How To Open Source a Light Module

Guidelines, and step-by-step instructions on how to open source and share a Light Module.

Prepare to Share

So - you have an idea - or an existing light module that you are ready to share? Great!

Please read this page on the Magnoila Documentation first: to get the big picture on why we chose npm, and the key guidelines for a quality shared module.

Finding and sharing light modules

Looking for an idea? Consider these three flavors:

1. Pure Magnolia functionality - Something working with navigation, a megamenu, breadcrumbs, a tool to reference content apps.
2. Provide a JS or CSS library to authors - Chart.js, Jquery plugins, Image galleries, etc.
3. Provide a web service to authors - Ad servers, social network widgets, maps, etc.

Here is the most simple example of a shared module, which you could clone from github to use as a boilerplate to get started sharing your own module:

- https://github.com/magnolia-cms/shareable-magnolia

Here's a few other examples of best-practice shared modules on npm & github (The npm package always contains a link to github):

- https://www.npmjs.com/package/language-switcher-magnolia
- https://www.npmjs.com/package/chartjs-magnolia
- https://www.npmjs.com/package/calculator-magnolia

Webinar on sharing light modules

At 22:00 - A Step by step walkthrough of these steps with a new light module.

Name

The name of a shared module should be the same in magnolia, on npm, and on github (or whichever public source code repository you use), and should work well in all of these contexts. For these and other reasons we recommend the pattern `<what-it-is>-magnolia`.

For example: `language-switcher-magnolia`, `chartjs-magnolia`, `pinterest-magnolia`.

Location

Find a good location on your harddrive for shared code. If you are working with an existing light module from a project, its a good idea to copy or move the module out of the project to make it easier to maintain. If you are creating a new module, you can use `mgnl create-light-module`.

To ensure that your module will work for others and really contains everything it needs, it's a good idea to polish it in a fresh Magnolia instance. To grab a bundle, you can use `mgnl jumpstart`.

Symbolic links are a convenient way to have a shared light module in one location on your harddrive, but use it in many locations. Open a terminal in your servers light-modules location and run:

```
ln -s <path to light module> <name of light module dir>
```

Example:

```
ln -s /Users/czimmermann/Documents/shared-projects/chartjs-magnolia chartjs-magnolia
```

License

Choose a license, and add a license file to your module.

We recommend the MIT license for modules that you are sharing publicly, this gives others freedom to use your module on any project.

README.md

Good README files are key to the success of the Magnolia light module ecosystem. Someone looking for a module should understand yours in 1 minute thanks to screenshots and clear descriptions.

For a template for sharing - use this file: https://raw.githubusercontent.com/magnolia-cms/shareable-magnolia/master/README.md

GIt

Initialize a git repository in your module directory. Code can be hosted in any publicly available repository, but we recommend github due to its popularity and comprehensive feature set. If you create a https://github.com account, and create a repository via the website, it provides you with very helpful instructions about how to connect it to the project on your harddrive.

(Note: The "npm init" command will then automatically use the git information.)

npm init

Run `npm init` in your module directory - this prepares your module for distribution on npm by creating a package.json file. Accept the default prompts, but change the version to 0.0.1 unless you know you are ready for first official release.

npm Keywords

Once the package.json is created, add keywords to the package.json to make it discoverable via search, and to assist automation.

- `magnolia-light-module` (Required)
- `magnolia-component` (Optional - for a single component)
- `magnolia-kit` (Optional - for a collection of templates)

Example:

```
"keywords": [
    "magnolia-light-module",
    "magnolia-component"
  ],
```

module.yaml

`module.yaml` is the Magnolia module descriptor, an optional file where you can specify any hard Magnoila dependencies. Specify this if your module uses a feature which became available in a certain version, such as decorations.

Example: https://raw.githubusercontent.com/magnolia-cms/shareable-magnolia/master/module.yaml

Other Files: _dev

All files that you would like to include in your github source control project, but are not part of the actual light module, should be placed in the `_dev` directory.

.npmignore

While the source control project should include everything to help a developer use and understand your module, the published package on npm should be just a light-module that is ready to use in production. The `.npmignore` file is used by npm to exclude files from being published to npm. It should be configured to ignore the `_dev` directory.

Example: https://raw.githubusercontent.com/magnolia-cms/shareable-magnolia/master/.npmignore

Demo content

A demo page with one or more instances of the component delivered in the module can be very helpful for someone evaluating your module.

Export your demo page(s) to xml bootstrap files, and place them in `_dev/demos`. Describe how to access them in the Demo section of the readme.

Tips:

- Name your page: `demo-<your module name>`
- You can use the mtk basic page template to host your component.
- To load any necessary webresources on the sample page, you can add an HTML component on the page and include script and style tags in the HTML content.

Java Jars

If your module depends on jars that are not in the standard Magnolia bundles, developers will need to install these jars manually. It is recommended that you include a link url to any jars in the "Usage" section of the readme file and explain how to install them. If you decide that you want to include the jars in your project - place them in `_dev/jars`.

Tips for handling resources and JS

- Component templates typically should not contain script or stylesheet link tags to bring in webresource files. Developers who use your module must have control over how they want to include the webresources. (Via resfn, themes, static includes, require.js or whatever approach they wish.)
- Component templates can include initialization JS which relies on those libraries.
- A component can be placed multiple times on the same page. It can help to have a unique id for each instance of the component. This can easily be created by using the id of the component node. (freemarker expression: `${content.@id}`)

Tips for enabling customization

It can be valuable to provide a way for developers to customize how your module functions in their project. For example, the languageSwitcher component can be customized to render either a selectbox or an unordered list by default.

- Template parameters can be used. A developer could change the template parameters directly in the template definition, but this would mean they cannot use the npm package as-is. They need to edit the template definition. A cleaner practice would be to use a decoration in their own project module to modify the template definition.
- Site parameters can be used. This approach has the advantage that the component can be configured to behave differently on each site in a multi-site project.

Checklist

Ready to publish to npm? Double check with this checklist.

	Item	Details
☐	Directory Name	<what it is>-magnolia
☐	magnolia-light-module	Keywords in package.json
☐	Description	In the README and the package.json
☐	README	Features, Usage, Demo
☐	README screenshots	
☐	Small	Your module provides one component, or a focussed set of functionality.
☐	Demo	
☐	Ready to use	npm package will be ready to be added to Magnoila resources directory.
☐	License file	
☐	Github	Source code is available. (Or other public source control repository)
☐	module.yaml	
☐	.npmignore	To exclude the `_dev` directory and other things not relevant to a production light module.
	Advanced	
☐	Buildable	Buildable with `npm run build`

Publish to npm

To publish your light module as a package to npm, run `npm publish`.

This interactive command will also help you setup a user on npm if you dont yet have one.

View your published package at `https://www.npmjs.com/package/<light module name>`!

Your package will be available via npm search shortly.

Let the world know

You are awesome. Thanks for contributing to the Magnolia community. Let everyone know by posting an announcement on the Magnolia forums. Mention @magnoila_cms in a tweet on twitter and we'll spread it to our followers. If you would like even more exposure consider writing a personal blog post about it. You could even contact us to discuss hosting it on the magnolia blogs.

Advanced frontend topics

npm dependencies, front-end builds, light projects?

See Sharing modules - Advanced Frontend Topics

Finding and using light modules

See Finding and using light modules