Minutes of the: *39th meeting of Ecma TC39*

in: *San Francisco, CA, USA*

on: *8-10 April 2014*

1 Opening, welcome and roll call

1.1 Opening of the meeting (Mr. Neumann)

Mr. Neumann has welcomed the delegates at Mozilla in San Francisco, CA, USA.

Companies / organizations in attendance:

Apple, Mozilla, Google, Microsoft, Intel, eBay, jQuery, Yahoo!, Netflix, Facebook, Samsung (guest,)

1.2 Introduction of attendees

John Neumann – Ecma International

Istvan Sebestyen – Ecma International, via Phone, part-time

Jafar Husain – Netflix

Filip Pizlo – Apple

Mark Hahnenberg – Apple

Oliver Hunt – Apple, part-time

Mark Miller – Google

Douglas Crockford – eBay

Allen Wirfs-Brock – Mozilla

Nicholas Matsakis – Mozilla

Waldemar Horwat – Google

Dimitry Lomov – Google

Eric Ferraiuolo – Yahoo!

Caridy Patino – Yahoo!

Juan Dopazo – Yahoo!

Rick Waldron – jQuery

Yehuda Katz – jQuery

Dave Herman – Mozilla

Brendan Eich (invited expert, part-time)

Rafael Weinstein – Google

Jeff Morrison – Facebook

Sebastian Markbage – Facebook

Luke Hoban – Microsoft

Brian Terlson – Microsoft

Ben Newman – Facebook

Rick Hudson – Intel,

Norbert Lindenberg – Mozilla, part-time

Alex Russel – Google, part-time

Tatiana Shpeisman – Intel, part-time

Brandon Benvie – Mozilla, part-time

Seo-Young Hwang – Samsung (invited expert)

Sung-Jae Lee – Samsung (invited expert)

Satish Chandra – Samsung (invited expert)

Domenic DeNicole – W3C (liaison)

Mathias Bynens – Opera (invited expert)

Jaswanth Sreeram -

 # April 8 2014 Meeting Participants:

Doug Crockford (DC), Brian Terlson (BT), Luke Hoban (LH), Dmitry Lomov (DL), Waldemar Horwat (WH), Allen Wirfs-Brock (AWB), John Neumann (JN), Rick Hudson (RH), Rick Waldron (RW), Eric Ferraiuolo (EF), Jafar Husain (JH), Jeff Morrison (JM), Sung-Jae Lee (SJL), Seo-Young Hwang, Mark Honenberg (MH), Caridy Patino (CP), Yehuda Katz (YK), Niko Matsakis (NM), Ben Newman (BN), Filip Pizlo (FP), Satish Chondra (SC), Domenic Denicola (DD), Mark Miller (MM)

 # April 9 2014 Meeting Participants:

Doug Crockford (DC), Brian Terleson (BT), Luke Hoban (LH), Dmitry Lomov (DL), Waldemar Horwat (WH), Allen Wirfs-Brock (AWB), John Neumann (JN), Rick Hudson (RH), Rick Waldron (RW), Eric Ferraiuolo (EF), Jafar Husain (JH), Jeff Morrison (JM), Seo-Young Hwang (SYH), Mark Honenberg (MH), Caridy Patino (CP), Yehuda Katz (YK), Niko Matsakis (NM), Ben Newman (BN), Filip Pizlo (FP), Norbert Lindenberg (NL), Sebastian Markbage (SM), Mathias Bynens (MB), Rafael Weinstein (RWS), Jaswanth Sreeram (JS), Alex Russell (AR), Istvan Sebestyen (IS), Mark Miller (MM), Tatiana Shpeisman (TS), Brandon Benvie (BB), Brendan Eich (BE)

 # April 10 2014 Meeting Participants:

Doug Crockford (DC), Brian Terleson (BT), Luke Hoban (LH), Dmitry Lomov (DL), Waldemar Horwat (WH), Allen Wirfs-Brock (AWB), John Neumann (JN), Rick Hudson (RH), Rick Waldron (RW), Eric Ferraiuolo (EF), Jafar Husain (JH), Jeff Morrison (JM), Seo-Young Hwang (SYH), Mark Honenberg (MH), Caridy Patino (CP), Yehuda Katz (YK), Niko Matsakis (NM), Ben Newman (BN), Sebastian Markbage (SM), Mathias Bynens (MB), Rafael Weinstein (RWS), Mark Miller (MM),

1.3 Host facilities, local logistics

On behalf of Mozilla **Allen Wirfs-Brock** welcomed the delegates and explained the logistics.

1.4 List of Ecma documents considered

Ecma/TC39/2014/007 Minutes of the 38th meeting of TC39, San Jose, January 2014

Ecma/TC39/2014/008	Non-Ecma-member contributions (“text” and “software”) to the TC39 RF TG Work
Ecma/TC39/2014/009 (Rev. 1)	Venue for the 39th meeting of TC39, San Francisco, April 2014
Ecma/TC39/2014/010	Agenda for the 39th meeting of TC39, San Francisco, April 2014
Ecma/TC39/2014/011	TC39 chairman's report to the CC, April 2014
Ecma/TC39/2014/012	Draft Standard ECMA-262 6th edition, Rev. 23, April 5

2 Adoption of the agenda (2014/009-Rev1)

Agenda for the: 39th meeting of Ecma TC39

```
in: San Francisco, California, USA
on: 08 - 10 April 2014
TIME: 10:00 till 17:00 PST on 08th and 09th of April 2014
      10:00 till 16:00 PST on 10th of April 2014
LOCATION:
    Mozilla Foundation
    2 Harrison Street
    San Francisco, CA 94105
    USA

CONTACT:
    Allen Wirfs-Brock <allenwb@mozilla.com>
    Brendan Eich <brendan@mozilla.org>
```

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 (2014/00?)
3. Approval of the minutes from Jan. 2014 (2014/0??)
4. ECMA-262 6th Edition
   i. Review Latest Draft (Allen)
   ii. Consider not throwing a TypeError when .next(value) is called on a newborn generator (Ben Newman)
      a. [Arv proposed we agree on es-discuss, after I misremembered agreeing last meeting. Let's ratify quicky. /be]
   iii. Object.assign and multiple source objects (Rick Waldron)
      a. http://esdiscuss.org/topic/object-assign-with-several-source-objects

   b. http://esdiscuss.org/topic/object-mixin-source-target-arguments-changed
   c. http://esdiscuss.org/topic/object-mixin-object-assing-with-multiple-args
iv. Object.getOwnPropertyDescriptors(O) (Rick Waldron)
   a. https://gist.github.com/WebReflection/9353781
v. Array.prototype.contains (Rick Waldron)
   a. https://mail.mozilla.org/pipermail/es-discuss/2014-February/036386.html
   b. https://mail.mozilla.org/pipermail/es-discuss/2014-March/036573.html
vi. Should parseInt handle octal and binary integer literals (Rick Waldron, Erik Arvidsson)
   a. https://bugs.ecmascript.org/show_bug.cgi?id=1585
vii. Consider changing [[OwnPropertyKeys]] return type from iterator to array and include invariant checks (Tom VC)
   a. http://esdiscuss.org/topic/object-getownpropertydescriptors-o-plural#content-34
viii. Finalize default argument semantics (arguments object mapping, TDZ, scoping) (Brian Terlson)
ix. `Object.prototype.__proto__ = proxy` & getPrototypeOf trap returning itself (Brian Terlson)
x. Module Feedback from MSFT (Brian Terlson)

5. ECMA-262 7th Edition
   i. Object.entries(), Object.values() (Rick Waldron)
      a. http://esdiscuss.org/topic/object-entries-object-values
   ii. Typed objects spec progress report (Dmitry Lomov, Niko Matsakis)
   iii. Extensible Structured Clone (Yehuda Katz, Niko Matsakis)
   iv. Decorators (Yehuda Katz) 10am April 10
   v. Parallel JS Spec Report (Rick Hudson) 11am April 9
   vi. Async/await topics (Luke Hoban)
6. Test 262 Status
7. ECMA-404 and IETF interactions (Allen, Istvan)
8. Report from the Ecma Secretariat (2014/011 Chairman's report)
   i. Istvan updates on GA, TC39, and IPR matters
9. Date and place of the next meeting(s)
   i. May 20 - 22, 2014 (Menlo Park - Facebook)
   ii. July 29 - 31, 2014 (Redmond, WA - Microsoft)
   iii. September 23 - 25, 2014 (Boston, MA - Bocoup)

iv. November 18 - 20, 2014 (San Jose - PayPal)

10. Closure

The agenda as posted on Github was approved.

3 Approval of the minutes from the January 2014 Meeting (2014/007)

Ecma/TC39/2014/007, the minutes of the 38th meeting of TC39, San Francisco, January 2014 was approved without modification.

4 On the work of the TC39 RF (Royalty Free) Task Force

As requested by the Ecma GA in December 2013 in Las Vegas since the January 2014 TC39 meeting the TC39 RF TG is operational. Since the April 2014 TC39 RF TG meeting all TC39 members interested in participating should have registered, otherwise they may only participate in the TC39 Plenary meetings but not in the TC39 RF TG, where the current technical work is being progressed. Invited experts have to fill in a special invited experts form where they express their willingness to follow the policies of the TC39 RF TG.

At the request of the TC39 RF TG, TC 39 met on 4/9/2014 and voted unanimously to designate the draft of ES 262 reviewed at its next meeting (May 2014) as the draft for Opt-out possibility. The Out-out 60 days period will be ending in approximately end of July, 2014. This will meet the requirements for TC39 action to provide an opt-out period prior to the approval of the next version of ES-262 (6). The plan is that everything in ECMA-262 so far, plus the new draft until May 2014 should be subject of an opt-out. After the opt-out period if no one opts-out the group will consider the latest draft version of ES6 as RF. Before the May 2014 TC39 meeting the Ecma Secretariat will formally notify the members of TC 39 of the start and end of the opt out period per the procedures identified in TC39 operating rules.

5 For details of the technical discussions see Annex 1

6 Status Reports

6.1 Report from Geneva

6.1.1 Brief report about the work of the IPR Group on 3rd party text- and software contributions

Mr. Sebestyen said that the IPR ad hoc group has finished its work on a combined text- and software contribution solution from non Ecma members. The CC has also agreed to it. So this policy change is now under letter ballot in Ecma and will end in May. Then ES 6 tests can be submitted both by Ecma members and non-members.

6.1.2 Ecma Text copyright suitability for TC39

The IPR committee has started to discuss the matter. The first version of a "Frequently Asked Question" has been prepared and published:

http://www.ecma-international.org/memento/Ecma%20copyright%20FAQ.htm

This FAQ explains what is allowed from Ecma's point of view and what not. This FAQ is a living document, so comments, additions, changes etc. are welcome. The FAQ does not contain the issue yet, under what conditions it is allowed to take ideas from the current TC39 specifications and to include it into a completely new and different language. So this discussion is not finished yet.

