# Tutorial to bundle CrossEcore and CrossEcore generated code with Webpack

> Disclaimer: The resulting code here is somehow not optimal. To use it in production, you might want a minified, production optimized code. This guide here is to show only the general concepts of the tooling. For more information regarding optimization you might want to have a look at the documentation of Webpack or at this guide that I found useful.

To bundle JS code, there are a couple of different bundle types you'll find information about here and here. However, CrossEcore is only delivered in the AMD module type for which you need a module loader and which is not state-of-the-art anymore and makes bundling difficult. Therefore, I decided to bundle it on my own until an ES6 implementation of CrossEcore will be released. The generated code by the CrossEcore generator is also included here. For this bundling process I use Webpack.

## 1. Setting Up Webpack

Setting up Webpack is quite easy, you just have to install it via npm using this command:

`npm install webpack webpack-cli --save-dev`

Webpack can be executed just be executing the command `npx webpack`.

## 2. Entry and Output

At first we need to specify the entrypoint to our code. It is the script that runs at the beginning when the website is called and can (if a module shall be bundled) e.g. consist of exports. The output is just the output file + directory.

The following file is a basic config file that includes the config described here in this section. It should be named `webpack.config.js` and should be located in the root directory of the project. Webpack can now be executed using this config file with:

`npx webpack --config webpack.config.js`

```
const path = require('path');

module.exports = {
  entry: './index.js',
  mode: 'development',
  output: {
    filename: 'crom.js',
    path: path.resolve(__dirname, 'output'),
  },
  devtool: 'inline-source-map'
};
```

## 3. Loading TypeScript

One of the main advantages of Webpack (e.g. over RollUp, which is another bundler) is the `ts-loader`. It automatically loads TypeScript, transpiles it to JavaScript and bundles it. This makes Webpack (configured correctly) more or less a one-stop-shop. The `ts-loader` has to be included and configured to be used.

**Configuration**

The Loader has to be included as a module. To bundle CrossEcore AND our generated code, we have to include the `node_modules` directory as well as we have to toggle `allowTsInNodeModules` so that the TS code of CrossEcore is used (and not the AMD bundled version).

```
const path = require('path');

module.exports = {
  entry: './index.ts',
  mode: 'development',
  output: {
    filename: 'crom.js',
    path: path.resolve(__dirname, 'output'),
  },
  devtool: 'inline-source-map',
  module: {
    rules: [
      {
        loader: 'ts-loader',
        //exclude: /node_modules/,
        options: { allowTsInNodeModules: true }
      },
    ],
  },
  resolve: {
    extensions: [ '.tsx', '.ts', '.js' ],
  },
};
```

## 4. Translate TypeScript Path Mappings

In the `tsconfig.json`, CrossEcore (and we as well) defined some path mappings, that TypeScript uses to ensure easily readable absolute paths. However, JavaScript cannot handle those. Therefore we have to include a plugin into Webpack that automatically translates these paths into relative JavaScript paths. Please remember that you have to install the plugin via

`npm install tsconfig-paths-webpack-plugin`

The complete including the plugin `webpack.config.js` looks therefore like the following:

```
const path = require('path');
const TsconfigPathsPlugin = require('tsconfig-paths-webpack-plugin');

module.exports = {
  entry: './index.ts',
  mode: 'development',
  output: {
    filename: 'crom.js',
    path: path.resolve(__dirname, 'output'),
  },
  devtool: 'inline-source-map',
  module: {
    rules: [
      {
        loader: 'ts-loader',
        //exclude: /node_modules/,
        options: { allowTsInNodeModules: true }
      },
    ],
  },
  resolve: {
    extensions: [ '.tsx', '.ts', '.js' ],
    plugins: [new TsconfigPathsPlugin({ configFile: "./tsconfig.json" })]
  },
};
```

## 5. Fix CrossEcore Compiler Errors

Since CrossEcore has some compiler errors, we have to fix them on our own. Some CrossEcore files import interfaces that they are defined and exported in the same file. Of course, that's not how it works. We have to delete those imports by hand.

The errors are all similar to the errors in the extract of the `EClass.ts` (but you should scan all the CrossEcore files)

```
{...}
import {EClass} from "./EClass";
{...}

export interface EClass
extends EClassifier

{...}
```

## 6. Finishing

Afterwards, the configuration is finished. We just execute

`npx webpack --config webpack.config.js`

Then, the bundling process will run. To test everything, just add some code to the `index.ts` (e.g. instantiating CrossEcore-objects and logging them to the console), write a minimal `index.html`, add the bundled file in a `<script>`-tag and open the `index.html` with a web browser.