**Introduction**

This boilerplate is targeted towards large, serious projects and assumes you are somewhat familiar with Webpack and vue-loader. Make sure to also read [vue-loader's documentation](http://vuejs.github.io/vue-loader/index.html) for common workflow recipes.

If you just want to try out vue-loader or whip out a quick prototype, use the [webpack-simple](https://github.com/vuejs-templates/webpack-simple) template instead.

**Quickstart**

To use this template, scaffold a project with [vue-cli](https://github.com/vuejs/vue-cli). **It is recommended to use npm 3+ for a more efficient dependency tree.**

$ npm install -g vue-cli

$ vue init webpack my-project

$ cd my-project

$ npm install

$ npm run dev

# Project Structure

.

├── build/ # webpack config files

│ └── ...

├── config/

│ ├── index.js # main project config

│ └── ...

├── src/

│ ├── main.js # app entry file

│ ├── App.vue # main app component

│ ├── components/ # ui components

│ │ └── ...

│ └── assets/ # module assets (processed by webpack)

│ └── ...

├── static/ # pure static assets (directly copied)

├── test/

│ └── unit/ # unit tests

│ │ ├── specs/ # test spec files

│ │ ├── index.js # test build entry file

│ │ └── karma.conf.js # test runner config file

│ └── e2e/ # e2e tests

│ │ ├── specs/ # test spec files

│ │ ├── custom-assertions/ # custom assertions for e2e tests

│ │ ├── runner.js # test runner script

│ │ └── nightwatch.conf.js # test runner config file

├── .babelrc # babel config

├── .postcssrc.js # postcss config

├── .eslintrc.js # eslint config

├── .editorconfig # editor config

├── index.html # index.html template

└── package.json # build scripts and dependencies

### build/

This directory holds the actual configurations for both the development server and the production webpack build. Normally you don't need to touch these files unless you want to customize Webpack loaders, in which case you should probably look at build/webpack.base.conf.js.

### config/index.js

This is the main configuration file that exposes some of the most common configuration options for the build setup. See [API Proxying During Development](https://vuejs-templates.github.io/webpack/proxy.html) and [Integrating with Backend Framework](https://vuejs-templates.github.io/webpack/backend.html) for more details.

### src/

This is where most of your application code will live in. How to structure everything inside this directory is largely up to you; if you are using Vuex, you can consult the [recommendations for Vuex applications](http://vuex.vuejs.org/en/structure.html).

### static/

This directory is an escape hatch for static assets that you do not want to process with Webpack. They will be directly copied into the same directory where webpack-built assets are generated.

See [Handling Static Assets](https://vuejs-templates.github.io/webpack/static.html) for more details.

### test/unit

Contains unit test related files. See [Unit Testing](https://vuejs-templates.github.io/webpack/unit.html) for more details.

### test/e2e

Contains e2e test related files. See [End-to-end Testing](https://vuejs-templates.github.io/webpack/e2e.html) for more details.

### index.html

This is the **template** index.html for our single page application. During development and builds, Webpack will generate assets, and the URLs for those generated assets will be automatically injected into this template to render the final HTML.

### package.json

The NPM package meta file that contains all the build dependencies and [build commands](https://vuejs-templates.github.io/webpack/commands.html).

# Build Commands

All build commands are executed via [NPM Scripts](https://docs.npmjs.com/misc/scripts).

### npm run dev

Starts a Node.js local development server. See [API Proxying During Development](https://vuejs-templates.github.io/webpack/proxy.html) for more details.

* Webpack + vue-loader for single file Vue components.
* State preserving hot-reload
* State preserving compilation error overlay
* Lint-on-save with ESLint
* Source maps

### npm run build

Build assets for production. See [Integrating with Backend Framework](https://vuejs-templates.github.io/webpack/backend.html) for more details.

* JavaScript minified with [UglifyJS](https://github.com/mishoo/UglifyJS2).
* HTML minified with [html-minifier](https://github.com/kangax/html-minifier).
* CSS across all components extracted into a single file and minified with [cssnano](https://github.com/ben-eb/cssnano).
* All static assets compiled with version hashes for efficient long-term caching, and a production index.html is auto-generated with proper URLs to these generated assets.

### npm run unit

Run unit tests in PhantomJS with [Karma](https://karma-runner.github.io/). See [Unit Testing](https://vuejs-templates.github.io/webpack/unit.html) for more details.

* Supports ES2015+ in test files.
* Supports all webpack loaders.
* Easy [mock injection](http://vuejs.github.io/vue-loader/en/workflow/testing-with-mocks.html).

### npm run e2e

Run end-to-end tests with [Nightwatch](http://nightwatchjs.org/). See [End-to-end Testing](https://vuejs-templates.github.io/webpack/e2e.html) for more details.

* Run tests in multiple browsers in parallel.
* Works with one command out of the box:
  + Selenium and chromedriver dependencies automatically handled.
  + Automatically spawns the Selenium server.

# Linter Configuration

This boilerplate uses [ESLint](http://eslint.org/) as the linter, and uses the [Standard](https://github.com/feross/standard/blob/master/RULES.md) preset with some small customizations.

If you are not happy with the default linting rules, you have several options:

1. Overwrite individual rules in .eslintrc.js. For example, you can add the following rule to enforce semicolons instead of omitting them:
2. // .eslintrc.js
3. "semi": [2, "always"]
4. Pick a different ESLint preset when generating the project, for example [eslint-config-airbnb](https://github.com/airbnb/javascript/tree/master/packages/eslint-config-airbnb).
5. Pick "none" for ESLint preset when generating the project and define your own rules. See [ESLint documentation](http://eslint.org/docs/rules/) for more details.

# Pre-Processors

This boilerplate has pre-configured CSS extraction for most popular CSS pre-processors including LESS, SASS, Stylus, and PostCSS. To use a pre-processor, all you need to do is installing the appropriate webpack loader for it. For example, to use SASS:

npm install sass-loader node-sass --save-dev

Note you also need to install node-sass because sass-loader depends on it as a peer dependency.

### Using Pre-Processors inside Components

Once installed, you can use the pre-processors inside your \*.vue components using the lang attribute on <style> tags:

<style lang="scss">

/\* write SASS! \*/

</style>

### A note on SASS syntax

* lang="scss" corresponds to the CSS-superset syntax (with curly braces and semicolons).
* lang="sass" corresponds to the indentation-based syntax.

### PostCSS

Styles in \*.vue files are piped through PostCSS by default, so you don't need to use a specific loader for it. You can simply add PostCSS plugins you want to use in build/webpack.base.conf.js under the vue block:

// build/webpack.base.conf.js

module.exports = {

// ...

vue: {

postcss: [/\* your plugins \*/]

}

}

See [vue-loader's related documentation](http://vuejs.github.io/vue-loader/en/features/postcss.html) for more details.

### Standalone CSS Files

To ensure consistent extraction and processing, it is recommended to import global, standalone style files from your root App.vue component, for example:

<!-- App.vue -->

<style src="./styles/global.less" lang="less"></style>

Note you should probably only do this for the styles written by yourself for your application. For existing libraries e.g. Bootstrap or Semantic UI, you can place them inside /static and reference them directly in index.html. This avoids extra build time and also is better for browser caching. (See [Static Asset Handling](https://vuejs-templates.github.io/webpack/static.html))

# Handling Static Assets

You will notice in the project structure we have two directories for static assets: src/assets and static/. What is the difference between them?

### Webpacked Assets

To answer this question, we first need to understand how Webpack deals with static assets. In \*.vuecomponents, all your templates and CSS are parsed by vue-html-loader and css-loader to look for asset URLs. For example, in <img src="./logo.png"> and background: url(./logo.png), "./logo.png" is a relative asset path and will be **resolved by Webpack as a module dependency**.

Because logo.png is not JavaScript, when treated as a module dependency, we need to use url-loader and file-loader to process it. This boilerplate has already configured these loaders for you, so you basically get features such as filename fingerprinting and conditional base64 inlining for free, while being able to use relative/module paths without worrying about deployment.

