Writing npm Packages

To create a new npm package:

```
mkdir my-package
cd my-package/
meteor npm init
```

The last command creates a `package.json` file and prompts you for the package information. You may skip everything but `name`, `version`, and `entry point`. You can use the default `index.js` for `entry point`. This file is where you set your package's exports:

```
// my-package/index.js
exports.myPackageLog = function() {
  console.log("logged from my-package");
};
```

Now apps that include this package can do:

```
import { myPackageLog } from 'my-package'

myPackageLog(); // > "logged from my-package"
```

When choosing a name for your npm package, be sure to follow the npm guidelines.

Including in your app

When you are developing a new npm package for your app, there are a couple methods for including the package in your app:

- **Inside node_modules**: Place the package in your app's `node_modules/` directory, and add the package to source control. Do this when you want everything in a single repository.

```
cd my-app/node_modules/
mkdir my-package
cd my-package/
meteor npm init
git add -f ./ # or use a git submodule
```

- **npm link**: Place the package outside your app's directory in a separate repository and use `npm link`. Do this when you want to use the package in multiple apps.

```
cd ~/
mkdir my-package
cd my-package/
meteor npm init
cd ~/my-app/
meteor npm link ~/my-package
```

Other developers will also need to run the `npm link` command.

After either method, edit the `dependencies` attribute of `my-app/package.json`, adding `"my-package": "1.0.0"` (use the same version number you chose during `meteor npm init`).

Publishing your package

You can share your package with others by publishing it to the npm registry. While most packages are public, you can control who may view and use your package with private modules).

To publish publicly, follow these instructions. When you're done, anyone can add your package to their app with `npm install --save your-package`.

If you want to share packages during development, we recommend using the above methods instead of the registry. If you use the registry, then every time you change the package, you need to increment the version number, publish, and then `npm update my-package` inside your app.

Overriding packages with a local version

If you need to modify a package to do something that the published version doesn't do, you can edit a local version of the package on your computer.

Let's say you want to modify the `left-pad` npm package. If you haven't already, run inside your app directory:

```
meteor npm install --save left-pad
```

Now `left-pad` is included in your `package.json`, and the code has been downloaded to `node_modules/left_pad/`. Add the new directory to source control with:

```
git add -f node_modules/left_pad/
```

Now you can edit the package, commit, and push, and your teammates will get your version of the package. To ensure that your package doesn't get overwritten during an `npm update`, change the default caret version range in your `package.json` to an exact version.

Before:

```
"left-pad": "^1.0.2",
```

After:

```
"left-pad": "1.0.2",
```

An alternative method is maintaining a separate repository for the package and changing the `package.json` version number to a git URL or tarball, but every time you edit the separate repo, you'll need to commit, push, and `npm update left-pad`.