A basic module

Contents

What is a module?

- A module is a function that takes an attribute set and returns an attribute set.
- It may declare options, telling which attributes are allowed in the final outcome.
- It may define values, for options declared by itself or other modules.
- When evaluated by the module system, it produces an attribute set based on the declarations and definitions.

The simplest possible module is a function that takes any attributes and returns an empty attribute set:

options.nix

```
{ ... }:
{
}
```

To define any values, the module system first has to know which ones are allowed. This is done by declaring *options* that specify which attributes can be set and used elsewhere.

1.1. Declaring options

Options are declared under the top-level `options` attribute with `lib.mkOption`.

Skip to main content

```nix
{ lib, ... }:
{
  options = {
    name = lib.mkOption { type = lib.types.str; };
  };
}
```

Note

The `lib` argument is passed automatically by the module system. This makes Nixpkgs library functions available in each module's function body.

The ellipsis `...` is necessary because the module system can pass arbitrary arguments to modules.

The attribute `type` in the argument to `lib.mkOption` specifies which values are valid for an option. There are several types available under `lib.types`.

Here we have declared an option `name` of type `str`: The module system will expect a string when a value is defined.

Now that we have declared an option, we would naturally want to give it a value.

1.2. Defining values

Options are set or *defined* under the top-level `config` attribute:

config.nix

```nix
{ ... }:
{
  config = {
    name = "Boaty McBoatface";
  };
}
```

In our option declaration, we created an option `name` with a string type. Here, in our option definition, we have set that same option to a string.

Skip to main content

Option declarations and option definitions don't need to be in the same file. Which modules will contribute to the resulting attribute set is specified when setting up module system evaluation.

1.3. Evaluating modules

Modules are evaluated by `lib.evalModules` from the Nixpkgs library. It takes an attribute set as an argument, where the `modules` attribute is a list of modules to merge and evaluate.

The output of `evalModules` contains information about all evaluated modules, and the final values appear in the attribute `config`.

default.nix

```nix
let
  pkgs = import <nixpkgs> {};
  result = pkgs.lib.evalModules {
    modules = [
      ./options.nix
      ./config.nix
    ];
  };
in
result.config
```

Here's a helper script to parse and evaluate our `default.nix` file with `nix-instantiate --eval` and print the output as JSON:

eval.bash

```
nix-shell -p jq --run "nix-instantiate --eval --json --strict | jq"
```

As long as every definition has a corresponding declaration, evaluation will be successful. If there is an option definition that has not been declared, or the defined value has the wrong type, the module system will throw an error.

Running the script (`./eval.bash`) should show an output that matches what we have configured:

Skip to main content

```
{
  "name": "Boaty McBoatface"
}
```