Env preset

Babel preset that automatically determines the Babel plugins you need based on your supported environments. Uses compat-table

GitHub (https://github.com/babel/babel-preset-env) · npm (https://www.npmjs.com/package/babel-preset-env) · Edit this page (https://github.com/babel/babel-preset-env/blob/master/README.md)

Shell Copy

```
npm install babel-preset-env --save-dev
```

Without any configuration options, babel-preset-env behaves exactly the same as babel-preset-latest (or babel-preset-es2015, babel-preset-es2016, and babel-preset-es2017 together).

JSON Copy

```
{
  "presets": ["env"]
}
```

You can also configure it to only include the polyfills and transforms needed for the browsers you support. Compiling only what's needed can make your bundles smaller and your life easier.

This example only includes the polyfills and code transforms needed for the last two versions of each browser, and versions of Safari greater than or equal to 7. We use browserslist (https://github.com/ai/browserslist) to parse this information, so you can use any valid query format supported by browserslist (https://github.com/ai/browserslist#queries).

JSON Copy

```
{
  "presets": [
    ["env", {
      "targets": {
        "browsers": ["last 2 versions", "safari >= 7"]
      }
    }]
  ]
}
```

Similarly, if you're targeting Node.js instead of the browser, you can configure babel-preset-env to only include the polyfills and transforms necessary for a particular version:

JSON Copy

```
{
  "presets": [
    ["env", {
      "targets": {
        "node": "6.10"
      }
    }]
  ]
}
```

For convenience, you can use `"node": "current"` to only include the necessary polyfills and transforms for the Node.js version that you use to run Babel:

JSON Copy

```
{
  "presets": [
    ["env", {
      "targets": {
        "node": "current"
      }
    }]
  ]
}
```

Check out the many options (especially `useBuiltIns` to polyfill less)!

- How it Works
- Install
- Usage
- Options
- Examples
- Caveats
- Other Cool Projects

How it Works

Determine environment support for ECMAScript features

Use external data such as `compat-table` (https://github.com/kangax/compat-table) to determine browser support. (We should create PRs there when necessary)

