Making a Standard: Inside the ECMAScript Sausage Factory

Allen Wirfs-Brock
Project Editor
ECMAScript 2015 Language Specification
@awbjs
ECMAScript 2015

“ES6”
What is ECMAScript?

- ECMAScript is the name of the international standard that defines the JavaScript programming language.
- Developed by Technical Committee 39 (TC-39) of Ecma International.
- Issued as document ECMA-262.
- Not part of W3C.

JavaScript Implementations:

- Google V8
- Mozilla SpiderMonkey
- Microsoft Chakra
- Webkit JSCore
SEVEN DAYS IN MAY

THE JOHN FRANKENHEIMER-JOEL PRODUCTION
PRODUCED BY EDWARD LEWIS • DIRECTED BY JOHN FRANKENHEIMER
A PARAMOUNT RELEASE
May 1995, Created in ten days by Brendan Eich at Netscape: “Mocha”

September 1995, shipped in beta of Netscape Navigator 2.0: “LiveScript”

December 1995, Netscape 2.0b3: “JavaScript”

August 1996, JavaScript cloned in Microsoft IE 3.0: “JScript”

1996-1997, Standardization ECMA-262 Ed. 1: ”ECMAScript” aka ES1

1999, ES3 – modern JS baseline
ECMAScript: Troubled Adolescence

• 2000: ES4, attempt 1
• 2003-4: E4X, XML extensions for ECMAScript
• 2005-8: ES4, attempt 2
• 2007: Work on ES 3.1 starts as TC39 side-project
• 2008: ES4 abandoned
• 2009: ES5: “use strict”, JSON, Object.create, etc.
The ECMAScript Standard Timeline


ES 2015 “ES6”

“Web 2.0” / AJAX

JS Performance Revolution

“ES4”  E4X  “ES4”
First Comprehensive Revision Since 1999

Some ECMAScript 2015 Enhancements

• More concise and expressive syntax
• Modules
• Class Declarations
• Block scoped declarations
• Control abstraction via iterators and generators
• Promises
• String interpolation/Internal DSL support
• Subclassable built-ins
• Binary Array Objects with Array methods
• Built-in hash Maps and Sets + weak variants.
• More built-in Math and String functions
• Improved Unicode support, Unicode RegExp

ES 2015 (June 2015): 566 pages
ES 5 (Dec. 2009): 252 pages
ES 3 (Dec. 1999): 188 pages
ES 1 (June 1997): 110 pages
TC-39 isn’t like either of these
Things TC-39 focused on for ES 2015

- Modularity
- Better Abstraction Capability
  - Better functional programming support
  - Better OO Support
- Expressiveness and Clarity
- Better Compilation Target
- Things that nobody else can do
What Kind of Language Is JavaScript?

- Functional?
- Object-oriented?
- Class-based?
- Prototype-based?
- Permissive?
- Secure?
Don’t Break the Web
Don’t Create a Franken-language
A common meta-tweet

ES6 <insert some feature> is based on <insert some other language>.
What language had the most influence on the design of ECMAScript class declarations?

a) Java
b) C++
c) Ruby
d) Dart
e) Smalltalk

✓ f) Something else: JavaScript
JavaScript Class “Constructor” Pattern

- Instance Objects
- Prototype Objects
- Methods (Function Objects)
- Constructor Functions
Classes ES5 vs ES 2015

//ES5 define Employee as subclass of Person

```
function Employee(name, id) {
    Person.call(name);
    this.id = id;
}
Employee.prototype = Object.create(Person.prototype);
Object.defineProperty(Employee.prototype, "constructor",
    {value: Employee, enumerable: false, configurable: true});
Employee.__proto__ = Person;
Employee.withId = function (id) {...}
Employee.prototype.hire = function () {...};
Employee.prototype.fire = function () {...};
```

//ES2015 define Employee as subclass of Person

class Employee extends Person {
    constructor(name, id) {
        super(name);
        this.id = id;
    }
    hire () {...}
    fire () {...}
    static withId (id) {...}
}

Both create the same object structure
Interconnections
The closure in loop problem

function f(x) {
  for (var p in x) {
    var v = doSomething(x, p);
    obj.addCallback(
      function(args){
        handle(v, p, args)
      }
    );
  }
}

... 
obj.runCallbacks();

[Orange underline] Every callback uses the same value for v and p
var hoisting causes the problem

function f(x) {
    var p;
    var v;
    for (var p in x) {
        var v = doSomething(x, p);
        obj.setCallback(
            function(args) {
                handle(v, p, args)
            }
        );
    }
}

...
ES6 can’t redefine the scoping of `var`
Fixing closure in loop problem:
Add a new block scoped declaration

```javascript
function f(x) {
    for (var let p in x) {
        var let v = doSomething(x, p);
        obj.setCallback(
            function(args){
                handle(v, p, args)
            }
        );
    }
    ...
}

obj.runCallbacks();
```

