Mastering Mockup Documentation

Release 1.0.0.dev0

Johannes Raggam, Franco Pellegrini

February 02, 2017

Contents

Training materials for the Mockup training.

CHAPTER 1

About Mastering Mockup

Note: This training materials are in a work in progress state. You can contribute and help getting it better!

This training was created by Johannes Raggam and Franco Pellegrini to teach about Mockup, the new Frontend library for Plone 5 at the Plone Conference 2014 in Bristol.

The basic idea about this training (and even the boilerplate for this materials) was stolen from Philip Bauer's and Patrick Gerken's Mastering Plone materials. That is:

> The aim is that anyone with the appropriate knowledge can give a training based on it and contribute to it.
> It is published as Open Source on Github and readthedocs.

This training materials are available on Github and Readthedocs.

1.1 Trainers

The following trainers have given trainings based on Mastering Mockup:

Johannes Raggam Johannes Raggam from Graz/Austria works most of the time with a technology stack based around Python, Plone, Pyramid and Javascript. As an active Open Source / Free Software developer he believes in the power of collaborative work. He is BlueDynamics Alliance Partner and Plone Core Contributor since 2009 and member of the Plone Framework Team since 2012.

Franco Pellegrini Franco Pellegrini is a software developer from Cordoba, Argentina. He started developing Plone in 2005 in a small software company, and as an independent contractor since 2011. He believes in free software philosophy, and so, he has been a Plone core developer since 2010 and Framework Team member since 2012.

1.2 Using the documentation for a training

Feel free to organize a training yourself. Please be so kind to contribute any bugfixes or enhancements you made to the documentation for your training.

The training is rendered using sphinx and builds in two flavors:

default The verbose version used for the online-documentation and for the trainer. Build it in sphinx with `make html` or use the online-version.

presentation A abbreviated version used for the projector during a training. It should uses more bullet-points than verbose text. Build it in sphinx with `make presentation`.

Note: By prefixing an indented block of text or code with `.. only:: presentation` you can control that this block is used for the presentation-version only.

To hide a block from the presentation-version use `.. only:: not presentation`

Content without a prefix will be included in both versions.

1.2.1 The readthedocs-theme

We slightly tweaked readthedocs-theme in `_static/custom.css` so that it works better with projectors:

- We start hiding the navbar much earlier so that it does not interfere with the text.
- We enlarge the default width of the content-area.

1.2.2 Exercises

Some additional javascript shows hidden solutions for exercises by clicking.

Just prepend the solution with this markup:

```
..  admonition:: Solution
    :class: toggle
```

Here is a full example:

```
Exercise 1
^^^^^^^^^^

Your mission, should you choose to accept it...

..  admonition:: Solution
    :class: toggle

    To save the world with only seconds to spare do the following:

    .. code-block:: python

        from plone import api
```

It will be rendered like this:

Exercise 1

Your mission, should you choose to accept it...

Solution

To save the world with only seconds to spare do the following:

```
from plone import api
```

1.2.3 Building the documentation locally

To build the documentation follow these steps:

```
$ git clone https://github.com/plone/mockup-training.git
$ cd mockup-training
$ virtualenv-2.7 .
$ source bin/activate
$ pip install -r requirements.txt
$ make html
```

You can now open the output from `_build/html/index.html`. To build the presentation-version use `make presentation` instead of `make html`. You can open the presentation at `presentation/index.html`.

1.3 Contributing

Everyone is **very welcome** to contribute. Minor bugfixes can be pushed directly in the repository, bigger changes should made as pull-requests and discussed previously in tickets.

1.4 License

The Mastering Mockup Training is licensed under a Creative Commons Attribution 4.0 International License.

Make sure you have filled out a Contributor Agreement.

If you haven't filled in a Contributor Agreement, you can still contribute. Contact the Documentation team, for instance via the mailinglist or directly send a mail to plone-docs@lists.sourceforge.net Basically, all we need is your written confirmation that you are agreeing your contribution can be under Creative Commons. You can also add in a comment with your pull request "I, <full name>, agree to have this published under Creative Commons 4.0 International BY".

CHAPTER 2

Agenda of the “Mastering Mockup” training

Breaks every two hours, lunch at noon.

Let’s see how far we come on day 1.

2.1 Day 1 (27-10-2014)

- Introducing each other (name, company, training motivation).
- About the training.
- Introduction to Mockup.
- Vagrant installation.
  - Good moment for taking a break *
- Installation and Bootstrapping.
- Overview of Mockup and Mockup Core.
- Creating your own Mockup project.
- Creating your first pattern.
- Including a pattern from Mockup.

2.2 Day 2 (28-10-2014)

- Developing patterns all the time, Improving Mockup with new patterns.

2.3 Tasklist for Day 2

- Create patterns for remaining scripts in plone_ecmascript.
  - [ ] cookie_functions CMPlone#282
  - [ ] inline_validation CMPlone#282
  - [giacamo] jquery.highlightsearchterms CMPlone#282
  - [ ] js-standards CMPlone#282

  - [ ] kss-bbb CMPlone#282
  - [alessandro] livesearch CMPlone#282
  - [ ] login CMPlone#282
  - [fulvio, agitator] mark_special_links CMPlone#282
  - [ ] popupforms CMPlone#282
  - [ ] table_sorter CMPlone#282
  - [andrea] unlockOnFormUnload CMPlone#282
- Make sure correct widgets from plone.app.widgets are used for dublin core fields CMPlone#281
- content rules javascript broken on plone 5 CMPlone#279

2.3.1 Eventually, if someone is up to

- Issues in https://github.com/plone/mockup/issues
- plone.app.widgets code and test needs to be moved to CMFPlone/plone.app packages CMPlone#280
- legacy js legacy import not importing correct order CMPlone#275

2.3.2 By the way...

These are the Plone 5 issues, part of them are Mockup related: https://github.com/plone/Products.CMFPlone/issues/184

2.3.3 Archivements

Eric: working on ABCJS Adrian: working on checkbox slider pattern Lewis: working on foundation framework accodrion plugin Arno: working on isotope Andrea: working on UnlockOnFormUnload pattern Alessandro: working on livesearch pattern Giacamo: working on highlightsearchterms Peter and Fulvio: working on mark_special_links chris: working on pattern integration into plone

Contents:

CHAPTER 3

Introduction

Mockup is the new frontend library for Javascript development, which is going to be used by plone.app.widgets and Plone 5.

