Search the doc

Generators are the building blocks of the Yeoman ecosystem. They're the plugins run by `yo` to generate files for end users.

In reading this section, you'll learn how to create and distribute your own.

> *Note: We built a generator-generator (https://github.com/yeoman/generator-generator) to help users get started with their own generator. Feel free to use it to bootstrap your own generator once you understand the below concepts.*

Organizing your generators

Setting up as a node module

A generator is, at its core, a Node.js module.

First, create a folder within which you'll write your generator. This folder must be named `generator-name` (where `name` is the name of your generator). This is important, as Yeoman relies on the file system to find available generators.

Once inside your generator folder, create a `package.json` file. This file is a Node.js module manifest. You can generate this file by running `npm init` from your command line or by entering the following manually:

```
{
  "name": "generator-name",
  "version": "0.1.0",
  "description": "",
  "files": [
    "generators"
  ],
  "keywords": ["yeoman-generator"],
  "dependencies": {
    "yeoman-generator": "^1.0.0"
  }
}
```

The `name` property must be prefixed by `generator-`. The `keywords` property must contain `"yeoman-generator"` and the repo must have a description to be indexed by our generators page (/generators/).

You should make sure you set the latest version of `yeoman-generator` as a dependency. You can do this by running: `npm install --save yeoman-generator`.

The `files` property must be an array of files and directories that is used by your generator.

Add other `package.json` properties (https://docs.npmjs.com/files/package.json) as needed.

Folder tree

Yeoman's functionality is dependent on how you structure your directory tree. Each sub-generator is contained within its own folder.

The default generator used when you call `yo name` is the `app` generator. This must be contained within the `app/` directory.

Sub-generators, used when you call `yo name:subcommand`, are stored in folders named exactly like the sub command.

In an example project, a directory tree could look like this:

```
├──package.json
└──generators/
    ├──app/
    |   └──index.js
    └──router/
        └──index.js
```

This generator will expose `yo name` and `yo name:router` commands.

Yeoman allows two different directory structures. It'll look in `./` and in `generators/` to register available generators.

The previous example can also be written as follows:

```
├──package.json
├──app/
|   └──index.js
└──router/
    └──index.js
```

If you use this second directory structure, make sure you point the `files` property in your `package.json` at all the generator folders.

```
{
  "files": [
    "app",
    "router"
  ]
}
```

Extending generator

Once you have this structure in place, it's time to write the actual generator.

Yeoman offers a base generator which you can extend to implement your own behavior. This base generator will add most of the functionalities you'd expect to ease your task.

In the generator's index.js file, here's how you extend the base generator:

```
var Generator = require('yeoman-generator');

module.exports = class extends Generator {};
```

We assign the extended generator to `module.exports` to make it available to the ecosystem. This is how we export modules in Node.js (https://nodejs.org/api/modules.html#modules_module_exports).

Overwriting the constructor

Some generator methods can only be called inside the `constructor` function. These special methods may do things like set up important state controls and may not function outside of the constructor.

To override the generator constructor, add a constructor method like so:

```
module.exports = class extends Generator {
  // The name `constructor` is important here
  constructor(args, opts) {
    // Calling the super constructor is important so our generator is correct
    super(args, opts);

    // Next, add your custom code
    this.option('babel'); // This method adds support for a `--babel` flag
  }
};
```

Adding your own functionality

Every method added to the prototype is run once the generator is called–and usually in sequence. But, as we'll see in the next section, some special method names will trigger a specific run order.

Let's add some methods:

```
module.exports = class extends Generator {
  method1() {
    this.log('method 1 just ran');
  }

  method2() {
    this.log('method 2 just ran');
  }
};
```

When we run the generator later, you'll see these lines logged to the console.

Running the generator

At this point, you have a working generator. The next logical step would be to run it and see if it works.

Since you're developing the generator locally, it's not yet available as a global npm module. A global module may be created and symlinked to a local one, using npm. Here's what you'll want to do:

On the command line, from the root of your generator project (in the `generator-name/` folder), type:

```
npm link
```

That will install your project dependencies and symlink a global module to your local file. After npm is done, you'll be able to call `yo name` and you should see the `this.log`, defined earlier, rendered in the terminal. Congratulations, you just built your first generator!

Finding the project root

While running a generator, Yeoman will try to figure some things out based on the context of the folder it's running from.

Most importantly, Yeoman searches the directory tree for a `.yo-rc.json` file. If found, it considers the location of the file as the root of the project. Behind the scenes, Yeoman will change the current directory to the `.yo-rc.json` file location and run the requested generator there.

The Storage module creates the `.yo-rc.json` file. Calling `this.config.save()` from a generator for the first time will create the file.

So, if your generator is not running in your current working directory, make sure you don't have a `.yo-rc.json` somewhere up the directory tree.

Where to go from here?

After reading this, you should be able to create a local generator and run it.

If this is your first time writing a generator, you should definitely read the next section on running context and the run loop (/authoring/running-context.html). This section is vital to understanding the context in which your generator will run, and to ensure that it will compose well with other generators in the Yeoman ecosystem. The other sections of the documentation will present functionality available within the Yeoman core to help you achieve your goals.

Show your love for **Yeoman**, wear our **merch**! (https://yeoman.threadless.com/)

(https://twitter.com/yeoman) (https://github.com/yeoman/yeoman)

(/blog/atom.xml) API (https://yeoman.github.io/generator/)

Improve this page (https://github.com/yeoman/yeoman.github.io/blob/source/app/authoring/index.md)