`openlane.flows` API

The Flow Module

An API for implementing new flows using the OpenLane infrastructure, as well as a number of built-in flows.

exception openlane.flows.**FlowError**

Bases: `RuntimeError`

A `RuntimeError` that occurs when a Flow, or one of its underlying Steps, fails to finish execution properly.

exception openlane.flows.**FlowException**

Bases: `FlowError`

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

- A `StepException` raised by an underlying Step
- Invalid inputs
- Mis-use of class interfaces of the `Flow`
- Other unexpected failures

class openlane.flows.**FlowProgressBar**(flow_name: str, starting_ordinal: int = 1)

Bases: `object`

A wrapper for a flow's progress bar, rendered using Rich at the bottom of interactive terminals.

PARAMETERS:

- **flow_name** (*str*)
- **starting_ordinal** (*int*)

`start()`

Starts rendering the progress bar.

`end()`

Stops rendering the progress bar.

`property started: bool`

RETURNS:

If the progress bar has started or not

`set_max_stage_count(count: int)`

A helper function, used to set the total number of stages the progress bar is expected to keep tally of.

PARAMETERS:

count (*int*) – The total number of stages.

`start_stage(name: str)`

Starts a new stage, updating the progress bar appropriately.

PARAMETERS:

name (*str*) – The name of the stage.

`end_stage(*, increment_ordinal: bool = True)`

Ends the current stage, updating the progress bar appropriately.

PARAMETERS:

increment_ordinal (*bool*) –

Increment the step ordinal, which is used in the creation of step directories.

You may want to set this to `False` if the stage is being skipped.

Please note that step ordinal is not equal to stages- a skipped step increments the stage but not the step ordinal.

`get_ordinal_prefix() → str`

RETURNS:

A string with the current step ordinal, which can be used to create a step directory.

RETURN TYPE:

str

```
class openlane.flows.Flow(config: str | PathLike | Mapping[str, Any] |
    Sequence[str | PathLike | Mapping[str, Any]], *, name: str | None =
    None, pdk: str | None = None, pdk_root: str | None = None, scl: str
    | None = None, design_dir: str | None = None,
    config_override_strings: Sequence[str] | None = None)
```

Bases: `ABC`

An abstract base class for a flow.

Flows encapsulate a the running of multiple `Step`s in any order. The sequence (or lack thereof) of running the steps is left to the Flow itself.

The Flow ABC offers a number of convenience functions, including handling the progress bar at the bottom of the terminal, which shows what stage the

flow is currently in and the remaining stages.

PARAMETERS:

- **config** (*AnyConfigs*) – Either a resolved `openlane.config.Config` object, or an input to `openlane.config.Config.load()`.
- **name** (*str*) –

  An optional string name for the Flow itself, and not a run of it.

  If not provided, there are two fallbacks:

  - The value of the `name` property (`NotImplemented` by default)
  - The name of the concrete `Flow` class
- **config_override_strings** (*Optional[Sequence[str]]*) – See `openlane.config.Config.load()`
- **pdk** (*Optional[str]*) – See `openlane.config.Config.load()`
- **pdk_root** (*Optional[str]*) – See `openlane.config.Config.load()`
- **scl** (*Optional[str]*) – See `openlane.config.Config.load()`
- **design_dir** (*Optional[str]*) – See `openlane.config.Config.load()`

VARIABLES:

- **Steps** –

  A list of `Step` **types** used by the Flow (not Step objects.)

  Subclasses of `Flow` are expected to override the default value as a class member- but subclasses may allow this value to be further overridden during construction (and only then.)

  `Flow` subclasses without the `Steps` class property declared are considered abstract and cannot be initialized.

- **config_vars** – A list of **flow-specific** configuration variables. These configuration variables are used entirely within the logic of the flow itself and are not exposed to `Step`(s).
- **step_objects** –

  A list of `Step` **objects** from the last run of the flow, if it exists.

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

- **run_dir** –

  The directory of the last run of the flow, if it exists.

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

- **toolbox** –

  The `Toolbox` of the last run of the flow, if it exists.

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

- **config_resolved_path** –

  The path to the serialization of the resolved configuration for the last run of the flow.

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

```
classmethod get_help_md() → str
```

RETURNS:

rendered Markdown help for this Flow

RETURN TYPE:

str

```
get_all_config_variables() → List[Variable]
```

RETURNS:

All configuration variables for this Flow, including universal configuration variables, flow-specific configuration variables and step-specific configuration variables.

RETURN TYPE:

List[*Variable*]

```
classmethod init_with_config(config_in: Config | str | PathLike |
    Dict, **kwargs)
```

Deprecated since version 2.0.0a29: Use the constructor for the class instead

PARAMETERS:

config_in (*Config* | *str* | *PathLike* | *Dict*)

```
start(with_initial_state: State | None = None, tag: str | None =
    None, last_run: bool = False, _force_run_dir: str | None = None,
    _no_load_previous_steps: bool = False, *, overwrite: bool =
    False, **kwargs) → State
```

The entry point for a flow.