3.1 Background

Mockup grew out from searching for a solution to get all JavaScript within Plone into a manageable, reproducible, stable and testable state.

Until now, JavaScript in Plone was added here and there by core modules and addons to the Resource Registry of Plone. It was only possible to define the order in which JavaScript resources should be loaded. The order could be defined relative to another JavaScript resource. Same is valid for CSS resources. It was not possible to declare dependencies to other JavaScript resources. Either they were available because they were loaded beforehand into the global namespace, or the script threw an error. None - or rarely any - of the JavaScript within Plone was tested. Developers needed to know a fair amount of Plone developing to contribute JavaScript or CSS resources to it, such as defining resources, registering those in the registry, or at least knowledge about the (now deprecated) portal_skins machinery. Therefore, mainly Python developers were contributing JavaScript code. It was too hard and boring for Frontend developers to help out here.

Plone's new theming engine `Diazo` solved some of the problems. But still, JavaScript code was untested, and without a defined set of dependencies, it did or did not work, often only by accident.

Mockup gives us an elegant solution to all of the problems outlined above: By using RequireJS modules (as well as the Common JS based node modules for the build infrastructure), we now have to declare every JavaScript dependency explicitly. RequireJS resolves the dependency chains and loads all resources in the correct order. Clobbering the global namespace is normally not necessary anymore. Mockup encourages us to also test all JavaScript code, and it does it by using well known testing frameworks which allows us to easily write unit and integration tests. All patterns are encapsulated and can be easily reused. Reusability is possible, because dependencies from JavaScript code on a specific DOM structure is gone. Patterns are only loaded and configured via HTML class names and data attributes only. Paradise.

TODO: difference to patternslib Around 2012, Mockup came to the stage. Rok Garbas, always aware about the power behind JavaScript, developed Mockup out of frustration with JavaScript. Mockup started as a fork of split-off *Patternslib* - at least it took its idea of a Pattern registry. *Patternslib* was developed by Cornelis Kolbach, Wichert Akkerman and fellows, including Florian Friesdorf with whom Rok was in intense debate about ways of joining efforts.

Mockup stayed a separate project, was accepted for inclusion in the Plone core by the Framework Team in 2013 and is one big part of the upcoming Plone 5.

This documentation should help developers to better understand Mockup, effectively quickstart projects and use Mockup in production.

3.2 Why mockup?

But wait, despite all of the well sounding hymns to mockup from above, aren't there more established alternatives?

Well ... yes, there are. Mockup grew up in a world where none of these alternatives were in sight. Now we have at least an upcoming Web Components W3C standards draft with concrete implementations (X-Tags, Polymer) and Angular JS Directives. Both of them could be used instead of Mockup.

And you can, if you want. Since mockup is so encapsulated, you can just switch over and use something else. Or only small parts of Mockup. It's up to you and your requirements.

But Mockup is only a small layer, which is loading others' JavaScript. The bigger payload is the JavaScript loaded by mockup. E.g. TinyMCE, Select2, ACE Editor and more. OK, Web components are doing the same and Angular JS also. Web Components, a very promising approach which solves exactly the same problems as Mockup does, is a W3C draft and its implementations only work for the most modern browsers. So no IE<11 support with that. Angular JS is also a very interesting framework, but it changes the way you work with JavaScript quite a bit. Angular forces you to use all of their concepts. What sense would it make to only use Angular Directives without controllers, services, etc.?

I expect Mockup to stay relevant for Plone until Web Components are standardized and stable and ECMAScript 6 is implemented by all Browsers we want to support. That will happen, hopefully, in the near future. Then Mockup could be changed to be a Web Components implementation, which again uses other top notch, well established Frameworks to satisfy the needs of frontend developers.

3.3 The Goals of Mockup

1. Standardize configuration of patterns implemented in js to use HTML data attributes, so they can be developed without running a backend server.
2. Use modern AMD approach to declaring dependencies on other js libs.
3. Full unit testing of js.

3.4 Technologies used with Mockup

Mockup is much about JavaScript, and so it uses a JavaScript toolset, which is quite common among JavaScript developers. This toolset includes:

- Bower (Github, Wikipedia) for package management.
- Grunt (Github) for running repetitive tasks like combining files, minifying JavaScript, creating bundles and more.
- RequireJS (Github) for defining dependencies between JavaScript modules. It uses the Asynchronous Module Definition approach, as opposed to the CommonJS Module Definition is defining. There is a document, explaining the reason behind AMD, which is worth a read. With the upcoming ECMA Script 6 standard, we will get JavaScript module definitions and dependency declarations built into the Language.
- Yeoman (Github, Wikipedia) for generating pattern scaffolds.
- LESS (Github, Wikipedia) as CSS preprocessor.
- Node JS (Github, Wikipedia) as a requirement for Grunt.
- Mocha
- PhantomJS

- React JS

‘<>‘_ (‘Github <>‘_, ‘Wikipedia <>‘_)

As always, some of these technologies can be discussed controversially. There are other options for package management, build infrastructure, declaring dependencies, preprocessing CSS - nearly for each aspect of Mockup. JavaScript has an insanely fast moving ecosystem. Fortunately, many Frameworks are quite excellent. In the final analysis, we had to decide for some of these Frameworks. Mockup is using well proven and widely used Frameworks. For sure, we will have to adapt Mockup to fit to changed conditions in the future, but we’re well off with the technologies chosen.

