Minutes of the: 43rd meeting of Ecma TC39

in: San Jose, CA, USA

on: 18-20 November 2014

1 Opening, welcome and roll call

1.1 Opening of the meeting (Mr. Neumann)

Mr. Neumann has welcomed the delegates at the Hilton in Santa Clara, CA, USA.

Companies / organizations in attendance:

Mozilla, Google, Microsoft, Intel, eBay, jQuery, Yahoo!, IBM, Facebook, Meteor, npm Inc., Twitter, Netflix, INRIA,

1.2 Introduction of attendees

Brian Terlson - Microsoft

Dmitry Lomov - Google

Taylor Woll - Microsoft

Jordan Harband – Twitter (invited expert)

David Herman - Mozilla

Ben Newman - Meteor Development Group

Arnaud Le Hors - IBM

Chip Morningstar - Paypal

Erik Arvidsson - Google

Peter Jensen - Intel

Erik Toth - Paypal

Domenic Penicola - Google

Caridy Patino - Yahoo

Rick Waldron - jQuery

Forrest Norvell - npm Inc

Waldemar Horwat - Google

Alor Schmitt - INRIA

Michael Ficerra

Eric Ferraiuolo - Yahoo

Jafar Husaik - Netflix

Allen Wirfs-Brock - Mozilla

Lee Byron - Facebook

Sebastian Markbage - Facebook

John Neumann - Chair

Jef Morrison - Facebook

Brendan Eich - Self

Mark Miller - Google

Domenic Denicola - Google

Yehuda Katz - jQuery

Kevin Smith - zenparsing@gmail.com

1.3 Host facilities, local logistics

On behalf of PayPal **Erik Toth** welcomed the delegates and explained the logistics.

1.4 List of Ecma documents considered

Ecma/TC39/2014/050	TC39 chairman's report to the CC, October 2014
Ecma/TC39/2014/051	Minutes of the 42nd meeting of TC39, Boston, September 2014
Ecma/TC39/2014/052	Submission for:JavaScript Object Notation (JSON)
Ecma/TC39/2014/053	Venue for the 43rd meeting of TC39, Santa Clara, November 2014
Ecma/TC39/2014/054	Agenda for the 43rd meeting of TC39, Santa Clara, November 2014 (Rev. 2)
Ecma/TC39/2014/055	TC39 RF TG form signed by Inria
Ecma/TC39/2014/056	Responses to the Ecma Contribution License Agreement (CLA), 14 November 2014

2 Adoption of the agenda (2014/054-Rev2)

Agenda for the: 43rd meeting of Ecma TC39

```
in: San Jose, CA
on: 18 - 20 Nov. 2014
TIME: 10:00 till 17:00 PST on 18th and 19th of Nov. 2014
      10:00 till 16:00 PST on 20th of Nov. 2014
LOCATION:
    Hilton - Santa Clara
    4949 Great America Parkway
    Santa Clara, California, 95054


Host (PayPal): Mr. Erik Toth's email: ertoth@paypal.com
Mr. Erik Toth's phone number: (517) 449-4340
```

1. Opening, welcome and roll call
   i. Opening of the meeting (Mr. Neumann)
   ii. Introduction of attendees
   iii. Host facilities, local logistics
2. Adoption of the agenda (2014/???)
3. Approval of the minutes from Sept 2014 (2014/???)

4. ECMA-262 6th Edition
    i. ✓ ES6 draft status report (Allen)
    ii. ✓ ES6 End game planning -- what's left to do?
    iii. ✓ Assignment to const. (Allen)
    iv. ✓ Array.prototype.contains breaks MooTools (Allen)
    v. ✓ Global let shadowing of global object properties (Allen)
    vi. ✓ Zepto broken by `new this.constructor` usage in some Array.prototype methods (Allen, Brian)
    vii. ✓ Exactly what did we decide to reserve to support future type syntax??
    viii. ✓ Template literal call site object caching. (Erik Arvidsson, Allen)
    ix. ✓ Array.isArray(new Proxy([], {})) (TomVC, Brendan Eich, Rick Waldron, Allen)
    x. ✓ RegExp subclassing fixes (Allen)
    xi. ✓ Performance issue: Object.defineProperties, Object.create, Object.assign hold on to the first error thrown and continue executing. (Brian Terlson, John David Dalton)
    xii. ✓ Should WeakMap/WeakSet have a .clear method? (MarkM)
    xiii. ✓ Add `async` as *FutureReservedWord* (Rick Waldron)
5. ECMA-262 7th Edition and beyond
    i. ✓ What do we do about `Array.prototype.contains` and `String.prototype.contains`? 1 2 (Domenic)
    ii. ✓ Pure functions strawperson (Domenic and Erik)
    iii. ✓ Abstract references as a solution to LTR composition and private state 1 (Kevin Smith)
    iv. I/O Streams as part of the ES standard library?? (Domenic)
    v. ✓ Array.prototype.includes: proposal to move to Stage 2.
    vi. ✓ Object.observe: proposal to move to Stage 3.
    vii. ✓ Can security monitors reliably detect monkey-patching of primordials? (Brendan, Michael Ficarra [invited expert])
    viii. ✓ `Map.prototype.map` and `Map.prototype.filter` (spec) + `Set` (Dmitry Soshnikov, Wednesday)
    ix. ✓ Revisit `Set` API (possible exclusion of `entries` and `keys`) (Dmitry Soshnikov)
6. ✓ Test 262 Status
7. ECMA-402 2nd Edition
    i. Status Update (Rick)
8. Report from the Ecma Secretariat
9. Date and place of the next meeting(s)

i. January 27 - 29, 2015 (San Francisco - Mozilla)
ii. March 24-26, 2015 (Europe)
iii. May 27 - 29, 2015 (San Francisco - Yahoo)
iv. July, 28 - 30, 2015 (Redmond, WA - Microsoft)
v. September 22 - 24, 2015 (Portland, OR - jQuery)
vi. November 17 - 19, 2015 (San Jose - Paypal)

10. Group Work Sessions
11. Closure

The agenda was approved.

3 Approval of the minutes from the September 2014 meeting (2014/051)

2014/051, the minutes of the 42th meeting of TC39, Boston, September 2014 were approved without modification.

4 General

The plan remains for GA approval of ES6 in June 2015.

TC39 will have "final draft" at Jan 2015 meeting. TC39 will vote to accept and forward to GA in March 2015. At that time TC39 RFTG need to have a final RF out-out period. It presumably starts at the March meeting;

The size and significance of ES6 for the GA as follows: ES5.1 approximately 250 pages. ES6 approximately 650 pages.

The plan is for "ES7" to have GA approval in June 2016 with yearly editions to follow.

Given the yearly release schedule TC39 would like to move away from informally talking about edition numbers (ES3, ES5, ES6, etc) and starting talking in terms like ES 2015, ES 2016, etc.). To encourage this, we would like to change the titling of ECMA-262 as follows:

Currently the document title is "ECMAScript Language Specification". TC39 would like ES6 to be titled: "ECMAScript 2015 Language Specification", ES7 would be "ECMAScript 2016 Language Specification", etc. We're just talking about the title text, the formal document numbering would remain the same, ie ECMA-262 6th Edition, ECMA-262 7th Edition, etc.

TC39 asks if that is possible ?

In June 2015 a little changed ECMA-402 (Internationalization) will also come out.

5 For details of the technical discussions see Annex 1.

6 Status Reports

6.1 Report from Geneva

Nothing new at this meeting.

6.2 Json

Matt Miller from Cisco has been approved as liaison manager between IETF and Ecma TC39. He took part in the TC39 Boston meeting.

7 Date and place of the next meeting(s)

Schedule 2015 meetings:

- January 27-29, 2015 (San Francisco, Mozilla)
- March 24-26, 2015 (Paris, France; INRIA)
- May 27-29, 2015 (San Francisco, Yahoo)
- July 28-30, 2015 (Redmond, Microsoft)
- Sept. 22-24, 2015 (Portland, jQuery)
- Nov. 17-19, 2015 (San Jose, Paypal)

8 Any other business

None.

9 Closure

Mr. Neumann thanked **PayPal** for hosting the meeting in Santa Clara, the TC39 participants for their hard work, and **Ecma International** for holding the social event dinner Wednesday evening. Special thanks goes to **Rick Waldron** to take the technical notes of the meeting.

Annex 1

# November 18 2014 Meeting Notes

Brian Terlson (BT), Taylor Woll (TW), Jordan Harband (JHD), Allen Wirfs-Brock (AWB), John Neumann (JN), Rick Waldron (RW), Eric Ferraiuolo (EF), Jeff Morrison (JM), Sebastian Markbage (SM), Erik Arvidsson (EA), Peter Jensen (PJ), Eric Toth (ET), Yehuda Katz (YK), Dave Herman (DH), Brendan Eich (BE), Ben Newman (BN), Forrest Norvell (FN), Waldemar Horwat (WH), Alan Schmitt (AS), Michael Ficarra (MF), Jafar Husain (JH), Lee Byron (LB), Dmitry Lomov (DL), Arnaud Le Hors (ALH), Chip Morningstar (CM), Caridy Patino (CP), Domenic Denicola (DD), Mark Miller (MM)

JN: Introduction

- Approval of 42nd Meeting Notes?