6.2 Json

After successful TC voting the JSON ballot (ECMA-404) has been completed on October 6, 2013. Since then the standard can be downloaded from the Ecma website and it is rather popular.

In the January 2014 TC39 meeting it was decided not to fast-track ECMA-404 to JTC 1. The April 2014 TC39 meeting confirmed that.

Also the current mode in Tc39 is, leave Edition 1 of ECMA-404 as it is. So likely no 2nd Edition of that standard will be prepared and voted at the end of 2014.

Mr. Sebestyen reported that the IAB (and through that IETF) would like to establish formal liaison status with Ecma and TC39. This has been approved by the IAB but formally not communicated yet. The current plans are that IETF is entitled to send liaison representative to Ecma TC39 meetings (who can contribute to Ecma work according to Ecma and TC39 rules), and Ecma TC39 can also send liaison representative to IETF meetings (who can contribute to IETF work according to IETF rules). Joint publications are at the moment not planned.

7 Date and place of the next meeting(s)

Schedule 2014 meetings:

- May 20 - 22, 2014 (Menlo Park, CA - Facebook)
- July 29 - 31, 2014 (Redmond, WA - Microsoft)
- September 23 - 25, 2014 (Boston, MA - BoCoup)
- November 18 - 20, 2014 (San Jose, CA - PayPal)

8 Any other business

TC39 during its session on 4/9/2014 unanimously agreed to support the recommendation of the chair of TC39 for Ecma to determine a way to allow **Mr. Brendan Eich** (one of the key experts on TC39 subject and work) to participate in TC39 for as long as he wants at no cost. TC39 recognizes that no voting rights would convey with this decision to extend honorary membership.

TC39 also requests that Ecma awards for **Dr. Waldemar Horwat** and **Mr. Brendan Eich** awarded at the next Ecma GA meeting.

9 Closure

Mr. Neumann thanked **Mozilla** for hosting the meeting in San Francisco, the TC39 participants their hard work, and **Ecma International** for holding the social event dinner Wednesday evening. Special thanks goes to **Mr. Rick Hudson** for taking the technical notes of the meeting.

Annex 1

Technical Notes (by Rick Hudson):

 # April 8 2014 Meeting Notes

Doug Crockford (DC), Brian Terleson (BT), Luke Hoban (LH), Dmitry Lomov (DL), Waldemar Horwat (WH), Allen Wirfs-Brock (AWB), John Neumann (JN), Rick Hudson (RH), Rick Waldron (RW), Eric Ferraiuolo (EF), Jafar Husain (JH), Jeff Morrison (JM), Sung-Jae Lee (SJL), Seo-Young Hwang, Mark Honenberg (MH), Caridy Patino (CP), Yehuda Katz (YK), Niko Matsakis (NM), Ben Newman (BN), Filip Pizlo (FP), Satish Chondra (SC), Domenic Denicola (DD), Mark Miller (MM)

(one late arrival, need name)

Welcome

JN: (open remarks)

AWB: logistics

Introductions

See attendee list

Agenda

https://github.com/tc39/agendas/blob/master/2014/04.md

JN: Agenda approved with changes.

Minutes

JN: Approval of minutes from Jan 2014, Ecma document 007

4.1 Review Latest Spec Draft

(Allen Wirfs-Brock)

Slides: [April-meeting-status-ES6-spec.pdf] (https://github.com/rwaldron/tc39-notes/blob/master/es6/2014-04/April-meeting-status-ES6-spec.pdf)

AWB: Updates to formatting, etc.

Summary of changes:
http://wiki.ecmascript.org/doku.php?id=harmony:specification_drafts#august_23_2013_draft_rev_17

YK: Concerns about let bindings in function parameters

...Let's make sure that anything in ES5 that does not have a let binding doesn't produce such, to avoid interop hazards.

BT: In IE11, you can't create a let binding whose identifier is the same as a function parameter

AWB/YK: Then this is a spec bug. (AWB: not clear what this is referring to, the behavior described above for IE11 sounds correct)

BT: Even `var x; let x;` is an error

AWB: There are lexical bindings that the closure captures and then there are static semantic rules that say you cannot have a `let` in the body that are same name as a `parameter`

... We need

LH: `let` is not designed to directly replace `var`, it's a benefit that these static rules exist.

RW: Agree (`let` is not the new `var`)

AWB: Cannot `import x` and then `let x`, this would be a redeclaration.

...There is an item to discuss `catch(e)`

WH: What are the ContainsExpressions rules in the function declaration instantiation spec for? As far as I can tell, they're an optimization to take out an invisible environment. They appear to have no visible effect.

AWB: Yes, that's invisible. Went back and forth about whether to express such optimizations in the spec.

AWB: (moving on)

- Lookahead grammar restriction created, to disambiguate `new super()` (AWB: but they aren't right yet. Stills needs some more work)

- Lookahead `let` restrictions added: IterationStatement (see notes)

- Temporary change return default for missing class constructor: Reverted.

AWB: First call to a generator (half of 4.ii on the agenda)

- Eliminate throw if argument is passed

- Argument is ignored and inaccessibe

- Differing recolltions on January discussion

- Most compelling reason: creates unnecessary difference between generator and manual implementaion of equivalent iterators.

4.2 Conclusion/Resolution

- removal of the thrown exception when a value is passed to next() for a newborn generator

- further discussion to be had by Ben Newman and Dave Herman re: what to do with that value.

AWB: (continuing)

```js
Array.from({0: 0, 4:4, length: 5});
```

... didn't produce a sparse array. Changed in Rev23.

Current spec says: in no way can `Array.from` produce a sparse array.

RW: This is good and makes the behaviour consistent with the iterator path in `Array.from`

YK: Generally, sparse arrays are dead.

(no opposition)

AWB: (continuing) Corrected RegExpExec so it correcly translates the match state of full unicode RegExps back to UTF-16 cature values and endIndex

WH: If you have a composed character, is it still considered two characters?

AWB: Yes, it operates at the level of code points or code units. Not a higher livel such as composed characters

YK: (requesting more information about the the ES5 compatibility change item added to Annex D)

RW: (note: it appears to be missing, Allen, please fill in/follow up?)

AWB: Call for Spec reviewers

BT: Need to create a "sign up" for task assignment for reviewing sections

AWB: Would like to see two TC39 members reviewing each of the sections

... September is the _end_, otherwise we risk delaying for another 6 months. The final reviewing needs to be done between now and July.

MM: Last meeting I said I was going to organize a security review. There are no draft implementations of the Realm API which will make the exploratory process of analyzing the system useless. There has to be some time to poke at a draft implementation before committing.

YK: We should wait to delve into this issue (modules) with Dave present

AWB: Agreed

MM: Putting the Realm aside, I have no concerns for the modules specification

JM: What exactly is missing, what can we work on to help make progress.

LH: What about the JS Loader projects that exist?

YK: There is no way to do the Realm API with those, however the Loader implementations

JM: Can `contextify` be used to implement Realm?

MM: There may be way to use this to discover holes. Let's take a look at it. There may be a way to expose issues in the same way.

AWB: There are changes to the Realm API, since the last meeting.

LH: Are there further expected changes?

AWB: One of the issues is the eval support is now at the Realm level. One of the change we got down from 4 methods to three methods re: eval.

YK: We should avoid discussing this without Dave present?

LH: There are API changes expected in Realms? Are there API changes in Loader?

AWB: Hopefully

LH: That's what expected

YK: Special casing class and function to create bindings at the parent level for `export default function foo` and `export default class foo`. We should special case those expression forms to behave like declaration forms

LH/BT: Yes.

AWB: If you want to treat it like an expression, put it in parens.

WH: What is `export default`?

YK: We've discussed this

AWB: (recapping the utility and specification of default exports)

WH: Can only have one default?

AWB/YK: Yes.

AWB: It's for modules that you want to export without a specific name.

YK: Compatibility with a desirable style of programming.

LH: We need to lock down API changes as soon as possible.

YK: It's possible to punt on Realm API (to ES7). Dave specifically designed this to separable.

MM: Where would you draw the line? If Realms were on the opposite side of the line, where would you draw the line?

AWB: I could draw a line: No API.

YK: Draw a line? No public Realm API, but the semeantics are still there.

LH: Our success critically relies on Loader API, this is where requirements are resolved.

...Discussion about Loader API

LH: All of the multiple phase complexity is needed

Discussion about Guy Bedford's work, es6-module-transpiler, traceur, etc.

YK: I'll open discussion with Guy to pinpoint hardships in implementation

LH: This will be helpful

EF: (to Brian) you're presenting implementation report?

BT: No, just feedback from TypeScript

Open Issues

AWB: for of/in, initialization expression scoping?

```
{
  let x = [0, 1];
  for (let x of x) console.log(x);
}
```

WH: Wouldn't it be the same as saying `let x = x`?

AWB: should be

- Current spec: of/in expression evaluated in enclosing scope:. Log: 0 1
- Possible alternative: extra scope with uninitialized x. Throws TDZ error

LH: This example has two scopes, the alternative has three?
...for creates a scope
...Why do you want to change?

AWB: There have been arguments that say `x` isn't yet initialized in the `for` scope

WH: Is the question: creating the scope before or after?

AWB: We have to create an extra scope where `x` is dead.

(ah-ha moment.)

AWB: Need input from implementors

MM: If no one has a strong reason, I suggest going through the second approach.

YK: err on the side of being more useful to the programmer

MM/AWB: Yes

WH: There might not be a dead zone violation

MM: it would be a dead zone violation in the second bullet
... There is only no dead zone violation if you don't touch it.

AWB: We need a consistent approach
... Already have this extra scope

AWB: Need to go back, we may initialize that outer binding and maybe we shouldn't

Conclusion/Resolution
- Be consistent
- Look at the initialization scope and be sure to intentionally _never_ initialize the for bindings in that scope
- Solution to open issue: extra scope with uninitialized x. Throws TDZ error

var hoisting and catch parameters

AWB:

```js
catch (x) {
  var x = 42;
```

}
```

Discussion about the existance of this code in real world.

BT: This exists in code found in the top 1k sites

AWB: i can remove the static semantics rule that's specifiically for catch, that says this is an error. In that case, it will be just like ES5, there will be a var and the var will be hoisted. Need to look at various place where simplifications of runtime semantics that assume this has been applied.

YK: Question about the actual binding.

WH: The same that happens inside a `with` that has captures a property with the same name as a var—the var hoists out.

YK: What is the negative effect of creating a binding that's undefined?

...Change incompatibility should have a higher bar than "it complicates the spec"

MM: Breaking changes aren't worth

WH: Keep the catch as a `let` binding, but make it not an error in Annex B, just for catch (not for let's in general)

LH/AWB: I like this

FP: It seems better to keep the existing behaviour?

MM: Is Annex B, normative? Normative optional?

YK: Confirms that Annex B is _required_ for browsers but normative/optional for non browsers

MM: Ok, no issue then.

Agreement that browser implementation will lead and others will follow for compatibility.

YK: Also need to make sure that this is not an error:

```js

```
catch(x) {
  function x() {
  }
}
```

AWB: This is an error

WH: Any reason for us to believe this exists in the wild?