Since these assets may be inlined/copied/renamed during build, they are essentially part of your source code. This is why it is recommended to place Webpack-processed static assets inside /src, along side other source files. In fact, you don't even have to put them all in /src/assets: you can organize them based on the module/component using them. For example, you can put each component in its own directory, with its static assets right next to it.

### Asset Resolving Rules

* **Relative URLs**, e.g. ./assets/logo.png will be interpreted as a module dependency. They will be replaced with an auto-generated URL based on your Webpack output configuration.
* **Non-prefixed URLs**, e.g. assets/logo.png will be treated the same as the relative URLs and translated into ./assets/logo.png.
* **URLs prefixed with ~** are treated as a module request, similar to require('some-module/image.png'). You need to use this prefix if you want to leverage Webpack's module resolving configurations. For example if you have a resolve alias for assets, you need to use <img src="~assets/logo.png"> to ensure that alias is respected.
* **Root-relative URLs**, e.g. /assets/logo.png are not processed at all.

### Getting Asset Paths in JavaScript

In order for Webpack to return the correct asset paths, you need to use require('./relative/path/to/file.jpg'), which will get processed by file-loader and returns the resolved URL. For example:

computed: {

background () {

return require('./bgs/' + this.id + '.jpg')

}

}

**Note the above example will include every image under ./bgs/ in the final build.** This is because Webpack cannot guess which of them will be used at runtime, so it includes them all.

### "Real" Static Assets

In comparison, files in static/ are not processed by Webpack at all: they are directly copied to their final destination as-is, with the same filename. You must reference these files using absolute paths, which is determined by joining build.assetsPublicPath and build.assetsSubDirectory in config.js.

As an example, with the following default values:

// config/index.js

module.exports = {

// ...

build: {

assetsPublicPath: '/',

assetsSubDirectory: 'static'

}

}

Any file placed in static/ should be referenced using the absolute URL /static/[filename]. If you change assetSubDirectory to assets, then these URLs will need to be changed to /assets/[filename].

We will learn more about the config file in the section about [backend integration](https://vuejs-templates.github.io/webpack/backend.html).

# Environment Variables

Sometimes it is practical to have different config values according to the environment that the application is running in.

As an example:

// config/prod.env.js

module.exports = {

NODE\_ENV: '"production"',

DEBUG\_MODE: false,

API\_KEY: '"..."' // this is shared between all environments

}

// config/dev.env.js

module.exports = merge(prodEnv, {

NODE\_ENV: '"development"',

DEBUG\_MODE: true // this overrides the DEBUG\_MODE value of prod.env

})

// config/test.env.js

module.exports = merge(devEnv, {

NODE\_ENV: '"testing"'

})

**Note:** string variables need to be wrapped into single and double quotes '"..."'

So, the environment variables are:

* Production
  + NODE\_ENV = 'production',
  + DEBUG\_MODE = false,
  + API\_KEY = '...'
* Development
  + NODE\_ENV = 'development',
  + DEBUG\_MODE = true,
  + API\_KEY = '...'
* Testing
  + NODE\_ENV = 'testing',
  + DEBUG\_MODE = true,
  + API\_KEY = '...'

As we can see, test.env inherits the dev.env and the dev.env inherits the prod.env.

### Usage

It is simple to use the environment variables in your code. For example:

Vue.config.productionTip = process.env.NODE\_ENV === 'production'

# Integrating with Backend Framework

If you are building a purely-static app (one that is deployed separately from the backend API), then you probably don't even need to edit config/index.js. However, if you want to integrate this template with an existing backend framework, e.g. Rails/Django/Laravel, which comes with their own project structures, you can edit config/index.js to directly generate front-end assets into your backend project.

Let's take a look at the default config/index.js:

// config/index.js

var path = require('path')

module.exports = {

build: {

index: path.resolve(\_\_dirname, 'dist/index.html'),

assetsRoot: path.resolve(\_\_dirname, 'dist'),

assetsSubDirectory: 'static',

assetsPublicPath: '/',

productionSourceMap: true

},

dev: {

port: 8080,

proxyTable: {}

}

}

Inside the build section, we have the following options:

### build.index

Must be an absolute path on your local file system.

This is where the index.html (with injected asset URLs) will be generated.

If you are using this template with a backend-framework, you can edit index.html accordingly and point this path to a view file rendered by your backend app, e.g. app/views/layouts/application.html.erb for a Rails app, or resources/views/index.blade.php for a Laravel app.

### build.assetsRoot

Must be an absolute path on your local file system.

This should point to the root directory that contains all the static assets for your app. For example, public/ for both Rails/Laravel.

### build.assetsSubDirectory

Nest webpack-generated assets under this directory in build.assetsRoot, so that they are not mixed with other files you may have in build.assetsRoot. For example, if build.assetsRoot is /path/to/dist, and build.assetsSubDirectory is static, then all Webpack assets will be generated in path/to/dist/static.

This directory will be cleaned before each build, so it should only contain assets generated by the build.

Files inside static/ will be copied into this directory as-is during build. This means if you change this prefix, all your absolute URLs referencing files in static/ will also need to be changed. See [Handling Static Assets](https://vuejs-templates.github.io/webpack/static.html) for more details.

### build.assetsPublicPath

This should be the URL path where your build.assetsRoot will be served from over HTTP. In most cases, this will be root (/). Only change this if your backend framework serves static assets with a path prefix. Internally, this is passed to Webpack as output.publicPath.

### build.productionSourceMap

Whether to generate source maps for production build.

### dev.port

Specify the port for the dev server to listen to.

### dev.proxyTable

Define proxy rules for the dev server. See [API Proxying During Development](https://vuejs-templates.github.io/webpack/proxy.html) for more details.

# API Proxying During Development

When integrating this boilerplate with an existing backend, a common need is to access the backend API when using the dev server. To achieve that, we can run the dev server and the API backend side-by-side (or remotely), and let the dev server proxy all API requests to the actual backend.

To configure the proxy rules, edit dev.proxyTable option in config/index.js. The dev server is using [http-proxy-middleware](https://github.com/chimurai/http-proxy-middleware) for proxying, so you should refer to its docs for detailed usage. But here's a simple example:

// config/index.js

module.exports = {

// ...

dev: {

proxyTable: {

// proxy all requests starting with /api to jsonplaceholder

'/api': {

target: 'http://jsonplaceholder.typicode.com',

changeOrigin: true,

pathRewrite: {

'^/api': ''

}

}

}

}

}

The above example will proxy the request /api/posts/1 to http://jsonplaceholder.typicode.com/posts/1.

## URL Matching

In addition to static urls you can also use glob patterns to match URLs, e.g. /api/\*\*. See [Context Matching](https://github.com/chimurai/http-proxy-middleware#context-matching) for more details. In addition, you can provide a filter option that can be a custom function to determine whether a request should be proxied:

proxyTable: {

'\*\*': {

target: 'http://jsonplaceholder.typicode.com',

filter: function (pathname, req) {

return pathname.match('^/api') && req.method === 'GET'

}

}

}

**Unit Testing**

An overview of the tools used by this boilerplate for unit testing:

* [Karma](https://karma-runner.github.io/): the test runner that launches browsers, runs the tests and reports the results to us.
* [karma-webpack](https://github.com/webpack/karma-webpack): the plugin for Karma that bundles our tests using Webpack.
* [Mocha](https://mochajs.org/): the test framework that we write test specs with.
* [Chai](http://chaijs.com/): test assertion library that provides better assertion syntax.
* [Sinon](http://sinonjs.org/): test utility library that provides spies, stubs and mocks.

Chai and Sinon are integrated using [karma-sinon-chai](https://github.com/kmees/karma-sinon-chai), so all Chai interfaces (should, expect, assert) and sinon are globally available in test files.

And the files:

* index.js

This is the entry file used by karma-webpack to bundle all the test code and source code (for coverage purposes). You can ignore it for the most part.