- - Approved

- Adoption of Agenda?

- - Approved
-

# 4.1 ES6 Draft Status Update
(Allen Wirfs-Brock)

AWB: One revision since last meeting, rev 28.
- http://wiki.ecmascript.org/doku.php?id=harmony:specification_drafts#october_14_2014_draft_rev_28

- Modules

- - Removed loader pipeline and Reflect.Loader API (moved to a new document)
- - Stream lined module linking semantics for declarative modules
- - Removed module declaration
- - Updated import decl. to include module imports
- - Updated defautl export syntax and semantics to support export of anonymous default functions
- - Added Module Env Records
- (Copy from slides)
-
- - There's a bug in rev r28 wrt module name normalization – should be relative to current module, was omitted (included in r29)

- Interim Subclass instantiation reform

- (Copy from slides)
- - Changed ordinary object creation to dispatch object allocation through [[CreateAction]] internal slot instead of @@create
- - Converted all @@create methods into [[CreateAction]] abstract operations

- - Eliminated Symbol.create and @@create
- - super without an immediately following
- (Copy from slides)
-
-

WH: What was the conclusion, not clear from the notes

AWB: (revisits problem statement and agreed upon solution)
- https://github.com/rwaldron/tc39-notes/blob/master/es6/2014-09/sept-24.md#object-instantiation-redo
- https://github.com/rwaldron/tc39-notes/blob/master/es6/2014-09/sept-24.md#conclusionresolution

EA: What about argument passing?

BT/RW: This was concretely included in conclusion

AWB: (Confirms)

- Rev28 Draft

- - ES6 eval semantics
- - Eliminated unused abstract operations PromiseAll PromiseCatch, PromiseThen
- - Modified Promise.all to specification internally uses a List instead of an Array
- - Added @@iterator property to %IteratorPrototype%
- - Added requirement that the object returned by ordinary object [[Enumerate]] must inherit from %IteratorPrototype%
- - Removed @@iterator from various standard iterators (inherited now)
- - Updated ToPropertyKey to accept Symbol wrapper objects, similar to how other priitive coercion abstract operations handler wrapper objects
- - ToNumber now recognizes binary and octal integers
- - Significant fix to destructuring assignent where the rest aassingment target is itself a destructuring pattern.
- - Updated Annex A Grammars to match ES6
-
-

AWB: (whiteboard)
Now allowed:
```js
[a[1], ...[f, ...rest]] = array;
```

## 4.2 End Game Planning
(Allen Wirfs-Brock)

- - Needed:
  - - One paragraph summary of ES6 goals for introduction
  - - Clause 4 - Language Overview. Needs to reflect ES6 features
- - Readers, reviewers
- - Ecma-402 2nd Edition, review
- - How will we resolve last minute issues?
-

DH: questions about March deadline, JN says deadling is important for patent issues, AWB says that last-minute changes could push publication date

DH: worst-case scenario: possibility of shipping a broken of ES6 to preserve the release momentum, getting the finished spec to the GA

WH: we can't ship a broken ES6

DH: respectfully disagree, need to get a quality ES6 out into the world, that momentum is very important to see

AWB: zero expectation that the ES6 spec is going to be perfect: "too much like software to have any expectations"

AWB: will open a bug tracker on bugs to deal with early errata, hit the next edition – turnaround in a year!

DH: if we do find issues, they won't be hanging out in the world as long

AWB: Revisiting confidence in current state of spec.
- Keep in mind next opt-out period: https://github.com/rwaldron/tc39-notes/blob/master/es6/2014-07/jul-30.md#rftg-admin-es6-opt-out-period

DH: If time before we ship, could we do more? Realms? Probably not. Consider revisiting try/finally restriction on generators?

BE: We didn't end up on that path.

AWB: Need to review iterator algorithms for any places that have abnormal exits to call `return()`

RW: (explanation of why the language overview draft is incomplete)
- Can we get a concrete due date?

AWB: Before the holidays

RW: (agreement)

AWB: 402 2nd Edition

RW: Currently updated to reflect the necessary changes for ES6. Allen and I have decided to coordinate 402 edition publication with 262 editions (eg Ecma-262 6th -> Ecma-402 2nd, and so forth)

TODO: Review with Allen

AWB: What to do about last minute changes?

BE: es-discuss, but not everyone reads this all the time.

AWB: Can we use the reflector to start conversations?

DH: We need to be prepared to have conference calls

## 4.3 Assignment to a const: static error?
(Allen Wirfs-Brock)

https://esdiscuss.org/topic/throwing-errors-on-mutating-immutable-bindings https://bugs.ecmascript.org/show_bug.cgi?id=3253

```javascript
const x = 42;
x = 32; // early error?
```

- es-discuss consensus: eliminate early error, because analysis during parsing is hard and no clear consensus that's work the parser should be taking on
- current spec. draft (leagcy) ES5 semantics only throws on assignment to an immutable binding in strict mode:

```js
"don't use strict";  // ?

Object.defineProperty(this, "globalReadOnly", { value: "readonly" });

var func = function f() {
  // silently skips assignment
  f = undefined;
  // silently skips assignment
  undefined = 42;
  Infinity = 0;
};
func(); // no exception thrown
```

- Should assingment to const also be silent in non-strict mode? Exception will require some new spec mechanisms.

WH: Leery about introducing new kinds of state that then make their way into various reflection APIs. This extra bit of state to distinguish const bindings from merely immutable ones will not be reflected to user code in any way, right?

AWB: Right.

#### Conclusion/Resolution

- Runtime assignment to const bindings (ie. bindings introduced by the const keyword) throw in all modes (strict and non-strict).
- Legacy const bindings (function name bindingin in function expressions) in sloppy mode continue to be no ops

## 4.4 Array.prototype.contains breaks MooTools
(Allen Wirfs-Brock)

https://esdiscuss.org/topic/having-a-non-enumerable-array-prototype-contains-may-not-be-web-compatible
https://esdiscuss.org/topic/array-prototype-contains-solutions

AWB: The issue is not "contains", but specifically in how they create their mixins

(General discussion to clarify: String.prototype.contains, `Array.prototype.contains`. Both are problematic)

Tabling until Domenic arrives.

BT: Conflict in Outlook web version, `Array.prototype.values`. This has been patched and the issue should dissappear in the next few months.
- Where there is one problem, there are many...

BE: Mark may have a proposal for fixing the `Array.prototype.contains` issue. If we're waiting for Domenic, can we wait for Mark? Is he coming?

AWB: We agreed to not worry about `Array.prototype.values`, because it can be fixed and the fix is quickly distributed.

BT: IE team has pushed forward on `Array.prototype.values` in the technical preview (small applause).

EA: We can roll it out again as well.

Discussion of alternative paths for `String|Array.prototype.contains`. Come back to it.

#### Conclusion/Resolution

- Revisit with Domenic present

## 4.5 Global let shadowing of global object properties
(Allen Wirfs-Brock)

AWB: issues:

- When are/arent' global lets allowed to shadow an aleady existing property of the global object
- Are buit-in global equivalent to global vars or are they just properties of global object
- Make it illegal to shadow a global property would mean future global properties are breaking changes

Proposal:
Runtime error when instantiating a script if a lexical declaration shadows a non-configurable property of global object.

WH: Any new non-configurable global properties would be a breaking change

DH:

AWB: Takes care of known issues, eg `let Infinity = ...`

WH: What does it solve?

DH: w/o this fix: the hazard is that any code can change the meaning (in an irrevocable way) of a global

WH: If it shadows locally, then it's local to scope

DH: This is in top level, script

BE: This is an issue for JITs, when we made non-configurable, JITs took advantage of this

DL: Yes

WH: Can you retroactively introduce a let?

EA: Yes.

DH: Which footgun is least problwem?

AWB: BZ claims there are security implications.

DH: Jason couldn't provide any security issues

AWB/BE: In BZ's email

WH: contained in your scope?

DH: The lexical contour is global

BE: (from BZ's email): `window.location`
... we should just do it.

AWB: Short of redesigning the entire global lexical scope contour.

BE: `window.location`, etc. are "own"

DH: Any time you ask for it, it must always be that exact property. If it's on the prototype, the chain can be mutated.

AWB: Properties non-configurable, function declarations didn't override them. Issues in ES5.

DH/BE: Always global:

- window.location
- window.top

BE: The minimum solution is to address only own properties of the global.

DH: Based on the current state of the global object.
- Have to specify when the check is done.

AWB: At var instantiation

DH: Mutated later to _become_ non-configurable, unaffected. Does not retroactively become an error.

JHD: ? About built-ins configurability

DH: Non-issue, we won't specify which _names_.
- No retroactive error because (dave can you fill this in)

WH: Why should they be allowed to make something non-configurable later?

BE/AWB: Just an object

BE: Might try to do a two way check? Not worth it.

WH: This will bite us at some point.

DH: Locking down environment against untrusted code, you've always had to be the first to run.

Clarification of who is setting up the invariants. User code vs. Browser.

BE: Browser wants to know later that when it makes access to `location` or `top` that it will get the binding that it created.

DH: The browser just has to lock it down before any other code. Just like user code that wants to lock down the environment.

WH: Why do this if security is an issue and not make them let bindings?

