Compiling Assets (Mix)

Introduction

Laravel Mix provides a fluent API for defining Webpack build steps for your Laravel application using several common CSS and JavaScript pre-processors. Through simple method chaining, you can fluently define your asset pipeline. For example:

```
mix.js('resources/js/app.js', 'public/js')
    .sass('resources/sass/app.scss', 'public/css');
```

If you've ever been confused and overwhelmed about getting started with Webpack and asset compilation, you will love Laravel Mix. However, you are not required to use it while developing your application; you are free to use any asset pipeline tool you wish, or even none at all.

Installation & Setup

Installing Node

Before triggering Mix, you must first ensure that Node.js and NPM are installed on your machine.

```
node -v
npm -v
```

By default, Laravel Homestead includes everything you need; however, if you aren't using Vagrant, then you can easily install the latest version of Node and NPM using simple graphical installers from their download page.

Laravel Mix

The only remaining step is to install Laravel Mix. Within a fresh installation of Laravel, you'll find a `package.json` file in the root of your directory structure. The default `package.json` file includes everything you need to get started. Think of this like your `composer.json` file, except it defines Node dependencies instead of PHP. You may install the dependencies it references by running:

```
npm install
```

Running Mix

Mix is a configuration layer on top of Webpack, so to run your Mix tasks you only need to execute one of the NPM scripts that is included with the default Laravel `package.json` file:

```
// Run all Mix tasks...
npm run dev
```

```
// Run all Mix tasks and minify output...
npm run production
```

Watching Assets For Changes

The `npm run watch` command will continue running in your terminal and watch all relevant files for changes. Webpack will then automatically recompile your assets when it detects a change:

```
npm run watch
```

You may find that in certain environments Webpack isn't updating when your files change. If this is the case on your system, consider using the `watch-poll` command:

```
npm run watch-poll
```

Working With Stylesheets

The `webpack.mix.js` file is your entry point for all asset compilation. Think of it as a light configuration wrapper around Webpack. Mix tasks can be chained together to define exactly how your assets should be compiled.

Less

The `less` method may be used to compile Less into CSS. Let's compile our primary `app.less` file to `public/css/app.css`.

```
mix.less('resources/less/app.less', 'public/css');
```

Multiple calls to the `less` method may be used to compile multiple files:

```
mix.less('resources/less/app.less', 'public/css')
    .less('resources/less/admin.less', 'public/css');
```

If you wish to customize the file name of the compiled CSS, you may pass a full file path as the second argument to the `less` method:

```
mix.less('resources/less/app.less', 'public/stylesheets/styles.css');
```

If you need to override the underlying Less plug-in options, you may pass an object as the third argument to `mix.less()`:

```
mix.less('resources/less/app.less', 'public/css', {
    strictMath: true
});
```

Sass

The `sass` method allows you to compile Sass into CSS. You may use the method like so:

```
mix.sass('resources/sass/app.scss', 'public/css');
```

Again, like the `less` method, you may compile multiple Sass files into their own respective CSS files and even customize the output directory of the resulting CSS:

```
mix.sass('resources/sass/app.sass', 'public/css')
    .sass('resources/sass/admin.sass', 'public/css/admin');
```

Additional Node-Sass plug-in options may be provided as the third argument:

```
mix.sass('resources/sass/app.sass', 'public/css', {
    precision: 5
});
```

Stylus

Similar to Less and Sass, the `stylus` method allows you to compile Stylus into CSS:

```
mix.stylus('resources/stylus/app.styl', 'public/css');
```

You may also install additional Stylus plug-ins, such as Rupture. First, install the plug-in in question through NPM (`npm install rupture`) and then require it in your call to `mix.stylus()`:

```
mix.stylus('resources/stylus/app.styl', 'public/css', {
    use: [
        require('rupture')()
    ]
});
```

PostCSS

PostCSS, a powerful tool for transforming your CSS, is included with Laravel Mix out of the box. By default, Mix leverages the popular Autoprefixer plug-in to automatically apply all necessary CSS3 vendor prefixes. However, you're free to add any additional plug-ins that are appropriate for your application. First, install the desired plug-in through NPM and then reference it in your `webpack.mix.js` file:

```
mix.sass('resources/sass/app.scss', 'public/css')
    .options({
        postCss: [
            require('postcss-css-variables')()
        ]
    });
```

Plain CSS

If you would just like to concatenate some plain CSS stylesheets into a single file, you may use the `styles` method.

```
mix.styles([
    'public/css/vendor/normalize.css',
    'public/css/vendor/videojs.css'
], 'public/css/all.css');
```

URL Processing

Because Laravel Mix is built on top of Webpack, it's important to understand a few Webpack concepts. For CSS compilation, Webpack will rewrite and optimize any `url()` calls within your stylesheets. While this might initially sound strange, it's an incredibly powerful piece of functionality. Imagine that we want to compile Sass that includes a relative URL to an image:

```
.example {
    background: url('../images/example.png');
}
```

Absolute paths for any given `url()` will be excluded from URL-rewriting. For example, `url('/images/thing.png')` or `url('http://example.com/images/thing.png')` won't be modified.

