "0" for False.
- If the value is numeric, it is converted to a string.
- Otherwise, the value is passed to `str()`.

PARAMETERS:

value (*Any*)

RETURN TYPE:

str

latest

```
abstract get_script_path() → str
```

protected

RETURNS:

A path to the Tcl script to be run by this step.

RETURN TYPE:

str

`get_command() → List[str]`

protected

This command should be overridden by subclasses and replaced with a command incorporating the appropriate tool: e.g. `openroad`, `yosys`, et cetera.

RETURNS:

A list of strings representing the command used to run the script, including the result of `get_script_path()`.

RETURN TYPE:

List[str]

`prepare_env(env: dict, state: State) → dict`

protected

Creates a copy of an environment dictionary, then converts all accessible `self.config` variables and state inputs to environment variables so they may be used as inputs to the scripts.

Inputs are assigned the keys `CURRENT_{ID}` where ID is the relevant `DesignFormat`'s enum name.

Outputs are assigned the keys `CURRENT_{ID}` where ID is the relevant `DesignFormat`'s enum name, although outputs with multiple values (SPEF, etc) will be skipped and a step is expected to handle creating variables for them on its own.

The values are converted to strings as per `value_to_tcl()`.

PARAMETERS:

- **env** (*dict*) – The input environment dictionary
- **state** (*State*) – The input state

RETURNS:

a copy of the environment dictionary where `self.config` variables

RETURN TYPE:

dict

```
run(state_in: State, **kwargs) → Tuple[Dict[DesignFormat, Path |
    List[Path] | Dict[str, Path | List[Path]] | None], Dict[str,
    Any]]
```

protected

This overridden `run()` function prepares configuration variables and inputs for use with Tcl: specifically, it converts them all to environment variables so they may be used by the Tcl scripts being called. See `prepare_env()` for more info.

Additionally, it logs the output to a `.log` file named after the script.

When overriding in a subclass, you may find it useful to use this pattern:

```
kwargs, env = self.extract_env(kwargs)
env["CUSTOM_ENV_VARIABLE"] = "1"
return super().run(state_in, env=env, **kwargs)
```

latest

This will allow you to add further custom environment variables to a call while still respecting an `env` argument further up the call-stack.

PARAMETERS:

- **state_in** (*State*) – See superclass.
- ****kwargs** – Passed on to subprocess execution: useful if you want to redirect stdin, stdout, etc.

RETURNS:

see superclass

RETURN TYPE:

Tuple[*Dict*[*DesignFormat*, *Path* | *List*[*Path*] | *Dict*[str, *Path* | *List*[*Path*]] | None], *Dict*[str, *Any*]]

```
run_subprocess(cmd: Sequence[str | PathLike], log_to: str | PathLike
    | None = None, silent: bool = False, report_dir: str | PathLike
    | None = None, env: Dict[str, str] | None = None, **kwargs) →
    Dict[str, Any]
```

protected

PARAMETERS:

- **cmd** (*Sequence[str | PathLike]*)
- **log_to** (*str | PathLike | None*)
- **silent** (*bool*)
- **report_dir** (*str | PathLike | None*)
- **env** (*Dict[str, str] | None*)

RETURN TYPE:

Dict[str, *Any*]

```
class openlane.steps.YosysStep(config: Config | None = None, state_in:
    State | None | Future[State] = None, *, id: str | None = None, name:
    str | None = None, long_name: str | None = None, flow: Any | None =
    None, _config_quiet: bool = False, _no_revalidate_conf: bool =
    False, **kwargs)
```

latest

Bases: `TclStep`

PARAMETERS:

- **config** (*Optional[Config]*)
- **state_in** (*Future[State]*)
- **id** (*str*)
- **name** (*str*)
- **long_name** (*str*)
- **flow** (*Optional[Any]*)
- **_config_quiet** (*bool*)
- **_no_revalidate_conf** (*bool*)

```
get_command() → List[str]
```

protected

This command should be overridden by subclasses and replaced with a command incorporating the appropriate tool: e.g. `openroad`, `yosys`, et cetera.