BE: Not backwards compatible

WH: Can't create new non-configurable properties

AWB: Anytime there are local scripts with top level var and let bindings, you have possible conflicts with other bindings. It just _is_. A good a reason to use modules.

#### Conclusion/Resolution

- Error when instantiating a script if a lexical declaration shadows an own, non-configurable property of global object.

## 4.6 Zepto broken by new this.construct usage in some Array.prototype methods
(Allen Wirfs-Brock, Brian Terlson)

BT: The checks that Zepto does internally to know what to construct is broken by changes in Array methods.

```js
var obj = {};
var obj.__proto__

```

The problem is that ES6 Array methods do not explicitly create new Arrays anymore, but instead call `this.constructor`. Zepto uses a plain object with __proto__ assigned...

fill in later

AWB: Spec text that breaks Zepto:

```
4. If O is an exotic Array object, then
   a. Let C be Get(O, "constructor").
   b. ReturnIfAbrupt(C).
   c. If IsConstructor(C) is true, then

      • i.  Let thisRealm be the running execution context’s Realm.
      • ii.  If SameValue(thisRealm, GetFunctionRealm(C)) is true, then
      • iii. Let A be the result of calling the [[Construct]] internal method of C with argument (0).

5. If A is undefined, then

      • a. Let A be ArrayCreate(0).
```

```

AWB: Forced me to revisit "species".

Discussion clarifying the cause.

AWB: We can fix this by doing one more level of indirection...

Zepto Proposed Fix

```
4. Let C be Get(O, "constructor")
5. ReturnIfAbrupt(C)
6. If IsConstructor(C) is true, then
    • a. Let thisRealm be the running execution context's Realm.
    • b. If SameValue(thisRealm, GetFunctionRealm(C)) is true, then
        o i. Let species be Get(C, @@species);
        o ii. ReturnIfAbrupt(species)
        o iii. If IsConstructor(species) is true, then
            ▪ 1. Let A be the resu;t of calling the [[Construct]] internal method of species with
              argument O.

7. If A is undefined, then
    • a. Let A be ArrayCreate(0).
```

WH: If @@species is intended to create copies of the current object, then why wouldn't Object.constructor have a @@species? Wanting to create copies of Objects is perfectly natural, but then we'd be back to the same Zepto problem. What you'd want is a @@speciesButDoNotDefineMeOnObjectConstructorUnderPentaltyOfBreakingZepto.

What color is the bikeshed?

- species
- copyConstructor

BT: Is this 100% back compatible?

AWB: It should be

Who is going to implement and test?

BE: put in spec.

Remaining differences...

AWB: ES5 always gave an array for these methods. For subclassing, we needed to change that.

DD: This works well for creating Promise subclasses.
- Promise and Array are only built-ins that have methods that make instances of themselves.
- eg. `new this.constructor()`
```

#### Conclusion/Resolution

- Allen's proposed fix accepted.
- It's called "species"

## 5.1 & 4.4 Array.prototype.contains and String.prototype.contains
(Domenic Denicola, Mark Miller)

DD/RW/BT: Just change them both to `includes`. It solves it directly.

DH: Consistency constraint? Any other ducktyping that expects "contains"?

Probably

DD: There are also DOM APIs that "look like" arrays, but have no Array.prototype methods, but _do_ have a `contains` method.

RW: DOMTokenList (and one other?)

BE: This is a naming game

BT: Let's just do `includes`

BE: (agrees)

MM: (explanation of analogous operations on Array to String.prototype.contains)

There is precedent for papering over the difference between substring and array elements

#### Conclusion/Resolution

- `String.prototype.contains` => `String.prototype.includes`
- `Array.prototype.contains` => `Array.prototype.includes`

Continues...

DH: We should be allowed to extend built-in prototypes. New syntax can't be the first solution.

Discussion about how, when and where it's appropriate to publish polyfills that adhere to specification bound features.

SM: Not possible to publish polyfills and know that users will be responsible with upgrades

JHD: Each change with the spec has to be a major version bump.

BE: Need to know when to risk

JHD: An es7-shim will likely have finer granularity in feature detection.

DD: Tests will never be complete enough

MM: Experience in SES is proof that the things that need to be tested for will always grow.

DH: We should reprise this conversation when Yehuda is here, he has concrete recommendations to share with authors, with regard to train model.

AWB: Is there something that modules can help with? `import ...` and get ES7 features?

DD: No, that's effectively "use es7";

AWB: Modules loaded for side effects?

MM: shouldn't encourage the pattern

## 4.8 Template literal call site object caching.

(Erik Arvidsson, Mark Miller, Allen Wirfs-Brock)

- https://bugs.ecmascript.org/show_bug.cgi?id=3305
- https://mail.mozilla.org/pipermail/es-discuss/2014-July/038343.html

AWB:

```js
let world = "world";
let t = "tag`hello, ${world}`";
eval(t);
eval(t);

new Function(t)();
new Function(t)();

tag`hello, ${world}`
```

How many unique callsites? 5, 3, 2 or 1?

AWB: Note that the following

```js
tag`hello, ${world}`
tag`hello, ${world}`
```

has two unique callsites.

BE: Identity can be observed

MM: If you adopt the answer 1, there is no communication channel open. Although mutable state is reachable from the callsite (nee template) object, the only such mutable state is the primordials (Array.prototype, etc). Thus, the memo must be per-realm -- by contrast with the global Symbol registry. The identity sharing is surprising. If we want to avoid that surprise, then 5 is the answer. The performance argument says 1. Whatever the `tag` function pre-computes, it will typically memoize based on the identity of the callsite (nee template) argument. Note that ES6 provides identity-based maps, but provides no content-equality-based maps for use on array or object keys.

AWB: "Callsite" is probably the wrong term.

WH: What is `tag`? System or user code?

BE: Tag is not the thing that we're discussing being memoized. It's the constant array that's passed to tag.

MM: (restating issue) The object that captures the literal part of the expession that's captured for the call.

DH: If we go with 5, the argument becomes: regain performance that's been lost? Can the programmer do it for themselves?
- So if I want the function to execute multiple times, how can implement the single "cached".
- Impossible to get the performance of 1, because the 5 would _always_ allocate an array.

In favor of 1:
- No way to get that performance manually
- If you want 5, you can manually do it with 1.

WH: We've tried to do call site memoization in the past. In ES3 we allowed behaviorally equivalent closures that didn't capture any free variables to share identity. We also made regexp literals share identity. Since then we've backed out of both of those decisions.

BE: ES3 left closure memoization up to implementations. ES5 forbade closure memoization.

MM: (revisiting const functions)
- const functions http://wiki.ecmascript.org/doku.php?id=strawman:const_functions , by freezing the function, safely enable the joining optimization that ES3 unsafely tried to allow.

MM: 1: the memoization is the raw string contents and the holes, only -- the information that goes into the callsite (nee template) object.

Discussion of performance via caching on "call sites"

MM: The consequences of 1 are easy to enumerate

WH: If we do 1, how would we implement that without leaking memory? An implementation would likely have a memoization map from strings to arrays. At what point can that map forget bindings? There is never a guarantee that a particular key string won't be used again.

("template identity" is better to describe the thing that has been referred to as "call site")

MM: For a given set of template contents, there exists no more than 1 template identity. That invariant is not violated by having 0. Within the implementation, one could have a weak-value map, mapping from the template contents to a weakly held template object. Weak value maps expose non-deterministic GC http://wiki.ecmascript.org/doku.php?id=strawman:weak_references#a_weakvaluemap , but this internal use of a weak value map does not expose any effects of GC to JS code.

BE:

```js
`hi, ${name}`
`hi, ${n}`
`hi, ${_}`
````

All of those would evaluate to the same template.

WH:

```js
`hi, ${a}, ${b}`
`hi, ${x}, ${x}`
```

Would those two be the same template or two different templates?

BE: Same.

#### Conclusion/Resolution

- The result would be 1

## What is the `this` binding at the top of a module?

DH: Makes sense going forward to access the global via the Reflect library. `this` should be undefined at the top level of a module.

DD: I previously thought we had consensus on `Reflect.global`, but it's not in the spec

MM: The idea of something like `Reflect.global` is a good idea, but nervous about putting it ES6. Libraries like Caja need to be able to virtualize. Things we've made available through committee defined modules have been authority free, but global is authority bearing. Need more experience living with the ES6 module system before deciding how to make authority bearing things available for import.

DD: Won't indirect eval give you a script context global?

MM: Don't need to add something to ES6 for this issue.

DH: Agree

#### Conclusion/Resolution

- `this` is undefined

## 4.9 Array.isArray(new Proxy([], {}))
(TomVC, Brendan Eich, Rick Waldron, Allen Wirfs-Brock)

AWB: (Explains the expectation of `Array.isArray(...)`)

BE: Tom believes that `Array.isArray(new Proxy([], {})) === true`

RW: (agrees)

AWB: Breaks the exotic array check

BE: But not the same

AWB: All the checks have been replaced with spec language re: exotic array object

MM: What are the observable differences of a proxied array

DH: (asserts that there are concrete cases for virtualization)

RW: (wants to provide reality based example, but can't get a fucking word in)

BE/MM: Allowing Array.isArray to behave this way is desirable

AWB: Unless the Proxy is poorly implemented

BE: We already decided that malicious or poorly implemented Proxy's don't restrict our

DH: Agree that `Array.isArray(new Proxy([], {})) === true`

AWB: Even if they override all the mop operations, and no longer behaves like an Array

DH: Yes.