BT: Looked for this pattern, with no findings, but absence in top 1k doesn't mean non-existence.

WH: Disallow this and see if there is any pushback.

Conclusion/Resolution
- Keep the catch as a `let` binding, but make it not an error in Annex B, just for `catch` (no other)
- If redeclaration with `let` or `const` is still a redeclaration error (this is no change)
- Continue to disallow function declaration in catch

MM: This could only exist in non-strict code.

YK: The same behaviour as:

```js
{
let x;
let x;
function x() {}
}
```

Confirm.

Presentation by Samsung Representatives

Slides: [ecma-tc39-talk-v1.0.pdf](https://github.com/rwaldron/tc39-notes/blob/master/es6/2014-04/ecma-tc39-talk-v1.0.pdf)

Presentation focus on intent of committee participation

- JS on wearable devices
- JS on microcontrollers (small memory devices)

Interested in building a small footprint JS engine
- Wearables and IoT
- A subset?
- Possibly as an open source project

We are here to ask your opinion about approach

- multi level subset of profile of ECMAScript
- is it reasonable to define a subset
- can this be covered can this committee

WH: 48K of RAM? That's the same as an Apple II.

WH: Just supporting Unicode takes a lot more memory than that.

MM: What about running existing code?

RW: (share experience of using JS subset in microcontroller environments)

SC: (more about goals of the platform, re: HTML/CSS)

LH: HTML and CSS are massive

RW: Platform dependencies themselves assume specification compliance

WH: One of the reasons for selecting a subset of JavaScript is that you can write applications for the device that also run in the browser.

WH: While libraries are big, a subset of the "good parts" of ECMAScript would also be useful for reducing memory usage.

?: How so?

WH: Some of the features of ECMAScript (such as peculiarities of eval) pretty much require efficient implementations to do speculative optimization and then have a mapping and a way to back out if the assumptions fail. That mapping and the extra copy of the code for deoptimization costs memory.

YK: So don't optimize. Use an interpreter.

WH: Lack of optimization would waste battery power on small devices.

RH: points about starting and reducing

LH: It's important to consider what are you willing to trade?

Mixed discussion about ES versions that might be considered subsets themselves.

AWB: There is clearly concern here, but it would be valuable to have you included and there are people here that are interested in small device platforms.

[[SetPrototypeOf]] circularity invariant

(Allen Wirfs-Brock)

- If Proxy involved, impossible to enforce on prototype chain
- Eliminate the invariant?
- Is there some weaker invalriant we might replace with?

LH: How?

AWB: getPrototypeOf can lie while the invariant is being checked

MM: Are there any recommendations beyond dropping the invariant?

FP: (Question about requirements)

MM: Any defensible options other than dropping the invariant?

AWB: No

WH: Yes. To prevent proxy cycles, something in the proxy (perhaps the target's prototype, perhaps something else) would record what the proxy thinks the prototype is. GetPrototypeOf does not run user code and only returns the current stored value. SetPrototypeOf runs user code and can change the stored value. The circularity check occurs at the time of the store.

AWB: Possible for ordinary objects, if the prototype chain only consists of ordinary objects, there must not be circularities.

Discussion about Proxy trap handlers for `getPrototypeOf` and `setPrototypeOf`

MM: Storing state related to the target

BN: In the same way there is RangeError for call recursion, is there a way to RangeError on length of prototype walk

MM: Waldemar's rule will give you the error earlier.

WH: It's better to pay the runtime cost in the less frequent cases that change prototypes rather than having to pay it every time a prototype chain is read.

NM: The more user code we can get out of the prototype look up, the better.

AWB: Ask Tom about intervening when user sets prototype

YK/MM: Agree.

Conclusion/Resolution

- Prevent proxy cycles. If the target is a proxy, the proxy target prototype records what it thinks the proxy prototype is, and returns the targets prototype and setPrototype is never called

- Whenever you change a prototype, you run a circulartity check immediately.

(the following is about whether [[SetPrototypeOf]] should be allowed modify the prototype of Object.prototype)

BT: Does the proxy's handler have get trap

WH: If it does, there is an infinite loop

MM: Even without Object.prototype, you can create this same loop

- When you mess around with prototypes, you can render the entire environment inoperable.

BT: Setting Object.prototype to a Proxy?

WH: I agree with you but not worried about it.

YK: Proxy is the only way to implement exotic behaviour in user code.

MM: Too late to draw the line about what prototypes can be assigned a Proxy.

(This addressed agenda item 4.9)

Promise then issue

```js
p.then(42, "43")
  .then(false, new Map);
```

- Error or default argument values if actual argument is not callable?
- if error throw or asynch error?

AWB: Area of disagreement that Andreas brought up:
- The arguments to then are supposed to be functions,
- If the first argument isn't provided and not explicitly `undefined` or `null`,
What do you do with non-callable arguments?

Possible solutions:
- The same thing if you provide no argument?
- Throw?

Currently throws.

LH: The current behaviour is throw

MM: Domenic wants it to not indicate an error?

AWB: Correct.
Here's an example:

```js
p.then(expr && callable);
```

```

If `expr` evaluates to false, this non-callable false value is passed.

AWB: The claim is that all libraries accept the non-callable and just use the default.

JH, WH: Do we unwrap eagerly or lazily?

AWB: We'll get to that issue.

LH: WinJS throws

JH: (jafar, this point was about sync and async errors, can you fill in?)

YK: (points about errors that are logged to the console)

Discussion about IDL

YK: There are two development modes, one that wants all errors logged and one that does not.

WH: Don't want some errors sync and some errors async

(Note: the point of this discussion is whether `p.then(non-callable)` should throw synchronously)

AWB reviews spec and finds other points in which then throws synchronously

LH: Where do you draw the line that differentiates what is a runtime error and a construction time error

JH: Why make a distinction?

LH: Right, it's very difficult and causes you to be very specific. Or just "give up"

AWB: Don't type check unless you need to

LH: The spec always does type checking up front, for example `[].map(null)`, this throws.

RW: (presents own position which agrees with LH)
(Present Domenic's position verbatim)
```

DD (via RW):

> Changing these to throws from existing promise code is a refactoring hazard. Similar to making methods non-enumerable. i'm ok with it rejecting, but not with it throwing, then you have to do .catch() AND catch { }, it's still a refactoring hazard, but at least one that doesn't screw you over by creating sync errors in code that previously only expected async ones.

AWB:

LH: Ok, so Domenic made two points:

- The Refactoring Hazard
- Making it an error means you have to decide whether it's sync or async

LH: 3 options

- Sync error
- Async error
- No Error

YK/RW: Domenic is ok with async error (ie. reject)

JM: If didn't want async error, is there a way to emulate?

No

LH:

JM: Promises exist in two stages: setup stage, execution stage. setup stage is the code that is written, the execution stage is async behaviour. setup stage errors should be sync

RW: agree

LH: it should never be required to put a try/catch around `p.then`. (restates JM)

YK: conflation of two modes

MM: Why is `.then(non-callable)` different then a malformed subclassing? (the latter is sync)

LH: let's narrow it down and take "no error" off the table.

Agreement.

2 options:

- Sync error
- Async error

LH: IDL currently does all type checks on parameters sync.

YK/LH: If you were to use `await`, always get all of your type checking inside the async flow.

YK: The way to look at this is to treat it like it's an async function.

(Domenic Denicola call in)

LH: (recap current discussion)

DD: First, strange that we are accepting null, but not false or 0. Why is null and undefined on one side, but other values on the other. Second, the refactor hazard.

MM: What does Q do?

DD: If Promise A+ does this and with the amount of code that already relies on this.

LH: Agree that accepting null and undefined is already

DD: Precendent: IgnoreNonCallable in JSON.parse and JSON.stringify, .valueOf; Array.prototype.sort throws for non-callable

MM: For sort?

DD: sort throws if you pass non-callable

(AWB: actually ES5 says that sort is implementation defined if argument is a non-callables other than undefined.)

YK: Enough evidence to leave it as is

LH: I'm convinced.

Discussion about IDL optional function vs. mandatory function

MM: Recalling that originally, no part of Promise.then executed synchronously.
Case in concern:

```js
Promise.resolve(instanceOfSubclass);
```

DD: returns a `Promise`

MM: How does `Promise.resolve()` handle `instanceOfSubclass`

DD: Treats it as an untrusted thenable

MM: Excellent

Review of `Promise.resolve`, focused on `Promise.resolve(untrustedObject).then(onFulfilled, onRejected)`

MM: Before subclassing, we had the safety property, given frozen Promise and Promise.prototype, that
Promise.resolve(untrustedObject).then(...untrustedArguments)
did not run user code synchronously during these two calls. Satisfied that we have
preserved this safety property.

YK: Enough evidence that we shouldn't error here and then all other errors are async.

AWB: Anything that's non-callable uses the default (not just null and undefined)

Discussion about precedent.

WH: Reluctant to go with no error. It's a bad idea to silently ignore things such as attempts to write read-only variables or call things that are not functions.

WH: No particular precedent here. Some places throw when you call a non-function, some don't. If the sort comparator is not undefined and not a function, the standard throws.

AWB: The presented arguments are 2/3 in JSON functions

LH: Do we want to set a precedent?

...The Loader API throws on non-callable.

MM: Crock, was there a reason you chose not to throw on non-callable?

DC: No particular criteria.

MM: Experience with the API?

DC: I'd be reluctant to do the same again.

The discussion is now:

- Async Error
- No Error

RW: Firefox Nightly doesn't throw on this

DD: Chrome doesn't throw

MM: Given the state, I don't think we should change.

LH: If making a change mattered, then we should make the change. If there are no benefits, then "do no damage" first.

LH: (discussion about the amount of time we spend on these things)

... p.then(42)? Doesn't matter. It's a minor corner case (we should go with what programmers expect)

The decision for how to handle this non-callable case could inform all cases that expect functions

MM/WH: Don't really like the silently tolerant behaviour in general. Want the general precedent to be to throw errors when calling something that can't be called, but willing to make an exception in this case due to the established library usage.

Conclusion/Resolution

- w/r to .then, respect libraries: no error.

- do not adopt this as a principle for non-callables passed where callables are expected, the general principal: if something is neither a callable or undefined, indicate an error.

DD: Optional callable vs. required callable. There is spec text that matches neither of the consequents.

?: Example is [].map(nonCallable). If you pre-check, this errors. If you "just call it", this does not error (since it never gets called.)

MM: Per the principle defined, you'd simply check if the argument is defined, if it is, call it.

@@iterator for arguments object

AWB:

- Own Property?
- Or should introduce a prototype object to contain it?

RW: There is a record of an agreement for this from several years ago

BT: Not sure why we want to do this for arguments?

JH: Don't want to encourage programmers to use arguments object in ES6

RW: Agrees, but not sure that's the right way to go.

RW: We already have consensus on iterator protocol for arguments object.

MM: Any notes on own vs prototype?

RW: No.