Feature name	Current browser	Compilers/polyfills																																	
	99%	56%	71%	25%	44%	18%	59%	18%	3%	11%	61%	83%	93%	34%	64%	67%	69%	70%	71%	74%	86%	88%	89%	91%	92%	92%	94%	94%	4%	25%	25%	37%	
		Traceur	Babel + core-js[2]	ES6 Trans-piler	Closure	JSX[3]	Type-Script + core-js	es6-shim	IE 10	IE 11	Edge 12[4]	Edge 13[4]	Edge 14[4]	FF 31 ESR	FF 38 ESR	FF 40	FF 41	FF 42	FF 43	FF 44	FF 45 ESR	FF 46	FF 47	FF 48	FF 49	FF 50 Beta	FF 51 Aurora	FF 52 Nightly	CH <19	CH 39, OP 26[1]	CH 40, OP 27[1]	CH 41, OP 28[1]	
Optimisation																																			
proper tail calls (tail call optimisation)	2/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	
Syntax																																			
default function parameters	7/7	4/7	4/7	4/7	4/7	0/7	5/7	0/7	0/7	0/7	0/7	0/7	7/7	3/7	3/7	3/7	3/7	3/7	4/7	4/7	4/7	4/7	4/7	4/7	4/7	4/7	6/7	6/7	0/7	0/7	0/7	0/7	
rest parameters	5/5	4/5	3/5	2/5	2/5	3/5	4/5	0/5	0/5	0/5	5/5	5/5	5/5	3/5	4/5	4/5	4/5	4/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	0/5	0/5	0/5	0/5	
spread (...) operator	15/15	15/15	13/15	8/15	12/15	2/15	4/15	0/15	0/15	0/15	12/15	15/15	15/15	11/15	15/15	15/15	15/15	15/15	15/15	15/15	15/15	15/15	15/15	15/15	15/15	15/15	15/15	15/15	0/15	0/15	0/15	0/15	
object literal extensions	6/6	6/6	6/6	6/6	4/6	5/6	6/6	0/6	0/6	0/6	6/6	6/6	6/6	0/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	0/6	0/6	0/6	0/6	
for..of loops	9/9	9/9	9/9	4/9	6/9	2/9	3/9	0/9	0/9	0/9	6/9	7/9	7/9	5/9	7/9	7/9	7/9	7/9	7/9	7/9	7/9	7/9	7/9	7/9	7/9	7/9	7/9	7/9	0/9	7/9	7/9	7/9	
octal and binary literals	4/4	2/4	4/4	2/4	4/4	0/4	4/4	2/4	0/4	0/4	4/4	4/4	4/4	2/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	0/4	0/4	0/4	4/4	
template literals	5/5	4/5	4/5	3/5	3/5	4/5	3/5	0/5	0/5	0/5	4/5	5/5	5/5	0/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	0/5	0/5	0/5	5/5	
RegExp "y" and "u" flags	5/5	3/5	3/5	0/5	0/5	0/5	0/5	0/5	0/5	0/5	2/5	5/5	5/5	2/5	2/5	2/5	2/5	2/5	2/5	2/5	2/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	0/5	0/5	0/5	0/5	
destructuring, declarations	22/22	20/22	21/22	14/22	18/22	12/22	15/22	0/22	0/22	0/22	0/22	0/22	21/22	13/22	19/22	19/22	19/22	19/22	19/22	19/22	19/22	19/22	21/22	21/22	21/22	21/22	21/22	21/22	0/22	0/22	0/22	0/22	
destructuring, assignment	24/24	23/24	24/24	17/24	16/24	11/24	19/24	0/24	0/24	0/24	0/24	0/24	23/24	14/24	20/24	20/24	21/24	21/24	21/24	21/24	21/24	21/24	23/24	23/24	23/24	23/24	23/24	23/24	0/24	0/24	0/24	0/24	
destructuring, parameters	23/23	19/23	20/23	15/23	17/23	12/23	15/23	0/23	0/23	0/23	0/23	0/23	22/23	12/23	18/23	18/23	18/23	18/23	18/23	18/23	18/23	18/23	19/23	19/23	19/23	19/23	20/23	20/23	0/23	0/23	0/23	0/23	
Unicode code point escapes	2/2	1/2	1/2	1/2	1/2	0/2	1/2	0/2	0/2	0/2	2/2	2/2	2/2	0/2	0/2	1/2	1/2	1/2	1/2	1/2	1/2	1/2	1/2	1/2	1/2	1/2	1/2	1/2	0/2	0/2	0/2	0/2	
new.target	2/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	1/2	2/2	0/2	0/2	0/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	0/2	0/2	0/2	0/2	
Bindings																																			
const	16/16	14/16	14/16	10/16	14/16	0/16	14/16	0/16	0/16	12/16	12/16	12/16	16/16	3/16	12/16	12/16	12/16	12/16	12/16	12/16	12/16	12/16	12/16	12/16	12/16	12/16	16/16	16/16	1/16	1/16	1/16	9/16	
let	12/12	10/12	10/12	8/12	10/12	0/12	10/12	0/12	0/12	10/12	10/12	10/12	12/12	0/12	0/12	0/12	0/12	0/12	0/12	10/12	10/12	10/12	10/12	10/12	10/12	10/12	12/12	12/12	0/12	0/12	0/12	6/12	
block-level function declaration[1][3]	Yes	Yes	Yes	No	Yes	No	No	No	No	Yes	Yes	Yes	Yes	No	No	No	No	No	No	No	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	Flag	Flag	Yes	
Functions																																			
arrow functions	13/13	11/13	9/13	8/13	10/13	8/13	9/13	0/13	0/13	0/13	8/13	13/13	13/13	8/13	8/13	9/13	10/13	10/13	11/13	11/13	13/13	13/13	13/13	13/13	13/13	13/13	13/13	13/13	0/13	0/13	0/13	0/13	
class	24/24	17/24	19/24	17/24	13/24	16/24	19/24	0/24	0/24	0/24	0/24	24/24	24/24	0/24	0/24	0/24	0/24	0/24	0/24	0/24	24/24	24/24	24/24	24/24	24/24	24/24	24/24	24/24	0/24	0/24	0/24	0/24	
super	8/8	7/8	4/8	7/8	5/8	7/8	7/8	0/8	0/8	0/8	0/8	8/8	8/8	0/8	0/8	0/8	0/8	0/8	0/8	0/8	8/8	8/8	8/8	8/8	8/8	8/8	8/8	8/8	0/8	0/8	0/8	0/8	
generators	27/27	24/27	24/27	0/27	16/27	0/27	0/27	0/27	0/27	0/27	0/27	27/27	27/27	14/27	20/27	20/27	20/27	20/27	21/27	21/27	25/27	25/27	25/27	25/27	25/27	25/27	25/27	25/27	0/27	16/27	16/27	16/27	