MM: Array is part of the primordials, don't have to specify how they come into existence, just how they behave once in existence.

BE:

```
  Array.isArray  | Result
---------------------------
        []       | true
new Proxy([], {})| true
Array Subclass   | true
new Nodelist()   | false
new Uint32Array()| false
```

WH: (adds row to BE's table, based on the proposed isArray pseudocode on the slide that turns exotic objects that share Array's constructor into being themselves arrays)
Any exotic object that inherits from Array | ?!@#

DH: let d = new Date; d.__proto__ = Array.prototype; Array.isArray(d)

DH: can allow typed objects where

BE: Any value object that inherits from Array, isArray => true

AWB: If an exotic object and inherits from Array, isArray => true

EA: @@isConcatSpreadable addresses the failure of Nodelist being unable to inherit from Array

AWB: Proxy with array as target may not behave anything liek an array

DH: But that's not what brands are about, simply about the bit that says "the brand"

AWB: Promise.isPromise wouldn't work if `new Proxy(new Promise(), {})`

DD: But `Array.isArray` is a special case.

- `Array.isArray` checks to see if its argument is a Proxy and then drills through to the target?

•

DH: Addressed the lack of `typeof ...`

DH: Could say that isArray is true IFF argument is an exotic array...

AWB: No, would break many things.

MM: Agreement with Dave, that these things should agree with eachother.

DH: A new term that means "is inductively like an exotic array object".

DH: The meaning of isArray, there is reflectively a bit that's set that differentiates the behaviour to use for all special cases. Any data structure in which Array.isArray is true, JSON.stringify should follow the array path.

eg.
```js
var a = [];
var p = new Proxy(a, {});

Array.isArray(p) === true;

var o = {
  a: p
};

JSON.stringify(o); === '"{"a":[]}"';
```

DH: Any Proxy whose target is an Array, is treated like an Array.

WH: What about objects that inherit from Array?

DH: Only objects that were created using Array's construction mechanism (from last meeting) would be arrays. Objects that merely monkey-patched the proto chain to inherit from Array would not be arrays.
AWB: post meeting note -- "created using Array's construction mechanism" is equivlant to says "is an exotic array object" because that mechanism is the only way to create exotic array objects..

DH: For proxies, remember whether the target object was an array when the proxy was created, and return the same answer. Revoked array proxies would still be arrays.

DH: No other objects would be arrays.

#### Conclusion/Resolution

- `Array.isArray` checks to see if its argument is a Proxy of an Array and returns true when it is

• - b/c of revocable proxy: at creation time, discover it's an Array

- Any data structure in which Array.isArray is true, JSON.stringify should take the Array path.
- The following:

• - Array.isArray

- - Array.prototype.concat
- - isConcatSpreadable
- - JSON.stringify

should replace the occurrence of "is exotic array object with the isArray interal check.

AWB: In post meeting discussions MM and AWB concluded that Array.isArray should throw when applied to a revoked proxy. This is more consistent with overall revoked proxy behavior and eliminates the need for additiounal mechanism for remembering the array-ness of revoked proxies.

## 4.10 RegExp subclassing fixes

(Allen Wirfs-Brock)

AWB: When ES6 refactored functions that either take a string or a RegExp there was an issue where the state on the RegExp instance was not set correctly.

AWB: These functions created a clone of the RegExp. But how do we do that when there are subclasses involved.

```javascript
new RegExp(regExp);
new RegExp(regExp, flags);  // Throws!
```

AWB: Seems like we should allow passing in flags in this case.

MM: Isn't there a property that gives you the source?

AWB: There is `source`.

WH: Is this a compatible change?

AWB: The match function does not use lastIndex and other state.

MM: Is the cloning too big of a hammer?

AWB: We are making a clone because we do not want to mutate the internal state.

AWB: What are the obvious reasons to subclass RegExp? Maybe one wants to add new flags? But RegExp has no way of getting the flags as whole.

```javascript
let re = /abc/mi;
re.??? === 'mi'
```

AWB: Suggests adding a `flags` getter, and extending the RegExp constructor so that it can take a RegExp *and* a flags string, instead of throwing as it does now.

DH: Why do we need this?

AWB: The double dispatch is needed allow subclassing of RegExp.

DH: Why the @@isRegExp symbol

AWB: It is needed due to the double dispatch in functions that take either a RegExp or a string. If we didn't have the symbol then we would blindly just do `toString`.

WH: We just ran across the same problem earlier today with arrays. Why solve it in two different ways in the standard?

MM: Why don't we use symbols for these double dispatch functions and then we don't need the extra symbol, `@@isRegExp`

BE: Does anyone want RegExp.prototype.match? Together with String.prototype.match, it's too confusing as to what is matching what.

DH: Can we revert or defer it?

AWB: This has been in the spec for years.

DH: There is a subtle difference between match and exec. If we now add match to RegExp it is going to be even more confusing.

AWB: We can just

AWB/BE: There are only four of these: match, replace, search, split

BE: Lets add symbol names for them.

BE: `Symbol.match`, `Symbol.replace`, `Symbol.search`, `Symbol.split`

WH: Do we need @@species any more?

AWB: We might still need it for the `new this.constructor`.

DD: We use `@@species` for Promises.

MM: If Arrays use @@species, TypedArrays use @@species, Promises use @@species, then RegExp should use it too.

#### Conclusion/Resolution

- Add RegExp.prototype.flags getter

- - Per discussion with JHD/AWB/BE: RegExp#flags should return a string of flags, but sorted alphabetically, to match #toString - eg /a/gim.flags === /a/igm.flags === 'gim'
- - Erratum from JHD:
-   - All implementations I tested (FF/Chr/Saf/IE/node) alphabetize the flags in RegExp#toString - the spec should make sure that's concrete for both #toString and #flags
-   - When there are no flags, the spec should probably specify that RegExp#flags returns an empty string
-   - question to be clarified: is `flags` an own property on a RegExp instance, like "source"? Or, is it a getter defined on RegExp.prototype?

- Make RegExp constructor not throw for (re: RegExp, flags: string).
    - Essentially, implicit conversion of re -> re.source when "flags" is provided?
- Rename the double dispatch methods to use use symbol names instead of string names.
- Get rid of @@isRegExp
- Add @@species for consistency.

## 4.13 Add async as FutureReservedWord

(Rick Waldron)

RW: We've reserved `await`, but not `async`. Should `async` be added as well?

DD: async is contextual. It is only valid as `async [nonewlinehere] function`.

EA: But async arrow function might need it.

```js
async (...) => {}
         ^ look ahead to here!

async(...)
```

EA: The existing cover grammar covers this almost completely already.

DH: The cover grammar is creeping me out
- Not likely to have async the module and async the contextual keyword in the same scope and if you do...

FN: Not insurmountable, if async (the module) is re-written for ES6 modules, nothing saying that it can't be renamed.

DH: Too much of an adoption tax

Conflict with existing Identifier use is not worth creating a _reserved_ _word_.

#### Conclusion/Resolution

- Will not reserve `async` as FutureReserveWord

## 4.11 Performance issue: `Object.defineProperties`, `Object.create`, `Object.assign`.
(Brian Terlson, John David Dalton)

BT: The performance issue arises when no error occurs, despite being specified that the first error thrown is to be held onto until the end of the operation.

MM/AWB: Hard to accept that this is specification related.

MM: Implementation effort should be spent, not spec change and user pain.

BT: Don't care at all about the determinism of the shape of the object when an error occurs. No library code does this, so why does the spec?

MM: The original specification focused on atomicity of the operation, which had actual performance costs and we backed out of that. This was the next best semantics.

WH: If more than one error, which do you get? Is that deterministic?

BT/MM/AWB: Always the first.

WH: First property or first temporally?

MM: First temporally. And yes, this leads to the same kind of "nondeterminism" in the choice of errors to throw.

AWB: Hard to believe this is the perf bottleneck. It doesn't seem credible.

BT: It's not the bottleneck.

MM: The burden is not big

JHD: Burden in shims or transpilers?

MM: Need for polyfills?

JHD: Still know of runtimes that need es5-shim

DD: If Object.assign is slow, no one will use it

RW: It will be a complete failure.

BE: The complaint is?

BT: If no one cares about this behaviour, why are we requiring it?

MM: I care.

JHD: Have a ticket in es6-shim for this, haven't implemented it due to the cost of try/catch

BE: If authors aren't testing for this and no one is paying attention to this...

AWB: But we don't want to leave it implementation dependent.

DD: In ES5 it's completely deterministic: you just throw the first

DL: No.

MM: If you consider the object as a bag of properties and not a sequence of properties.

JHD: Not a determinism issue, completely deterministic in all cases. The first error is always thrown

AWB: All the properties that can be computed will be computed.

MM: No disagreement

Discussion about who owns the burden of these performance

RW: Why does Object.assign also behave this way?

BT: b/c we made O.pD and O.c do this, and for consistency

RW: But my original proposal said nothing of doing this. Developer expectation would be: this behaves like jQuery, YUI, Dojo, etc.

MM: Do not object to `Object.assign` being specified without the try/catch because the operation is just a put and when the target is a non-Proxy it may still have setters invoked on put.