CHAPTER 4

Installation and Bootstrapping

4.1 Normal Installation

The normal way to install *Mockup* is to directly install it to your system, using the system's *Node* version and other tools. If this is not possible, because you are using Windows or don't want to pollute your System with libraries not necessarily needed, use the *Installing Mockup with Vagrant* method.

4.1.1 Prerequisites

To build and hack on Mockup you will need recent versions of `git`, `node`, `npm`, `PhantomJS` and `make`.

- Installing Node: You need to have Node 0.10 or newer. Normally, NPM is installed together with node. For installation instructions see Installation via package manager or Building Node.
- Installing PhantomJS: Use your package manager or download and install).

Note: There is a useful tool, which helps you to install different versions of Node, if you need it. The Node Version Manager, `nvm`. Try it, if you run into problems with your system's Node version.

Right now development of this project is being done primarily on Linux and OS X, so setting up the tooling on MS Windows might be an adventure for you to explore – though, all of the tools used have equivalent versions for that platform, so with a little effort, it should work!

4.1.2 Installing Mockup

Installing *Mockup* is as easy as cloning and then running *make bootstrap*:

```
$ git clone https://github.com/plone/mockup.git
$ cd mockup
$ make bootstrap
```

This bootstraps the whole application. It cleans up the directory and installs node and bower dependencies.

4.2 Installing Mockup with Vagrant

Vagrant is a scripting environment for virtual machine hosts like VirtualBox. There are two configuration files, `Vagrantfile` and `provision.sh` which are used to bootstrap a whole system including any dependencies, which should be installed on the guest system.

Vagrant can be a great choice, if you don't want to install Mockup and its dependencies directly to your machine, possibly polluting your environment (but normally, Mockup doesn't install anything globally). Vagrant is also a great choice, if you want to provide the same environment for every developer. Therefore we chose it as the recommended installation method for our Mockup training.

With this method, Mockup is installed by installing a virtual machine with Vagrant by using the `Vagrantfile` and `provision.sh` files, which are included in Mockup. A guest VM (Ubuntu 14.04) is started with the Vagrantfile and provisioned with Mockup prerequisites. Then a bootstrap script is run on the guest VM. Follow these steps:

1. Install VirtualBox: https://www.virtualbox.org. Use your system's package manager, if you have one.
2. Install Vagrant: http://www.vagrantup.com. If you have such a thing like a system package manager, I recommend to use it only, if it includes a recent version of Vagrant.
3. If you are using Windows, install the Putty ssh kit: http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html. Install all the binaries, or at least putty.exe and plink.exe.

Now we can install Mockup itself:

```
$ # Run the following on the host OS:
$ git clone https://github.com/plone/mockup.git
$ cd mockup
$ vagrant up
```

Now, go for lunch or a long coffee break. "vagrant up" is going to download a virtual box kit (unless you already happen to have a match Windows, it will also generate an ssh key pair that's usable with Putty.

Note: While running "vagrant up", feel free to ignore messages like "stdin: is not a tty" and "warning: Could not retrieve fact fqdn". They have no significance in this context.

Look to see if the install ran well. The virtual machine should be running at this point:

```
$ vagrant reload
$ vagrant ssh
```

Now you are logged into your virtual machine:

```
$ # Run the following on the guest VM:
$ cd /vagrant
$ git pull
$ make bootstrap
```

Now you have the complete source code for all Patterns from Mockup. From here on you generate bundles of common functionality and minify them.

You're ready to start working on testable, modular and beautiful JavaScript!

Note: Parts of this instructions are based on the plonedev.vagrant README.rst. Have a look for it, if you need more information and troubleshooting instructions.

4.2.1 Using Vagrant

Understanding Vagrant in depth is out of this document scope. The most important commands for using vagrant are listed below.

- `vagrant up`

  This command will start the virtual environment. When running it for the first time it will install and configure all needed packages. NOTE: Some of the output will be in red, this is normal.

- `vagrant reload`

  This command will make the virtual environment restart. You need to do this the first time

- `vagrant ssh`

  Once the virtual environment is up and running, this command will ssh into the machine. This is like ssh'ing into any computer, all you need to do to exit is "Ctrl + D"

- `vagrant halt`

  This command will turn off the virtual machine, issuing an ACPI shutdown, so it's safe to use it at any time.

- `vagrant destroy`

  This command will destroy the virtual environment. Be aware that this will remove the entire virtual machine. Be careful and know when you're using it.

For additional reading, go to Vagrant homepage.

CHAPTER 5

Working with Mockup

5.1 Building the documentation and examples

To see it in action, you must compile everything once with:

```
$ make docs
```

Then, start the python test server like so:

```
$ python -m SimpleHTTPServer
```

After that, access the served site in a web browser via the url http://localhost:8000

5.2 Running tests

Run tests with PhantomJS and continue to listen for changes:

```
$ make test
```

Run tests with Chrome:

```
$ make test-dev
```

Or run the tests for an individual plugin:

```
$ make test-once pattern=select2
```

5.3 More Makefile commands

The `Makefile` provides this list of commands:

```
all                 Tests everything once, creates all default bundles and builds the documentation.
bootstrap           Bootstrap Mockup. Cleans the environment (deletes node_modules and bower_componer
bootstrap-common    Common tasks for other bootstrap tasks. Not intended to be run manually.
bootstrap-nix       Bootstraps Mockup for NixOS environments. It installs all dependencies via the ni
bundle-filemanager  Builds the resourceeditor filemanager bundle.
bundle-plone        Builds the Plone bundle.
bundle-resourceregistry Builds the bundle for the new resource registry.
bundle-structure    Builds the structure bundle (wildcard.foldercontents content browser).
```

```
bundle-widgets      Builds the widgets bundle.
bundles             Builds all the default bundles (bundle-widgets, bundle-structure, bundle-plone).
clean               Clean the environment by removing the build, node_modules and bower_components di
clean-deep          Clean the environment like with ``clean`` and additionally clean bower's and node
docs                Builds the Mockup documentation.
jshint              Run the code quality suite (jshint and jscs).
publish-docs        Publish the github pages documentation.
test                Run Mockup's tests and keep watching for file changes. Accepts the option [--patt
test-ci             Run the tests on the Continious Integration server environment.
test-dev            Run Mockup's tests in the Chromium browser and keep watching for file changes. Ac
test-once           Run Mockup's tests only once. Accepts the [--pattern=PATTERNNAME] option to defin
watch               Watches for file changes and rebuilds Mockup.
```

All tests also accept the experimental `--debug` and `--verbose` options to help with debugging by changing the verbosity of the log messages.