* specs/

This directory is where you write your actual tests. You can use full ES2015+ and all supported Webpack loaders in your tests.

* karma.conf.js

This is the Karma configuration file. See [Karma docs](https://karma-runner.github.io/) for more details.

**Running Tests in More Browsers**

You can run the tests in multiple real browsers by installing more [karma launchers](https://karma-runner.github.io/1.0/config/browsers.html) and adjusting the browsers field in test/unit/karma.conf.js.

**Mocking Dependencies**

This boilerplate comes with [inject-loader](https://github.com/plasticine/inject-loader) installed by default. For usage with \*.vue components, see [vue-loader docs on testing with mocks](http://vue-loader.vuejs.org/en/workflow/testing-with-mocks.html).

# End-to-end Testing

This boilerplate uses [Nightwatch.js](http://nightwatchjs.org/) for e2e tests. Nightwatch.js is a highly integrated e2e test runner built on top of Selenium. This boilerplate comes with Selenium server and chromedriver binaries pre-configured for you, so you don't have to mess with these yourself.

Let's take a look at the files in the test/e2e directory:

* runner.js

A Node.js script that starts the dev server, and then launches Nightwatch to run tests against it. This is the script that will run when you run npm run e2e.

* nightwatch.conf.js

Nightwatch configuration file. See [Nightwatch's docs on configuration](http://nightwatchjs.org/guide#settings-file) for more details.

* custom-assertions/

Custom assertions that can be used in Nightwatch tests. See [Nightwatch's docs on writing custom assertions](http://nightwatchjs.org/guide#writing-custom-assertions) for more details.

* specs/

Your actual tests! See [Nightwatch's docs on writing tests](http://nightwatchjs.org/guide#writing-tests) and [API reference](http://nightwatchjs.org/api) for more details.

### Running Tests in More Browsers

To configure which browsers to run the tests in, add an entry under "test\_settings" in [test/e2e/nightwatch.conf.js](https://github.com/vuejs-templates/webpack/blob/master/template/test/e2e/nightwatch.conf.js#L17-L39) , and also the --env flag in [test/e2e/runner.js](https://github.com/vuejs-templates/webpack/blob/master/template/test/e2e/runner.js#L15). If you wish to configure remote testing on services like SauceLabs, you can either make the Nightwatch config conditional based on environment variables, or use a separate config file altogether. Consult [Nightwatch's docs on Selenium](http://nightwatchjs.org/guide#selenium-settings)for more details.

**Prerendering for SEO**

If you want to prerender routes that will not significantly change once pushed to production, use this Webpack plugin: [prerender-spa-plugin](https://www.npmjs.com/package/prerender-spa-plugin), which has been tested for use with Vue. For pages that *do*frequently change, [Prerender.io](https://prerender.io/) and [Netlify](https://www.netlify.com/pricing) both offer plans for regularly re-prerendering your content for search engines.

**Using prerender-spa-plugin**

1. Install it as a dev dependency:

npm install --save-dev prerender-spa-plugin

1. Require it in **build/webpack.prod.conf.js**:

// This line should go at the top of the file where other 'imports' live in

var PrerenderSpaPlugin = require('prerender-spa-plugin')

1. Configure it in the plugins array (also in **build/webpack.prod.conf.js**):

new PrerenderSpaPlugin(

// Path to compiled app

path.join(\_\_dirname, '../dist'),

// List of endpoints you wish to prerender

[ '/' ]

)

If you also wanted to prerender /about and /contact, then that array would be [ '/', '/about', '/contact' ].

1. Enable history mode for vue-router:
2. const router = new VueRouter({
3. mode: 'history',
4. routes: [...]

})

# TypeScript 支持

在 Vue 2.5.0 中，我们大大改进了类型声明以更好地使用默认的基于对象的 API。同时此版本也引入了一些其它变化，需要开发者作出相应的升级。阅读[**博客文章**](https://medium.com/the-vue-point/upcoming-typescript-changes-in-vue-2-5-e9bd7e2ecf08)了解更多详情。

## 发布为 NPM 包的官方声明文件

静态类型系统能帮助你有效防止许多潜在的运行时错误，而且随着你的应用日渐丰满会更加显著。这就是为什么 Vue 不仅仅为 Vue core 提供了针对 [**TypeScript**](https://www.typescriptlang.org/) 的[**官方类型声明**](https://github.com/vuejs/vue/tree/dev/types)，还为 [**Vue Router**](https://github.com/vuejs/vue-router/tree/dev/types) 和 [**Vuex**](https://github.com/vuejs/vuex/tree/dev/types) 也提供了相应的声明文件。

而且，我们已经把它们[**发布到了 NPM**](https://cdn.jsdelivr.net/npm/vue/types/)，最新版本的 TypeScript 也知道该如何自己从 NPM 包里解析类型声明。这意味着只要你成功地通过 NPM 安装了，就不再需要任何额外的工具辅助，即可在 Vue 中使用 TypeScript 了。

我们还计划在近期为 vue-cli 提供一个选项，来初始化一个立即可投入开发的 Vue + TypeScript 项目脚手架。

## 推荐配置

|  |
| --- |
| // tsconfig.json  {  "compilerOptions": {  // 与 Vue 的浏览器支持保持一致  "target": "es5",     // 这可以对 `this` 上的数据属性进行更严格的推断  "strict": true,     // 如果使用 webpack 2+ 或 rollup，可以利用 tree-shake:  "module": "es2015",  "moduleResolution": "node"  }  } |

这句选项告诉 TypeScript 不要处理 ES 模块引入语句 (译注：import .. from ..)。这样 webpack 2 就可以充分利用其基于 ES 模块的 tree-shaking (译注一种在抽象语法树中减除未被使用的死代码的优化技术，简称摇树优化)。

参阅 [**TypeScript 编译器选项文档 (英)**](https://www.typescriptlang.org/docs/handbook/compiler-options.html) 了解更多。

## 开发工具链

要使用 TypeScript 开发 Vue 应用程序，我们强烈建议您使用 [**Visual Studio Code**](https://code.visualstudio.com/)，它为 TypeScript 提供了极好的“开箱即用”支持。

如果你正在使用[**单文件组件**](https://cn.vuejs.org/v2/guide/single-file-components.html) (SFC), 可以安装提供 SFC 支持以及其他更多实用功能的 [**Vetur 插件**](https://github.com/vuejs/vetur)。

## 基本用法

要让 TypeScript 正确推断 Vue 组件选项中的类型，您需要使用 Vue.component 或 Vue.extend 定义组件：

|  |
| --- |
| import Vue from 'vue'  const Component = Vue.extend({  // 类型推断已启用  })  const Component = {  // 这里不会有类型推断,  // 因为TypeScript不能确认这是Vue组件的选项  } |

请注意，当使用 Vetur 与 SFC 时，类型推断将自动应用于默认导出，因此不需要将其包装在 Vue.extend 中：

|  |
| --- |
| <template>  ...  </template>  <script lang="ts">  export default {  //类型推断已启用  }  </script> |

## 基于类的 Vue 组件

如果您在声明组件时更喜欢基于类的 API，则可以使用官方维护的 [**vue-class-component**](https://github.com/vuejs/vue-class-component) 装饰器：

|  |
| --- |
| import Vue from 'vue'  import Component from 'vue-class-component'  // @Component 修饰符注明了此类为一个 Vue 组件  @Component({  // 所有的组件选项都可以放在这里  template: '<button @click="onClick">Click!</button>'  })  export default class MyComponent extends Vue {  // 初始数据可以直接声明为实例的属性  message: string = 'Hello!'  // 组件方法也可以直接声明为实例的方法  onClick (): void {  window.alert(this.message)  }  } |

## 增强类型以配合插件使用

插件可以增加 Vue 的全局/实例属性和组件选项。在这些情况下，在 TypeScript 中制作插件需要类型声明。庆幸的是，TypeScript 有一个特性来补充现有的类型，叫做[**模块补充 (module augmentation)**](https://www.typescriptlang.org/docs/handbook/declaration-merging.html#module-augmentation)。