BE: if we made a mistake in not specifying order for Object.defineProperties or Object.assign, that's on us -- not a reason to inflict held-first-exception workaround for our mistake on devs of engines and polyfills

#### Conclusion/Resolution

Continue tomorrow.

# November 19 2014 Meeting Notes

Brian Terlson (BT), Taylor Woll (TW), Jordan Harband (JHD), Allen Wirfs-Brock (AWB), John Neumann (JN), Rick Waldron (RW), Eric Ferraiuolo (EF), Jeff Morrison (JM), Sebastian Markbage (SM), Erik Arvidsson (EA), Peter Jensen (PJ), Eric Toth (ET), Yehuda Katz (YK), Dave Herman (DH), Brendan Eich (BE), Ben Newman (BN), Forrest Norvell (FN), Waldemar Horwat (WH), Alan Schmitt (AS), Michael Ficarra (MF), Jafar Husain (JH), Lee Byron (LB), Dmitry Lomov (DL), Arnaud Le Hors (ALH), Chip Morningstar (CM), Caridy Patino (CP), Domenic Denicola (DD), Mark Miller (MM), Yehuda Katz (YK), Dmitry Soshnikov (DS), Kevin Smith (KS)

## 6. Test 262 Status
(Brian Terlson)

BT:
- Lots of activity happening on Github
- Harness improvements ongoing, unblocking usage by polyfill authors is a top priority
- Missing collateral for ES6 features
- Move all tests to folders based on feature name

- - Improves use and navigation
- - Problem is that 15k tests need to be assigned to folders
- - 10k are categorized so far
-

AWB: How will we know when we have enough tests that cover a reasonably minimal amount

BT: Existance tests?

DD: When all implementors submit their suites

AWB: Focus on edge cases?

...

## 4.11 Performance issue: `Object.defineProperties`, `Object.create`, `Object.assign`.
(revisit)

AWB: (Recap)

MM: After consideration of the issue at hand, and in keeping with principle of least surprise: there is no other place where the spec defines similar behaviour. All other speced action before propagating a throw, such as the "return" from for-of loops, is cleanup action rather than continuation of the original operation. Given that the order in which the properties are tried is deterministic and with the balance of the other arguments, I agree that the first error should throw at the point of exception.

#### Conclusion/Resolution

- Remove pendingException semantics
- Error throws at point of exception

## 4.12 Should WeakMap/WeakSet have a .clear method? (MarkM)
(Mark Miller)

MM: In the absense of clear, we have a security property: the mapping from weakmap/key pair value can only be observed or affected by someone who has both the weakmap and the key. With clear(), someone with _only_ the WeakMap would've been able to affect the WeakMap-and-key-to-value mapping.

#### Conclusion/Resolution

- Remove `clear` from WeakMap and WeakSet

## 4.7 Clarify the syntax reserved?
(Allen Wirfs-Brock)

AWB: What did we reserve?

WH: Like this but curious about how it's done.

WH, BE: (discussion re: specifics of BindingIdentifier grammar)

WH: Will validate the grammar by January

AWB: The colon-less form of object literal

Various: Discussion of the annoying ambiguities in using colons to add types to destructuring patterns.

#### Conclusion/Resolution

- No colon after BindingIdentifier
- Jeff Morrison and Brian Terlson to chamption refinements

## 5.8 Map.prototype.map and Map.prototype.filter (spec) + Set
(Dmitry Soshnikov)

request slides

DS: Map.prototype extensions
- map
- filter
- more...

These exist on Arrays, hopefully this is straight forward

Proposed spec: https://gist.github.com/DmitrySoshnikov/a218700746b2d7a7d2c8

MM: Similar to parallel js, lets keep an eye on the higher order operations to ensure that operations remain parallelizable

Design Choices

- Directly on Map.prototype
- On generic "map-like"? %CollectionPrototype%
- Binding `::` operator on iter tools?

User-level API

1. Simple value map:

```js
new Map([["x", 10], ["y", 20]]).map((v, k, m) => {
  return v * 2;
});
=> [["x", 20], ["y", 40]]
```

2. Entries map:

```js
new Map([["x", 10], ["y", 20]]).map((v, k, m) => {
  if (k == "x") return ["z", v];
  return [k, v * 2];
});

=> [["z", 10], ["y", 10]]
```

WH: Common Lisp has a menagerie of map-like functions. One of the most useful ones is one that allows the per-element function to decide to skip an element instead of always producing one.

YK/DD: (discussion re: Ruby style map operations)

MM: If you want to get the default, without being verbose

DD: Put a symbol on the map this[@@collectionConstructor] to find the right constructor

AWB: Smalltalk uses an abstract above that has a species property to determine what to create.

Discussion of lazy mode

YK: Lazy: do not accumulate intermediaries. Non-lazy: accumulate

YK/DD: Not important to determine this right now.

WH: How would you directly turn a map into an array, i.e. provide a mapping function that takes a (key, value, m) and produces array values?

?: Array.from(m.entries.map(...)) instead of new Map(m.entries.map(...))

WH: But that wouldn't work if, as was just being discussed, we got rid of the outer new Map by making it implicit.

- Prototcol design choices
(need slide)

Re: %CollectionProtocol% https://esdiscuss.org/topic/map-filter-map-and-more#content-33

AWB: Want to avoid intermediary collection creation

Possibly?
```js
import { map } from "itertools";

var nmap = omap::map((k, v) => [k + 1, v + 1]);
```

DD: we have the IteratorPrototype

- Overall

- - To correlate with `map.forEach` better to be `map.map` and `map.filter`, not `map::map`
- - Direct `Map.prototype` or `%CollectionPrototype%` - to be discussed
-

YK/DD/DS: (discussion of need for additions to Map.prototype)

- Concern about forEach as stopgap for transition to for-of
- DH/others: transition still ongoing
- BE: transition means "stopgap" code on web endures
- BE: anyway we want high-order "interior" iteration forever
- BE: JS is a TIMTOWtDI language

RW: These can be made generic enough for both Map and Set if `map` defaults to the thing that `mapEntries` is doing. Then works the same for both.

YK: No consensus on a map-specific solution.

DD: Don't think we should put anything else on `Map.prototype`, it should be on `map.entries()`?
- Not consistent in each case

DH: Disagree. Most common case gets the position of being default. Methods on class get to be the domininant, default case.
- We have keys, values, entries and need iterator

DD: This example seems to have values as the default

DH/YK/RW: Which is wrong.

MM: map vs mapEntries is differentiating in what it does with the callback's return value. These do not differ on the arguments they provide to the callback.

DD: Entries is the default

YK/DH/RW: Agree.

DS: Remove forEach?

WH: Why? I don't see anything wrong with it, and it's analogous to the other mappers.

BE: forEach is not just transitional. We need it.

RW: This isn't a trade.

DH/YK: (discussion re: adding a new CollectionPrototype to the chain?)

BE: Someone needs to do the work to see if this can be done. We want batteries included interators, higher order operations etc.

DH: Might be that we have several inheritance heirarchies? Similar or same methods?

YK: Can have a higher class that gets delegated to.

WH: (itemizes each of the cases: map, filter and forEach)
- How do you turn on type into another?

YK: This is what we were discussing re: the symbol

[several discussions at once]

AWB: An iterator that wraps another that says "iterate, but of a specific type".

BE: Need slides, DD doing that.

```js
Map.prototype.entries = function () {
  return new MapIterator({ collectAs: this.constructor[Symbol.species] });
};

class MapIterator extends Iterator {
  constructor({ collectAs }) {
    super();
    this[Symbol.collectAs] = collectAs;
  }
}

Iterator.prototype.collect = function () {
  return this[Symbol.collectAs].from(...);
};

myMap.entries()                    // proposal: you can remove .entries()
  .filter(([k, v]) => k % 2 === 0)
  .map(([k, v]) => [k * 2, v])
  .forEach(...);
  // or .reduce(...) also forces
  // or for-of
  // or .collectAs(Array)
  // or .collect(), which uses [Symbol.collectAs] as a default
```

(mixed discussion)

MM/DH: classic lazyness, "force" means done building up the lazy stuff and want to actually force it to execute and get a result now

WH: Not convinced, what goes in the ... on line 13? (i.e. does the collectAs constructor get passed values or [key, value] pairs? This makes the difference between getting the result ['x', 'y', 'z'] and [[0: 'x'], [1: 'y'], [2: 'z']].)

DH: collectAs is the the way to go from one final thing to another final thing.

Confirmed.

DD: call keys, you get keys, call values, you get values, call entries, get entries.

WH: Very confusing. In for loops "entries" means the format of your input, not what you'll eventually output.

DH: That's just our established terminology for items in Map or Set

WH: Yes, but it's used *on the input* to specify the kind of the output instead of the kind of the input.

Domenic adds to example: "Proposal: you can remove `entries()`"

```js
  myMap.entries()
    .filter(([k, v]) => k % 2 === 0)
    .map(([k, v]) => [k * 2, v])
    .map(([k, v]) => v)
    .collectAs(Array)
```

To generate example that collects just the values (not the pairs) into an array.

WH: To clarify, this doesn't collapse the intermediate results into any maps, so there would be no issues with .map(([k, v]) => v) duplicates getting undesirably coalesced?

DD: Right.