5.4 Using Bower directly

After making changes to bower.json, you don't have to run `make bootstrap`, which wipes all dependencies and starts installing them all over again. You can use bower directly:

```
$ bower search PACKAGENAME  # search online for a package in the bower registry
$ bower list  # list all dependencies and possible updates
$ bower install  # install all dependencies listed in bower.json
$ bower update  # update all dependencies to the versions specified in bower.json
```

For more information, see the bower API documentation.

5.5 Including a local mockup-core checkout for developing

If you want to also hack on mockup-core together with mockup, clone mockup-core into a directory on your machine and just symlink it into bower_components:

```
$ cd ..
$ git clone https://github.com/plone/mockup-core
$ cd mockup/bower_components
$ rm -R mockup-core
$ ln -s ../../mockup-core .
```

Note: You can also point bower.json to a local git checkout. You have to point bower directly to the *.git* subdirectory and declare the branch name in order to be able to use a local checkout. For that, replace the *mockup-core* line in *bower.json* with something like the following:

```
"mockup-core": "file:///PATH/TO/mockup-core/.git/#master"
```

Please note, you have to commit any changes on mockup-core and then run `bower install`, `bower update` or `make bootstrap` in mockup again.

CHAPTER 6

Overview of Mockup

There is also a basic minimal pattern with comments all over the source code, which explains the structure of a Mockup project. You can find it here: https://github.com/collective/mockup-minimalpattern

6.1 Mockup Nomenclature

Pattern Patterns are units of JavaScript, defined by a RequireJS/AMD style module. Patterns may require other patterns to operate, and may also require third party libraries. Think of a pattern as a module – encapsulated and separate, and providing a widget or tool to be used by other patterns or in html.

Bundle Bundles are defined in a similar way to *Patterns* – they are encapsulated bits of JavaScript that define requirements for a bundle and have some extra code in them that's useful for integrating the required patterns into Plone products.

6.2 RequireJS vs Common JS modules

Mockup is using RequireJS and Common JS modules at the same time. For all frontend related stuff, RequireJS with its configuration in *js/config.js* and *define* method is used. For all build infrastructure related stuff, CommonJS modules, with its *module.exports* declarations are used.

If you are aware about this distinction, you can avoid some head wrangling moments.

6.3 Mockup Core Project Structure

Mockup Core defines the basic infrastructure for Mockup: the base Grunt tasks, a base pattern, the pattern registry and the documentation building framework. You can reuse these components for mockup projects.

`bower.json`: All frontend related, bower managed dependencies.

`bower_components/`: Here, all bower managed dependencies are installed.

`Gruntfile.js`: Defines the tasks for running jshint and the tests.

`js/config.js`: RequireJS configuration. This is the file, where all JavaScript dependencies are defined, so that RequireJS is able to find them via a name.

`js/docs/`: Infrastructure for creating mockup's documentation from comment-sections in pattern files, following a specific convention.

`js/grunt.js`: Mockup Core's grunt infrastructure.

`js/pattern.js`: Base Pattern.

`js/registry.js`: Mockup pattern registry.

`Makefile`: `GNU make` Makefile, which defines common actions for developing with mockup-core. It uses Grunt to a large extend.

`node_modules/`: Node / npm managed dependencies are in here. These are all not-frontend related JavaScript dependencies for running grunt, bower, tests and the like.

`package.json`: Node / npm package dependencies and metadata for Mockup Core's infrastructure. The dependencies defined in here land in `node_modules`.

`tests/`: Contains all tests, including general setup and configuration code.

6.4 Mockup Project Structure

`bower.json`: All frontend related, bower managed dependencies.

`bower_components/`: Here, all bower managed dependencies are installed.

`build/`: Contains the builded bundles. This are combined, optimized, and minimized JavaScript code, as well as the compiled CSS (Less) and media files from a bundle's dependencies.

`docs/`: Mockup documentation files and examples built with `make docs`.

`Gruntfile.js`: Defines the tasks for creating bundles, documentation and running the tests. It depends on mockup-core's grunt base file.

`index.html`: Documentation index.html file. This is the entry file when viewing the documentation via `http://localhost:8000` after starting `python -m SimpleHTTPServer` in mockup root.

`js/bundles/`: The directory, where Mockup's bundles are defined.

`js/config.js`: RequireJS configuration. This is the file where all JavaScript dependencies are defined, so that RequireJS is able to find them via a name.

`js/i18n.js`: Fork of jarn.jsi18n, a JavaScript i18n framework and integration layer for Plone message catalogs.

`js/router.js`: Framework to add routing capabilities, execute callbacks on routing, and manipulating the browser history on routing.

`js/ui/`: Mockup UI components for reuse in patterns (Buttons, Toolbars, etc).

`js/utils.js`: Generic reusable utilities for patterns. Current available utilities include: `generateId`, `Loading` animation, `QueryHelper` for generating request query strings and more.

`less/`: All LESS style files, which are needed for bundles or patterns. Grunt's less task default behavior is to compile a CSS file from the less file with the same name as the bundle name. For more information se the grunt.js file in `mockup-core`.

`lib/`: Extra frontend libraries, which are not available via bower. In our case it's the jquery.event.drag.js and jquery.event.drop.js files, which provide drag and drop events for jQuery.