例如，声明一个 string 类型的实例属性 $myProperty：

|  |
| --- |
| // 1. 确保在声明补充的类型之前导入 'vue'  import Vue from 'vue'  // 2. 定制一个文件，设置你想要补充的类型  // 在 types/vue.d.ts 里 Vue 有构造函数类型  declare module 'vue/types/vue' {  // 3. 声明为 Vue 补充的东西  interface Vue {  $myProperty: string  }  } |

在你的项目中包含了上述作为声明文件的代码之后 (像 my-property.d.ts)，你就可以在 Vue 实例上使用 $myProperty 了。

|  |
| --- |
| var vm = new Vue()  console.log(vm.$myProperty) // 将会顺利编译通过 |

你也可以声明额外的属性和组件选项：

|  |
| --- |
| import Vue from 'vue'  declare module 'vue/types/vue' {   // 可以使用 `VueConstructor` 接口   // 来声明全局属性  interface VueConstructor {     $myGlobal: string   }  }  // ComponentOptions 声明于 types/options.d.ts 之中  declare module 'vue/types/options' {  interface ComponentOptions<V extends Vue> {  myOption?: string  }  } |

上述的声明允许下面的代码顺利编译通过：

|  |
| --- |
| // 全局属性  console.log(Vue.$myGlobal)  // 额外的组件选项  var vm = new Vue({  myOption: 'Hello'  }) |

←

## [Introduction](https://babeljs.io/learn-es2015/#introduction)

ECMAScript 2015 is an ECMAScript standard that was ratified in June 2015.

ES2015 is a significant update to the language, and the first major update to the language since ES5 was standardized in 2009. Implementation of these features in major JavaScript engines is [underway now](https://kangax.github.io/es5-compat-table/es6/).

See the [ES2015 standard](http://www.ecma-international.org/ecma-262/6.0/index.html) for full specification of the ECMAScript 2015 language.

## [ECMAScript 2015 Features](https://babeljs.io/learn-es2015/#ecmascript-2015-features)

### [Arrows and Lexical This](https://babeljs.io/learn-es2015/#ecmascript-2015-features-arrows-and-lexical-this)

Arrows are a function shorthand using the => syntax. They are syntactically similar to the related feature in C#, Java 8 and CoffeeScript. They support both expression and statement bodies. Unlike functions, arrows share the same lexical this as their surrounding code. If an arrow is inside another function, it shares the “arguments” variable of its parent function.

**Try**

**Copy**

// Expression bodies

var odds = evens.map(v => v + 1);

var nums = evens.map((v, i) => v + i);

// Statement bodies

nums.forEach(v => {

if (v % 5 === 0)

fives.push(v);

});

// Lexical this

var bob = {

\_name: "Bob",

\_friends: [],

printFriends() {

this.\_friends.forEach(f =>

console.log(this.\_name + " knows " + f));

}

};

// Lexical arguments

function square() {

let example = () => {

let numbers = [];

for (let number of arguments) {

numbers.push(number \* number);

}

return numbers;

};

return example();

}

square(2, 4, 7.5, 8, 11.5, 21); // returns: [4, 16, 56.25, 64, 132.25, 441]

### [Classes](https://babeljs.io/learn-es2015/#ecmascript-2015-features-classes)

ES2015 classes are a simple sugar over the prototype-based OO pattern. Having a single convenient declarative form makes class patterns easier to use, and encourages interoperability. Classes support prototype-based inheritance, super calls, instance and static methods and constructors.

**Try**

**Copy**

class SkinnedMesh extends THREE.Mesh {

constructor(geometry, materials) {

super(geometry, materials);

this.idMatrix = SkinnedMesh.defaultMatrix();

this.bones = [];

this.boneMatrices = [];

//...

}

update(camera) {

//...

super.update();

}

static defaultMatrix() {

return new THREE.Matrix4();

}

}

### [Enhanced Object Literals](https://babeljs.io/learn-es2015/#ecmascript-2015-features-enhanced-object-literals)

Object literals are extended to support setting the prototype at construction, shorthand for foo: foo assignments, defining methods and making super calls. Together, these also bring object literals and class declarations closer together, and let object-based design benefit from some of the same conveniences.

**Try**

**Copy**

var obj = {

// Sets the prototype. "\_\_proto\_\_" or '\_\_proto\_\_' would also work.

\_\_proto\_\_: theProtoObj,

// Computed property name does not set prototype or trigger early error for

// duplicate \_\_proto\_\_ properties.

['\_\_proto\_\_']: somethingElse,

// Shorthand for ‘handler: handler’

handler,

// Methods

toString() {

// Super calls

return "d " + super.toString();

},

// Computed (dynamic) property names

[ "prop\_" + (() => 42)() ]: 42

};

The \_\_proto\_\_ property requires native support, and was deprecated in previous ECMAScript versions. Most engines now support the property, but [some do not](https://kangax.github.io/compat-table/es6/#__proto___in_object_literals). Also, note that only [web browsers](http://www.ecma-international.org/ecma-262/6.0/index.html#sec-additional-ecmascript-features-for-web-browsers) are required to implement it, as it's in [Annex B](http://www.ecma-international.org/ecma-262/6.0/index.html#sec-object.prototype.__proto__). It is available in Node.

### [Template Strings](https://babeljs.io/learn-es2015/#ecmascript-2015-features-template-strings)

Template strings provide syntactic sugar for constructing strings. This is similar to string interpolation features in Perl, Python and more. Optionally, a tag can be added to allow the string construction to be customized, avoiding injection attacks or constructing higher level data structures from string contents.

**Try**

**Copy**

// Basic literal string creation

`This is a pretty little template string.`

// Multiline strings

`In ES5 this is

not legal.`

// Interpolate variable bindings

var name = "Bob", time = "today";

`Hello ${name}, how are you ${time}?`

// Unescaped template strings

String.raw`In ES5 "\n" is a line-feed.`

// Construct an HTTP request prefix is used to interpret the replacements and construction

GET`http://foo.org/bar?a=${a}&b=${b}

Content-Type: application/json

X-Credentials: ${credentials}

{ "foo": ${foo},

"bar": ${bar}}`(myOnReadyStateChangeHandler);

### [Destructuring](https://babeljs.io/learn-es2015/#ecmascript-2015-features-destructuring)

Destructuring allows binding using pattern matching, with support for matching arrays and objects. Destructuring is fail-soft, similar to standard object lookup foo["bar"], producing undefined values when not found.

**Try**

**Copy**

// list matching

var [a, ,b] = [1,2,3];

a === 1;

b === 3;

// object matching

var { op: a, lhs: { op: b }, rhs: c }

= getASTNode()

// object matching shorthand

// binds `op`, `lhs` and `rhs` in scope

var {op, lhs, rhs} = getASTNode()

// Can be used in parameter position

function g({name: x}) {

console.log(x);

}

g({name: 5})

// Fail-soft destructuring

var [a] = [];

a === undefined;

// Fail-soft destructuring with defaults

var [a = 1] = [];

a === 1;

// Destructuring + defaults arguments

function r({x, y, w = 10, h = 10}) {

return x + y + w + h;

}

r({x:1, y:2}) === 23

### [Default + Rest + Spread](https://babeljs.io/learn-es2015/#ecmascript-2015-features-default-rest-spread)

Callee-evaluated default parameter values. Turn an array into consecutive arguments in a function call. Bind trailing parameters to an array. Rest replaces the need for arguments and addresses common cases more directly.

**Try**

**Copy**

function f(x, y=12) {

// y is 12 if not passed (or passed as undefined)

return x + y;

}

f(3) == 15

**Try**

**Copy**

function f(x, ...y) {

// y is an Array

return x \* y.length;

}

f(3, "hello", true) == 6

**Try**

**Copy**

function f(x, y, z) {

return x + y + z;

}

// Pass each elem of array as argument

f(...[1,2,3]) == 6

### [Let + Const](https://babeljs.io/learn-es2015/#ecmascript-2015-features-let-const)

Block-scoped binding constructs. let is the new var. const is single-assignment. Static restrictions prevent use before assignment.