RETURNS:

A list of strings representing the command used to run the script, including the result of `get_script_path()`.

RETURN TYPE:

List[str]

```
abstract get_script_path() → str
```

protected

RETURNS:

A path to the Tcl script to be run by this step.

RETURN TYPE:

str

```
run(state_in: State, **kwargs) → Tuple[Dict[DesignFormat, Path |
    List[Path] | Dict[str, Path | List[Path]] | None], Dict[str,
    Any]]
```

latest

protected

This overridden `run()` function prepares configuration variables and inputs for use with Tcl: specifically, it converts them all to environment variables so they may be used by the Tcl scripts being called. See `prepare_env()` for more info.

Additionally, it logs the output to a `.log` file named after the script.

When overriding in a subclass, you may find it useful to use this pattern:

```
kwargs, env = self.extract_env(kwargs)
env["CUSTOM_ENV_VARIABLE"] = "1"
return super().run(state_in, env=env, **kwargs)
```

This will allow you to add further custom environment variables to a call while still respecting an `env` argument further up the call-stack.

PARAMETERS:

- **state_in** (*State*) – See superclass.
- ****kwargs** – Passed on to subprocess execution: useful if you want to redirect stdin, stdout, etc.

RETURNS:

see superclass

RETURN TYPE:

Tuple[*Dict*[*DesignFormat*, *Path* | *List*[*Path*] | *Dict*[str, *Path* | *List*[*Path*]] | None], *Dict*[str, *Any*]]

class openlane.steps.**OpenROADAlert**(cls: Literal['warning', 'error'], code: str | None, message: str)

Bases: `object`

Data structure encapsulating an alert (warning or error) from OpenROAD.

PARAMETERS:

- **cls** (*Literal['warning', 'error']*)
- **code** (*str* | *None*)
- **message** (*str*)

class openlane.steps.**OpenROADOutputProcessor**(*a

Bases: `OutputProcessor`

A special output processor for steps leveraging OpenROAD-based subprocesses.

It captures *[ERROR]* and *[WARNING]* lines into a data structure where they can be further processed by the step itself rather than simply printed to the terminal.

```
process_line(line: str)
```

If a line contains an OpenROAD error/warning, it is processed and handed over to the step's `on_alert` method.

PARAMETERS:

line (*str*) – The line in question

RETURNS:

`True` if the line has alerts, `False` if the line has no alerts

```
result() → List[OpenROADAlert]
```

RETURNS:

A list of OpenROAD alerts captured by this output processor

RETURN TYPE:

List[*OpenROADAlert*]

```
class openlane.steps.SupportsOpenROADAlerts(*args, **kwargs)
```

Bases: `Protocol`

A listener for `OpenROADOutputProcessor`. Fires whenever a line contains an alert.

```
on_alert(alert: OpenROADAlert) → OpenROADAlert
```

PARAMETERS:

alert (*OpenROADAlert*) – The alert found in the processed line

RETURNS:

The alert once again, modified at the step object's leisure

RETURN TYPE:

OpenROADAlert

latest

```
class openlane.steps.OpenROADStep(config: Confi
    State | None | Future[State] = None, *, id: str | None = None, name:
    str | None = None, long_name: str | None = None, flow: Any | None =
```

```
None, _config_quiet: bool = False, _no_revalidate_conf: bool =
False, **kwargs)
```

latest

Bases: `TclStep`

PARAMETERS:

- **config** (*Optional[Config]*)
- **state_in** (*Future[State]*)
- **id** (*str*)
- **name** (*str*)
- **long_name** (*str*)
- **flow** (*Optional[Any]*)
- **_config_quiet** (*bool*)
- **_no_revalidate_conf** (*bool*)

```
abstract get_script_path() → str
```

protected

RETURNS:

A path to the Tcl script to be run by this step.

