ECMAScript6

ECMAScript 2015 (6th Edition) is the current version of the ECMAScript Language Specification standard. Commonly referred to as "ES6", it defines the standard for the JavaScript implementation in JavaScript within SpiderMonkey the engine used in Firefox and other Mozilla applications. ES6 is a superset, and backward compatible with ES5, which means that all of your ES5 code bases are already valid ES6. This helps tremendously with adopting ES6-specific features, because you can do so incrementally.

Benefits:

- Less boilerplate code to handle parameters
- Anyone reading your code immediately sees which parameters are optional, without having reading the function body (where they might or might not be on top of the function).
- Destructuring in function parameters can even support defaults
- We make the properties the object parameter requires explicit. A common issue is seeing a function takes an object, but not knowing what kind of object. Now that's solved.
- We can have defaults. No need to check if the property is there
- Promises are a way of dealing with asynchronous code. Code using promises can make asynchronous code easier to compose, and can also avoid the common issue of "callback hell". They are a first class representation of a value that may be made available in the future.
- Arrows is a new syntax for functions, which brings several benefits. It automatically binds `this` to the surrounding code's context
- The syntax allows an implicit return when there is no body block, resulting in shorter and simpler code in some cases
- The arrow operator, => is shorter and simpler than `function`, although stylistic issues are often subjective
- Asynchronous code execution can be harnessed using Generators. Which yield values. The function's execution pauses at the yielded value, and the next time the generator is called, it resumes from where it left off. In the next iteration. Similar to how "*continue*" works.
- It's better than using something like CoffeeScript, because you're not slapping arbitrary features on top of your JS – the features will be available in the future versions of JS.

Hint!:

- One potential gotcha of this syntax is that variables without default parameters still receive `undefined` as their default, rather than throwing an error if not called with enough parameters.

Features

- arrows
- classes
- enhanced object literals
- template strings
- destructuring
- default + rest + spread
- let + const
- iterators
- generators
- unicode
- modules
- module loaders
- map + set + weakmap + weakset
- proxies
- symbols
- subclassables built in
- promises
- math + number + array + string + object APIs
- binary and octal literals
- reflect API
- tail calls

Tools

Editors

- es6 syntax highlighting for Sublime Text and TextMate
- es6 syntax support in WebStorm and PhpStorm, compilation to ES5 with Traceur file watcher
- DocPad plugin for Traceur

Parsers

- Esprima Harmony branch - experimental branch of the Esprima parser which can parse ES6 features to SpiderMonkey AST format.

- Acorn - a small, fast, JavaScript-based JavaScript parser with ES6 support, parses toSpiderMonkey AST format.
- esparse - ES6 parser written in ES6.
- Traceur compiler - also has built-in parser available under `traceur.syntax.Parser`.

Transpilers

- babel - turn ES6+ code into vanilla ES5 with no runtime.
- traceur compiler - includes classes, generators, promises, destructuring patterns and default parameters
- es6ify - traceur wrapped as browserfy
- babelify - babel wrapped as browserfy
- es6-transpiler - includes classes, destructuring, default parameters, spread
- bitovi - converts ES6 to AMD
- regexpu - transforms unicode-aware ES6 regex to ES5

CI for Transpilation

grunt tasks

- es6 transpiler - https://github.com/sindresorhus/grunt-es6-transpiler
- sample grant file - https://github.com/thomasboyt/grunt-microlib
- babel - https://github.com/babel/grunt-babel
- traceur - https://github.com/aaronfrost/grunt-traceur
- module - https://github.com/joefiorini/grunt-es6-module-transpiler
- regenerator - https://github.com/sindresorhus/grunt-regenerator

gulp plugins

- babel: gulp-babel
- traceur: gulp-traceur
- regenerator: gulp-regenerator
- es6 module transpiler: gulp-es6-module-transpiler
- es6-transpiler: gulp-es6-transpiler - ES6 ES
- es6-jstransform: gulp-jstransform - ES6 ES5 using FB's jstransform
- regexpu: gulp-regexpu

Boilerplates

- es6-boilerplate -tooling to allow the community to use es6 now via traceur in conjunction with AMD and browser global modules, with source maps, concatenation, minification, compression, and unit testing in real browsers.

Karma plugins

- babel: karma-babel-preprocessor
- traceur: karma-traceur-preprocessor

Mocha plugins

- Mocha Traceur - A simple plugin for Mocha to pass JS files through the Traceur compiler

Browser plugins

- Scratch JS - A Chrome/Opera DevTools extension to run ES6 on a page with either Babel or Traceur

Module Loaders

Module systems bring support for modules to ES5 browsers (Node.js has a built-in module system). That way, you can build your app out of modules – your own and library modules. Popular module systems are:

- RequireJS: is a loader for AMD modules, which can be statically created via TypeScript, Traceur or Babel. *Loader plugins* (based on Traceur and Babel) enable it to load ES6 modules.
- Browserify: packages CommonJS modules (including ones installed via npm) so that they can be loaded in browsers. Supports ES6 modules via*transforms* (plugins) based on Traceur and Babel.
- webpack: a packager and loader for either CommonJS modules (including ones installed via npm) or AMD modules (including ones

installed via Bower). Supports ES6 modules via *custom loaders* (plugins) based on Traceur and Babel.
- SystemJS: A module system based on the ES6 Module Loader Polyfill that supports ES6 modules and the ES5 module formats CommonJS, AMD and “ES6 module loader API”.

- ES6 Module Loader polyfill (compat with latest spec and Traceur)
- js-loaders - Mozilla's spec-compliant loader prototype
- JSPM - ES6, AMD, CJS module loading/package management
- beck.js - toolkit for ES6 Module Loader pipelines, shim for legacy environments

Code generation

- generator-node-esnext - Yeoman generator for Traceur apps
- generator-es6-babel - Yeoman generator for Babel apps
- grunt-init-es6 - scaffold node modules with unit tests, authored in ES6
- Loom generators with ES6 ember modules
- brunch plugin for ES6 module transpilation

Linting

Test tools such as Jasmine and mocha can mostly be used as is, because they work with the transpiled code and don’t have to understand the original ES6 code. The following linters all support ES6, but to varying degrees:

- JSLint (focus: enforcing coding practices)
- JSHint (focus: enforcing coding practices)
- ESLint (focus: letting people implement their own style rules)
- JSCS (focus: enforcing code style)

Snippets

- ECMA6 Snippets

Resources

- https://github.com/lukehoban/es6features/blob/master/README.md
- https://github.com/addyosmani/es6-tools
- http://babeljs.io/docs/setup/
- http://codeutopia.net/blog/2015/01/06/es6-what-are-the-benefits-of-the-new-features-in-practice/
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/New_in_JavaScript/ECMAScript_6_support_in_Mozilla