**Try**

**Copy**

function f() {

{

let x;

{

// this is ok since it's a block scoped name

const x = "sneaky";

// error, was just defined with `const` above

x = "foo";

}

// this is ok since it was declared with `let`

x = "bar";

// error, already declared above in this block

let x = "inner";

}

}

### [Iterators + For..Of](https://babeljs.io/learn-es2015/#ecmascript-2015-features-iterators-for-of)

Iterator objects enable custom iteration like CLR IEnumerable or Java Iterable. Generalize for..in to custom iterator-based iteration with for..of. Don’t require realizing an array, enabling lazy design patterns like LINQ.

**Try**

**Copy**

let fibonacci = {

[Symbol.iterator]() {

let pre = 0, cur = 1;

return {

next() {

[pre, cur] = [cur, pre + cur];

return { done: false, value: cur }

}

}

}

}

for (var n of fibonacci) {

// truncate the sequence at 1000

if (n > 1000)

break;

console.log(n);

}

Iteration is based on these duck-typed interfaces (using [TypeScript](http://typescriptlang.org/) type syntax for exposition only):

**Copy**

interface IteratorResult {

done: boolean;

value: any;

}

interface Iterator {

next(): IteratorResult;

}

interface Iterable {

[Symbol.iterator](): Iterator

}

#### Support via polyfill

In order to use Iterators you must include the Babel [polyfill](https://babeljs.io/docs/usage/polyfill).

### [Generators](https://babeljs.io/learn-es2015/#ecmascript-2015-features-generators)

Generators simplify iterator-authoring using function\* and yield. A function declared as function\* returns a Generator instance. Generators are subtypes of iterators which include additional next and throw. These enable values to flow back into the generator, so yield is an expression form which returns a value (or throws).

Note: Can also be used to enable ‘await’-like async programming, see also ES7 await [proposal](https://github.com/lukehoban/ecmascript-asyncawait).

**Try**

**Copy**

var fibonacci = {

[Symbol.iterator]: function\*() {

var pre = 0, cur = 1;

for (;;) {

var temp = pre;

pre = cur;

cur += temp;

yield cur;

}

}

}

for (var n of fibonacci) {

// truncate the sequence at 1000

if (n > 1000)

break;

console.log(n);

}

The generator interface is (using [TypeScript](http://typescriptlang.org/) type syntax for exposition only):

**Copy**

interface Generator extends Iterator {

next(value?: any): IteratorResult;

throw(exception: any);

}

#### Support via polyfill

In order to use Generators you must include the Babel [polyfill](https://babeljs.io/docs/usage/polyfill).

### [Comprehensions](https://babeljs.io/learn-es2015/#ecmascript-2015-features-comprehensions)

Removed in Babel 6.0

### [Unicode](https://babeljs.io/learn-es2015/#ecmascript-2015-features-unicode)

Non-breaking additions to support full Unicode, including new unicode literal form in strings and new RegExp u mode to handle code points, as well as new APIs to process strings at the 21bit code points level. These additions support building global apps in JavaScript.

**Try**

**Copy**

// same as ES5.1

"𠮷".length == 2

// new RegExp behaviour, opt-in ‘u’

"𠮷".match(/./u)[0].length == 2

// new form

"\u{20BB7}" == "𠮷" == "\uD842\uDFB7"

// new String ops

"𠮷".codePointAt(0) == 0x20BB7

// for-of iterates code points

for(var c of "𠮷") {

console.log(c);

}

### [Modules](https://babeljs.io/learn-es2015/#ecmascript-2015-features-modules)

Language-level support for modules for component definition. Codifies patterns from popular JavaScript module loaders (AMD, CommonJS). Runtime behaviour defined by a host-defined default loader. Implicitly async model – no code executes until requested modules are available and processed.

**Try**

**Copy**

// lib/math.js

export function sum(x, y) {

return x + y;

}

export var pi = 3.141593;

**Try**

**Copy**

// app.js

import \* as math from "lib/math";

console.log("2π = " + math.sum(math.pi, math.pi));

**Try**

**Copy**

// otherApp.js

import {sum, pi} from "lib/math";

console.log("2π = " + sum(pi, pi));

Some additional features include export default and export \*:

**Try**

**Copy**

// lib/mathplusplus.js

export \* from "lib/math";

export var e = 2.71828182846;

export default function(x) {

return Math.exp(x);

}

**Try**

**Copy**

// app.js

import exp, {pi, e} from "lib/mathplusplus";

console.log("e^π = " + exp(pi));

#### Module Formatters

Babel can transpile ES2015 Modules to several different formats including Common.js, AMD, System, and UMD. You can even create your own. For more details see the [modules docs](https://babeljs.io/docs/plugins/).

### [Module Loaders](https://babeljs.io/learn-es2015/#ecmascript-2015-features-module-loaders)

#### Not part of ES2015

This is left as implementation-defined within the ECMAScript 2015 specification. The eventual standard will be in WHATWG's [Loader specification](https://whatwg.github.io/loader/), but that is currently a work in progress. What is below is from a previous ES2015 draft.

Module loaders support:

* Dynamic loading
* State isolation
* Global namespace isolation
* Compilation hooks
* Nested virtualization

The default module loader can be configured, and new loaders can be constructed to evaluated and load code in isolated or constrained contexts.

**Try**

**Copy**

// Dynamic loading – ‘System’ is default loader

System.import("lib/math").then(function(m) {

alert("2π = " + m.sum(m.pi, m.pi));

});

// Create execution sandboxes – new Loaders

var loader = new Loader({

global: fixup(window) // replace ‘console.log’

});

loader.eval("console.log(\"hello world!\");");

// Directly manipulate module cache

System.get("jquery");

System.set("jquery", Module({$: $})); // WARNING: not yet finalized

#### Additional polyfill needed

Since Babel defaults to using common.js modules, it does not include the polyfill for the module loader API. Get it [here](https://github.com/ModuleLoader/es6-module-loader).

#### Using Module Loader

In order to use this, you'll need to tell Babel to use the system module formatter. Also be sure to check out [System.js](https://github.com/systemjs/systemjs)

### [Map + Set + WeakMap + WeakSet](https://babeljs.io/learn-es2015/#ecmascript-2015-features-map-set-weak-map-weak-set)

Efficient data structures for common algorithms. WeakMaps provides leak-free object-key’d side tables.

**Try**

**Copy**

// Sets

var s = new Set();

s.add("hello").add("goodbye").add("hello");

s.size === 2;

s.has("hello") === true;

// Maps

var m = new Map();

m.set("hello", 42);

m.set(s, 34);

m.get(s) == 34;

// Weak Maps

var wm = new WeakMap();

wm.set(s, { extra: 42 });

wm.size === undefined

// Weak Sets

var ws = new WeakSet();

ws.add({ data: 42 });

// Because the added object has no other references, it will not be held in the set

#### Support via polyfill

In order to support Maps, Sets, WeakMaps, and WeakSets in all environments you must include the Babel [polyfill](https://babeljs.io/docs/usage/polyfill).

### [Proxies](https://babeljs.io/learn-es2015/#ecmascript-2015-features-proxies)

Proxies enable creation of objects with the full range of behaviors available to host objects. Can be used for interception, object virtualization, logging/profiling, etc.

