
Moved

This package has moved and is now available at [@rollup/plugin-babel](https://github.com/rollup/plugin-babel). Please update your dependencies. This repository is no longer maintained.

rollup-plugin-babel

Seamless integration between Rollup and Babel.

Why?

If you're using Babel to transpile your ES6/7 code and Rollup to generate a standalone bundle, you have a couple of options:

- run the code through Babel first, being careful to exclude the module transformer, or
- run the code through Rollup first, and *then* pass it to Babel.

Both approaches have disadvantages – in the first case, on top of the additional configuration complexity, you may end up with Babel's helpers (like `classCallCheck`) repeated throughout your code (once for each module where the helpers are used). In the second case, transpiling is likely to be slower, because transpiling a large bundle is much more work for Babel than transpiling a set of small files.

Either way, you have to worry about a place to put the intermediate files, and getting sourcemaps to behave becomes a royal pain.

Using Rollup with rollup-plugin-babel makes the process far easier.

Installation

babel 7.x

```
1 npm install --save-dev rollup-plugin-babel@latest
```

babel 6.x

```
1 npm install --save-dev rollup-plugin-babel@3
```

Usage

Command Line (rollup)

Configuration `rollup.config.js` (docs):

```
1 import babel from 'rollup-plugin-babel';
2 import pkg from './package.json';
3
4 const config = {
5   input: 'src/index.js',
6   output: [
7     {
8       file: pkg.module,
9       format: 'esm',
10    },
11  ],
12  plugins: [babel()],
13 };
14
15 export default config;
```

Programmatic

```
1 import { rollup } from 'rollup';
2 import babel from 'rollup-plugin-babel';
3
4 rollup({
5   input: 'main.js',
6   plugins: [
7     babel({
8       exclude: 'node_modules/**'
9     })
10  ]
11 }).then(...)
```

All options are as per the Babel documentation, plus the following:

- `options.externalHelpers`: a boolean value indicating whether to bundle in the Babel helpers
- `options.include` and `options.exclude`: each a minimatch pattern, or array of minimatch patterns, which determine which files are transpiled by Babel (by default, all files are transpiled)
- `options.externalHelpersWhitelist`: an array which gives explicit control over which babelHelper functions are allowed in the bundle (by default, every helper is allowed)

-
- `options.extensions`: an array of file extensions that Babel should transpile (by default the Babel defaults of `.js`, `.jsx`, `.es6`, `.es`, `.mjs` are used)

Babel will respect `.babelrc` files – this is generally the best place to put your configuration.

External dependencies

Ideally, you should only be transforming your source code, rather than running all of your external dependencies through Babel – hence the `exclude: 'node_modules/**'` in the example above. If you have a dependency that exposes untranspiled ES6 source code that doesn't run in your target environment, then you may need to break this rule, but it often causes problems with unusual `.babelrc` files or mismatched versions of Babel.

We encourage library authors not to distribute code that uses untranspiled ES6 features (other than modules) for this reason. Consumers of your library should *not* have to transpile your ES6 code, any more than they should have to transpile your CoffeeScript, ClojureScript or TypeScript.

Use `babelrc: false` to prevent Babel from using local (i.e. to your external dependencies) `.babelrc` files, relying instead on the configuration you pass in.

Helpers

In some cases Babel uses *helpers* to avoid repeating chunks of code – for example, if you use the `class` keyword, it will use a `classCallCheck` function to ensure that the class is instantiated correctly.

By default, those helpers will be inserted at the top of the file being transformed, which can lead to duplication. This rollup plugin automatically deduplicates those helpers, keeping only one copy of each one used in the output bundle. Rollup will combine the helpers in a single block at the top of your bundle. To achieve the same in Babel 6 you must use the `external-helpers` plugin.

Alternatively, if you know what you're doing, you can use the `transform-runtime` plugin. If you do this, use `runtimeHelpers: true`:

```
1 rollup.rollup({
2   ...,
3   plugins: [
4     babel({ runtimeHelpers: true })
5   ]
6 }).then(...)
```

By default `externalHelpers` option is set to `false` so babel helpers will be included in your bundle.

If you do not wish the babel helpers to be included in your bundle at all (but instead reference the global `babelHelpers` object), you may set the `externalHelpers` option to **true**:

```
1 rollup.rollup({
2   ...,
3   plugins: [
4     babel({
5       plugins: ['external-helpers'],
6       externalHelpers: true
7     })
8   ]
9 }).then(...)
```

Modules

This is not needed for Babel 7 - it knows automatically that Rollup understands ES modules & that it shouldn't use any module transform with it. The section below describes what needs to be done for Babel 6.

The `env` preset includes the `transform-es2015-modules-commonjs` plugin, which converts ES6 modules to CommonJS – preventing Rollup from working. Since Babel 6.3 it's possible to deactivate module transformation with `"modules": false`. So there is no need to use the old workaround with `babel-preset-es2015-rollup`, that will work for Babel <6.13. Rollup will throw an error if this is incorrectly configured.

However, setting `modules: false` in your `.babelrc` may conflict if you are using `babel-register`. To work around this, specify `babelrc: false` in your rollup config. This allows Rollup to bypass your `.babelrc` file. In order to use the `env` preset, you will also need to specify it with `modules: false` option:

```
1 plugins: [
2   babel({
3     babelrc: false,
4     presets: [['env', { modules: false }]],
5   }),
6 ];
```

Configuring Babel 6

The following applies to Babel 6 only. If you're using Babel 5, do `npm i -D rollup-plugin-babel@1`, as version 2 and above no longer supports Babel 5

```
1 npm install --save-dev rollup-plugin-babel@3 babel-preset-env babel-plugin-external-helpers
```

```
1 // .babelrc
2 {
3   "presets": [
4     [
5       "env",
6       {
7         "modules": false
8       }
9     ]
10  ],
11  "plugins": [
12    "external-helpers"
13  ]
14 }
```

Custom plugin builder

`rollup-plugin-babel` exposes a plugin-builder utility that allows users to add custom handling of Babel's configuration for each file that it processes.

`.custom` accepts a callback that will be called with the loader's instance of `babel` so that tooling can ensure that it using exactly the same `@babel/core` instance as the loader itself.

It's main purpose is to allow other tools for configuration of transpilation without forcing people to add extra configuration but still allow for using their own `babelrc` / `babel` config files.

Example

```
1 import babel from 'rollup-plugin-babel';
2
3 export default babel.custom(babelCore => {
4   function myPlugin() {
5     return {
6       visitor: {},
7     };
8   }
9
10  return {
11    // Passed the plugin options.
12    options({ opt1, opt2, ...pluginOptions }) {
13      return {
14        // Pull out any custom options that the plugin might
15        // have.
16        customOptions: { opt1, opt2 },
17      };
18    }
19  };
20 }
```

```
17         // Pass the options back with the two custom options
18         removed.
19         pluginOptions,
20     },
21
22     config(cfg /* Passed Babel's 'PartialConfig' object. */, { code
23     , customOptions }) {
24         if (cfg.hasFilesystemConfig()) {
25             // Use the normal config
26             return cfg.options;
27         }
28
29         return {
30             ...cfg.options,
31             plugins: [
32                 ...(cfg.options.plugins || []),
33
34                 // Include a custom plugin in the options.
35                 myPlugin,
36             ],
37         };
38
39     result(result, { code, customOptions, config, transformOptions
40     }) {
41         return {
42             ...result,
43             code: result.code + '\n// Generated by some custom
44             plugin',
45         };
46     },
47
48     });
49 });
```

License

MIT