YK: Foresee a problem

RW: agree that we should do it, purely for consistency, but practioners need to discourage the use of `arguments` in ES6 code.

AWB: 2 Cases:

- want each by name
- want to parse the arguments list

Discussion about default params pattern and `...arguments` delegation.

Back to consensus on inclusion

MM: Strong against introducing Arguments.prototype

RW: Same.

JH: Ideally the @@iterator is on a prototype to avoid the allocation costs.

YK: Implementations can optimize.

Conclusion/Resolution
- @@iterator on arguments object
- specified as an own property (no creation of exposed Arguments.prototype)
- Updated: https://bugs.ecmascript.org/show_bug.cgi?id=1114

Default parameters
(Brian)

BT:
```js
function f(a = 1) {
  console.log(a, arguments[0]); // 1, undefined ?
}

f(undefined);
```

BN: That's consistent with the difference between arguments and parameters.

BT: If a default or rest param, you get non-mapped arguments.

MM: What is arguments.length?

BT: Always based on what is _passed_

RW: There is a rule that was created, which is relevant to this discussion: https://github.com/rwaldron/tc39-notes/blob/6d88efa7c4eba2d7a8b6fd5801f2415c2f29c94c/es6/2013-07/july-25.md#consensusresolution

MM: Avoidance of micromodes if possible. (Mark, can you fill this in? It was re: craziness)

Conclusion/Resolution

- An argument list that only has identifiers in sloppy mode has mapped arguments object. ALL other arguments objects (ie. any that contain default, destructuring, or rest, or those in strict mode) don't map.

Initializer in for-in

AWB:

```js
for (var x = "nothing" in foo) { ... }
```

Previously convinced that this was safe to remove. Apple implementors reported breakage caused by removing this.

MH: (confirms) Have encountered sites that have this code.

YK: I understand the desire to clean up, but if clean up causes breakage then why bother?

AWB: Need to address the let case (AWB: this was more about consistacy with the let case. let case does not have the initializer but as new syntax that doesn't introduce any compatability issues.)

YK: Nothing happens in the let case,

MM: Except for the side effects of the expression

AWB: So basically it was a matter of seeing if we can

YK: As a rule, we should err on the side of caution when faced with these minor changes that create incompatibilities once we find real-world problems.

Conclusion/Resolution

- Punt on a decision until May

Function "as" Block

BT:

```js
if (true) function f() {

}
```

MM/BT: Already illegal in strict mode

BT: In all browsers, you can call `f()` after this, but doesn't hoist.

WH: That's a contradiction, you said it doesn't hoist but can be called.

(This is the problem?)

BT: On the list, Brendan was opposed to adding this to Annex B

WH: Is this hypothetical or real breakage? Where does this actually occur in the wild?

BT: ...haven't seen this anywhere in the existing code

WH: Inclined to keep the new semantics simple and ignore this.

MM: Not seeing in the wild? Back off

YK: If we can get implementors to agree to make this an error, then we'll specify it as such

AWB: note: file bug to have this tested in Firefox

Conclusion/Resolution
- Allen will file bug to have this tested in Firefox

Name property of bound functions and toMethod functions

AWB:
- Currently neither have an own name property
- Should either or both get one?
- If so, what should it be?
  - "bound foo"? (for the bound case)

YK: Why this naming?

RW: In the current ES6 specification, for the name property of function, accessors are prefixed by "get " and "set " (trailing space intended).

AWB: Don't think about these name conventions as any kind of type information, just as an a means to distinguish

MM: what about toMethod?

AWB: Suggest no qualifier

Reviewing toMethod

MM: What about function properties named "get foo"?

MM: I guess this is ok. These names are not high-integrity -- mostly intended for debugging info.

Conclusion/Resolution

- toMethod: no prefix qualifier
- Bound functions: "bound foo"
- Adopt uniformly:
  - "get "
  - "set "
  - "bound "
- Fix empty string property bug

AWB: Concerned about backwards injection attacks.

MM: Too late to

(AWB: I don't recall what this was about)

new %TypedArray%(iterable)

- Currently constructor doesn't recognize iterables, but requires an array like. This is the behavior inherited from the Khronos spec.
- Need to use:
- %TypedArray%.from(iterable)
- Should constructor work like %TypedArray%.from?

Conclusion/Resolution

- Make all %TypedArray% constructors accept iterable

new %TypedArray%("2")

DL: %TypedArray% actually coerces this to a number first to create a typed array with length 2. The previous consensus would have to occur _after_ the coercion step to preserve the expected behaviour.

Conclusion/Resolution

- Do string coercion to number before handling iterable to avoid mishandling a string argument.

Map Constructor and Duplicate Keys

```js
let map = new Map([["x", 1], ["x", 2]]);
```

RW: Revisiting:

- https://github.com/rwaldron/tc39-notes/blob/master/es6/2013-07/july-23.md#consensusresolution-1
- https://github.com/rwaldron/tc39-notes/blob/master/es6/2013-07/july-24.md#consensusresolution-2

Conclusion/Resolution

- No error, last property wins, eg. map.get("x") === 2

 # April 9 2014 Meeting Notes

Doug Crockford (DC), Brian Terleson (BT), Luke Hoban (LH), Dmitry Lomov (DL), Waldemar Horwat (WH), Allen Wirfs-Brock (AWB), John Neumann (JN), Rick Hudson (RH), Rick Waldron (RW), Eric Ferraiuolo (EF), Jafar Husain (JH), Jeff Morrison (JM), Seo-Young Hwang (SYH), Mark Honenberg (MH), Caridy Patino (CP), Yehuda Katz (YK), Niko Matsakis (NM), Ben Newman (BN), Filip Pizlo (FP), Norbert Lindenberg (NL), Sebastian Markbage (SM), Mathias Bynens (MB), Rafael Weinstein (RWS), Jaswanth Sreeram (JS), Alex Russell (AR), Istvan Sebestyen (IS), Mark Miller (MM), Tatiana Shpeisman (TS), Brandon Benvie (BB), Brendan Eich (BE)

RF Status

(John Neumann)

The May version of the specification is the opt-out version

See:

http://www.ecma-international.org/memento/TC39%20policy/Ecma%20Experimental%20TC39%20Royalty-Free%20Patent%20Policy.pdf (E.4-E.6)

http://www.ecma-international.org/memento/TC39%20policy/Opt%20Out%20Form%20for%20a%20TC39%20Royalty%20Free%20%28RF%29%20Task%20Group%20%28TG%29.pdf

Conclusion/Resolution

- May document will be opt-out

RF/TG Item

(John Neumann)

JN: Vote to request lifetime membership waiver for Brendan Eich

Motion to approve? [WH raises motion]

Yes, seconded

No discussion, No objection

Conclusion/Resolution

- unanimously in favor

Object.observe Update

(Rafael Weinstein)

RWS: The next step was spec text review.

YK: Reviewed but have feedback

AWB: Concerned not enough attention from committee members

YK: Similar concern, but additionally want to see more practical application testing of new features in general.

AR: Can you be more concrete, with regard to Object.observe

YK: There is a need for filtering of change records

RWS: We had hoped to work on this post 1.0

YK: We tried this and it's messy and complicated, so it's either a matter of being messy and complicated in user code or in V8

LH: Valid and being addressed... How does this correlate to the status of the spec.

YK: The spec seems fine, but this is an issue discovered when I tried to make something with it.

LH: More effort has gone into review of this feature than any other

YK: But needs to be useful to library code.

AWB: (continuing) I'd feel more confident if more committee members reviewed the spec for correctness.

RWS: What is the purpose of the reviewer role, in this stage? I assumed it was for the mechanics, not the feature itself. I'm confident we can work through the existing concerns.

YK: I think that practitioners need to be given an opportunity to review in real use

RWS: This should be done much earlier in the process.

JH: How long is needed to feel comfortable with this?

YK: We already made implementation attempts and ultimately view them as failures. We wrote real code and discovered real problems.

LH: So you're not comfortable giving consensus at this point, based on your actual experience.

...Concern that consensus blocking this late

RWS: Propose that we don't move to stage 3, but I want a commitment from Yehuda that to prioritize working through the remaining issues.

YK: Confirm

RWS: We should view this as feedback for the new process

YK: There was feedback, but no action

RWS: There have been many updates and if there were issues, that was on you to follow up... If there is criticism, it needs to be on record.

AWB: In addition to meeting notes record, file bugs to track progress

YK: In general, if there are concerns, there's two sides: person A thinks "I should push harder" vs. person B saying "that's fine" without giving it much thought

AWB: Still need to be more active reviewing in the committee.

Relevant: https://github.com/rwaldron/tc39-notes/blob/master/es6/2013-09/sept-19.md#7-objectobserve-status-report (incomplete notes about the

discussion about filtering in general)

Conclusion/Resolution

- RWS withdrawing request for stage 3
- YK commits to prioritize working through remaining issues prior to next meeting.
- RW commits to reviewing for spec mechanics
- AWB: "Don't approve stuff without reading the spec text, dammit" :)

(Istvan Sebestyen joins remotely)

RF Status

JN: (recaps decisions)

IS: Issues:

1. Status of Third Party contribution
- Currently under ballot
- expect it to pass

AWB: Will the forms be available prior to the conclusion of the ballot?

IS: Yes, will make available as soon as we can.

AWB: We have a back log of contributions waiting to be accepted. Would like to be able to provide immediate instruction

2. IPR has prepared FAQ addressing Ecma copyright policy
- http://www.ecma-international.org/memento/Ecma%20copyright%20FAQ.htm

3. Liason with IETF

4. JSON

AWB: The first edition is done, no sign of interest for a second edition.

IS: Fast track ISO?

(will follow up)

RW: Istvan, please send link to FAQ for inclusion.

confirmed

BT: CLA needs to be electronic and checked automatically for each pull request (bot)

RW: Needs to provide access to committee members to confirm contributor agreement.

EF: Is the vote for just the forms/agreements?

AWB: The vote is to approve the policy, the forms already exist.

Conclusion/Resolution
- RW/BT/AWB to send CLA requirements (noted above) as need to be successful

Parallel JS Spec Report
(Rick Hudson)

Slides: [TC39PJSApril2013.pdf](https://github.com/rwaldron/tc39-notes/blob/master/es6/2014-04/TC39PJSApril2013.pdf)

RH:
Summary

- Major Focus on implementation

- Use case driven perf tuning
- Harvey Mudd collab
- Prohect implementing OpenCV
- Demos rewritten to use typed object API

Design Goals

- Ease of use
- Deterministic where possible
- Follow current syntac seantics, and security

Platorm Independent
- Supporte all kinds of Platforms, Parallel or not
- Perform well on different parallel architectures, multi-core, GPUs SIMD

Extracting reasonable performance out of parallet hardware
- Extracting all performance a secondary goal

Key Insight: Temporal Immutability

- During Concurrent Execution
- A computation can read or write its local data
- A computation can read shared state
- Parent waits patiently
- Whitelist thread-safe/temporally immutable primitives
- violations or best effort faulure result in a seuqnetial schedule
- Otherwise
- Nothing changes
- Current JavaScript programs are unaffected

→ The sweet spot between Functional and OO

WH: Examples of whitelisting?