PARAMETERS:

- **with_initial_state** (*State* | *None*) – An optional initial state object to use. If not provided: * If resuming a previous run, the latest `state_out.json` (by filesystem modification date) * If not, an empty state object is created.
- **tag** (*str* | *None*) –

  A name for this invocation of the flow. If not provided, one based on a date string will be created.

  This tag is used to create the "run directory", which will be placed under the directory `runs/` in the design directory.

- **last_run** (*bool*) –

  Use the latest run (by modification time) as the tag.

  If no runs exist, a `FlowException` will be raised.

  If `last_run` and `tag` are both set, a `FlowException` will also be raised.

- **overwrite** (*bool*) – If true and a run with the desired tag was found, the contents will be deleted instead of appended.
- **_force_run_dir** (*str* | *None*)
- **_no_load_previous_steps** (*bool*)

RETURNS:

`(success, state_list)`

RETURN TYPE:

State

```
abstract run(initial_state: State, **kwargs) → Tuple[State,
    List[Step]]
```

protected

The core of the Flow. Subclasses of flow are expected to override this method.

PARAMETERS:

initial_state (*State*) – An initial state object to use.

RETURNS:

A tuple of states and instantiated step objects for inspection.

RETURN TYPE:

Tuple[*State*, *List*[*Step*]]

```
dir_for_step(step: Step) → str
```

protected

May only be called while `run_dir` is not None, i.e., the flow has started. Otherwise, a `FlowException` is raised.

RETURNS:

A directory within the run directory for a specific step, prefixed with the current progress bar stage number.

PARAMETERS:

step (*Step*)

RETURN TYPE:

str

```
start_step(step: Step, *args, **kwargs) → State
```

protected

A helper function that handles passing parameters to `Step.start`.'

It is essentially equivalent to:

```
step.start(
    toolbox=self.toolbox,
    step_dir=self.dir_for_step(step),
)
```

See `Step.start()` for more info.

PARAMETERS:

- **step** (*Step*) – The step object to run
- **args** – Arguments to *step.start*
- **kwargs** – Keyword arguments to *step.start*

RETURN TYPE:

State

`start_step_async(step: Step, *args, **kwargs) → Future[State]`

protected

An asynchronous equivalent to `start_step()`.

PARAMETERS:

- **step** (*Step*) – The step object to run
- **args** – Arguments to *step.start*
- **kwargs** – Keyword arguments to *step.start*

RETURNS:

A `Future` encapsulating a State object, which can be used as an input to the next step (where the next step will wait for the `Future` to be realized before calling `Step.run()`)

RETURN TYPE:

Future[*State*]

`set_max_stage_count(count: int)`

protected

Alias for `self.progress_bar`'s `FlowProgressBar.set_max_stage_count()`.

Deprecated since version 2.0.0a46: Use .progress_bar.set_max_stage_count

PARAMETERS:

count (*int*)

`start_stage(name: str)`

protected

Alias for `self.progress_bar`'s `FlowProgressBar.start_stage()`.

Deprecated since version 2.0.0a46: Use .progress_bar.start_stage

PARAMETERS:

name (*str*)

```
end_stage(increment_ordinal: bool = True)
```

protected

Alias for `self.progress_bar`'s `FlowProgressBar.end_stage()`.

Deprecated since version 2.0.0a46: Use .progress_bar.end_stage

PARAMETERS:

increment_ordinal (*bool*)

```
class FlowFactory
```

Bases: `object`

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

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

classmethod **register**(registered_name: str | None = None) → Callable[[Type[Flow]], Type[Flow]]

A decorator that adds a flow type to the registry.

PARAMETERS:

registered_name (*str* | *None*) –

An optional registered name for the flow.

If not specified, the flow will be referred to by its Python class name.

RETURN TYPE:

Callable[[*Type*[*Flow*]], *Type*[*Flow*]]

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

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

PARAMETERS:

name (*str*) – The registered name of the Flow. Case-sensitive.

RETURN TYPE:

Type[*Flow*] | None

classmethod **list**() → List[str]

RETURNS:

A list of strings representing all registered flows.

RETURN TYPE:

List[str]

factory

alias of `FlowFactory`

class openlane.flows.**SequentialFlow**(*args, Substitute: Dict[str, str |

```
Type[Step] | None] | None = None, **kwargs)
```

Bases: `Flow`

The simplest Flow, running each Step as a stage, serially, with nothing happening in parallel and no significant inter-step processing.

All subclasses of this flow have to do is override the `Steps` abstract property and it would automatically handle the rest. See *Classic* in Built-in Flows for an example.