`Makefile`: `GNU make` Makefile, which defines common actions for developing with mockup. It uses Grunt to a large extent. E.g. `make bootstrap`, `make test`, `make docs` and `make bundles`. For more information see: *More Makefile commands*.

`mockup/`: Files for integrating `mockup` in Plone, e.g. in `plone.app.widgets`. This is only for development purposes, so that the task of copying bundle files to `plone.app.widgets` isn't necessary. Still, you have to do a `make bundle-BUNDLENAME` for compiling the files, which are accessed by `plone.app.widgets`.

`node_modules/`: Node / npm managed dependencies are in here. These are all not-frontend related JavaScript dependencies for running grunt, bower, tests and the like.

`package.json`: Node / npm package dependencies and metadata for Mockup's infrastructure. The dependencies defined in here land in `node_modules`.

`patterns/`: Here, the actual patterns are defined. For each pattern, one directory. Some patterns include LESS resource files, templates and submodules.

`setup.py`: Setuptools based Python package infrastructure. This file is needed to include Mockup in Plone for development, e.g. in `plone.app.widgets`.

`tests/`: Contains all tests for patterns and bundles, including general setup and configuration code.

CHAPTER 7

Create your own mockup project

As seen earlier in this training, one alternative to work with Mockup is to pull the complete source from github, do your changes, removals and modifications and finally compile. This process has obvious disadvantages, and for this we have created a generator, that allows you to generate the bare minimum you need to start your project.

7.1 Dependencies

7.1.1 System wide installation

If you feel comfortable installing packages in your system, all you need for using the generator is npm and yo. If you're on Ubuntu :

```
$ sudo apt-get install npm nodejs-legacy
$ npm install -g yo
$ npm install -g generator-plonemockup
```

7.2 Creating your project

We will now create our first project

7.2.1 Without using Vagrant

Just create a directory where your project will be. We'll call it "myproject". Inside this directory we can run our generator.

7.2.2 Using Vagrant

Just pull the development version of the generator:

```
$ git clone git@github.com:collective/generator-plonemockup.git
```

The relevant files are "Vagrantfile" and "provision.sh". Understanding and using Vagrant is out of this document's scope. For now, all you need to care about is running any of the following commands inside the directory you have just cloned, that holds the Vagrantfile:

- `vagrant up`

Make sure you run "vagrant ssh" to log in to your virtual environment. When inside, you will be user "vagrant", which has full sudo privileges inside that machine.

the /vagrant directory will be our working directory, and is the same as the directory outside the virtual environment.

So, if you:

```
vagrant@mockup-generator:~$ cd /vagrant/
vagrant@mockup-generator:/vagrant$ ls
```

You should see the same content as you had from "the outside".

Go to /vagrant and create a directory for your project, we'll call it "myproject" and inside, we'll run our generator

7.2.3 Common steps

From this point on, instructions are the same whether you are running vagrant or not:

```
$ cd myproject/
$ yo plonemockup
```

The generator will ask a bunch of questions for generating the project and populating the packages.json among other things. They are detailed below:

- `Name for the project`

This is the name of the whole project, it is used for class and file names. You can put anything here, except for hyphens and spaces (This will get fixed eventually), so for now, we are going to name it "myproject"

- `Version`

Self-explanatory. The version of your project. Make sure you use x.y.z format (ie. 3 numbers)

- `Description`

An optional description, you can use any character in here.

- `Homepage`

Optional, specify your project's URL

- `Repository`

If you intend to publish this in a version control system, you can specify the URL. It will default to https://github.com/collective/{packageName}, in our case https://github.com/collective/myproject

- `Author full name, email and webpage`

You can specify your name, email address and a web site to be included as metadata with your project

- `License`

Choose the license for your project. At the moment, GPLv2, GPLv3 and MIT are supported. You can modify the package later to provide your own if needed.

- `Name for your pattern`

Here you can specify the name for a pattern you intend to develop. There is no way to not choose one at the moment, there will be in future versions. If you don't intend to develop a pattern, just put any name here, and you can remove it manually later. We will be developing a new pattern, and we are going to call it "mypattern". Just like the project name, avoid spaces and hyphens (This will also be fixed in future releases)

After answering all questions, your package is created and all dependencies are pulled in using bower and npm. Just wait until it finishes. If you get an error at this moment, you can re-run 'npm install' and 'bower install' as needed.

When running the automated process for the first time, it may happen that the process just hangs. This might be because at one point, bower asked:

```
[?] May bower anonymously report usage statistics to improve the tool over time?
```

and all the npm install output hides it... don't worry, just type 'yes' or 'no' and hit 'Enter'. Or if you intend to respond 'yes' anyway, just hit 'Enter' directly. You will see the above question printed again...

7.3 Working with your new package

This will assume we have named our project 'myproject' and our pattern, 'mypattern'

7.3.1 Structure

The newly created package has the full structure of files and directories ready to start developing:

```
-- bower.json
-- config.js
-- dev
|   -- dev.html
|   -- dev.js
-- Gruntfile.js
-- js
|   -- bundles
|   |   -- myproject.js
|   -- patterns
|       -- mypattern.js
-- less
|   -- myproject.less
-- package.json
-- README.md
-- tests
    -- config.js
    -- pattern-mypattern-test.js
```

7.3.2 The 'dev' directory

This is a helper folder. It provides a dev.html which already includes basic html to start developing your pattern and includes the needed javascript files. The 'dev.js' file is the one that loads your bundle and pattern(s) automatically so you can start developing right away.

7.3.3 The 'js' directory

This is where your bundle and pattern(s) will be located. You will be working mostly in js/patterns/mypattern.js if you are developing a new pattern, or in js/bundles/myproject.js if you are bringing additional patterns from mockup.

7.3.4 The 'less' directory

This is where you will include the less files for your project.