**Try**

**Copy**

// Proxying a normal object

var target = {};

var handler = {

get: function (receiver, name) {

return `Hello, ${name}!`;

}

};

var p = new Proxy(target, handler);

p.world === "Hello, world!";

**Try**

**Copy**

// Proxying a function object

var target = function () { return "I am the target"; };

var handler = {

apply: function (receiver, ...args) {

return "I am the proxy";

}

};

var p = new Proxy(target, handler);

p() === "I am the proxy";

There are traps available for all of the runtime-level meta-operations:

**Try**

**Copy**

var handler =

{

// target.prop

get: ...,

// target.prop = value

set: ...,

// 'prop' in target

has: ...,

// delete target.prop

deleteProperty: ...,

// target(...args)

apply: ...,

// new target(...args)

construct: ...,

// Object.getOwnPropertyDescriptor(target, 'prop')

getOwnPropertyDescriptor: ...,

// Object.defineProperty(target, 'prop', descriptor)

defineProperty: ...,

// Object.getPrototypeOf(target), Reflect.getPrototypeOf(target),

// target.\_\_proto\_\_, object.isPrototypeOf(target), object instanceof target

getPrototypeOf: ...,

// Object.setPrototypeOf(target), Reflect.setPrototypeOf(target)

setPrototypeOf: ...,

// for (let i in target) {}

enumerate: ...,

// Object.keys(target)

ownKeys: ...,

// Object.preventExtensions(target)

preventExtensions: ...,

// Object.isExtensible(target)

isExtensible :...

}

#### Unsupported feature

Due to the limitations of ES5, Proxies cannot be transpiled or polyfilled. See support in [various JavaScript engines](https://kangax.github.io/compat-table/es6/#test-Proxy).

### [Symbols](https://babeljs.io/learn-es2015/#ecmascript-2015-features-symbols)

Symbols enable access control for object state. Symbols allow properties to be keyed by either string (as in ES5) or symbol. Symbols are a new primitive type. Optional name parameter used in debugging - but is not part of identity. Symbols are unique (like gensym), but not private since they are exposed via reflection features like Object.getOwnPropertySymbols.

**Try**

**Copy**

(function() {

// module scoped symbol

var key = Symbol("key");

function MyClass(privateData) {

this[key] = privateData;

}

MyClass.prototype = {

doStuff: function() {

... this[key] ...

}

};

// Limited support from Babel, full support requires native implementation.

typeof key === "symbol"

})();

var c = new MyClass("hello")

c["key"] === undefined

#### Limited support via polyfill

Limited support requires the Babel [polyfill](https://babeljs.io/docs/usage/polyfill). Due to language limitations, some features can't be transpiled or polyfilled. See core.js's [caveats section](https://github.com/zloirock/core-js#caveats-when-using-symbol-polyfill) for more details.

### [Subclassable Built-ins](https://babeljs.io/learn-es2015/#ecmascript-2015-features-subclassable-built-ins)

In ES2015, built-ins like Array, Date and DOM Elements can be subclassed.

**Try**

**Copy**

// User code of Array subclass

class MyArray extends Array {

constructor(...args) { super(...args); }

}

var arr = new MyArray();

arr[1] = 12;

arr.length == 2

#### Partial support

Built-in subclassability should be evaluated on a case-by-case basis as classes such as HTMLElement **can** be subclassed while many such as Date, Array and Error **cannot** be due to ES5 engine limitations.

### [Math + Number + String + Object APIs](https://babeljs.io/learn-es2015/#ecmascript-2015-features-math-number-string-object-apis)

Many new library additions, including core Math libraries, Array conversion helpers, and Object.assign for copying.

**Try**

**Copy**

Number.EPSILON

Number.isInteger(Infinity) // false

Number.isNaN("NaN") // false

Math.acosh(3) // 1.762747174039086

Math.hypot(3, 4) // 5

Math.imul(Math.pow(2, 32) - 1, Math.pow(2, 32) - 2) // 2

"abcde".includes("cd") // true

"abc".repeat(3) // "abcabcabc"

Array.from(document.querySelectorAll("\*")) // Returns a real Array

Array.of(1, 2, 3) // Similar to new Array(...), but without special one-arg behavior

[0, 0, 0].fill(7, 1) // [0,7,7]

[1,2,3].findIndex(x => x == 2) // 1

["a", "b", "c"].entries() // iterator [0, "a"], [1,"b"], [2,"c"]

["a", "b", "c"].keys() // iterator 0, 1, 2

["a", "b", "c"].values() // iterator "a", "b", "c"

Object.assign(Point, { origin: new Point(0,0) })

#### Limited support from polyfill

Most of these APIs are supported by the Babel [polyfill](https://babeljs.io/docs/usage/polyfill). However, certain features are omitted for various reasons (e.g. String.prototype.normalize needs a lot of additional code to support). You can find more polyfills [here](https://github.com/addyosmani/es6-tools#polyfills).

### [Binary and Octal Literals](https://babeljs.io/learn-es2015/#ecmascript-2015-features-binary-and-octal-literals)

Two new numeric literal forms are added for binary (b) and octal (o).

**Try**

**Copy**

0b111110111 === 503 // true

0o767 === 503 // true

#### Only supports literal form

Babel is only able to transform 0o767 and not Number("0o767").

### [Promises](https://babeljs.io/learn-es2015/#ecmascript-2015-features-promises)

Promises are a library for asynchronous programming. Promises are a first class representation of a value that may be made available in the future. Promises are used in many existing JavaScript libraries.

**Try**

**Copy**

function timeout(duration = 0) {

return new Promise((resolve, reject) => {

setTimeout(resolve, duration);

})

}

var p = timeout(1000).then(() => {

return timeout(2000);

}).then(() => {

throw new Error("hmm");

}).catch(err => {

return Promise.all([timeout(100), timeout(200)]);

})

#### Support via polyfill

In order to support Promises you must include the Babel [polyfill](https://babeljs.io/docs/usage/polyfill).

### [Reflect API](https://babeljs.io/learn-es2015/#ecmascript-2015-features-reflect-api)

Full reflection API exposing the runtime-level meta-operations on objects. This is effectively the inverse of the Proxy API, and allows making calls corresponding to the same meta-operations as the proxy traps. Especially useful for implementing proxies.

**Try**

**Copy**

var O = {a: 1};

Object.defineProperty(O, 'b', {value: 2});

O[Symbol('c')] = 3;

Reflect.ownKeys(O); // ['a', 'b', Symbol(c)]

function C(a, b){

this.c = a + b;

}

var instance = Reflect.construct(C, [20, 22]);

instance.c; // 42

#### Support via polyfill

In order to use the Reflect API you must include the Babel [polyfill](https://babeljs.io/docs/usage/polyfill).

### [Tail Calls](https://babeljs.io/learn-es2015/#ecmascript-2015-features-tail-calls)

Calls in tail-position are guaranteed to not grow the stack unboundedly. Makes recursive algorithms safe in the face of unbounded inputs.

**Try**

**Copy**

function factorial(n, acc = 1) {

"use strict";

if (n <= 1) return acc;

return factorial(n - 1, n \* acc);

}

// Stack overflow in most implementations today,

// but safe on arbitrary inputs in ES2015

factorial(100000)

#### Temporarily Removed in Babel 6

Only explicit self referencing tail recursion was supported due to the complexity and performance impact of supporting tail calls globally. Removed due to other bugs and will be re-implemented.

## Introduction

ECMAScript 6, also known as ECMAScript 2015, is the latest version of the ECMAScript standard. ES6 is a significant update to the language, and the first update to the language since ES5 was standardized in 2009. Implementation of these features in major JavaScript engines is [underway now](http://kangax.github.io/es5-compat-table/es6/).

See the [ES6 standard](http://www.ecma-international.org/ecma-262/6.0/) for full specification of the ECMAScript 6 language.