KS: Want to transform one interator into another, but dont want to use the defined function, want to use my own. How does this propagate this through?

DH: Syntax doesn't give you a nice way to do this, generator is just a function that takes an argument

KS: Yes.

DH: No `collectAs`?

YK: Provide a default

DH: But as Kevin says, this couldn't be implemented with Generators

AWB: Need to explore this, but the subclassability of generators comes into play here. Need to work through it .

DH: Can't instantiate a subclass of generator as a generator

...

MM: Agree that lack of map and filter is a point of confusion.

YK/DH: We all agree with this.

YK: If all in agreement, then we're in consensus to at least do _something_, but not sure what that is.
- Also needs to be subclassable

MF: have a reservation about the entire premise. What about the
- Non-empty list in? Guarantee out?

DH: If input type of filter is

MF: Two functor laws:

1. If map the id function over a functor, the functor that we get back should be the same as the original functor.
2. Composing two functions and then mapping the resulting function over a functor should be the same as first mapping one function over the functor and then mapping the other one.

(break)

#### Conclusion/Resolution

- Needs work.
- Sufficient issues
- Iterator prototype first

- - How does that translate to the collection api proposal itself.
-
-
-

## 5.9 Revisit Set API (possible exclusion of entries and keys)
(Dmitry Soshnikov)

DS: (proposes removal of second arg to set.forEach)

RW: The matching arguments in map.forEach and set.forEach were designed to match array.forEach for consistency.

JM: Is there any code that would rely on this?

AWB: If set only has value iterator, how do you create a map without keys or entries?

- If you remove this from ES6, how do we create a map from a set?

JM: Not pleasant.

AWB: Correct.

RW: What is the actual value of removal?

AWB/LB: There is value in consistency

#### Conclusion/Resolution

- No removal of argument to `set.forEach`

## Abstract references as a solution to LTR composition and private state
(Kevin Smith)

KS: Abstract referemces, gives a way to provide an abstraction for the records base component, for that record. Antoher way to think about it is virtual properties.

Using the IteratorPrototype Problem to illustrate

- - We want iterator methods!
    - - Left to Right composition
    - - Userland FTw
    -

Conflicting goals

- - Don't want users to extend built-in prototypes
- - Don't want to wait for TC39
-

A General Problem?

The user has an object.

- The user has a function
    - But...
    -

There's no convenient way to all the function as a method of the object.
Have to use right-to-left. All chains to the left.

A (More) General Problem?

The user has an object L

- The user has a function R
    -
    -

A General Solution

L :: R

base = L
referenced name = R

AWB: Are you evaluating R?

MM: R is an expression. Only the value it evaluates to is significant. In this regard, it is more similar to square bracket indexing than it is to dot.

Dereferencing behaviour is delegated to the _referenced_ _name_ object.

- `Symbol.referenceGet`
- `Symbol.referenceSet`
- `Symbol.referenceDelete`

Examples

```
/* gets evaluated approximately like */
L::R

    • R[Symbol.referenceGet](L)


L::R = expr
  R[Symbol.referenceSet](L)

L::R()

    • R[Symbol.referenceGet](L).call(L)

```

-

WH: Asks clarifying question about what goes into the base, name, and strict slots of the generated Reference object.

[KS flips to the semantics slide]

WH: Does this mean that the call fails whenever L === undefined?

MM/KS: Yes.

WH: Which is the base?

MM/KS: The base is L

WH: In your examples above, it's R

MM: In the desugaring, the references

MM: The value L is the reference's base. The value of R is the reference's "name"

MM: The semantics is normative in the proposal, not the desuraring.

WH: The desugaring is not helpful because it misleads to wrong conclusions about the proposal, such as the base and the behavior when L === undefined.

Built-In Support: Maps

Maps have inherent virtual property semantics:
(need to copy from slides)

Private State

```js
// Because the Map has "get", "set", and "delete"
const X = new PrivateMap();
const Y = new PrivateMap();

class Point {
  constructor(x, y) {
    this::X = x;
    this::Y = y;
  }
}

// Private State + Sugar
private X, Y;

class Point {
  constructor(x, y) {
    this::X = x;
    this::Y = y;
  }
}
```

YK: (question about inheritance, Yehuda can fill that in?)

MM: If you want protected state (visible through inheritance), you can define that.

MM: Think of `::` as more like "base["

YK: FWIW, some explicit private state syntax is desirable.

Weaknessess

The user must bring the virtual property object into scope as a variable. Fortunately, we now have better tools to manage scope:
- Modules
- Lexical Declarations
- Destructuring

More Information

- https://github.com/zenparsing/es-abstract-refs

Examples:
- https://gist.github.com/zenparsing/611b8788ff8ffcfcc20e

DD: This doesn't work outside of certain cases, the symbols are added

BE: T

DD: Wouldn't unreified private state, by analogy to spec-internal properties, have the same non-transparency across proxies that spec-internal properties do?

MM: The only non-transparency in both cases is over non-membrane uses of proxies, whose ad hoc nature cause many necessary non-transparencies. For example, applying Data.prototype.getYear to a proxy for a Date. But in a full membrane scenario, you'd apply a proxy for getYear to a proxy for Date, which turns into applying the real getYear to the real Date on the other side of the membrane. Non-reified private state would avoid the membrane tranparency issues in the same way. These issues only arise once you reify the designator into something first class which enables access to that private state.

YK: The only real disagreement is whether the mental model is private state or the model that [Mark just described]

MM: Lead to Relationships: impossible to each "private state designator.get state bearing object" (meta-mm: Rick, I don't understand what you recorded here and it doesn't ring any bells. If no one else remembers, please delete it as uninformative noise. Thanks.)

YK: everybody agrees on the private state semantics, but the argument is whether we need a special-case syntax for private state, or a general-purpose extraction that everyone will need to learn

AWB: Generalization that :: can have use on both sides.

JM: When you see the call example it makes more sense.

DD: But you're not "calling", you're "symbol.referenceGet"ing
- Ill serving private state
- Ill serving the bind

MM: Non reifable private state, such as in Java, pure textual. No transparency across membrange problem. The issue cannot arise becase theure is no ability to reify the designator of that slot.
- Nice conservative starting point because it directly reflects internal slots.

YK: There's a curse of expert knowledge in the room, need to step back and think of it in simpler / easier terms

MM: Given reification of the designator, we're only transparent across membranes if the access invokes the reified state designator with the state bearing object as argument, rather than vice versa.

DD: would work just as well, if this :: only worked with private maps. The generalization that :: can intercept an object with get, set, MM: (Added after discussion) Only transparent across membranes if :: also works with proxies for these maps. There's no need to make a special rule for proxies to these maps vs other proxies, so we shouldn't. If we don't then it also doesn't make sense to impose a private-map-only restriction in the first place.

WH: Main objection is to choice of the particular syntax used. :: would be more naturally used for either type annotations (due to the existing usage of : in object literals and destructuring) or namespace-like qualifiers (like in C++).

WH: (other than choice of syntax): interesting, but not fully investigated. fear of locking ourselves into using weak-maps-like mechanism for private state. This of course lets you stick "X" onto anything with `::`, not just objects of your own class.

AWB: But weakmaps have essentially grown into a thing that provides a separate set of "private slots", keyed on whatever object.

RW: Yes. We're using this pattern significantly in hardware abstractions, where the key is the instance respresnting some device and the "private state" contains the raw readings from the physical device. Keeps this data at hand, but away from user muddling.

CM: In practice, there will be a profusion of extended objects and people won't realize that they're using this `::` vs. ?

DD:

```js
class Point {
  private X, Y; // desugars to
  // new PrivateMap(Point) ?

  constructor(x, y) {
    this::X = x;
    this::Y = y;


      •     X.set(foo, bar);

  }
  method(o) {
    o::X = "foo"; // X.set(o, "foo");
  }
}

```

WH's objection, as presented by MM: method(o) can be called on any o, not just Points, and can be used to pollute other objects. This leads to issues via various confused deputy integrity bugs.

?: May want to attach private properties to any object.

MM/WH: Sure, fine to allow that case too, but it should require an affirmative step by the programmer. The path of least resistance should be simple private.

DD: What if `private` means that the map will only accept an instance of Point?

MM: In the

DH: There is a lot we want to achieve with "private" and I don't think we even have consensus on what we want to "private" to be. There is a lot machinery in this proposal. Missing a natural correspondance ot an existing mental model

YK: Or another language

MM: ? re: non-reified private state

DD: (updates)

```js
class Point {
  private X, Y; // causes constructor to do
  // X.set(this, undefined); Y.set(this, undefined);

  constructor(x, y) {
    this::X = x;
    this::Y = y;


      •      X.set(foo, bar);

  }
  method(o) {
    o::X = "foo";
    // if (!X.has(o)) {
    //   throw new TypeError();
    // }
    // X.set(o, "foo");
  }
}
```

DH: Hypothetically: if `::` was only usable for private. No generalization. Throwing this extra baggage nto class private is doubling down on private being only usable with class. I

Should be a requirement: usable outside of classes.

MM: accessor not be reified AND be usable outside of classes?

DH: Hypothetically, not reified.

MM: Non reified privacy indicator outside of classes?

DH:

MM: Doesn't reify? It should work

AWB: not class specific, you can put private in object literal.

