Writing and Using Plugins

OpenLane plugin modules are written in Python and can be added to `PYTHONPATH` or installed to your Python `site-packages` (e.g. installed either inside or outside a venv). OpenLane detects and imports all Python modules found that have the prefix `openlane_plugin_`.

Plugins are useful to add support for more utilities other than those included with OpenLane; either alternative open-source EDA utilities that are not part of the built-in flows, or proprietary utilities that can never be a part of the built-in parts.

> **Note**
>
> Plugins are not supported in OpenLane Docker containers.

Registering new Flows and Steps

Plugins, much like OpenLane itself, may create and register steps and flows into global registries, where the steps and flows can then be accessed programmatically via their ID.

You can use the Python decorators `@Flow.factory.register()` and `@Step.factory.register()` to register flows and steps respectively.

Registered flows and steps can be accessed in configuration files as shown in Writing Custom Flows, but they can also be accessed programmatically as follows:

```
MyCustomStep = Step.factory.get("ToolName.MyCustomStep")
MyCustomFlow = Flow.factory.get("MyCustomFlow")
```

For information on writing the flows and steps themselves, see Writing Custom Flows and Writing Custom Steps.

Including Tools

Naturally, one of the most important uses of plugins is the ability to write steps for other tools. There are multiple ways to include these tools:

Bring-your-own-tools

You may require users to bring their own tools by installing them on their own and adding them to `PATH`, then running OpenLane with the plugin separately.

This is usually the only option for proprietary utilities.

You can still install the plugin itself with Nix while requiring the tool be separate- see the section below.

With Nix

> **Note**
>
> This section requires a strong understanding of the Nix programming language, including the Flakes feature, to follow.

You may bundle either the plugin alone or the plugin and the tool using Nix.

The flake for OpenLane contains a function named `createOpenLaneShell` in its outputs. This creates a devshell, which is an executable script that drops you into an environment with OpenLane, its dependencies, and optionally plugins, installed.

The plugins are to be regular Python Nix derivations, built using the `buildPythonPackage` function with one addition: Plugins must have the attribute `includedTools`, declaring tool dependencies (so they may be available to the user standalone in the environment).

For example, if a plugin, for some reason, depends on the Python `numpy` library and incorporates `bash`, it can declare its dependencies as follows:

```
propagatedBuildInputs = includedTools ++ [
  openlane
  …
];
```

Then to create a devshell with both OpenLane and the plugin available, use this Nix expression:

```
devShells = openlane2.inputs.nix-eda.forAllSystems { withInputs = [openlane2 s
  default = callPackage (openlane2.createOpenLaneShell {
    extra-python-packages = [
      pkgs.openlane_plugin_example
    ];
  }) {};
});
```

For a full example of this, see `openlane_plugin_example`.