RETURN TYPE:

str

```
prepare_env(env: dict, state: State) → dict
```

protected

Creates a copy of an environment dictionary, then converts all accessible `self.config` variables and state inputs to environment variables so they may be used as inputs to the scripts.

Inputs are assigned the keys `CURRENT_{ID}` where ID is the relevant `DesignFormat`'s enum name.

Outputs are assigned the keys `CURRENT_{ID}` where ID is the relevant `DesignFormat`'s enum name, although outputs with multiple values (SPEF, etc) will be skipped and a step is expected to handle creating variables for them on its own.

latest

The values are converted to strings as per `value_to_tcl()`.

PARAMETERS:

- **env** (*dict*) – The input environment dictionary
- **state** (*State*) – The input state

RETURNS:

a copy of the environment dictionary where `self.config` variables

RETURN TYPE:

dict

```
run(state_in, **kwargs) → Tuple[Dict[DesignFormat, Path | List[Path]
    | Dict[str, Path | List[Path]] | None], Dict[str, Any]]
```

The *run()* override for the OpenROADStep class handles two things:

1. Before the *super()* call: It creates a version of the lib file minus cells that are known bad (i.e. those that fail DRC) and pass it on in the environment variable *_PNR_LIBS*.

2. After the *super()* call: Processes the *or_metrics_out.json* file and updates the State's *metrics* property with any new metrics in that object.

RETURN TYPE:

Tuple[*Dict*[*DesignFormat*, *Path* | *List*[*Path*] | *Dict*[str, *Path* | *List*[*Path*]] | None], *Dict*[str, *Any*]]

```
get_command() → List[str]
```

protected

This command should be overridden by subclasses and replaced with a command incorporating the appropriate tool: e.g. `openroad`, `yosys`, et cetera.

RETURNS:

A list of strings representing the command used to run the script, including the result of `get_script_path()`.

RETURN TYPE:

List[str]

latest

```
layout_preview() → str | None
```

RETURNS:

An HTML tag that could act as a preview for a specific stage or `None` if a preview is unavailable for this step.

RETURN TYPE:

str | None

```
class openlane.steps.OdbpyStep(config: Config | None = None, state_in:
    State | None | Future[State] = None, *, id: str | None = None, name:
    str | None = None, long_name: str | None = None, flow: Any | None =
    None, _config_quiet: bool = False, _no_revalidate_conf: bool =
    False, **kwargs)
```

latest

Bases: `Step`

PARAMETERS:

- **config** (*Optional[Config]*)
- **state_in** (*Future[State]*)
- **id** (*str*)
- **name** (*str*)
- **long_name** (*str*)
- **flow** (*Optional[Any]*)
- **_config_quiet** (*bool*)
- **_no_revalidate_conf** (*bool*)

```
run(state_in, **kwargs) → Tuple[Dict[DesignFormat, Path | List[Path]
    | Dict[str, Path | List[Path]] | None], Dict[str, Any]]
```

protected

The "core" of a step.

This step is considered per-object private, i.e., if a Step's run is called anywhere outside of the same object's `start()`, its behavior is undefined.

PARAMETERS:

- **state_in** –

  The input state.

  Note that `self.state_in` is stored as a future and would need to be resolved before use first otherwise.

  For reference, `start()` is responsible for resolving it for `.run()`.

- ****kwargs** – Passed on to subprocess execution: useful if you want to redirect stdin, stdout, etc.