DH: Not confident we have agreement of what private outside of class should do or look like.
examples:

(whiteboard image)

DH: If we decide second class, then additional set of concerns

MM: In the absense of reifying, we can faithfully explain what's going on in terms of static semantics.

DH: if first class, allows use as a general mechanism

MM: Proposes that private sugar (for declaring private properties) be defined only within classes. The language already has plenty of mechanisms to define it ad hoc in other contexts. On the other hand, the :: usage sugar would be usable throughout the language.

It's basically impossible to follow (rapid topic-hopping, analogous to frequency-hopping radios :))

When members review the contents of this discussion, they will have to fill in their own summaries.

BE: for this proposal, take the @ vs :: objection to heart, separate "privacy" from the L-to-R order.

#### Conclusion/Resolution

- Resolve the existing issues
- Separate "privacy" from left to right

Meta MM: Rick -- These conclusions do not reflect anything I remember being concluded from the discussion. As far as I remember, we all came to a much better joint understanding, but there was no attempt to articulate an agreed overall conclusion from the discussion.

## 5.7 Can security monitors reliably detect monkey-patching of primordials?
(Brendan, Michael Ficarra [invited expert])

MF: The problem space: "How can the developer be more confident in how their code will evaluate in any arbitrary environment"
- "First Class Realms"

DH: Sorry to interrupt, that work is done. It's part of ES7: Realms.
- Allows to execute a string of code.
- Completely isolated
- Creator can setup it's environment as they like

Can think of it similar to a DOM-less iframe or Worker.

(Questions about the analogy)

Synchronous, unmediated. Owner can reach in, etc.

DH: We should take this offline and compare notes.

MF: Envisioned as a function that has a directive, can cast values from the outside. Cannot reuse. Intended to run as an IIB

DH: What happens if there lexical references?

MF: Any lexical references are resolved.

DD: What if you have `[[Global]]Array === [[Realm]]Array`, which Array?

MM: Concern, if an adversary's code has run first. No code that runs after can know that it didn't run first.

DH:

MF: you can't rely on the trusted version of a built-in

MM: Code hosted on a server in a different domain, served in response to HTTP GET, not carrying the header `ACCESS-CONTROL-ALLOW-ORIGIN`
(does not allow cross-origin get). Script tag can still read the results of a cross origin get, without access-control header, but xhrs cannot.

MF: Hoping that the security benefits are better for spec analysis. Developer sanity, alleviate working around issues. Performance _may_ be a benefit, but I won't go into that.

MF: The benefit of this proposal over Dave's is that you're not doing string programming and have access to things around you.

DH: Operating under two assumptions:
Mine: assume cannot operate under an "pwned" environment
Other: assume can operate under an "pwned" environment

Only accident you have worry about is someone trashing `Reflect.Realm`. At that point, its game over anyway. If malice is the concern, then there is nothing you can do and we can't design that way.

MM: I agree in practice. There is a very narrow threat model in which the advesary can run first, but in which it cannot rewrite the defender's code. If the defender's code is hosted on a server in a different domain, and served -- in response to an HTTP GET -- without the ACCESS-CONTROL-ALLOW-ORIGIN, and if the adversay *has no other server of its own on the network* that can issue the HTTP GET to the defender's server. However, this is such a narrow and impractical threat model that we should not add any mechanism that support only it.

DH: Can't comment, it's out of my wheelhouse
- Dislike the proposal in that it creates a false security.

MM: Relies on assumption with no malicious server.

MF: Our motivation is to protect our customer's users.

MM: Assuming adversary code is already running in a browser that your bank software is about to load, the adversary can reach out from an adversary server, which fetches the code from the defender, and sends that code, via xhr say, to the adversary code running in the browser. It doesn't matter whether the adversary rewrites it on the server or the browser. The point is, the adversary will still succeed in the rewriting attack, and the defender will not know it is running in the matrix.

(Discussion re: threat models, tit for tat.)

YK: ServiceWorker: any HTTP request can be intercepted and pwned.

- Discussion that lead to GreaseMonkey
- extensions are more dangerous

BE: Should talk about this more.

MM: The APIs in the Realm must be set by the creator of the Realm

DH: Need to determine what things appear in a Realm when you ask for the default

MM: The convenience of "give me the host provided stuff" is to great, the rule should come from us.

DH: Could also maintain a document that specifies what things are in the default

MM: Can say that the default is "at least populated by the built-ins" and anything the host wants to include

DH: Unclear what things should not be included.

DH: Need to provide the list of what should be there and what should not be there.

#### Conclusion/Resolution

- Use the Realm API
- The idea that we can hang something on the Realm API to help protect against possibly-malicious extensions.
- The Realm initialization API probably should be enhanced to take a whitelist as argument, so it can include only the subset of the initial primordials enumerated by that whitelist.

## 5.2 Pure Functions

Casual discussion, not sure we've actually started this...

?: Why not define pure functions as passing function source code and then eval'ing it in an empty environment?

MM: It's similar, but defining pure function abstractions allows for better implementation optimization opportunities.

WH: What does structured cloning do with proxies?

Various simultaneously: It's a mess, implementation-defined, not expressible in ECMAScript, and/or breaks proxies.

DH: Separate out design problem of what a frozen environment is.

WH: The implementations discussed so far sound heavy-weight: structured cloning of parameters, possibly eval'ing function body. My impression was that pure function would be used to customize parallel code by mapping a function over a data structure in parallel, etc. These functions are often tiny such as (x, y) -> (x > y) and we'd want a very lightweight pure function model.

WH: In addition, a lot of the pure functions we'd want to map over data structures in parallel are actually pure closures. The closed-over state would be cloned.

MM: [presents write barrier model of thinking about pure functions. Mark all state allocated before the fork; any attempt by a pure function to modify that state would throw.]

WH: This brings up the library issue of what happens when a pure function calls a library method. C++11 faced the same issue when they added concurrency to the language. Multiple threads are allowed to read the same data unsynchronized, but if at least one thread does a write (and they aren't all synchronized) then the behavior is undefined. For simple accesses such as reading an int, that's clear. But what about calling methods on library functions? What do they do internally? Can readers keep and update caches? The C++ committee decided to stick with the prevailing practice and declare that any library functions that look like they're readers (in C++ indicated by declaring them const) shall not mutate any internals (at least not unless they use special language features to achieve proper synchronization). In ECMAScript we'd want to do something similar.

WH: However, the above makes proxies break pure functions and vice versa. Consider an otherwise transparent proxy of an object O that behaves the same as O but also counts the number of times each method is invoked. That proxy breaks all usage of O in a pure function.

DH: [Skeptical about parallel ECMAScript.]

MM: ES6 specifies that Function.prototype.toString(call) of a normal JS-written function results in an evaluable expression that, if evaled in an adequately similar environment, results in a function object with the same [[Call]] behavior as the original. Thus, given the original (SES-like) assumption that the primordials of the receiving environment are frozen, you can use the same trick as used by the old proposed "there" function http://wiki.ecmascript.org/doku.php?id=strawman:concurrency#there -- stringify the function on the sending side, and then safely eval it on the receiving side. For this eval (or call to Function constructor) to be safe, the receiving environment must be so much like SES that it may as well be SES.

MM: (In response to something by DH) If the primordials are naively frozen, you'll face the same usability issue we faced with SES -- the override mistake http://wiki.ecmascript.org/doku.php?id=strawman:fixing_override_mistake , resulting in innocent code like "Point.prototype.toString = ..." failing. To fix this, you'll need to tamper proof the primorials instead, replacing each data property with an accessor whose setter emulates how assignment would have worked in the absence of the override mistake.

# November 20 2014 Meeting Notes

Brian Terlson (BT), Taylor Woll (TW), Jordan Harband (JHD), Allen Wirfs-Brock (AWB), John Neumann (JN), Rick Waldron (RW), Eric Ferraiuolo (EF), Jeff Morrison (JM), Sebastian Markbage (SM), Erik Arvidsson (EA), Peter Jensen (PJ), Eric Toth (ET), Yehuda Katz (YK), Dave Herman (DH), Brendan Eich (BE), Ben Newman (BN), Forrest Norvell (FN), Waldemar Horwat (WH), Alan Schmitt (AS), Michael Ficarra (MF), Jafar Husain (JH), Lee Byron (LB), Dmitry Lomov (DL), Arnaud Le Hors (ALH), Chip Morningstar (CM), Caridy Patino (CP), Domenic Denicola (DD), Mark Miller (MM), Yehuda Katz (YK), Dmitry Soshnikov (DS), Kevin Smith (KS), Rafael Weinstein (RWS)

## 5.5 Array.prototype.includes() proposal to move to Stage 2.
(Domenic Denicola)

DD: Has all of the requisite items complete.
- Not ready for Step 3 because we need more patches landed.
- Ready for Allen to review specification text.

AWB: Move stuff to TC39 github?

DD: Yes.

AWB: For Ecma record, need PDF

#### Conclusion/Resolution

- Proceeds to Stage 2

## 5.6 Object.observe: proposal to move to Stage 3.
(Rafael Weinstein)

RWS: Move Object.observe to Stage 3?

RW: Can you list out the updates?