Every callback uses a distinct binding for v and p
function f(x, x) {
    var x;
    for (var x in obj) {
        if (obj[x] === somethingSpecial) {
            var x = 0;
            ...
        }
    }
}

function x() { doSomething();
x();
}
Want to avoid new `let` WTFs

// duplicate declarations
function f() {
    let x = 1;
    let x = 2;
}

// duplicate let and var
function g() {
    let x = 1;
    var x = 2;
}

// duplicate let and parameter
function h(x) {
    let x = 1;
}

// duplicate let and function
function h() {
    let x = 1;
    function x() {}
}

// hoist var to/over let
function ff() {
    let x = 1;
    if (pred) {
        var x;
    }
}
Some ES6 Declaration Rules

• Single unique binding for any name in a scope.

• Multiple `var` and top-level `function` declarations for the same name are allowed. (Still one binding per name)  Just like ES1-5

• All other multiple declarations are errors: `var`/`let`, `let`/`let`, `let`/`const`, `class`/`function`, etc.

• `var` declarations hoist to top level and auto initialized to `undefined`.  Just like ES1-5

• Can’t hoist a `var` over any other declaration of same name (except a top-level function, just like ES1-5)

• Runtime error, for accessing or assigning to an uninitialized binding

• `let`, `const`, class declarations are dead until initialized (TDZ).
So, What’s Next?

ECMAScript 2030 ?

1132 pages ?
The ECMAScript Standard Timeline

Release trains are now leaving the station

ES 1  ES 2  ES 3  ES 5  ES 5.1  ES 2015  ES 2016  ES 2017  …

“ES4”  E4X  “ES4”

Annual June Incremental Updates
The TC39 Process

The Ecma TC39 committee is responsible for evolving the ECMAScript programming language and authoring the specification. The committee operates by consensus and has discretion to alter the specification as it sees fit. However, the general process for making changes to the specification is as follows.

### Development

Changes to the language are developed by way of a process which provides guidelines for evolving an addition from an idea to a fully specified feature, complete with acceptance tests and multiple implementations. There are four “maturity” stages. The TC39 committee must approve acceptance for each stage.

<table>
<thead>
<tr>
<th>Stage</th>
<th>Purpose</th>
<th>Entrance Criteria</th>
<th>Acceptance Signifies</th>
<th>Spec Quality</th>
<th>Post-Acceptance Changes Expected</th>
<th>Implementation Types Expected</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 Strawman</td>
<td>Allow input into the specification</td>
<td>None</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>
| 1 Proposal | • Make the case for the addition  
• Describe the shape of a solution  
• Identify potential challenges | • Identified “champion” who will advance the addition  
• Pros outlining the problem or need and the general shape of a solution  
• Illustrative examples of usage  
• High-level API  
• Discussion of key algorithms, abstractions and semantics  
• Identification of potential “cross-cutting” concerns and implementation challenges/complexity | The committee expects to devote time to examining the problem space, solutions and cross-cutting concerns | None | Major | Polyglot / demos |
| 2 Draft | Precisely describe the syntax and semantics using formal spec language | • Above  
• Initial spec text | The committee expects the feature to be developed and eventually included in the standard | Draft: all major semantics, syntax and API are covered, but TODOs, placeholders and editorial issues are expected | Incremental | Experimental |
| 3 Candidate | Indicate that further refinement will require feedback from implementations and users | • Above  
• Complete spec text  
• Designated reviewers have signed off on the current spec text  
• The ECMAScript editor has signed off on the current spec text | The solution is complete and no further work is possible without implementation experience, significant usage and external feedback. | Complete: all semantics, syntax and API are completed described | Limited: only those deemed critical based on implementation experience | Spec compliant |
| 4 Finished | Indicate that the addition is ready for inclusion in the formal ECMAScript standard | • Above  
• Test 262 acceptance tests have been written for mainline usage scenarios  
• Two compatible implementations which pass the acceptance tests  
• Significant in-the-field experience with shipping implementations, such as that provided by two independent VMs  
• The ECMAScript editor has signed off on the current spec text | The addition will be included in the soonest practical standard revision | Final: All changes as a result of implementation experience are integrated | None | Shipping |

Proposals:  [https://github.com/tc39/ecma262/blob/master/README.md](https://github.com/tc39/ecma262/blob/master/README.md)
ECMAScript 2016, June 2016

New Features

• ["a", "b", "c"].includes("b")  //true
• 3 ** 2  //9, the exponentiation operator

Missed the 2016 Train

• async functions
• SIMD support
• String padStart, padEnd
• Etc.
• It’s real
• The specification is done
• Transpilers and polyfills available today
• It’s being implemented in your favorite browsers right now
• It’s the foundation for the next 10-20 years of JavaScript evolution

It Has Legs

Allen Wirfs-Brock

http://www.wirfs-brock.com/Allen
allen@wirfs-brock.com
@awbjs