We can periodically run build-data.js (https://github.com/babel/babel-preset-env/blob/master/scripts/build-data.js) which generates plugins.json (https://github.com/babel/babel-preset-env/blob/master/data/plugins.json).

Ref: #7 (https://github.com/babel/babel-preset-env/issues/7)

Maintain a mapping between JavaScript features and Babel plugins

> Currently located at plugin-features.js (https://github.com/babel/babel-preset-env/blob/master/data/plugin-features.js).

This should be straightforward to do in most cases. There might be cases where plugins should be split up more or certain plugins aren't standalone enough (or impossible to do).

Support all plugins in Babel that are considered latest

> Default behavior without options is the same as `babel-preset-latest`.

It won't include `stage-x` plugins. env will support all plugins in what we consider the latest version of JavaScript (by matching what we do in `babel-preset-latest` (http://babeljs.io/docs/plugins/preset-latest/)).

Ref: #14 (https://github.com/babel/babel-preset-env/issues/14)

Determine the lowest common denominator of plugins to be included in the preset

If you are targeting IE 8 and Chrome 55 it will include all plugins required by IE 8 since you would need to support both still.

Support a target option "node": "current" to compile for the currently running node version.

For example, if you are building on Node 6, arrow functions won't be converted, but they will if you build on Node 0.12.

Support a browsers option like autoprefixer

Use browserslist (https://github.com/ai/browserslist) to declare supported environments by performing queries like `> 1%, last 2 versions`.

Ref: #19 (https://github.com/babel/babel-preset-env/pull/19)

Install

With npm (https://www.npmjs.com):

Shell Copy

```shell
npm install --save-dev babel-preset-env
```

Or yarn (https://yarnpkg.com):

Shell Copy

```
yarn add babel-preset-env --dev
```

Usage

The default behavior without options runs all transforms (behaves the same as babel-preset-latest (https://babeljs.io/docs/plugins/preset-latest/)).

JSON Copy

```
{
  "presets": ["env"]
}
```

Options

For more information on setting options for a preset, refer to the plugin/preset options (http://babeljs.io/docs/plugins/#plugin-preset-options) documentation.

targets

`{ [string]: number | string }`, defaults to `{}`.

Takes an object of environment versions to support.

Each target environment takes a number or a string (we recommend using a string when specifying minor versions like `node: "6.10"`).

Example environments: `chrome`, `opera`, `edge`, `firefox`, `safari`, `ie`, `ios`, `android`, `node`, `electron`.

