# Babel

## What is Babel?

Babel is a highly configurable JavaScript compiler and transpiler. It is used to convert a preferred programming language into compatible ES5 JavaScript. Commonly, this is used to convert ES6 into ES5 in order to make our code compatible with a wider range of browsers and environments.

You can find the documentation for Babel here: <https://babeljs.io/docs/en/>

### Using Babel

Babel is often integrated into web frameworks such as React, but it is possible to use Babel in a standalone manner.

### Babel REPL

Prototyping with Babel is easy using the Babel online REPL. This allows you to experiment with Babel without installing it.

It can be found here: <https://babeljs.io/repl>

### Using Babel Locally

#### Installation

Adding Babel to an existing project is straightforward. Install Babel CLI using the following command:

npm install --save-dev @babel/core @babel/cli

#### Usage

Add the **build** command to the scripts section of **package.json:**

{

"name": "babel-test",

"version": "1.0.0",

"scripts": {

"build": "babel src -d lib"

},

"devDependencies": {

"@babel/cli": "^7.2.3",

"@babel/core": "^7.4.0"

}

}

Now you can run the following in your terminal. This will process the contents of the **src/** directory and place the result in the **lib/** directory.

npm run build

#### Configuration

If you run the build now it won’t do much. We need to add a configuration to tell Babel what to do. The easiest thing we can do is install the **env preset** which will enable transforms for ES6+. Install this like so:

npm install @babel/preset-env --save-dev

Next, create a file names **.babelrc** in the root of your project with the following content to tell Babel to use the preset:

{

"presets": [

"@babel/preset-env"

]

}

Now with Babel configured running **npm run build** will correctly convert any **.js** files from ES6+ syntax to compatible ES5.

## Babel in Action

Using the setup previously detailed, we can code in ES6 and transform it into ES5 JavaScript. The following examples show some examples of what might be changed.

// Defining variables

const array = ['a', 'b', 'c'];

// Destructuring

const [first, ...rest] = array;

// Arrow Functions

const myFunc = () => 'example';

Input ES6

"use strict";

// Defining variables

var array = ["a", "b", "c"];

// Destructuring

var first = array[0],

rest = array.slice(1);

// Arrow Functions

var myFunc = function myFunc() {

return "example";

};

Output ES5

This example highlights some of the simpler changes that Babel does. Common features like const, let, destructuring, and arrow functions are changed into ES5 compatible equivalents.

While this example is very simple it’s clear that the ES5 output is more verbose and therefore takes up more space. This problem is compounded when using the more complex structures in React, so expect the output of Babel to contain larger files than what is input. A quick demonstration of this would be difference in size of ES6 classes and the ES5 equivalent.

## Babel in React

JSX is not natively supported in browsers, so in addition to what was previously demonstrated React uses Babel to transform JSX into ES5 JavaScript. It achieves this using the **React preset**. This enables us to transform this:

const TestComp = () => (<div>Hello, world!</div>);

Into this ES5 equivalent:

var TestComp = function TestComp() {

return React.createElement("div", null, "Hello, world!");

};

If you’re using React this process is for the most part abstracted away from the user. React uses Babel to automatically convert your code to compatible ES5 during builds. This however doesn’t mean you should not be aware of this process as bugs, although rarely, sometimes present themselves only after the code has been through the conversion process.

Occasionally you may see an error message that includes ‘Babel’ it is important that when this occurs that you understand the function of Babel and can decide for yourselves where you need to investigate further.