ES6 includes the following new features:

* [arrows](https://github.com/lukehoban/es6features#arrows)
* [classes](https://github.com/lukehoban/es6features#classes)
* [enhanced object literals](https://github.com/lukehoban/es6features#enhanced-object-literals)
* [template strings](https://github.com/lukehoban/es6features#template-strings)
* [destructuring](https://github.com/lukehoban/es6features#destructuring)
* [default + rest + spread](https://github.com/lukehoban/es6features#default--rest--spread)
* [let + const](https://github.com/lukehoban/es6features#let--const)
* [iterators + for..of](https://github.com/lukehoban/es6features#iterators--forof)
* [generators](https://github.com/lukehoban/es6features#generators)
* [unicode](https://github.com/lukehoban/es6features#unicode)
* [modules](https://github.com/lukehoban/es6features#modules)
* [module loaders](https://github.com/lukehoban/es6features#module-loaders)
* [map + set + weakmap + weakset](https://github.com/lukehoban/es6features#map--set--weakmap--weakset)
* [proxies](https://github.com/lukehoban/es6features#proxies)
* [symbols](https://github.com/lukehoban/es6features#symbols)
* [subclassable built-ins](https://github.com/lukehoban/es6features#subclassable-built-ins)
* [promises](https://github.com/lukehoban/es6features#promises)
* [math + number + string + array + object APIs](https://github.com/lukehoban/es6features#math--number--string--array--object-apis)
* [binary and octal literals](https://github.com/lukehoban/es6features#binary-and-octal-literals)
* [reflect api](https://github.com/lukehoban/es6features#reflect-api)
* [tail calls](https://github.com/lukehoban/es6features#tail-calls)

## ECMAScript 6 Features

### Arrows

Arrows are a function shorthand using the => syntax. They are syntactically similar to the related feature in C#, Java 8 and CoffeeScript. They support both statement block bodies as well as expression bodies which return the value of the expression. Unlike functions, arrows share the same lexical this as their surrounding code.

// Expression bodies

var odds = evens.map(v => v + 1);

var nums = evens.map((v, i) => v + i);

var pairs = evens.map(v => ({even: v, odd: v + 1}));

// Statement bodies

nums.forEach(v => {

if (v % 5 === 0)

fives.push(v);

});

// Lexical this

var bob = {

\_name: "Bob",

\_friends: [],

printFriends() {

this.\_friends.forEach(f =>

console.log(this.\_name + " knows " + f));

}

}

More info: [MDN Arrow Functions](https://developer.mozilla.org/en/docs/Web/JavaScript/Reference/Functions/Arrow_functions)

### Classes

ES6 classes are a simple sugar over the prototype-based OO pattern. Having a single convenient declarative form makes class patterns easier to use, and encourages interoperability. Classes support prototype-based inheritance, super calls, instance and static methods and constructors.

class SkinnedMesh extends THREE.Mesh {

constructor(geometry, materials) {

super(geometry, materials);

this.idMatrix = SkinnedMesh.defaultMatrix();

this.bones = [];

this.boneMatrices = [];

//...

}

update(camera) {

//...

super.update();

}

get boneCount() {

return this.bones.length;

}

set matrixType(matrixType) {

this.idMatrix = SkinnedMesh[matrixType]();

}

static defaultMatrix() {

return new THREE.Matrix4();

}

}

More info: [MDN Classes](https://developer.mozilla.org/en/docs/Web/JavaScript/Reference/Classes)

### Enhanced Object Literals

Object literals are extended to support setting the prototype at construction, shorthand for foo: foo assignments, defining methods, making super calls, and computing property names with expressions. Together, these also bring object literals and class declarations closer together, and let object-based design benefit from some of the same conveniences.

var obj = {

// \_\_proto\_\_

\_\_proto\_\_: theProtoObj,

// Shorthand for ‘handler: handler’

handler,

// Methods

toString() {

// Super calls

return "d " + super.toString();

},

// Computed (dynamic) property names

[ 'prop\_' + (() => 42)() ]: 42

};

More info: [MDN Grammar and types: Object literals](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Grammar_and_types#Object_literals)

### Template Strings

Template strings provide syntactic sugar for constructing strings. This is similar to string interpolation features in Perl, Python and more. Optionally, a tag can be added to allow the string construction to be customized, avoiding injection attacks or constructing higher level data structures from string contents.

// Basic literal string creation

`In JavaScript '\n' is a line-feed.`

// Multiline strings

`In JavaScript this is

not legal.`

// String interpolation

var name = "Bob", time = "today";

`Hello ${name}, how are you ${time}?`

// Construct an HTTP request prefix is used to interpret the replacements and construction

POST`http://foo.org/bar?a=${a}&b=${b}

Content-Type: application/json

X-Credentials: ${credentials}

{ "foo": ${foo},

"bar": ${bar}}`(myOnReadyStateChangeHandler);

More info: [MDN Template Strings](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/template_strings)

### Destructuring

Destructuring allows binding using pattern matching, with support for matching arrays and objects. Destructuring is fail-soft, similar to standard object lookup foo["bar"], producing undefined values when not found.

// list matching

var [a, , b] = [1,2,3];

// object matching

var { op: a, lhs: { op: b }, rhs: c }

= getASTNode()

// object matching shorthand

// binds `op`, `lhs` and `rhs` in scope

var {op, lhs, rhs} = getASTNode()

// Can be used in parameter position

function g({name: x}) {

console.log(x);

}

g({name: 5})

// Fail-soft destructuring

var [a] = [];

a === undefined;

// Fail-soft destructuring with defaults

var [a = 1] = [];

a === 1;

More info: [MDN Destructuring assignment](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Destructuring_assignment)

### Default + Rest + Spread

Callee-evaluated default parameter values. Turn an array into consecutive arguments in a function call. Bind trailing parameters to an array. Rest replaces the need for arguments and addresses common cases more directly.

function f(x, y=12) {

// y is 12 if not passed (or passed as undefined)

return x + y;

}

f(3) == 15

function f(x, ...y) {

// y is an Array

return x \* y.length;

}

f(3, "hello", true) == 6

function f(x, y, z) {

return x + y + z;

}

// Pass each elem of array as argument

f(...[1,2,3]) == 6

More MDN info: [Default parameters](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/Default_parameters), [Rest parameters](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/rest_parameters), [Spread Operator](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Spread_operator)

### Let + Const

Block-scoped binding constructs. let is the new var. const is single-assignment. Static restrictions prevent use before assignment.

function f() {

{

let x;

{

// okay, block scoped name

const x = "sneaky";

// error, const

x = "foo";

}

// error, already declared in block

let x = "inner";

}

}

More MDN info: [let statement](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/let), [const statement](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/const)

### Iterators + For..Of

Iterator objects enable custom iteration like CLR IEnumerable or Java Iterable. Generalize for..in to custom iterator-based iteration with for..of. Don’t require realizing an array, enabling lazy design patterns like LINQ.

let fibonacci = {

[Symbol.iterator]() {

let pre = 0, cur = 1;

return {

next() {

[pre, cur] = [cur, pre + cur];

return { done: false, value: cur }

}

}

}

}

for (var n of fibonacci) {

// truncate the sequence at 1000

if (n > 1000)

break;

console.log(n);

}

Iteration is based on these duck-typed interfaces (using [TypeScript](http://typescriptlang.org/) type syntax for exposition only):

interface IteratorResult {

done: boolean;

value: any;

}

interface Iterator {

next(): IteratorResult;

}

interface Iterable {

[Symbol.iterator](): Iterator

}

More info: [MDN for...of](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/for...of)

### Generators

Generators simplify iterator-authoring using function\* and yield. A function declared as function\* returns a Generator instance. Generators are subtypes of iterators which include additional next and throw. These enable values to flow back into the generator, so yield is an expression form which returns a value (or throws).

Note: Can also be used to enable ‘await’-like async programming, see also ES7 await proposal.

var fibonacci = {

[Symbol.iterator]: function\*() {

var pre = 0, cur = 1;

for (;;) {

var temp = pre;

pre = cur;

cur += temp;

yield cur;

}

}

}

for (var n of fibonacci) {

// truncate the sequence at 1000

if (n > 1000)

break;

console.log(n);

}

The generator interface is (using [TypeScript](http://typescriptlang.org/) type syntax for exposition only):

interface Generator extends Iterator {

next(value?: any): IteratorResult;

throw(exception: any);

}

More info: [MDN Iteration protocols](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Iteration_protocols)

### Unicode

Non-breaking additions to support full Unicode, including new Unicode literal form in strings and new RegExp u mode to handle code points, as well as new APIs to process strings at the 21bit code points level. These additions support building global apps in JavaScript.