The data (https://github.com/babel/babel-preset-env/blob/master/data/plugins.json) for this is generated by running the build-data script (https://github.com/babel/babel-preset-env/blob/master/scripts/build-data.js) which pulls in data from compat-table (https://kangax.github.io/compat-table).

targets.node

`number | string | "current" | true`

If you want to compile against the current node version, you can specify `"node": true` or `"node": "current"`, which would be the same as `"node": process.versions.node`.

targets.browsers

`Array<string> | string`

A query to select browsers (ex: last 2 versions, > 5%) using browserslist (https://github.com/ai/browserslist).

Note, browsers' results are overridden by explicit items from `targets`.

targets.uglify

`true`

When using `uglify-js` to minify your code, you may run into syntax errors when targeting later browsers since `uglify-js` does not support any ES2015+ syntax.

To prevent these errors - set the `uglify` option to `true`, which enables all transformation plugins and as a result, your code is fully compiled to ES5. However, the `useBuiltIns` option will still work as before and only include the polyfills that your target(s) need.

> Uglify has support for ES2015 syntax via uglify-es (https://github.com/mishoo/UglifyJS2/tree/harmony). If you are using syntax unsupported by `uglify-es`, we recommend using babel-minify (https://github.com/babel/minify).

> Note: This option is deprecated in 2.x and replaced with a `forceAllTransforms` option (https://github.com/babel/babel-preset-env/pull/264).

spec

`boolean`, defaults to `false`.

Enable more spec compliant, but potentially slower, transformations for any plugins in this preset that support them.

loose

`boolean` , defaults to `false` .

Enable “loose” transformations for any plugins in this preset that allow them.

modules

`"amd" | "umd" | "systemjs" | "commonjs" | false` , defaults to `"commonjs"` .

Enable transformation of ES6 module syntax to another module type.

Setting this to `false` will not transform modules.

debug

`boolean` , defaults to `false` .

Outputs the targets/plugins used and the version specified in plugin data version (https://github.com/babel/babel-preset-env/blob/master/data/plugins.json) to `console.log` .

include

`Array<string>` , defaults to `[]` .

> NOTE: `whitelist` is deprecated and will be removed in the next major in favor of this.

An array of plugins to always include.

Valid options include any:

- Babel plugins (https://github.com/babel/babel-preset-env/blob/master/data/plugin-features.js) - both with (`babel-plugin-transform-es2015-spread`) and without prefix (`transform-es2015-spread`) are supported.
- Built-ins (https://github.com/babel/babel-preset-env/blob/master/data/built-in-features.js), such as `map` , `set` , or `object.assign` .

This option is useful if there is a bug in a native implementation, or a combination of a non-supported feature + a supported one doesn’t work.

For example, Node 4 supports native classes but not spread. If `super` is used with a spread argument, then the `transform-es2015-classes` transform needs to be `include`d, as it is not possible to transpile a spread with `super` otherwise.

> NOTE: The `include` and `exclude` options *only* work with the plugins included with this preset (https://github.com/babel/babel-preset-env/blob/master/data/plugin-features.js); so, for example, including `transform-do-expressions` or excluding `transform-function-bind` will throw errors. To use a plugin *not* included with this preset, add them to your config (https://babeljs.io/docs/usage/babelrc/) directly.

exclude

`Array<string>`, defaults to `[]`.

An array of plugins to always exclude/remove.

The possible options are the same as the `include` option.

This option is useful for "blacklisting" a transform like `transform-regenerator` if you don't use generators and don't want to include `regeneratorRuntime` (when using `useBuiltIns`) or for using another plugin like fast-async (https://github.com/MatAtBread/fast-async) instead of Babel's async-to-gen (http://babeljs.io/docs/plugins/transform-async-generator-functions/).

useBuiltIns

`boolean`, defaults to `false`.

A way to apply `babel-preset-env` for polyfills (via "babel-polyfill").

> NOTE: This does not currently polyfill experimental/stage-x built-ins like the regular "babel-polyfill" does. This will only work with npm >= 3 (which should be used with Babel 6 anyway)

```
npm install babel-polyfill --save
```

Copy

This option enables a new plugin that replaces the statement `import "babel-polyfill"` or `require("babel-polyfill")` with individual requires for `babel-polyfill` based on environment.

