INTEGRATING YEOMAN

Every time you run a generator, you're actually using the `yeoman-environment`. The environment is a base system that is decoupled from any UI component and can be abstracted away by any tool. When you run `yo`, you're basically just running a terminal UI façade on top of the core Yeoman environment.

The basics

The first thing you need to know is the environment system is contained in the `yeoman-environment` package. You can install it by running:

```
npm install --save yeoman-environment
```

This module provides methods to retrieve installed generators, register and run generators. It also provides the user interfaces adapter that generators are using. We provide a full API documentation (https://yeoman.github.io/environment/) (which is the terse list of methods available.)

Using `yeoman-environment`

A simple usage example

Let's start with a simple usage example of `yeoman-environment` before we move to deeper topics.

In this example, let's assume `npm` wants to provide a `npm init` command to scaffold a `package.json`. Reading the other pages of the documentation, you already know how to create a generator - so let's assume we already have a `generator-npm`. We'll see how to invoke it.

First step is to instantiate a new environment instance.

```
var yeoman = require('yeoman-environment');
var env = yeoman.createEnv();
```

Then, we'll want to register our `generator-npm` so it can be used later. You have two options here:

```
// Here we register a generator based on its path. Providing the namespace
// is optional.
env.register(require.resolve('generator-npm'), 'npm:app');

// Or you can provide a generator constructor. Doing so, you need to provide
// a namespace manually
var GeneratorNPM = generators.Base.extend(/* put your methods in here */);
env.registerStub(GeneratorNPM, 'npm:app');
```

Note that you can register as many generators as you want. Registered generators are just made available throughout the environment (to allow composability for example).

At this point, your environment is ready to run `npm:app`.

```
// In its simplest form
env.run('npm:app', done);

// Or passing arguments and options
env.run('npm:app some-name', { 'skip-install': true }, done);
```

There you go. You just need to put this code in a `bin` runnable file and you can run a Yeoman generator without using `yo`.

Find installed generators

But what if you wish to provide access to every Yeoman generator installed on a user machine? Then you need to execute a lookup of the user disk.

```
env.lookup(function () {
  env.run('angular');
});
```

`Environment#lookup()` takes a callback that'll be called once Yeoman is done searching for installed generators. Every found generator is going to be registered on the environment.

In case of namespace conflicts, local generators will override global ones.

Get data about registered generator

Calling `Environment#getGeneratorsMeta()` will return an object describing the meta data the lookup task registered.

Each object keys is a generator namespace, and the value object contains these keys:

- `resolved` : the resolved path to a generator
- `namespace` : the namespace of the generator

For example:

```
{
  "webapp:app": {
    "resolved": "/usr/lib/node_modules/generator-webapp/app/index.js",
    "namespace": "webapp:app"
  }
}
```

Note: Generators registered using `#registerStub()` will have `"unknown"` as `resolved` value.

Providing a custom User Interface (UI)

Yeoman uses *adapters* as an abstraction layer to allow IDE, code editor and the like to easily provide user interfaces necessary to run a generator.

An adapter is the object responsible for handling all the interaction with the user. If you want to provide a different interaction model from the classical command line, you have to write your own adapter. Every method to interact with a user is passing through this adapter (mainly: prompting, logging and diffing).

By default, Yeoman provides a Terminal Adapter (https://github.com/yeoman/environment/blob/master/lib/adapter.js). And our test helpers provide a Test Adapter (https://github.com/yeoman/yeoman-test/blob/master/lib/adapter.js) who's mocking prompts and silencing the output. You can use these as reference for your own implementation.

To install an adapter, use the third parameter of `yeoman.createEnv(args, opts, adapter)`.

An adapter should provide at least three methods.

`Adapter#prompt()`

It provides the question-answer functionality (for instance, when you start `yo`, a set of possible actions is prompted to the user). Its signature and behavior follows these of Inquirer.js (https://github.com/SBoudrias/Inquirer.js). When a generators call `this.prompt`, the call is in the end handled by the adapter.

`Adapter#diff()`

Called internally when a conflict is encountered and the user ask for a diff between the old and the new file (both files content is passed as arguments).

`Adapter#log()`

It's both a function and an object intended for generic output. See `lib/util/log.js` (https://github.com/yeoman/environment/blob/master/lib/util/log.js) for the complete list of methods to provide.

Example implementations

Here's a list of modules/plugins/app using `yeoman-environment`. You can use them as inspiration.

- yo (https://github.com/yeoman/yo)
- yeoman-app (https://github.com/yeoman/yeoman-app)