// same as ES5.1

"𠮷".length == 2

// new RegExp behaviour, opt-in ‘u’

"𠮷".match(/./u)[0].length == 2

// new form

"\u{20BB7}"=="𠮷"=="\uD842\uDFB7"

// new String ops

"𠮷".codePointAt(0) == 0x20BB7

// for-of iterates code points

for(var c of "𠮷") {

console.log(c);

}

More info: [MDN RegExp.prototype.unicode](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/RegExp/unicode)

### Modules

Language-level support for modules for component definition. Codifies patterns from popular JavaScript module loaders (AMD, CommonJS). Runtime behaviour defined by a host-defined default loader. Implicitly async model – no code executes until requested modules are available and processed.

// lib/math.js

export function sum(x, y) {

return x + y;

}

export var pi = 3.141593;

// app.js

import \* as math from "lib/math";

alert("2π = " + math.sum(math.pi, math.pi));

// otherApp.js

import {sum, pi} from "lib/math";

alert("2π = " + sum(pi, pi));

Some additional features include export default and export \*:

// lib/mathplusplus.js

export \* from "lib/math";

export var e = 2.71828182846;

export default function(x) {

return Math.log(x);

}

// app.js

import ln, {pi, e} from "lib/mathplusplus";

alert("2π = " + ln(e)\*pi\*2);

More MDN info: [import statement](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/import), [export statement](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/export)

### Module Loaders

Module loaders support:

* Dynamic loading
* State isolation
* Global namespace isolation
* Compilation hooks
* Nested virtualization

The default module loader can be configured, and new loaders can be constructed to evaluate and load code in isolated or constrained contexts.

// Dynamic loading – ‘System’ is default loader

System.import('lib/math').then(function(m) {

alert("2π = " + m.sum(m.pi, m.pi));

});

// Create execution sandboxes – new Loaders

var loader = new Loader({

global: fixup(window) // replace ‘console.log’

});

loader.eval("console.log('hello world!');");

// Directly manipulate module cache

System.get('jquery');

System.set('jquery', Module({$: $})); // WARNING: not yet finalized

### Map + Set + WeakMap + WeakSet

Efficient data structures for common algorithms. WeakMaps provides leak-free object-key’d side tables.

// Sets

var s = new Set();

s.add("hello").add("goodbye").add("hello");

s.size === 2;

s.has("hello") === true;

// Maps

var m = new Map();

m.set("hello", 42);

m.set(s, 34);

m.get(s) == 34;

// Weak Maps

var wm = new WeakMap();

wm.set(s, { extra: 42 });

wm.size === undefined

// Weak Sets

var ws = new WeakSet();

ws.add({ data: 42 });

// Because the added object has no other references, it will not be held in the set

More MDN info: [Map](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Map), [Set](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Set), [WeakMap](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/WeakMap), [WeakSet](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/WeakSet)

### Proxies

Proxies enable creation of objects with the full range of behaviors available to host objects. Can be used for interception, object virtualization, logging/profiling, etc.

// Proxying a normal object

var target = {};

var handler = {

get: function (receiver, name) {

return `Hello, ${name}!`;

}

};

var p = new Proxy(target, handler);

p.world === 'Hello, world!';

// Proxying a function object

var target = function () { return 'I am the target'; };

var handler = {

apply: function (receiver, ...args) {

return 'I am the proxy';

}

};

var p = new Proxy(target, handler);

p() === 'I am the proxy';

There are traps available for all of the runtime-level meta-operations:

var handler =

{

get:...,

set:...,

has:...,

deleteProperty:...,

apply:...,

construct:...,

getOwnPropertyDescriptor:...,

defineProperty:...,

getPrototypeOf:...,

setPrototypeOf:...,

enumerate:...,

ownKeys:...,

preventExtensions:...,

isExtensible:...

}

More info: [MDN Proxy](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Proxy)

### Symbols

Symbols enable access control for object state. Symbols allow properties to be keyed by either string (as in ES5) or symbol. Symbols are a new primitive type. Optional description parameter used in debugging - but is not part of identity. Symbols are unique (like gensym), but not private since they are exposed via reflection features like Object.getOwnPropertySymbols.

var MyClass = (function() {

// module scoped symbol

var key = Symbol("key");

function MyClass(privateData) {

this[key] = privateData;

}

MyClass.prototype = {

doStuff: function() {

... this[key] ...

}

};

return MyClass;

})();

var c = new MyClass("hello")

c["key"] === undefined

More info: [MDN Symbol](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Symbol)

### Subclassable Built-ins

In ES6, built-ins like Array, Date and DOM Elements can be subclassed.

Object construction for a function named Ctor now uses two-phases (both virtually dispatched):

* Call Ctor[@@create] to allocate the object, installing any special behavior
* Invoke constructor on new instance to initialize

The known @@create symbol is available via Symbol.create. Built-ins now expose their @@create explicitly.

// Pseudo-code of Array

class Array {

constructor(...args) { /\* ... \*/ }

static [Symbol.create]() {

// Install special [[DefineOwnProperty]]

// to magically update 'length'

}

}

// User code of Array subclass

class MyArray extends Array {

constructor(...args) { super(...args); }

}

// Two-phase 'new':

// 1) Call @@create to allocate object

// 2) Invoke constructor on new instance

var arr = new MyArray();

arr[1] = 12;

arr.length == 2

### Math + Number + String + Array + Object APIs

Many new library additions, including core Math libraries, Array conversion helpers, String helpers, and Object.assign for copying.

Number.EPSILON

Number.isInteger(Infinity) // false

Number.isNaN("NaN") // false

Math.acosh(3) // 1.762747174039086

Math.hypot(3, 4) // 5

Math.imul(Math.pow(2, 32) - 1, Math.pow(2, 32) - 2) // 2

"abcde".includes("cd") // true

"abc".repeat(3) // "abcabcabc"

Array.from(document.querySelectorAll('\*')) // Returns a real Array

Array.of(1, 2, 3) // Similar to new Array(...), but without special one-arg behavior

[0, 0, 0].fill(7, 1) // [0,7,7]

[1, 2, 3].find(x => x == 3) // 3

[1, 2, 3].findIndex(x => x == 2) // 1

[1, 2, 3, 4, 5].copyWithin(3, 0) // [1, 2, 3, 1, 2]

["a", "b", "c"].entries() // iterator [0, "a"], [1,"b"], [2,"c"]

["a", "b", "c"].keys() // iterator 0, 1, 2

["a", "b", "c"].values() // iterator "a", "b", "c"

Object.assign(Point, { origin: new Point(0,0) })

More MDN info: [Number](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Number), [Math](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Math), [Array.from](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/from), [Array.of](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/of), [Array.prototype.copyWithin](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/copyWithin), [Object.assign](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/assign)

### Binary and Octal Literals

Two new numeric literal forms are added for binary (b) and octal (o).

0b111110111 === 503 // true

0o767 === 503 // true

### Promises

Promises are a library for asynchronous programming. Promises are a first class representation of a value that may be made available in the future. Promises are used in many existing JavaScript libraries.

function timeout(duration = 0) {

return new Promise((resolve, reject) => {

setTimeout(resolve, duration);

})

}

var p = timeout(1000).then(() => {

return timeout(2000);

}).then(() => {

throw new Error("hmm");

}).catch(err => {

return Promise.all([timeout(100), timeout(200)]);

})

More info: [MDN Promise](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise)

### Reflect API

Full reflection API exposing the runtime-level meta-operations on objects. This is effectively the inverse of the Proxy API, and allows making calls corresponding to the same meta-operations as the proxy traps. Especially useful for implementing proxies.

// No sample yet

More info: [MDN Reflect](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Reflect)

### Tail Calls

Calls in tail-position are guaranteed to not grow the stack unboundedly. Makes recursive algorithms safe in the face of unbounded inputs.

function factorial(n, acc = 1) {

'use strict';

if (n <= 1) return acc;

return factorial(n - 1, n \* acc);

}

// Stack overflow in most implementations today,

// but safe on arbitrary inputs in ES6

factorial(100000)