It should be noted, for Steps with duplicate IDs, all Steps other than the first one will technically be subclassed with no change other than to simply set the ID to the previous step's ID with a suffix: i.e. the second instance of `Test.MyStep` will have an ID of `Test.MyStep1`, and so on.

PARAMETERS:

- **Substitute** (*Optional[Dict[str, Union[str, Type[Step], None]]]*) –

  Substitute all instances of one *Step* type by another *Step* type in the `Steps` attribute for this instance only.

  You may also use the string Step IDs in place of a *Step* type object.

  Duplicate ID normalization is re-run after substitutions.

- **args** – Arguments for `Flow`.
- **kwargs** – Keyword arguments for `Flow`.

VARIABLES:

gating_config_vars – A mapping from step ID (wildcards) to lists of Boolean variable names. All Boolean variables must be True for a step with a specific ID to execute.

```
run(initial_state: State, frm: str | None = None, to: str | None =
    None, skip: Iterable[str] | None = None, reproducible: str |
    None = None, **kwargs) → Tuple[State, List[Step]]
```

protected

The core of the Flow. Subclasses of flow are expected to override this method.

PARAMETERS:

- **initial_state** (*State*) – An initial state object to use.
- **frm** (*str* | *None*)
- **to** (*str* | *None*)
- **skip** (*Iterable[str]* | *None*)
- **reproducible** (*str* | *None*)

RETURNS:

A tuple of states and instantiated step objects for inspection.

RETURN TYPE:

Tuple[*State*, *List*[*Step*]]

```
openlane.flows.cloup_flow_opts(*, config_options: bool = True,
    run_options: bool = True, sequential_flow_controls: bool = True,
    sequential_flow_reproducible: bool = False, pdk_options: bool =
    True, log_level: bool = True, jobs: bool = True,
    accept_config_files: bool = True, volare_by_default: bool = True,
    volare_pdk_override: str | None = None, _enable_debug_flags: bool =
    False) → Callable[[Callable[[...], Any]], Callable[[...], Any]]
```

Creates a wrapper that appends a number of OpenLane flow-related flags to a function decorated with @cloup.command (https://cloup.readthedocs.io/en/stable/autoapi/cloup/index.html#cloup.command).

The following keyword arguments will be passed to the decorated function. * Those postfixed ‡ are compatible with the constructor for `Flow`. * Those postfixed § are compatible with the `Flow.start()`.

- **Flow configuration options (if parameter `config_options` is `True`):**
  - `flow_name`: `Optional[str]`: A valid flow ID to be used with `Flow.factory.get()`
  - `config_override_strings`‡: `Optional[Iterable[str]]`
- **Sequential flow controls (if parameter `sequential_flow_controls` is `True`)**
  - `frm``§: ``Optional[str]`: Start from a step with this ID. Supported by sequential flows.
  - `to``§: ``Optional[str]`: Stop at a step with this id. Supported by sequential flows.
  - `skip``§: ``Iterable[str]`: Skip these steps. Supported by sequential flows.
- **Sequential flow reproducible (if parameter `sequential_flow_reproducible` is `True`)**
  - `reproducible``§: ``str`: Create a reproducible for a step with is ID, aborting the flow afterwards. Supported by sequential flows.
- **Flow run options (if parameter `run_options` is `True`):**
  - `tag``§: ``Optional[str]`
  - `last_run``§: ``bool`: If `True`, `tag` is guaranteed to be None.
  - `with_initial_state``§: ``Optional[State]`
- **PDK options**
  - `use_volare`: `bool`
  - `pdk_root`‡: `Optional[str]`
  - `pdk`‡: `str`
  - `scl`‡: `Optional[str]`
- `config_files`: `Iterable[str]`: Paths to configuration files (if parameter `accept_config_files` is `True`)

PARAMETERS:

- **config_options** (*bool*) – Enables flow configuration and starting CLI flags

- **sequential_flow_controls** (*bool*) – Enables flow control CLI flags
- **flow_run_options** – Enables tag CLI flags
- **pdk_options** (*bool*) – Enables PDK CLI flags
- **log_level** (*bool*) – Enables `--log-level` CLI flag
- **jobs** (*bool*) – Enables `-j/--jobs` CLI flag
- **accept_config_files** (*bool*) – Accepts configuration file paths as CLI arguments
- **volare_by_default** (*bool*) – If `pdk_options` is `True`, this changes whether Volare is used by default for this CLI or not.
- **run_options** (*bool*)
- **sequential_flow_reproducible** (*bool*)
- **volare_pdk_override** (*str* | *None*)
- **_enable_debug_flags** (*bool*)

RETURNS:

The wrapper

RETURN TYPE:

Callable[[*Callable*[[...], *Any*]], *Callable*[[...], *Any*]]