RH: Math.sin

WH: Calculating the sine might be parallelizable, but are the lookup of the name and object used to invoke it parallelizable? My biggest concerns are with the glue stuff such as variable and object accesses, which might hit a proxy, observer, or whatnot.

RH: It's a QoS issue. Start with everything blacklisted, then let implementations figure out what to optimize/whitelist.

WH: If I dont know what the impl does, what am i allowed to do to stay in this efficient realm

NM: Building tools that will help developers see what operations are causing de-opts, general jit feedback

MM: With the JIT optimization, you're comfortable knowing the jit will do its job

WH/YK/MM/AR: (Disagreement)

AR: I maintain a hierarchical constraint solver and must pay close attention to keeping math operations on the same numeric path (note: maintaining values as integer or float)

NM: For best performance, parallel code wants to do the same. We expect better developer tools will be very helpful for both cases.

WH: I want to know specifically how to stay within the good performance safe-harbor

NM: Yes, but we don't have a complete picture.

WH: Want at least a safe harbor subset described. Having users study the idiosyncrasies of a specific implementation to discover it just leads to fragmentation.

TS: Recommend best practices document, vs. specification text

WH: Standards can clearly suggest. For example, the C++ standard has done that by naming the safe harbor conditions that allow the RVO and NRVO optimizations. Implementations can go even further than the cases named in the standard, but users can expect to have at least those optimization cases work efficiently.

NM: I can see an addendum that includes this, but we're not there yet. It takes time to figure out what the common subset is. I expect we'll eventually be able to say this with more precision.

WH: Where on the spectrum between stylized mostly machine-generated code (asm.js) and general-purpose user-written code can we expect the optimizable subset to be?

NM: Much closer to general purpose user-written code.

RH: There will be a sweet spot where this will be very effective. Progress will show

RH: (continuing)

Parallel JavaScript API (ES7)

- Extend JavaScript's

(Need slide)

Sum using reducePar

```js
// Sequential
var i;
var a = [1,2,3,4];
var sum = 0;
for (i = 0; i< a.length; i++) {
  sum+= a[i];
}
```

```js
// Data Parallel
var pa = [1,2,3,4];
... need slide.

```

Reverse
```js
var pa = Array.buildPar(4, i => i);
var reversed = pa.scatterPar((a, index, c) => c.length - index - 1); // [3,2,1,0]
```

Positive
```js
```

```
var pa = [1,-7,3,5];
var positive = pa.filterPar(e => e > 0); // [1, 3, 5]
```

WH: Is there a similar thing to (common lisp) mapcan? [In Common Lisp mapcan is a map which calls a function on each element. The function returns a list and mapcan returns the concatenation of those lists. The JS equivalent would be the same except using arrays instead of lists.] When writing code I found mapcan to be one of the most useful mappers.

NM: We don't currently have a flatMap operation. It can be done in parallel but requires multiple passes. Could be done in user code.

WH: Doing it in user code makes it hard to optimize well. A built-in would offer more opportunities for an implementation to optimize it by expressing what the user is doing much more directly.

NM: Let's follow up off line

LH: I've noticed that these are all defined on arrays, but used to be on Typed Array

NM: On both

LH: These are consuming and producing arrays, is there a lot of overhead? You'll have an allocation

DL: (Dmitry, please fill in)

NM: The API is important, but we haven't worked on it.

RH: (continuing)

Non-Determinism

(see slide)

MM, WH: Why do the arrows on the slide cross? Why require reduce combinators to be commutative?

NM: For parallelism.

MM, WH: Why? Only associative is needed for parallelism.

TS: eg. if you try to do addition, 100 threads on GPU and doing addition as atomic operation, can't guarantee commutativity

WH: If you're able to compile the reduce operation into an atomic add, just do it. If you know enough to compile it into an atomic add, you also know that atomic add is commutative.

WH: If you have something more complex, assume it's associative but don't assume it's commutative. It significantly limits the applicability of the algorithms.

NM: I'll do some more measurements and come back with data.

MM: When the elemental function is commutative, in that case, use whatever technique you want to preserve associativity

YK: You may want to allow people to say that

WH: I want to see, in practice, if that makes a difference

TS: What's a non-trivial example of a noncommutative reduce operator?

MM: Matrix multiplication is associative and non-commutative

RH: (continuing)

Spec Wording

reducePar and scanPar use values from the original `O` array and results pusged onto an `A` array

"repeat in an arbitary and implementation dependent order len-1 times."

"select 2 previously unselected indices, k1 and k2 from `O` and `A`"

WH: What is the result of reducePar when the input is an empty array?

Agreement to follow the same empty array behaviour currently defined by `Array.prototype.reduce`

Why Parallel Versions

Sufficiently sophisticated copiler argument

- new semantics to reduce, scan

... need slide.

What we have learned

- We can see the horizon and there are no show stoppers
- Multiple prototypes: Intel(FF, [V8/Crosswalk](https://github.com/crosswalk-project/v8-crosswalk))
- Production: Mozilla closely tracking spec
- Scaling is achievable in parallelizale parts of the application
- Falling back to sequential schedule better than throw
- Out pointers to kernal functions are useful for reducing memory pressure and avoidng copying
- Allocation pressure is crucial to performance in larger kernels

Pressure on Memory Management Latency

(just wait for slides)

MM: Earlier said whitelisting things known to be immutable or threadsafe?

RH: No, have to define "thread safe"

NM: have not done that, want to replay executions without side effects

Next Steps

(copy slides)

(end)

LH: There was other work that Brendan had shown, re: SIMD. Is that still for ES7?

NM: (recapping current progress, no answer for ES7)

Mixed discussion about comprehensions, the introduction of .buildPar, filterPar, reducePar, mapPar

TS: As we added parallel versions of map, reduce, etc. needed parallel of comprehesions which is buildPar

MM/JH: (substantial use of comprehensions in other languages)

WH: Desire to have a parallel comprehension syntax.

?: Why not just optimize the current comprehension syntax?

WH: For exactly the same reason why we're discussing parallel map et al instead of reusing map: it's hard in general to tell whether calls to functions can be reordered.

LH: (subjectively) the comprehension form is more attractive

Signature of Array.from map callback

(Allen Wirfs-Brock)

AWB: Currently differs in callback arguments:

.from

```js
callback.call(thisValue, v)
```

.map:

```js
callback.call(thisValue, v, i, source);
```

RW: The change is supported in Map.prototype.forEach, Set.prototype.forEach, where the last argument is the set or map

AWB: But what can you actually do with that source object if it is just an iterator object?

JH: What

AWB: Consider scenario where someone might want to use the same map callback function for old-style map callback functions + Array.from

AWB: Functions that try to index into an iterator (3rd param) would get undefined, because its an iterator -- not an array

RW: There's a 3rd case for index, eg. tracking index for conditional execution (eg. even/odd element). Might concede the 3rd argument

AWB: Map.prototype.forEach() passes key (for "index" param). Set.prototype.forEach() passes element

AWB: Obscuring errors if arguments don't match expectations

RW: This is a user-code problem

JH: Could we leave it out and require a free variable?

RW: Would be weird that the Array mappers don't leave out the 3rd argument

RH: We find the index very valuable for picking up stuff from free vars

RW: Let's provide index/key for both paths, but not the 3rd param. If you need that you can use a closed over free-var

Conclusion/Resolution

- Change usingIterator path callback signature to: value, index
- Change array like path callback signature to: value, index
- Removes "items" from 17.d.3.1 (array like path)
- Adds "k" to 8.g.7.1

(Noted: https://bugzilla.mozilla.org/show_bug.cgi?id=904723)

Bug 1571 RegExp Syntax

https://bugs.ecmascript.org/show_bug.cgi?id=1571

AWB: ES5 changed (?=) and (?!) from zero-width atoms to assertions

- Doesn't match reality
- Why was this change made?
- Should we roll it back?

LH: If web reality matches 1571

- we don't know the motivation
- we don't know who or why

MB, WH: Roll the change back to ES3, all browsers match ES3 behaviour

AWB: Need to update the ES6 web reality

DC: Did this make it into the bug suites?

LH: Must not have, because no major browser is failing.

WH: How much does test262 test detection of syntax errors?

BT: A lot but probably not enough

Conclusion/Resolution

- Roll back the ES5 change

Change “EscapeSequence 0 [lookahead ∉ DecimalDigit]” to match reality

https://bugs.ecmascript.org/show_bug.cgi?id=1553

AWB: Strict mode explicitly disallows octal escapes.

MB: Technically `\08` isn't an octal escape sequence though since `8` is not an octal digit; not sure if the strict mode rule applies

LH: We should take this offline

WH: Not a DecimalIntegerLiteral, which can only have one digit if it starts with a zero.

?: So '\08' is a null followed by '8'.

WH: No, it's not that either. The grammar has a lookahead restriction that states that a decimal integer escape cannot be followed by a digit. It's a syntax error.

Conclusion/Resolution

- Follow up on the bug report

(AWB: who/how? Action items without names don't get done...)

Implementation Dependencies in String.prototype.replace

AWB: 21.1.3.14 "Table 40 — Replacement Text Symbol Substitutions": someone should research what implementations do here (any differences or “web reality” to match?)

This issue dates back to ES3

Conclusion/Resolution

- LH to follow up

RegExp toString escaping not fully specified

AWB: 21.2.5.13 may need a more explicit spec.

WH: This was intentional when we defined toString in ES3. If the RegExp is constructed from a string, the string is not always usable as-is due to issues such as //, ///, and others. Evolution of regexps to support non-BMP unicode may introduce other cases due to the cover grammar.

WH: Given that the source string is not usable as-is, there is no obvious unique value that toString ought to return. An implementation might choose to optimize or simplify regexp patterns (example: replace /[zzzzz][.]\u0041/ with /z\.A/), and implementations may differ in how far they go in such optimizations. In ES3 we decided to specify the behavior of the returned string: if eval'd, it must produce an identically behaving RegExp. Don't see anything that would invalidate that decision since then.

MB: Browsers currently re-use the exact pattern, e.g. `String(/a/)` vs. `String(/\x61/)` – normalization/serialization would be welcome

RW: ES3 spec says "...src may or may not be identical to the source property...." when referring to whether `RegExp.prototype.toString()` should return the same pattern given to it

...This language disappears in ES5

- ES3 15.10.6.4

- ES5 15.10.6.4

- ES6 21.2.5.13

MB: `eval( RegExp(string) )` may not result as intended e.g. if string is `'/'` or `'"'`.

WH: specifying toString completely is a can of worms; instead we should add a requirement that leaves the exact `toString` behavior up to the implementation, as long as `eval( RegExp(string))` returns a regular expression that has identical behavior.

?: Not all browsers correctly implement the rule that RegExp toString results must be evalable into the same regular expression.

WH: That would be a browser bug.