> NOTE: Only use `require("babel-polyfill");` once in your whole app. Multiple imports or requires of `babel-polyfill` will throw an error since it can cause global collisions and other issues that are hard to trace. We recommend creating a single entry file that only contains the `require` statement.

In

JavaScript Try Copy

```
import "babel-polyfill";
```

Out (different based on environment)

JavaScript Try Copy

```
import "core-js/modules/es7.string.pad-start";
import "core-js/modules/es7.string.pad-end";
import "core-js/modules/web.timers";
import "core-js/modules/web.immediate";
import "core-js/modules/web.dom.iterable";
```

This will also work for `core-js` directly (`import "core-js";`)

```
npm install core-js --save
```

Copy

Examples

Export with various targets

JavaScript Try Copy

```
export class A {}
```

Target only Chrome 52

.babelrc

JSON Copy

```
{
  "presets": [
    ["env", {
      "targets": {
        "chrome": 52
      }
    }]
  ]
}
```

Out

JavaScript Try Copy

```
class A {}
exports.A = A;
```

Target Chrome 52 with webpack 2/rollup and loose mode

.babelrc

JSON Copy

```
{
  "presets": [
    ["env", {
      "targets": {
        "chrome": 52
      },
      "modules": false,
      "loose": true
    }]
  ]
}
```

Out

JavaScript Try Copy

```
export class A {}
```

Target specific browsers via browserslist

.babelrc

JSON Copy

```
{
  "presets": [
    ["env", {
      "targets": {
        "chrome": 52,
        "browsers": ["last 2 versions", "safari 7"]
      }
    }]
  ]
}
```

Out

JavaScript Try Copy

```
export var A = function A() {
  _classCallCheck(this, A);
};
```

Target latest node via `node: true` or `node: "current"`

.babelrc

JSON Copy

```
{
  "presets": [
    ["env", {
      "targets": {
        "node": "current"
      }
    }]
  ]
}
```

Out

JavaScript Try Copy

```
class A {}
exports.A = A;
```

Show debug output

.babelrc

JSON Copy

```
{
  "presets": [
    [ "env", {
      "targets": {
        "safari": 10
      },
      "modules": false,
      "useBuiltIns": true,
      "debug": true
    }]
  ]
}
```

stdout

Shell Copy

```shell
Using targets:
{
  "safari": 10
}

Modules transform: false

Using plugins:
  transform-exponentiation-operator {}
  transform-async-to-generator {}

Using polyfills:
  es7.object.values {}
  es7.object.entries {}
  es7.object.get-own-property-descriptors {}
  web.timers {}
  web.immediate {}
  web.dom.iterable {}
```

Include and exclude specific plugins/built-ins

> always include arrow functions, explicitly exclude generators

JSON Copy

```
{
  "presets": [
    ["env", {
      "targets": {
        "browsers": ["last 2 versions", "safari >= 7"]
      },
      "include": ["transform-es2015-arrow-functions", "es6.map"],
      "exclude": ["transform-regenerator", "es6.set"]
    }]
  ]
}
```

Caveats

If you get a `SyntaxError: Unexpected token ...` error when using the object-rest-spread (https://github.com/babel/babel/tree/master/packages/babel-plugin-transform-object-rest-spread) transform then make sure the plugin has been updated to, at least, `v6.19.0`.

Other Cool Projects

- babel-preset-modern-browsers (https://github.com/christophehurpeau/babel-preset-modern-browsers)
- ?

Community Discussion

1 reply

ConAntonakos 20 Apr

Does `babel-preset-env` change the minification from UglifyJS to babili? Therefore, I don't need to consider adding a plugin for my bundler such as Webpack?

Is it necessary to declare `{ target: { uglify: true } }` if you're using an UglifyJS plugin in your Webpack config for example?

Also, when do I need to consider using `useBuiltIns`? Is it recommended?

Thanks!