RETURN TYPE:

Tuple[*Dict*[*DesignFormat*, *Path* | *List*[*Path*] | *Dict*[str, *Path* | *List*[*Path*]] | None], *Dict*[str, *Any*]]

```
class openlane.steps.MagicStep(config: Config |
    State | None | Future[State] = None, *, id:
    str | None = None, long_name: str | None = None, flow: Any | None =
```

latest

```
None, _config_quiet: bool = False, _no_revalidate_conf: bool =
False, **kwargs)
```

Bases: `TclStep`

PARAMETERS:

- **config** (*Optional[Config]*)
- **state_in** (*Future[State]*)
- **id** (*str*)
- **name** (*str*)
- **long_name** (*str*)
- **flow** (*Optional[Any]*)
- **_config_quiet** (*bool*)
- **_no_revalidate_conf** (*bool*)

`abstract get_script_path() → str`

protected

RETURNS:

A path to the Tcl script to be run by this step.

RETURN TYPE:

str

`get_command() → List[str]`

protected

This command should be overridden by subclasses and replaced with a command incorporating the appropriate tool: e.g. `openroad`, `yosys`, et cetera.

RETURNS:

A list of strings representing the command used to run the script, including the result of `get_script_path()`.

RETURN TYPE:

List[str]

`prepare_env(env: dict, state: State) → dict`

protected

Creates a copy of an environment dictionar latest
accessible `self.config` variables and state inputs to environment variables so they may be used as inputs to the scripts.

Inputs are assigned the keys `CURRENT_{ID}` where ID is the relevant `DesignFormat`'s enum name.

Outputs are assigned the keys `CURRENT_{ID}` where ID is the relevant `DesignFormat`'s enum name, although outputs with multiple values (SPEF, etc) will be skipped and a step is expected to handle creating variables for them on its own.

The values are converted to strings as per `value_to_tcl()`.

PARAMETERS:

- **env** (*dict*) – The input environment dictionary
- **state** (*State*) – The input state

RETURNS:

a copy of the environment dictionary where `self.config` variables

RETURN TYPE:

dict

```
run(state_in: State, **kwargs) → Tuple[Dict[DesignFormat, Path |
    List[Path] | Dict[str, Path | List[Path]] | None], Dict[str,
    Any]]
```

protected

This overridden `run()` function prepares configuration variables and inputs for use with Tcl: specifically, it converts them all to environment variables so they may be used by the Tcl scripts being called. See `prepare_env()` for more info.

Additionally, it logs the output to a `.log` file named after the script.

When overriding in a subclass, you may find it useful to use this pattern:

```
kwargs, env = self.extract_env(kwargs)
env["CUSTOM_ENV_VARIABLE"] = "1"
return super().run(state_in, env=env, **kwargs)
```

latest

This will allow you to add further custom environment variables to a call while still respecting an `env` argument further up the call-stack.

PARAMETERS:

- **state_in** (*State*) – See superclass.
- ****kwargs** – Passed on to subprocess execution: useful if you want to redirect stdin, stdout, etc.

RETURNS:

see superclass

RETURN TYPE:

Tuple[*Dict*[*DesignFormat*, *Path* | *List*[*Path*] | *Dict*[str, *Path* | *List*[*Path*]] | None], *Dict*[str, *Any*]]

```
class openlane.steps.NetgenStep(config: Config | None = None, state_in:
    State | None | Future[State] = None, *, id: str | None = None, name:
    str | None = None, long_name: str | None = None, flow: Any | None =
    None, _config_quiet: bool = False, _no_revalidate_conf: bool =
    False, **kwargs)
```

latest

Bases: `TclStep`

PARAMETERS:

- **config** (*Optional[Config]*)
- **state_in** (*Future[State]*)
- **id** (*str*)
- **name** (*str*)
- **long_name** (*str*)
- **flow** (*Optional[Any]*)
- **_config_quiet** (*bool*)
- **_no_revalidate_conf** (*bool*)

`abstract get_script_path() → str`

protected

RETURNS:

A path to the Tcl script to be run by this step.

RETURN TYPE:

str

`get_command() → List[str]`

protected

This command should be overridden by subclasses and replaced with a command incorporating the appropriate tool: e.g. `openroad`, `yosys`, et cetera.

RETURNS:

A list of strings representing the command used to run the script, including the result of `get_script_path()`.

RETURN TYPE:

List[str]