AWB: Some browsers haven't been paying close attention to this.

WH: Best fixable by putting a few test cases in test262.

Conclusion/Resolution

- Leave `RegExp.prototype.toString` definition as-is, but consider adding requirement https://bugs.ecmascript.org/show_bug.cgi?id=2609

Allen's TODO Summary

- Lots of Module related cleanup and refinement
- new eval semantics
- MOP/Proxy property enumeration API
- Cleanup completion reform and issues (nothing insurmountable)
- Need to write Annex B spec for HTML-like comments

BT: Can you describe the "MOP/Proxy property enumeration API" item?

AWB: There are outstanding enumeration issues that we need to finally address.

Introduction and Language Overview

- Need ES6 paragraph for intro (Brendan?)
- Need somebody to update language overview
- In rev23 added some material about classes and how they related to the prototype discussion

Conclusion/Resolution

- RW volunteers to write first draft of language overview

LH: Is the recent work you've done on scope complete?

AWB: Still have work to do on eval scope

BT: That's whether or not there is an implied block

RW: This is not to say that _literally_ the difference is the addition of "{" + source + "}", but that you could reason about the result in such a way

AWB/LH: Approximately.

...re: Modules

JM: Can work to help Dave extract Module knowledge

RW: Review, find missing parts, report it (bugs, to Dave, etc)

AWB: HTML comments syntax should be specified for Annex B https://bugs.ecmascript.org/show_bug.cgi?id=2610

ES7 Process, New Proposal Home

https://github.com/tc39/ecma262

AWB/BT/LH/RW: mixed discussion and agreement to replace the wiki.

Conclusion/Resolution

- All in favor!

RW: Should this github repo contain actual proposal info?

BT: Haven't decided yet, but seems reasonable

RW: Will hold off until the organizational story is complete.

RW: We should have a guideline on format for linked proposals (see http://wiki.ecmascript.org/doku.php?id=strawman:string_padding as a possible example)

Discussion about what topics should be in this guideline

- History
- Use Cases
- Problems/Pain points addressed

BT: (to YK) send a PR with your proposal proposal.

JN: Please review the review sign up sheet and sign up.

4.4 Object.getOwnPropertyDescriptors
(Rick Waldron)

RW: this is the analog of defineProperties
This is for ES7.

MM: Returns array or iterator?

RW: It returns a plain object that can be passed directly to defineOwnProperties

MM: Does it inherit from Object.prototype?

RW: Uknown. Open issue.

WH: What if you have a proxy that represents a massively infinite structure, what does this do?

RW: Unknown. Open issue.
No reason to rush, this is an ES7 thing.

AWB: Must be same answer as what you get from getOwnPropertyNames

RW: Makes sense.

DC: Returns things that are not enumerable?

RW: Yes.

MM: I have no problem with this. What about symbol properties?

AWB: Open issue.

MM: This one is a trivial polyfill.

AWB: Any library can implement this. They should experiment and figure out what's useful.

YK: There are many issues. Would be best for us to decide what the right behavior is.

AWB: If you have a complex object you don't necessarily want to create all these property descriptors... Lots of allocations that get thrown out immediately. May not be a good idea.

YK: Maybe you want a fn with a callback that yields in the value

AWB: Or something like iterator but it doesn't necessarily fit.

MM: There might already be a polyfill in the Traits.js library.

WH: Why does it return an object instead of matching the behavior of Reflect.ownKeys?

RW: To pass to O.dPs

BN: Why no 'own' in the name even though they make own properties.

AWB: History - you can imagine dealing with own properties and inherited properties, you have to make a choice about which you deal with. Methods explicitly have own in it. In situations where the only thing you could deal with drop the 'own'. Defining implies own.

BN: Descriptors is so verbose... why?

RW: You get back a descriptor. There is already a getOwnPropertyDescriptor.

ACHIEVEMENT UNLOCKED: Longer API name than getOwnPropertyDescriptor!

Conclusion/Resolution

- Pursue for ES7. RW has spec text. Approved for Stage 0.
- RW to send a PR to add to github.com/tc39/ecma262 tracker

4.5 Array.prototype.contains

(Rick Waldron)

RW: ES5 adds String.prototype.contains. Seems oversight that we don't have the same thing in Array. But, we can wait until ES7.

BT: Any spec text?

RW: Not yet, just mailing list discussions.

WH: Are you searching for elements or subsequences? If goal is to be analogous with string you would be searching for subsequences.

DC: We didn't do that for `indexOf`... Rick suggests continuing in that tradition.

WH: That's confusing. If it's not analogous, it should have a different name...

MB: Would be nice for DOM (would get rid of abstractions). DOM has `contains` for classList.

YK: Why not use `has` like set?

MB: Because `classList` already uses `contains`.

BT: Is this different than `indexOf`?

RW: Open question.

Conclusion/Resolution

- Sounds good, need strawman to approve stage 0.

4.6 Updates to parseInt

(Rick Waldron)

RW: Should `parseInt` handle new octal and binary integer literal syntax? [bug #1585](https://bugs.ecmascript.org/show_bug.cgi?id=1585)

WH: Which octal syntax? 0123 or 0o123?

RW: 0[OoBb]<suitabledigits> only.

AWB: Essentially add the new literal syntax to `parseInt`.

WH: Sounds great, love to do it. But are there security problems? For example, let's say you have a website that parses the same 0o123 integer twice, one time uses `parseInt`, other time uses something else that isn't aware of the new prefixes and thinks that parseInt would return 0. Validation might pass but actual value would be wrong.

RW: Doesn't this exist with hex integer literals?

WH: No, hex literal has been around since the beginnning... These would be new. It's a breaking change.

MM: What is the rule that you propose to recognize an octal literal?

RW: Those in the spec: 0b/0B or 0O/0o.

LH: This works today (returns `0`). This is a breaking change...

AWB: Applications may not want this behavior...

MB: You can use `Number("0b11")` or `Number("0o42")` instead.

LH: Was that a breaking change? Looks like, but we went from `NaN` to actually returning a number.

WH: It's a breaking change. parseInt looks for a valid prefix and ignores the rest. parseInt('0o123') currently returns 0 because it sees the starting 0.

Conclusion/Resolution

It's dead. RW to close the bug wontfix.

4.3, Update Object.assign to accept multiple sources

(Rick Waldron)

RW: `Object.assign` is useful. Multiple real-world APIs do this, but most allow multiple sources.

AWB: Some have additional options (enumerable vs. non-enumerable, shallow vs. deep copy).

RW: We selected an appropriate name (assign) that set it apart from existing APIs (ie. extend, or merge). We got consensus on this. Problem is that the response from practitioners has been negative. They want a multiple sources version.

EF: Didn't we solve that with reduce?

RW: Somewhat, but it introduces a bug.

RW: I tend to agree it falls short of the cowpath. We should fix for ES6.

YK: one of the common use cases for assign where you're supporting multiple mixins

BT: Why is name important?

AWB: Anyone else who has defined `Object.<that name>` could clobber existing stuff.

SM: Want an immutable version?

RW: Workaround - just use first source as empty object.

AWB: We considered that in the future we might want to add an options record. We weren't trying to provide an end-user solution but a primitive.

EF: If we only had support for one source and target and people used the reduce pattern we could break the web in the future if we want to extend this later.

RW: It will be a WTF that we ignored the cowpath.

AWB: That's why we chose a different name. This isn't `extends`.

YK: It's about real use cases.

WH: Why do the existing methods ignore exceptions?

MM: If some props cause exceptions and others don't, rather than have it be random which prop took, you have a gaurantee that all non-exceptional properties took. Do same thing with DP.

BT: What options?

AWB: Filter function, whether symbols are used.

:: Discussion about whether and what was discussed previously, confirmed prevous consensus was target and source previously, but we didn't explicitly say multiple sources was out ::

YK: We decided that we were going to do assign now, and punt on more complex APIs for ES7. We don't need copying APIs for very specific use cases.

RW: this is a super common use case.

AWB: Most common is single source + target

YK: People will want multiple sources and won't see it as different API.

AWB: Concern that for people who are doing mixins, this is the wrong primitive thing.

YK: Won't happen. Doesn't rebind `super`. So we need ES7 to handle `super` correctly.

BE: Who's against?

AWB: I'm not enthusiastic. Won't stand in the way.

BE: Seems strictly winning to have multiple arguments.

Conclusion/Resolution

- Object.assign gets multiple source objects.

5.1 Object.entries, Object.values

(Rick Waldron)

RW: ES5 added `Object.keys`. For ES7, `Object.entries` and `Object.values` make sense. These return arrays.

BT: Array of arrays for `entries`?

RW: Yes. You can pass to Map constructor.

YK: Important to be iterable.

AWB: Why not return iterator?

RW: Agree it's crappy but it makes sense because keys returns array.

AWB: Alternatively we could add this to a standard `Dict` module.

BT: Assuming we get standard modules?

AWB: We'll get them.

Conclusion/Resolution

- Need strawman for stage 0

Test 262 Update

BT: A lot of pull requests piling up

... Domenic will port promise tests

... Awaiting the contributor agreement form

BN: Will contribute Generator tests

BT: Also need syntax tests

DL: V8 is beginning to implement ES6 and will want to contribute back the tests

RW: We'll need to update the current PRs with any guidelines

MM: For tests that are not specifically specified as strict-only or sloppy-only, currently the test262 harness only tests these in sloppy mode. Must test these in both strict and sloppy by default.

BT: Issues with the error message string varying across platforms

BT: Also need cross realm association testing

...Naively, we could write in-browser tests, but ideally we want host-agnostic testing.

AWB: Need to stay up to date with reviewing

BT: Rick and I have been doing this

DL: V8 team can assist as well

BT: Will need to support a variety of disparate test systems

MB: What about Annex B tests? They'd have to run in a browser; test runner needs to support that somehow.

MM: Detect Annex B features, then if they're present, test if they behave as per the spec, if not, fail silently.

BT: Continue to report test coverage gaps.

Conclusion/Resolution

- BT to publish test criteria/guidelines
- Inform current contributors of guidelines to prepare
- Establish rules for testing Annex B

Async Functions Question

(Jafar Husain)

JH: Question about the omission of the `close()` method from iterator

BE: `close` came from Python, was on generator objects. We unified send and next, we didn't include `close`.

JH: To be specific, I want the object returned by `@@iterator` to include a `close()` method.

BE: The problem with `close` in python is that it leaks GC semantics.

JH: I understand this isn't based on IEnumerator, but think there is a important case being missed.

MM: I like this addition

BE: we solved by making `close` automatic

DL: If you have `close`, and have `try`/`finally` there is no guarantee that the `close` is called.

BE: In a browser, to avoid denial of service, `finally` is not guaranteed to run.

AWB: Iterator prototocol doesn't require implementing throw, so that the iterator must only implement `next`

BE: Andy Wingo presented cases for removal https://mail.mozilla.org/pipermail/es-discuss/2013-May/030683.html

Discussion about whether this is a breaking change, resolution - it is but probably not a big deal

The name "return" is more accurate then "close"

LH: We need a write up that explains why this is critical

RW: (to JH) confirm a write up for tomorrow

Discussion about "reserving" names on iterator objects.