RWS: Conferred with Ember team, issues were theoretical

YK: Conceptual

RWS: Appears to be objections from YK and Ember, provided examples that should overcome

YK: Yes, but the actual solution does address the problem with the specification of the feature.
- There are process issues. I've repeatedly provided feedback and repeatedly dismissed.
- Technical Issues:

- - As a result of async callback, the decision was "don't ever drop anything"
    - - All intermediate values are kept, of all properties in all states, even if these are completely unnecessary.

EA: What if someone needs all the changes?

YK: There should be a mechanism explicitly say what you want

WH: What happens if you change a property repeatedly in a tight loop without switching microtasks?

YK: You will accumulate _every_ change

WH: And eventually run out of memory

DH: Similar concerns from Mozilla engineers

JM: Thoughts on a solution?

YK: Approach that's been discussed is an API mechanism that allows me to specify the properties I want.

EA: I think you're making a bigger problem of this, it's inherently dangerous

RW: Waldemar just illutrated a danger

YK: (gives more examples of problems)

RWS: Restate:
- Too many allocations, all of the change records.
- Retaining all of the changes

WH: Does it create a record for _every_ change, say a numeric property: 1, 2, 3, 4... One for each?

YK: Yes.

DH/YK: Overwhelming number of changes when properties you don't care about are also changed.

BT: On the other side, there are a lot of developers waiting for this feature. It's the number one requested.

YK: Not opposed to the feature, just want to get it right.

AWB: We need address issues that we'vle identified and gain consensus.

DH: Look at it in two parts:
- "I want an efficient way of being notified of changes that I care about"
- "I want a full change log"

Both are very useful, but the second is likely to give you more data than you ever need. The default should be minimal, but should allow for opting into the full change log.

BT: I was operating under the assumption that V2 could add the less expensive path on top of the more expensive path.

DH: I would prefer the less expensive path to start with

DD: No, wouldn't you want all the data and then add mechanism to limit?

YK: Want use case

RWS: If you have a generic write barrier on the array, you'd have to hold onto the original array and compute the difference, which is expensive. Providing the intermediary values significantly reduces that expense.

JM: Is there an objection to adding the configurability?

EA/YK: (discussion re: performance of configurability)

JM: But still no objection to configurability, which can be omitted and give you all the change records.

RWS: (provides background of the design)

YK: Changes:
1. List of properties that I want to observe
2. Don't hold onto all intermediaries, the change record has only the last value

AWB: Would you want multiple notifications if multiple changes occur before the record is processed?

YK: No, just the last

AWB: How do you know when you need to start making notifications again?

YK: Same as now, when the changes are delivered, start over.
- Recapping the concern and what's desired.

RWS: Concerned that what you want will start to unravel the design.

(Discussion of authoring responsibilities).

Agreement to work this out offline.

AWB: More concerns:
- I'd like the specification to be written in terms of ES6 mechanisms.
- Need to use the Job mechanisms, likely everything you need is there for you.
- Need to define the interactions with Proxy's (alot of "[[...]]" stuff that doesn't correspond to MOP in ES6)

RWS: My understanding was that Proxy's wouldn't do anything automatic with regard to `Object.observe`

EA: Right now the spec is defined to tie into DefineOwnProperty. If Plain Object or Exotic Array, then it works. Some other Exotic? No.

AWB: If you wanted to?

EA: No text yet, but could add this
- Exotic Objects are Exotic. How much do we want to polish that turd?

DD: DOM updated with internal slots. Boris Zbarsky has identified issues with holding all the values as well.

EA: Will add normative text for exotic objects. Will help update the DOM.

(discussion about notification handler changes and special cases that still need to be addressed)

#### Conclusion/Resolution

- Draft revisions for filtering
- Spec terminology updates
- Normative section about exotic objects

## 402 Status Update

(Rick Waldron)

RW: Next version will coincide with ES6. Future versions will coincide with future versions of 262.

RW: After ES6 the Yahoo team (Eric Ferraiuolo, Caridy Patino) is taking over.

RW: Intl 1 provides ...

RW: Intl 2 provides `Array.prototype.toLocaleString`

RW: Intl 2 will align subclassing semantics with ES6. Refactoring [[Construct]] and use [[CreateAction]]. Update to use new spec algorithms/abstract operations.

EA: There was previous talk about big action items for the next version of Intl.

RW: This spec only includes minor fixes and bringing the spec up to date. Will leave the big feature items for future versions of the Intl spec.

AWB: There will be an rf opt out for this at the same time as for the ES6 spec.

#### Conclusion/Resolution

- See above.

## Break out sesssions

- Object.observe
- Async generators/iterators. Get your yield on
- Value Objects

## 5.4 I/O Streams as part of the ES standard library
(Domenic Denicola)

DD: Presents: https://streams.spec.whatwg.org/

- Some suggestion to move into ES
- Specified as platform agnostic (browser and node)

- Possibly too narrow

MM: Abstraction is not data specific?

DD: Correct.

- Walk through back pressure

MM: Explain the "too narrow"

DD: Just that it might be a step too far

RW: agrees

MM: Is the specification bigger than it needs to be?

DD: No, it's as specified as it needs to be, but seems like a step too large

JH: Don't see a conflict with async generators

Discussion on the merit of different io mechanisms

DD: With modules, do we want to extend to cover many different aspects? fetch, etc.

AWB: Programming languages need basic IO? Yes. Need to think about the ES standard built-in library. This could easily be an Ecma spec for a standard module.

RW: Had this same conversation independently and propose this as a TC39 guided spec, not dependent on Ecma-262, but normatively referenced.

WH: Any requirements for asynchrony, execution turns?

DD: No. Specifically designed away from such

AWB: Prefer that TC39 handle these things to avoid too much platform specific design.

Discussion about appropriate groups to foster development. (Compare to Intl)

Discussion re: oversight?

DD: Joint deliverable?

RW: RFTG?

DD:

BT: Honest: The best way you'll get good participation from MS is in TC39, because the legal processes and agreements are in place to protect work here. We can work in w3c for same reason. Whatwg is less clear, and therefore not easy for us to be involved.

AWB: Very complex, legally. To be resolved for whatwg, they'd have to become a recognized standaUrds organization. In the legal/governmental recognition sense.

MM: We've provided plenty of reason why TC39 would take this, can you explain why you'd rather work in whatwg?

DD: Personal preference.

- Can provide snapshots for ecma if necessary

AWB: Not the issue, the issue is the legality of contributions/contributors.

DH: Hixie is also willing to work with companies on these things.

Heated discussion about specification ownership

MM: If the proposal is to allow the editor unilateral control, vs. committee refinement, consensus, and agreement, then the proposal should be withdrawn.

DD: Then withdrawn.

#### Conclusion/Resolution

- Withdrawn

## Async Generators
(Jafar Husain)

https://github.com/jhusain/asyncgenerator

JH: Propose to move to Stage 1

Major digression re: process.

...

DH: Want to see a comparative survey of how other languages and systems have approached this problem space. What does it look like, how does it work. Would like to return to providing this sort of detail.
- Rust has RFC process that nicely tracks feature work history and development.

Slides

```js
interface Iterable {
  Generator @@iterator(Generator);
}

interface Observable {
  Generator observer(Generator);
}

Array.prototype[@@observer] = function(generator) {
  var decoratedGenerator = Object.create(generator),
    done = false;

  ["throw","return"].forEach(method => {
    decoratedGenerator[method] = v => {
      var superMethod = generator[method];
      done = true;
      if (superMethod) {
        return superMethod.call(generator, v);
      }
    }
  });

  for (let x of this) {
    decoratedGenerator.next(x)
    if (done) {
      break;
    }
  }
  if (!done) {
    decoratedGenerator.return();
  }

  return decoratedGenerator;
}

[1,2,3][@@observer]({
  next: function(value) {
    console.log(value);
    if (value === 2) {
      this.return();
    }
  }
});

for (let x of [1,2,3]) {
  console.log(value);
  if (x === 2) {
    break;
  }
}

async function test() {
  for (let x on [1,2,3]) {
    console.log(value);
    if (x === 2) {
      break;
    }
  }
}

document.addEventListener("mousemove", function next(e) {

});
```

DH: Concerns about symmetry

JH: Any iterable can be an observable.

DH: (Clarifying gist) If I am a data source (collection), I call myself iterable if I want to some consumer to "pull" values. If I am a data source, I call myself "observable" if I want to "push" my values.

Examples of `for-on`, `@@observer`

BE: I wouldn't try for Stage 1 quite yet. Let's see more explanation in the form of examples (gists, etc).

DH: need to find more ways to discuss this stuff other than the output of a transpilation.
- Need it to explain at a user level

DS: Needs a real use case

MMj, WH: These help.

DH: Need to look at this from a "what is the problem space", then try to solve that. There are two competing solutions (`async function *`) and this

KS: (explains his approach with async iterator)

```js
Interator<Promise> or Observable

asyc function * v() {
  let values = toAsycInterable(new Websocket(...));
  for (let promise of values) {
    let value = await promise;
    yield value + 1;
  }
}
```

Discussion re: back pressure

WH: Does either approach do buffering by default? [Prefer to transfer/work on one item at a time unless buffering is explicitly requested to avoid blowing up if a turn takes a while.]

JH: (shows an example of `for-on` to illustrate support for back pressure)

More discussion about backpressure

#### Conclusion/Resolution

- Comparison documentation

Thanks to Paypal for the meeting accomodations. Thanks to Ecma.