By default, Laravel Mix and Webpack will find `example.png`, copy it to your `public/images` folder, and then rewrite the `url()` within your generated stylesheet. As such, your compiled CSS will be:

```
.example {
    background: url(/images/example.png?d41d8cd98f00b204e9800998ecf8427e);
}
```

As useful as this feature may be, it's possible that your existing folder structure is already configured in a way you like. If this is the case, you may disable `url()` rewriting like so:

```
mix.sass('resources/app/app.scss', 'public/css')
    .options({
        processCssUrls: false
    });
```

With this addition to your `webpack.mix.js` file, Mix will no longer match any `url()` or copy assets to your public directory. In other words, the compiled CSS will look just like how you originally typed it:

```
.example {
    background: url("../images/thing.png");
}
```

Source Maps

Though disabled by default, source maps may be activated by calling the `mix.sourceMaps()` method in your `webpack.mix.js` file. Though it comes with a compile/performance cost, this will provide extra debugging information to your browser's developer tools when using compiled assets.

```
mix.js('resources/js/app.js', 'public/js')
    .sourceMaps();
```

Style Of Source Mapping

Webpack offers a variety of source mapping styles. By default, Mix's source mapping style is set to `eval-source-map`, which provides a fast rebuild time. If you want to change the mapping style, you may do so using the `sourceMaps` method:

```
let productionSourceMaps = false;

mix.js('resources/js/app.js', 'public/js')
    .sourceMaps(productionSourceMaps, 'source-map');
```

Working With JavaScript

Mix provides several features to help you work with your JavaScript files, such as compiling ECMAScript 2015, module bundling, minification, and concatenating plain JavaScript files. Even better, this all works seamlessly, without requiring an ounce of custom configuration:

```
mix.js('resources/js/app.js', 'public/js');
```

With this single line of code, you may now take advantage of:

- ES2015 syntax.
- Modules
- Compilation of `.vue` files.
- Minification for production environments.

Vendor Extraction

One potential downside to bundling all application-specific JavaScript with your vendor libraries is that it makes long-term caching more difficult. For example, a single update to your application code will force the browser to re-download all of your vendor libraries even if they haven't changed.

If you intend to make frequent updates to your application's JavaScript, you should consider extracting all of your vendor libraries into their own file. This way, a change to your application code will not affect the caching of your large `vendor.js` file. Mix's `extract` method makes this a breeze:

```
mix.js('resources/js/app.js', 'public/js')
    .extract(['vue'])
```

The `extract` method accepts an array of all libraries or modules that you wish to extract into a `vendor.js` file. Using the above snippet as an example, Mix will generate the following files:

- `public/js/manifest.js`: *The Webpack manifest runtime*
- `public/js/vendor.js`: *Your vendor libraries*
- `public/js/app.js`: *Your application code*

To avoid JavaScript errors, be sure to load these files in the proper order:

```
<script src="/js/manifest.js"></script>
<script src="/js/vendor.js"></script>
<script src="/js/app.js"></script>
```

React

Mix can automatically install the Babel plug-ins necessary for React support. To get

started, replace your `mix.js()` call with `mix.react()`:

```
mix.react('resources/js/app.jsx', 'public/js');
```

Behind the scenes, Mix will download and include the appropriate `babel-preset-react` Babel plug-in.

Vanilla JS

Similar to combining stylesheets with `mix.styles()`, you may also combine and minify any number of JavaScript files with the `scripts()` method:

```
mix.scripts([
    'public/js/admin.js',
    'public/js/dashboard.js'
], 'public/js/all.js');
```

This option is particularly useful for legacy projects where you don't require Webpack compilation for your JavaScript.

A slight variation of `mix.scripts()` is `mix.babel()`. Its method signature is identical to `scripts`; however, the concatenated file will receive Babel compilation, which translates any ES2015 code to vanilla JavaScript that all browsers will understand.