BE: A symbol for the name of the method?

The risk is not big enough

JH: ES6 for-of won't look for a `return()` method, but ES7 for-of will look for it and invoke.

BE: Deeper issue, we had `close`, but we got rid of it. Was it because no one used it as presented, or was because Andy's points were sufficiently convincing.

YK: Having `close()` exist cause hazards?

BE: No

BN: You might break the for-of to exhaust the iterator elsewhere.

MM: Under this proposal, when the for-of exits early (break) it would cause the generator to take the exit path. If we do this, we have to do this in ES6. It's sufficiently weird enough that I'm not convinced.

YK: You'd opt in?

LH/MM: no

MM: If the for-of exits early, the `return()` on the generator is called

This _must_ be done in ES6, if done at all.

JH: Similar to `Object.unobserve`, sometimes you want to stop doing something that you've started.

Discussion about ways around and reasons for use.

MM: What if we modify: no explicit close/return on exit of for-of. generator objects have this method, but must be explicitly called.

JH: That allows for a future syntactic form?

MM: Probably wouldn't add syntactic form.

LH: Worried about the whole notion of having a return mechanism on generator objects (referring to the example given by Andy Wingo, item 2, surprising yield behaviour).

JH: If for-of doesn't explicitly call the `return()`?

BE: That's addressed by item 3 in Andy's concerns

MM: You could still do this by writing out manually

JH: Yes

BE: Recap: no explicit for-of semantics, has a `return()` that can be called if needed?

BN: may want `return()` to exit without triggering catch, vs. `throw()`

MM: Please present use cases that illustrate

JH: Confirmed.

~~Conclusion/Resolution~~

- ~~Generator.prototype.return()~~

(continued to April 10 2014)

 # April 10 2014 Meeting Notes

Doug Crockford (DC), Brian Terleson (BT), Luke Hoban (LH), Dmitry Lomov (DL), Waldemar Horwat (WH), Allen Wirfs-Brock (AWB), John Neumann (JN), Rick Hudson (RH), Rick Waldron (RW), Eric Ferraiuolo (EF), Jafar Husain (JH), Jeff Morrison (JM), Seo-Young Hwang (SYH), Mark Honenberg (MH), Caridy Patino (CP), Yehuda Katz (YK), Niko Matsakis (NM), Ben Newman (BN), Sebastian Markbage (SM), Mathias Bynens (MB), Rafael Weinstein (RWS), Mark Miller (MM),

YK: Revisit new TC39 process. Phase 2?

LH: Phase 2 is experimental

YK: Yesterday, was told Phase 2 is too late for feedback

LH: My sense was that the issue is communication

4.10 Modules Feedback
(Brian Terlson)

https://gist.github.com/bterlson/e68d34b691254a640841

(need slides)

BT: This is feedback, not necessarily requests for change.

Key Points

- Module + Import Syntax
- Hard to remember which does what?
- Having two import syntax forms seems undesirable

- Conceptual model is a departure from what folks are used to
- Reason for departure is not immediately clear (cycles?)

YK: When I teach modules, I don't teach the `module ...` form at all
...Doesn't

BT: The only way to get a binding to the module itself

LH: Can be used almost as a first-class object reference

YK: Cutting at this stage is not bad, I can champion researching the possibility

CP: This may break the ability to statically verify the module?

LH: If you did:

```js
```

module foo from "bar";

foo.whatever
```

If `foo.whatever` didn't exist, would be an error

Question about this being in the spec?

YK: The solution is to go back and determine whether or not `module foo from "..."` is necessary or can be cut.

(This does not affect default imports)

AWB/LH: "module" was for the `Math` case?

JM/RW: `Math` case could very well be defined as a default export.

YK: Can use the following to bring in a module solely for side effects:

```js
import "foo";
System.get("foo");
```

WH: How is work divided between the import and the System.get?

YK: `import "foo"` is just the mechanism to ensure the import of a module without specific bindings. For side-effecting modules.

WH: So the System.get is not neeed to bring in a module solely for side effects.

YK: It's there if you want to get a handle to the module.

LH: This is very close to the semantics of the `module` keyword, unless missing something

Discussion re: is the issue superficial syntax?
```

LH: Maybe (recommended by AWB) `import module Foo from "..."` to replace `module Foo from "..."`?

YK: May not be trivial
... The feedback is that the default export being the main module _and_ a property is confusing.

LH: (confirm)

YK: (walks through explanation)

BT: (continuing)

- Lots of ways to import and export
- As many as 3 different ways you would consume jQuery depending on what jQuery exports
- See: https://gist.github.com/bterlson/e68d34b691254a640841#possible-confusion

```js
import $ from "jQuery";
var get = $.get;

// vs.

import { get } from "jQuery";
```

RW: The first two serve different purposes

```js
var Foo = require("foo");

// and

var Foo = require("foo").Foo;

// The latter is the same as:
var f = require("foo);
var Foo = f.Foo;
```

Discussion about general understanding of node module design

YK: The important point is that if the first is used where the second is needed, user will get an error immediately and will change to the correct form.

LH: As much as I think this may be confusing, I don't want to change it because it I don't want to risk breaking cycle support.

YK: Cycles are not and should not be a thing that the user has to think about.

RW: (prompting to move on)

BT: (continuing)

- Lack of support for dynamic scenarios

BT: want write a function that took a module name as a param, import that module , mutate it and re-export. it would look like importing the original module, but actually getting the modified.

AWB: Can do this at the Loader level

BT: Confirms.

CP: We've been able to prove that this works quite well.

Conclusion/Resolution

- Revisit `module foo from "..."` (YK)

Reviewer sign up

Revisiting: Initializer in for-in

AWB: New information:

OH: To clarify JSC encountered breakage in the syntax. Only case we've seen is (presumably) author changing their mind about type of loop, so we _no-op_ the expression. No execution at all.

...Unclear about what we're saying about the let case. I would be opposed to leaking the redundant initialiser into for (let blah of/in …)

... Would be great to eventually nuke it.

AWB: Likely won't have any new information in May?

OH: All internal builds have this change, so we'll have users reporting.

AWB: Do we move forward nuking it, or as a syntax with no semantics? In main body or an annex?

WH: If we retain the syntax, it should do what it has always done. Why should we have it do something different?

MM: If we leave it in, we have to accept the side effects it causes.

YK: Why does JSC want to ignore evaluation?

OH: To discourage its use.

WH: If we wish to discourage its use, we should move the initializer syntax to Annex B (and have the Annex B semantics reflect its historical behavior, not new behavior).

RW: Any code that has been run through JSLint/JShint will not have this syntax, rejected outright.

AWB: Continue to have it removed, allow the syntactic form in Annex B

MM: Another observable thing... no properties, the var decl is visible after the loop

... The only reason to leave it is concern about breaking something bizarre?

WH: Isn't the var always visible? It's hoisted

MM: Yes, but the initialization value being assigned is only visible if there are no properties

WH: What's the impetus to keep the initializer syntax but change the semantics of the syntax to something different in Annex B?

AWB: Easier to put in the syntax with no semantics

MM: Either flush it, or put it in with the old semantics

AWB: Flush it and let implementations do what they do

BT: If implementors aren't getting rid of this, new runtimes will implement to match the web

MM: If we expect all browsers to implement, we should codify. If we expect not to implement, then don't codify. Cross browser content

Discussion, re: comparison to `__proto__`

MM: Other than JSC, any other impl?

BT: If JSC doesn't feel comfortable, unlikely that IE will

AWB: There has only been discussion at Mozilla

OH: We have evangelized, no one heeds

MM: Mozilla evangelized successfully

OH: Performance punishment (log messages for features that need to be discouraged)

BT: Willing to wait for Mozilla to guide

WH: Does test262 cover this?

BT: Probably, the test will have to be removed.

...Spec should reflect reality.

WH: In favor of taking it out and seeing if we can get away with it. We can put it back in Annex B. The work to take it out is not wasted.

BT: If all three browsers continue supporting, we agree to put it in the spec, at least in Annex B

Conclusion/Resolution

- Mozilla willing to ship a patch in nightly for experimentation

Revisiting: Generator Issues

(Jafar Husain)

Slides: [Allowing-Generators-to-Compose-over-IO.pdf](https://github.com/rwaldron/tc39-notes/blob/master/es6/2014-04/Allowing-Generators-to-Compose-over-IO.pdf)

JH: Generators can't abstract over sync or async IO.

Two Ways to Iterate

```js
var nums = function*() {
  yield 1;
  yield 2;
}
```

Iterable

```js
function Iterable(generatorFunction) {
  this[@@iterator] = generatorFunction;
}
```

Creating Iterables

```js
let nums = new Iterable(function*() {
  yield 1;
  yield 2;
  yield 3;
});
```

Iterating Iterables

```js
for(let x of nums()) {
  console.log(x);
}

//...becomes...
```

```
let iterator = nums()[@@iterator],
  pair;

while(!(pair = iterator.next()).done) {
  console.log(pair.value);
}
```

IO Streams as Iterables

```js
function getLines(fileName) {
  return new Iterable(function*() {
    let reader = new SyncReader(fileName);
    try {
      while(!reader.eof) {
        yield reader.readLine();
      }
    }
    finally {
      reader.close();
    }
  })
}
```

Iterable Composition

```js
Iterable.prototype.map = function(projection) {
   let self = this;
  return new Iterable(function*() {
    for(let x of self) {
      yield projection(x);
    }
  });
};
```

```
Iterable.prototype.takeWhile = function(predicate) {
  let self = this;
  return new Iterable(function*() {
    for(let x of self) {
      if (!predicate(x)) {
        break;
      }
      yield x;
    }
  });
};
```

YK: Need to address Brendan's objections from yesterday

JH: The crucial problem: the iterator claims to be an iterable and it's not.
...If async generators return Iterables, not Iterators.

WH: How

JH: If I accept an iterable and invoke `@@iterator`, I would expect to get a fresh iterator from each call

MM: everythime you for-of you expect to see "the stuff"

JH: Iterators are claiming to Iterables, which is a lie.

NM: An Iterable creates Iterators, Ite

AWB: An iterator that itself has an @@iterator is an iterable

WH: The distinciton is similar to C++'s distinction between input iterators and forward iterators. Input iterators are not replayable.

JH: This is a leaky abstraction. Two calls to @@iterator on the same iterator type will cause the same iterator to be shared unexpectedly in two different code paths.

Discussion re: the process for handling this.

YK: This should've been presented to the feature champions first.

MM: If these are issues that we're going to address, they need to be addressed in ES6

AWB: If we address it in ES6, the only way is to drop Generators from ES6

YK: The best approach is to first discuss with champions of feature

MM: Let's first hear the issues

YK: Disagree with first hearing of the issue in this forum. Jafar could've been more effective by bringing this to Dave and Brendan first.