7.3.5 The 'tests' directory

This is where automated tests for your patterns will be written (And of course you will write them ;))

7.3.6 config.js

In case you add new patterns, libraries, or need to tweak some paths, this is the file where you should do that.

7.4 Compiling and testing

Once developing is done and you want to compile your work, just go to your project's root directory and run 'make':

```
$ make
```

If you don't get any errors, you should end up with a new directory called build, where you will find your files ready to use.

CHAPTER 8

Creating your first pattern

Now that you have generated your first package, we will be creating a pattern. The generator already creates the boilerplate for a pattern and the needed bits to use it, so we will be just adding code to it.

8.1 Write the pattern

Open the `js/patterns/mypattern.js` file. Its sections are already commented:

```
define([
    'jquery',
    'mockup-patterns-base'
], function($, Base) {
    'use strict';

var mypatternPattern = Base.extend({
    // The name for this pattern
    name: 'mypattern',

    defaults: {
        // Default values for attributes
    },

    init: function() {
        // The init code for your pattern goes here
        var self = this;
        // self.$el contains the html element
        self.$el.append('<p>Your Pattern "' + self.name + '" works!</p>');
    }
});

return mypatternPattern;

});
```

8.2 See it in action

There are two ways of seeing the pattern through the web:

8.2.1 1) Open it in a browser

The best way to understand how all this works is to see it working. Open `dev/dev.html` in your browser

WARNING: Working locally will make the browser to complain with:

```
XMLHttpRequest cannot load ... Cross origin requests are only supported for HTTP.
```

This is a security feature. To go around it in Chrome, you need to start it with a `--disable-web-security` parameter

8.2.2 2) Use Python's SimpleHTTPServer

In the project root, start Python's SimpleHTTPServer like so:

```
$ python -m SimpleHTTPServer 8000
```

And open your web browser pointing to http://localhost:8000/dev/dev.html

Now, after this, you should see the message `Your Pattern "mypattern" works!`. This is because the pattern we have just created does this in its `init` class (`self.$el.append('<p>Your Pattern "' + self.name + '" works!</p>');`) where, `self.$el` is the HTML element which loads the pattern.

8.3 Let's make something interesting

8.3.1 Add some HTML

Open `dev/dev.html` file in your editor and replace:

```
<div class="pat-mypattern">
</div>
```

with:

```
<div class="pat-mypattern">
    <p class="target">This will get a background color</p>
    <button class="trigger">Press me!</button>
</div>
```

8.3.2 Add some CSS

Open `less/myproject.less` file in your editor and add:

```
.red-background {
    background-color: red;
}

.blue-background {
    background-color: blue;
}
```

8.3.3 Write your pattern

Open `js/patterns/mypattern.js` file and replace everything with:

```
define([
    'jquery',
    'mockup-patterns-base'
], function($, Base) {
    'use strict';

var mypatternPattern = Base.extend({
    name: 'mypattern',

    defaults: {
        initial_color: 'red'
    },

    change_color: function ($this) {
        var self = this;
        self.$el.find('p.target').removeClass(self.$current_color+'-background');
        if ( self.$current_color === 'red' ){
            self.$current_color = 'blue';
        }
        else {
            self.$current_color = 'red';
        }
        self.$el.find('p.target').addClass(self.$current_color+'-background');
    },

    init: function() {
        var self = this;
        self.$el.find('button.trigger').on('click', function(e) {
            self.change_color();
        });
        self.$current_color = self.options.initial_color;
        self.$el.find('p.target').addClass(self.$current_color+'-background');
    }
});

return mypatternPattern;

});
```

So, let's explain what the things are that we added:

- We modified the `init` method, so:
    1. It will subscribe to an event when pressing the button to call the `change_color` method
    2. It will get the default value of `initial_color`, defined in `defaults` and save it in an internal variable
    3. We assign the class to the <p> element
- We defined a default initial `red` value for the `initial_color`. More on this later.
- We created a new method, called `change_color` that will change from `red` to `blue` and back.

Now, if you refresh your browser, the paragraph should have a red background, but when pressing the button, it switches to blue, and then back to red when pressed again.

8.4 Defining initial default values

As we saw before, we defined an `initial_color` variable under `defaults` in our pattern. Variables defined here are the ones that we are going to be able to modify with data attributes from our HTML, so if you plan on developing a reusable pattern that you can use in several ways, this is the way to do it.

In our example, if we change our HTML as follows:

```
<div class="pat-mypattern" data-pat-mypattern="initial_color:blue;">
    <p id="target">This will get a background color</p>
    <button id="trigger">Press me!</button>
</div>
```

Then, instead of our paragraph starting as `red`, it will first be `blue` and change to red when first pressing the button.

As you can see, all default variables defined under `defaults` will be available under `self.options`

8.5 Isolation

One great thing about patterns is that they only affect the HTML code where they were applied. For this, you should always work with the `self.$el` element, as we did in our example. In order to understand this idea better, open your `dev/dev.html` file again, and replace:

```
<div class="pat-mypattern">
    <p class="target">This will get a background color</p>
    <button class="trigger">Press me!</button>
</div>
```

With:

```
<div class="pat-mypattern">
    <p class="target">This will start with a red background color</p>
    <button class="trigger">Press me!</button>
</div>

<div>
    <p class="target">This will get no background color</p>
    <button class="trigger">Press me!</button>
</div>

<div class="pat-mypattern" data-pat-mypattern="initial_color:blue;">
    <p class="target">This will start with a blue background color</p>
    <button class="trigger">Press me!</button>
</div>
```

If you now refresh your browser, you'll see that even though we did no changes to the javascript code, and just by defining some classes and data attributes, we can change the functionality, but have it be specific to a portion of the HTML.

CHAPTER 9

Including a pattern from Mockup

So, you have seen all the patterns already bundled with the mockup project in the examples page and you want to use one (or several). You'll find this incredible easy.

We will be assuming you have created your project using the instructions from the previous chapter. If not, then adjust your variables and file names accordingly.