Custom Webpack Configuration

Behind the scenes, Laravel Mix references a pre-configured `webpack.config.js` file to get you up and running as quickly as possible. Occasionally, you may need to manually modify this file. You might have a special loader or plug-in that needs to be referenced, or maybe you prefer to use Stylus instead of Sass. In such instances, you have two choices:

Merging Custom Configuration

Mix provides a useful `webpackConfig` method that allows you to merge any short Webpack configuration overrides. This is a particularly appealing choice, as it doesn't require you to copy and maintain your own copy of the `webpack.config.js` file. The `webpackConfig` method accepts an object, which should contain any Webpack-specific configuration that you wish to apply.

```
mix.webpackConfig({
    resolve: {
        modules: [
            path.resolve(__dirname, 'vendor/laravel/spark/resources/assets/js')
        ]
    }
});
```

Custom Configuration Files

If you would like to completely customize your Webpack configuration, copy the `node_modules/laravel-mix/setup/webpack.config.js` file to your project's root directory. Next, point all of the `--config` references in your `package.json` file to the newly copied configuration file. If you choose to take this approach to customization, any future upstream updates to Mix's `webpack.config.js` must be manually merged into your customized file.

Copying Files & Directories

The `copy` method may be used to copy files and directories to new locations. This can be useful when a particular asset within your `node_modules` directory needs to be relocated to your `public` folder.

```
mix.copy('node_modules/foo/bar.css', 'public/css/bar.css');
```

When copying a directory, the `copy` method will flatten the directory's structure. To maintain the directory's original structure, you should use the `copyDirectory` method instead:

```
mix.copyDirectory('resources/img', 'public/img');
```

Versioning / Cache Busting

Many developers suffix their compiled assets with a timestamp or unique token to force browsers to load the fresh assets instead of serving stale copies of the code. Mix can handle this for you using the `version` method.

The `version` method will automatically append a unique hash to the filenames of all compiled files, allowing for more convenient cache busting:

```
mix.js('resources/js/app.js', 'public/js')
    .version();
```

After generating the versioned file, you won't know the exact file name. So, you should use Laravel's global `mix` function within your views to load the appropriately hashed asset. The `mix` function will automatically determine the current name of the hashed file:

```
<script src="{{ mix('/js/app.js') }}"></script>
```

Because versioned files are usually unnecessary in development, you may instruct the versioning process to only run during `npm run production`:

```
mix.js('resources/js/app.js', 'public/js');

if (mix.inProduction()) {
    mix.version();
}
```

Browsersync Reloading

BrowserSync can automatically monitor your files for changes, and inject your changes into the browser without requiring a manual refresh. You may enable support by calling the `mix.browserSync()` method:

```
mix.browserSync('my-domain.test');

// Or...

// https://browsersync.io/docs/options
mix.browserSync({
    proxy: 'my-domain.test'
});
```

You may pass either a string (proxy) or object (BrowserSync settings) to this method. Next, start Webpack's dev server using the `npm run watch` command. Now, when you modify a script or PHP file, watch as the browser instantly refreshes the page to reflect your changes.

Environment Variables

You may inject environment variables into Mix by prefixing a key in your `.env` file with `MIX_`:

```
MIX_SENTRY_DSN_PUBLIC=http://example.com
```

After the variable has been defined in your `.env` file, you may access via the `process.env` object. If the value changes while you are running a `watch` task, you will need to restart the task:

```
process.env.MIX_SENTRY_DSN_PUBLIC
```

Notifications

When available, Mix will automatically display OS notifications for each bundle. This will give you instant feedback, as to whether the compilation was successful or not. However, there may be instances when you'd prefer to disable these notifications. One such example might be triggering Mix on your production server. Notifications may be deactivated, via the `disableNotifications` method.

```
mix.disableNotifications();
```

Become a Laravel Partner

Laravel Partners are elite shops providing top-notch Laravel development and consulting. Each of our partners can help you craft a beautiful, well-architected project.

Our Partners

Laravel

Highlights

- Release Notes
- Getting Started
- Routing
- Blade Templates
- Authentication
- Authorization
- Artisan Console
- Database

Resources

- Laracasts
- Laravel News
- Laracon
- Laracon EU
- Laracon AU
- Jobs
- Certification
- Forums

Partners

- Vehikl
- Tighten Co.
- Kirschbaum
- Byte 5
- 64Robots
- Cubet
- DevSquad
- Ideil

Ecosystem

- Vapor
- Forge
- Envoyer
- Horizon
- Lumen
- Nova
- Echo
- Valet

Laravel is a web application framework with expressive, elegant syntax. We believe development must be an enjoyable and creative experience to be truly fulfilling. Laravel attempts to take the pain out of development by easing common tasks used in most web projects.

Eloquent ORM
Testing

Cyber-Duck
ABOUT YOU
Become A Partner

Mix
Spark
Cashier
Homestead
Dusk
Passport
Scout
Socialite