webpack
MODULE BUNDLER

WELCOME

This small tutorial will guide you through a simple example.

You'll learn:

- How to install webpack
- How to use webpack
- How to use loaders
- How to use the development server

INSTALLING WEBPACK

You need to have node.js installed.

```
$ npm install webpack -g
```

> *This makes the webpack command available.*

SETUP THE COMPILATION

Start with a empty directory.

Create these files:

add `entry.js`

```
document.write("It works.");
```

add `index.html`

```
<html>
    <head>
        <meta charset="utf-8">
    </head>
    <body>
        <script type="text/javascript" src="bundle.js" charset="utf-8"></script>
    </body>
</html>
```

Then run the following:

```
$ webpack ./entry.js bundle.js
```

It will compile your file and create a bundle file.

If successful it displays something like this:

```
Version: webpack 1.12.11
Time: 51ms
    Asset     Size  Chunks             Chunk Names
bundle.js  1.42 kB       0  [emitted]  main
chunk    {0} bundle.js (main) 28 bytes [rendered]
    [0] ./tutorials/getting-started/setup-compilation/entry.js 28 bytes {0} [built]
```

Open `index.html` in your browser. It should display `It works.`

SECOND FILE

Next, we will move some code into an extra file.

add `content.js`

```
module.exports = "It works from content.js.";
```

update `entry.js`

```
- document.write("It works.");
+ document.write(require("./content.js"));
```

And recompile with:

```
$ webpack ./entry.js bundle.js
```

Update your browser window and you should see the text `It works from content.js.`.

It works from content.js.

> *webpack will analyze your entry file for dependencies to other files. These files (called modules) are included in your* `bundle.js` *too. webpack will give each module a unique id and save all modules accessible by id in the* `bundle.js` *file. Only the entry module is executed on startup. A small runtime provides the* `require` *function and executes the dependencies when required.*

THE FIRST LOADER

We want to add a css file to our application.

webpack can only handle JavaScript natively, so we need the `css-loader` to process CSS files. We also need the `style-loader` to apply the styles in the CSS file.

Run `npm install css-loader style-loader` to install the loaders. (They need to be installed locally, without `-g`) This will create a `node_modules` folder for you, in which the loaders will live.

Let's use them:

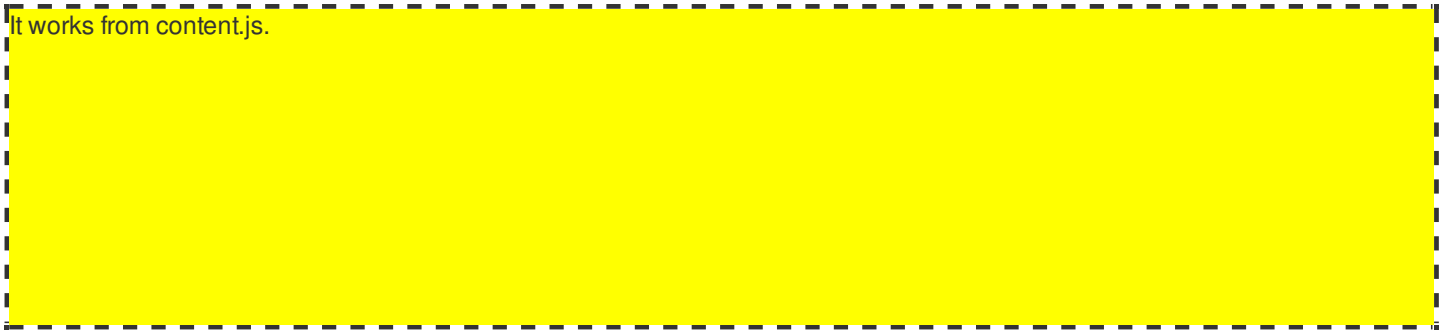
add `style.css`

```
body {
    background: yellow;
}
```

update `entry.js`

```
+ require("!style!css!./style.css");
  document.write(require("./content.js"));
```

Recompile and update your browser to see your application with yellow background.



> *By prefixing loaders to a module request, the module went through a loader pipeline. These loaders transform the file content in specific ways. After these transformations are applied, the result is a JavaScript module.*

BINDING LOADERS

We don't want to write such long requires `require("!style!css!./style.css");`.

We can bind file extensions to loaders so we just need to write: `require("./style.css")`

update `entry.js`

update entry.js

```
- require("!style!css!./style.css");
+ require("./style.css");
  document.write(require("./content.js"));
```

Run the compilation with:

```
webpack ./entry.js bundle.js --module-bind 'css=style!css'
```

> *Some environments may require double quotes: –module-bind “css=style!css”*

You should see the same result:

It works from content.js.

A CONFIG FILE

We want to move the config options into a config file:

add `webpack.config.js`

```
module.exports = {
    entry: "./entry.js",
    output: {
        path: __dirname,
        filename: "bundle.js"
    },
    module: {
        loaders: [
            { test: /\.css$/, loader: "style!css" }
        ]
    }
};
```

Now we can just run:

```
webpack
```

to compile:

```
Version: webpack 1.12.11
Time: 379ms
    Asset     Size  Chunks             Chunk Names
bundle.js  10.7 kB       0  [emitted]  main
chunk    {0} bundle.js (main) 8.86 kB [rendered]
    [0] ./tutorials/getting-started/config-file/entry.js 65 bytes {0} [built]
    [1] ./tutorials/getting-started/config-file/style.css 943 bytes {0} [built]
    [2] ../~/css-loader!./tutorials/getting-started/config-file/style.css 201 bytes {0} [built]
    [3] ../~/css-loader/lib/css-base.js 1.51 kB {0} [built]
    [4] ../~/style-loader/addStyles.js 6.09 kB {0} [built]
    [5] ./tutorials/getting-started/config-file/content.js 45 bytes {0} [built]
```

> *The webpack command-line will try to load the file `webpack.config.js` in the current directory.*

A PRETTIER OUTPUT

A PRETTIER OUTPUT

If the project grows the compilation may take a bit longer. So we want to display some kind of progress bar. And we want colors…

We can achieve this with

```
webpack --progress --colors
```

WATCH MODE

We don’t want to manually recompile after every change…

```
webpack --progress --colors --watch
```

Webpack can cache unchanged modules and output files between compilations.

> *When using watch mode, webpack installs file watchers to all files, which were used in the compilation process. If any change is detected, it’ll run the compilation again. When caching is enabled, webpack keeps each module in memory and will reuse it if it isn’t changed.*

DEVELOPMENT SERVER

The development server is even better.

```
npm install webpack-dev-server -g
```

```
webpack-dev-server --progress --colors
```

This binds a small express server on localhost:8080 which serves your static assets as well as the bundle (compiled automatically). It automatically updates the browser page when a bundle is recompiled (SockJS). Open http://localhost:8080/webpack-dev-server/bundle in your browser.

> *The dev server uses webpack’s watch mode. It also prevents webpack from emitting the resulting files to disk. Instead it keeps and serves the resulting files from memory.*