9.1 Assume the following HTML

This step is not really needed in order to include a dependency, we will use it just as an example. Let's say you have the following HTML:

```
<table border="1" style="text-align:center;">
    <thead>
        <tr>
            <th>Name</th>
            <th>Surname</th>
            <th>Arbitrary Number</th>
        </tr>
    </thead>
    <tbody>
        <tr>
            <td>Linus</td>
            <td>Torvalds</td>
            <td>4</td>
        </tr>
        <tr>
            <td>Ada</td>
            <td>Lovelace</td>
            <td>1</td>
        </tr>
        <tr>
            <td>Alan</td>
            <td>Turing</td>
            <td>3</td>
        </tr>
        <tr>
            <td>Dennis</td>
            <td>Ritchie</td>
            <td>2</td>
        </tr>
        <tr>
```

```
            <td>Richard</td>
            <td>Stallman</td>
            <td>5</td>
        </tr>
    </tbody>
</table>
```

You can include this in your `dev/dev.html` file from your project. Now, if you open it using your web browser, you will notice you have a basic table with no functionality whatsoever. But we want to be able to sort its content by clicking the titles.

9.2 Pick the pattern to use

For this exercise, we will pick the Table Sorter pattern. To use it, we need to add the special css class `pat-tablesorter` to the HTML snippet we have, so replace:

```
<table border="1" style="text-align:center;">
```

with:

```
<table border="1" style="text-align:center;" class="pat-tablesorter">
```

Be aware that you can include several classes at the same time, as you would normally do. That doesn't affect mockup functionality. Now refresh your browser. You will see that nothing has changed, and you still cannot order your table. Don't despair. Inside your project folder, look for the `config.js`. This is the file where all dependencies are listed. We see that Table Sorter pattern 'internal name' is 'mockup-patterns-tablesorter'.

9.3 Include the dependency in your bundle

Now that we know the internal name for the pattern we want to use, let's include it in our bundle. Go to your project directory, and open `js/bundles/myproject.js`. You will notice a section as follows:

```
define([
    'jquery',
    'mockup-registry',
    'mockup-patterns-base',
    //   Uncomment the line below to include all patterns from plone-mockup
    //   'mockup-bundles-widgets',
    //   <!~~ Add patterns below this line ~~!>
    'myproject-patterns-mypattern'
], function($, Registry, Base) {
    'use strict';
```

Just edit this list, and include our dependency. Friendly reminder: Be aware that this is a Javascript list, and as such it should NOT include a comma at the end of the last item.

That section should now look as follows:

```
define([
    'jquery',
    'mockup-registry',
    'mockup-patterns-base',
    //   Uncomment the line below to include all patterns from plone-mockup
    //   'mockup-bundles-widgets',
    //   <!~~ Add patterns below this line ~~!>
```

```
    'mockup-patterns-tablesorter',
    'myproject-patterns-mypattern'
], function($, Registry, Base) {
    'use strict';
```

That's it, now refresh again... you should be able to sort your table by clicking the column headers. If you now run `make`, Table sorter will be included in the compiled Javascript file.

9.4 Include a pattern with style dependencies

We have seen a very basic example of a small pattern that doesn't need much. Let's try again, but now we will include the Autotoc pattern.

9.4.1 Consider the following HTML

```
<div>
    <h1>Title 1</h1>
    <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nulla ultrices <br/> tempus purus vel
    <h2>Title 1.1</h2>
    <p>Aenean suscipit ligula nibh, vel congue dui mattis vel.<br/>  Phasellus ut nulla eget nisi veh
    <h3>Title 1.1.1</h3>
    <p>Phasellus interdum <br/> interdum nisi eu sagittis. Ut hendrerit feugiat <br/> nisl auctor ver
    <h2>Title 1.2</h2>
    <p>Fusce posuere turpis a lacus laoreet, in <br/> blandit mauris vehicula. Cras mattis vitae ex e
    <h1>Title 2</h1>
    <p>Sed vel tellus et <br/> ligula aliquet volutpat eget nec ante. Nulla eu pretium est. <br/> Mor
    <br/><br/><br/><br/><br/><br/><br/><br/><br/><br/><br/><br/><br/><br/><br/><br/><br/><br/><br/><b
</div>
```

We will now try to use the "Autotoc" pattern, so again, we search for its internal name in the `config.js` file, and include it as dependency in our bundle. So replace:

```
define([
    'jquery',
    'mockup-registry',
    'mockup-patterns-base',
    //   Uncomment the line below to include all patterns from plone-mockup
    //   'mockup-bundles-widgets',
    //   <!~~ Add patterns below this line ~~!>
    'mockup-patterns-tablesorter',
    'myproject-patterns-mypattern'
], function($, Registry, Base) {
    'use strict';
```

With:

```
define([
    'jquery',
    'mockup-registry',
    'mockup-patterns-base',
    //   Uncomment the line below to include all patterns from plone-mockup
    //   'mockup-bundles-widgets',
    //   <!~~ Add patterns below this line ~~!>
    'mockup-patterns-tablesorter',
    'mockup-patterns-autotoc',
```

```
    'myproject-patterns-mypattern'
], function($, Registry, Base) {
    'use strict';
```

And add the proper class to the outer div of our example, so it will look like this:

```
<div class="pat-autotoc">
    <h1>Title 1</h1>
    <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nulla ultrices <br/> tempus purus vel
    <h2>Title 1.1</h2>
    <p>Aenean suscipit ligula nibh, vel congue dui mattis vel.<br/>  Phasellus ut nulla eget nisi veh
    <h3>Title 1.1.1</h3>
    <p>Phasellus interdum <br/> interdum nisi eu sagittis. Ut hendrerit feugiat <br/> nisl auctor ver
    <h2>Title 1.2</h2>
    <p>Fusce posuere turpis a lacus laoreet, in <br/> blandit mauris vehicula. Cras mattis vitae ex e
    <h1>Title 2</h1>
    <p>Sed vel tellus et <br/> ligula aliquet volutpat eget nec ante. Nulla eu pretium est. <br/> Mor
    <br/><br/><br/><br/><br/><br/><br/><br/><br/><br/><br/><br/><br/><br/><br/><br/><br/><br/><br/><b
</div>
```

After refreshing your browser, you can see that the TOC has been generated, but is missing the styling.

9.4.2 Include less

Open `less/myproject.less` file and add this line to the end of it:

```
@import "../bower_components/plone-mockup/patterns/autotoc/pattern.autotoc.less";
```

Now refresh again, and you will see your TOC has a default styling applied.

9.4.3 Configure the pattern

As a final exercise, this pattern allows configuration to be passed from the HTML, so let's make the duration of the movement to be fast and to only include h1 and h2 in the TOC:

```
<div class="pat-autotoc" data-pat-autotoc="scrollDuration:fast;levels:h1,h2;">
    <h1>Title 1</h1>
    <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nulla ultrices <br/> tempus purus vel
    <h2>Title 1.1</h2>
    <p>Aenean suscipit ligula nibh, vel congue dui mattis vel.<br/>  Phasellus ut nulla eget nisi veh
    <h3>Title 1.1.1</h3>
    <p>Phasellus interdum <br/> interdum nisi eu sagittis. Ut hendrerit feugiat <br/> nisl auctor ver
    <h2>Title 1.2</h2>
    <p>Fusce posuere turpis a lacus laoreet, in <br/> blandit mauris vehicula. Cras mattis vitae ex e
    <h1>Title 2</h1>
    <p>Sed vel tellus et <br/> ligula aliquet volutpat eget nec ante. Nulla eu pretium est. <br/> Mor
    <br/><br/><br/><br/><br/><br/><br/><br/><br/><br/><br/><br/><br/><br/><br/><br/><br/><br/><br/><b
</div>
```

CHAPTER 10

Writing documentation for Mockup

The documentation for Mockup is automatically generated from comments in pattern code. The structure is as follows:

```
/* PATTERN TITLE
 *
 * Options:
 *    OPTION_TITLE(TYPE): DESCRIPTION
 *
 * Documentation:
 *   # Markdown title
 *
 *   Markdown structured description text
 *
 *   # Example
 *
 *   {{ EXAMPLE_ANCHOR }}
 *
 * Example: EXAMPLE_ANCHOR
 *    <div class="pat-PATTERN_NAME"></div>
 *
 * License:
 *   License text, if it differs from the package's license, which is
 *   declared in package.json.
 *
 */
```

CHAPTER 11

Mockup and Alternatives

Mockup was created at a time when no alternatives were in sight. But nowadays, with the explosion of JavaScript Frameworks, there are plenty of them.

11.1 Web Components

Web Components are a W3C draft and will likely gain a lot of momentum once browsers better support it. Web components can already be used by using Google's Polymer or Mozilla's X-Tags libraries. The only drawback is that those libraries only work for recent browser generations.

When Web Components gather more momentum, they will make Mockup obsolete. It's hard to say when this will happen. Maybe in one or two years? However, Mockup can probably be changed to use Web Components, so there will be an upgrade path.

11.2 Angular JS Directives

Angular JS also provides a framework to build reusable components. Actually, everything DOM related is implemented as a directive (TODO: PROOF). The Angular JS way of defining directives feels a lot like defining patterns in Mockup. Of course, Angular JS grew so much that Mockup has a hard time to compete with it. However, Angular JS provides a lot more than Mockup and for projects who do not want to use this fully fledged, all inclusive Framework but need something to manage JavaScript snippets of widgets, Mockup is a good choice.

CHAPTER 12

Working with the new resource registry

If you want to include a pattern in Plone 5's new resource registry, follow these instructions.

12.1 Include a pattern from the `Mockup` package in Plone core

The `Mockup` package is already a core dependency of `Products.CMFPlone` and Mockup's pattern directory is available as a browser resource and thus accessible for Plone.

All you have to do is to edit `Products/CMFPlone/profiles/dependencies/registry.xml` and add the a new record for your pattern. You have to define a name, like `plone.resources/mockup-patterns-select2` and the paths to your JS and LESS files. Don't forget to rerun the profile again (the name of this profile is: "Mandatory dependencies for a Plone site").

If you want to include your pattern by default, include it in the `Products/CMFPlone/static/plone.js` bundle, compile it client-side via the new resource registry (`@@resourceregistry-controlpanel`), which puts the compiled file back to the filesystem, commit it and push it.

12.2 Include a pattern from an external Mockup based project in your Plone project

If your pattern is in a separate package than `Mockup`, then you have to make sure that your pattern is accessible for Plone via a browser resource. You also have to provide a Generic Setup profile for plone.app.registry (`registry.xml`) and add an entry for your pattern in it.

CHAPTER 13

Mockup development process

13.1 Code style

Use jshint, and it will cover most code style issues. We included a `.jshint` file with Mockup's code style configuration. You can use this one in your editor to automatically display syntax errors (E.g. by the plugin `syntastic` in vim, if you use it).

We use spaces instead of tabs and an indentation level of 2 spaces per tab.

13.2 Contributions

For each feature, create a branch and make pull-requests on Github. Try to inclue all your changes in one commit only, so that our commit history stays clean. Still, you can do many commits to not accidentliy loose changes and still commit to the last commit by doing:

```
git commit --amend -am"my commit message".
```

Don't forget to also include a changelog entry in the `CHANGES.rst` file.

13.3 Documentation

Besides documenting your changes in the `CHANGES.rst` file, also include user and developer documentation as appropriate.

For patterns, the user documentation is included in a comment in the header of the pattern file, as described in *Writing documentation for Mockup*.

For function and methods, write an API documentation, following the apidocjs standard. You can find some examples throughout the source code.