WH: I want to understand the issues. This is the best forum we have at this moment.

AWB: This is a long resolved and accepted feature, may have non-optimal characteristics, but agreed on. We're long past changes like this.

LH: Let's focus on the problem, not the solution

(New example. copy from slide when available)

OH: you need `self` to finish somehow after the `break`

JH: you need the `return` semantic

...not a problem in Python, because they have StopIteration. This impl had StopIteration and close and both removed.

LH: (Luke please fill in the comments made re: StopIteration and finally clauses)

WH: Please explain what you're arguing for:

JH: Add a `return()` to generators

LH: return would trigger the `finally` clauses

WH: why in addtion to `throw`?

LH: throw hits `catch` blocks

WH: for-of would call that if you break?

LH: Yes, but that's a separate level of the proposal

AWB: No way to run only the `finally` blocks and not the `catch` blocks

WH: Is there some other aspect?

...Iterator is a once through, cannot go back

JH: Make function * to return Iterable

YK: The most salient use case: break but want to continue to use the generator

AWB: Let's just talk about return, no way to force the finally to run

...Specific issues that Andy brought up need to be addressed. Additionally need to convince why the agreement to those issues was incorrect.

YK: The break is ambiguous

JH: As long as you never use the for-of syntax, you're fine

Proposal:

- for...of always assume generator creation
- for...of alwayss terminates generator function

Lost in cross talk...

NM: If you know that iterable creates a fresh iterator, it's very reasonable to close it every time. If you don't know that, then Andy's argument holds.

JH: The value proposition of for-of is that you don't need to work with iterators

AWB: We _could_ add a `return()` method.

... It's a bigger change, but we could make for-of invoke `return()` on exit or completion

YK: doubt we can come to a conclusion on this

AWB: We _must_ come to a tentative conclusion

JH: Confirm, let's try.

MM: Agree with Allen

Plausible Solution:

MM:

- continue to have iterators have an @@iterator, classified as iterables

- continue to have generators return an iterator+

- continue to for-of on an iterator

- we change:

- generator instances have a `return` that when invoked, caused the yield point to take the return path, returning the argument that was provided to the return method. [AWB: actually, needs to return a IteratorResult object: {done: true, value: returnMethodArgument}]

- the for-of behaviour is extended with an equivalent of a finally cause that feature tests for presence of return, if present, call it on any abnormal exit from the loop, normal or abnormal. [AWB:no change is necessary for normal loop complextion because the return from the generator body that sets the completion object to {done: true} will run any finally code in the generator body]

- AWB: a 'return' call that interrupts a yield* needs to be propagate to the inner iterator, just like a 'throw' call.

YK: Recalling Brendan's opposition

MM: The new information: if you want to partially consume an iterator, don't use for-of.

Discussion about process.

RW: (interupt) We can't have a discussion with the champions without considering the issue an dhaving a meaningful record of the discussion. (consider a process note)

LH: Notable, C# does what we're proposing. C# also doesn't allow `yield` in `finally`. No guarantee that we'll run to resource completion.

AWB: No guarantee

MM: No asking for consensus?

AWB: I am. This room needs consensus before we bother including champions

MM: Not a TC39 consensus, a provisional consensus until the champions have been involved.

Discussion about process.

AWB: If we dont decide something, this will float to next meeting.

YK: Proposal: Jafar, Allen, Ben, Dave and Brendan need to work this out offline.

MM: Any dissent from the proposal stated being presented to the champions and allow a decision to be made without further discussion.

WH: Objects to this problem *not* being solved. Object to the status quo of doing nothing. Some solution (such as MM's above) must be found.

Conclusion/Resolution

- Unanimous agreement to allow the champions, Allen Wirfs-Brock, Brendan Eich, Dave Herman, Andy Wingo, Jafar Husain, Ben Newman to solve this problem.

- Allen Wirfs-Brock to originate the email for discussion.

- Reference material: https://mail.mozilla.org/pipermail/es-discuss/2013-May/030683.html

Decorators for ES7

(Yehuda Katz)

Slides: (need slides)

YK: Presenting aspects of common use cases not yet covered by ES6 `class`.

Knockout.js example (compute the value of a property)

WH: Do you want to use functors to produce functions that are per-class (i.e. on the prototype) or per-instance?

AWB: Per instance wants to be handled in the constructor

YUI example (a readonly property)

LH/YK: Sometimes you want to say a method is readOnly

AWB: No declarative way to describe the per instance state

Angular example

LH: (explanation) when I declare a class, I also want to register it with some other system

ES6 Experiments: Angular

```js
@NgDirective('[ng-bind]')
class NgBind {
  @Inject([Element])
  constructor(element) {
    this.element = element;
  }
}
```

AWB: The "@" used to define an annotation

JH: Point out that this is inert meta data

Userland Classes: Ember

```js
App.Pserson = Ember.Object.create({
firstName: null,
lastName: null,
fullName: Em.computed(function() {
return this.get('firstName') + ' ' + this.get('lastName');
}, 'firstName', 'lastName')
});
```

ES6 Experiments: Ember

```js
class Person extends Ember.Object {
  - dependsOn('firstName', 'lastName')
  get fullName() {
    return this.get('firstName') + ' ' + this.get('lastName');
  }

  - on('init')
  - obeserves('fullName)
  fullNameChanged() {
    // ...deal with the change
  }
}
```

YK: Goals:

- Decoration of methods and accessors
- Decoration of future declarattive property syntax
- modifcation of the property descriptor in addition to its value
- can wr

...need slide.

Property Decorators

```js
function readonly(prototype, name, descriptor) {
  descriptor.writable = false;
  return descriptor;
}

class PostComment extends HTMLElement {
  - readonly
  - on('click')
  clicked() {

  }

  - observes('value')
```

```
  valueChanged() {

  }
}
```

Close desugaring...

```js
// after PostComment declared

Object.defineProperty(PostComment.prototype, {
  clicked: {
    value: function() { ... },
    writable: false
  }
});

```

MM: What order?

YK: Bottom up

Memoize

```js
function memoize(...dependencies) {
  return function(prototype, name, descriptor) {
 // ... see slide for specifics
  };
}

class {
  - memoize('firstName', 'lastName')
  get fullName() {
    return `${this.firstName} ${this.lastName}`;
  }
```

}
```

NM: This is very similar to Python and there is a large history of people doing amazing and terrifying things

MM: The descriptor provided is the descriptor that, in the absense of the annotation, would've been installed on the property?

YK: Yes, only providing the descriptor

MM: Only the descriptor returned by the final annotation is used to install on the property.

AWB: Example:

```js
class {
  - dynamic(function() { ... })
  boringMethod() {}
}
```

YK: Yes, but there is another class annotation proposal better suited

Metadata

(extensive slides)

MM: Order of accessor case?

YK: Statically

Static Semantics

- `MathodDefinition` and `static MethodDefinition` have a list of `DecoratorExpressions` (AssignmentExpression)

WH: How does the decorator know whether you're decorating a getter or a setter from a get/set pair? It's reasonable to request different decorations for the two — decorate the getter as public and setter as private.

YK: It can't. Right now the proposal merges the get/set descriptors into an aggregated descriptors.
```

AWB: This assumes that the property is created with an aggregated descriptor, but DefineOwnProperty knows to only set the properties provided and doesn't set the other properties to undefined if they're not present.

MM: If you want the accessor to be non-configurable, you have to set them at once.

YK: If you don't aggregate, using a single decorator for a get/set pair is impossible.

WH: If you do aggregate, using a decorator for just a get or a set is impossible. That's a necessary thing to do and more closely matches the syntax.

MM: The getter for "foo" is non-configurable non-enumerable, and the setter for "foo" is non-configurable enumerable, it's impossible

YK: Need to find a solution to disallow creating contradictory descriptors

AWB: Overriding ClassMethodDefinition?

MM: Overriding MemberOfLiteral definition?

(Note: ClassMethodDefinition and MemberOfLiteral don't exist)

More Custom Syntax

```js
class Article {
  + hasMany('comment')
  + belongsTo('user')
  + attr('title')
  + attr('author')
  + attr('body')

  constructor() {

  }

  - dependsOn('title', 'author)
  get byline() {
    return `${this.title} by ${this.author}`;
```

```
  }
}
```

MM: The reason not using "@" was for two mechanisms?

YK: Yes

LH: Don't want to put a method decorator on a class level annotation.
... May not be the best solution to this problem. What you want is something declarative.

WH: How do you create prototype properties out of thin air?

YK: Use the + form.

WH: Then you could do this, right?

```js
class Article {
  + hasMany('comment')
  + belongsTo('user')
  + attr('title')
  + attr('author')
  + attr('body')

  - dependsOn('title', 'author)
  + generateMeAMethod

  - dependsOn('title', 'author)
  get byline() {
    return `${this.title} by ${this.author}`;
  }
}
```

YK: No, it's passed the class

AWB: The + form, does it involve the descriptor?

YK: No. The + form takes a class object, not a property descriptor.

WH: The cool thing is that - is composable. It would be weird to not be able to apply it to prototype properties that happen to be created out of thin air

Discussion about the problem faced across instance and prototype properties.

WH: Want declarative properties too.

MM: Use method decorators on computed declarative properties.

YK: Yes

WH: Concerned about syntax ambiguities

WH, AWB: Need to coordinate this with declarative properties proposals

AWB: Careful abt not using this to do things that we may do in language. This is targetted at library designers

MM: Creating this whole "module" thing, but pre-populating it with system modules. Could pre-populate with system decorators.

RW: There is long precedent, appropriate, etc.

RW/NM: Decorators made available in a decorators modules, avoids restricting user code

AWB: We should try to follow the same guidelines that we define with regard to defining library code in the language vs. library code in user space.

MM: All basic descriptor manipulation could easily be pre-populated.

YK/RW: configurable (sets configurable: true), nonconfigurable (sets configurable: false)

LH: Encourage the Angular designers to bring forward their proposal as well.

Conclusion/Resolution

- Start a strawman

- work together with others that have class syntax extension proposals
- consider other proposals

Ecma 402 Updates

AWB: updates need to be made to Ecma 402, but I dont have the time/bandwidth

Conclusion/Resolution
- RW volunteers to attempt
- AWB will provide basic summary information

Preview of asnyc/await
(Luke Hoban)

https://github.com/lukehoban/ecmascript-asyncawait/

A few updates since last time:

1. Ben Newman has implemented async/await in regenerator
2. Async arrows feedback
3. await at top level
4. Is await* needed?
5. Ordering of static + async: static async foo() { }

Discussion of async arrows from https://github.com/lukehoban/ecmascript-asyncawait/issues/13

LH: If we cared about being forward compatible with eliding () in arrows, we could be conservative with [no line terminator] : https://github.com/lukehoban/ecmascript-asyncawait/issues/13#issuecomment-40124678

Several: async(foo, bar) => ... looks too much like a function call.

LH: If 'async' were 'function', you likely wouldn't say that. Once editors colorize, this will be less of a concern.

WH: you're not proposing to make "async" a keyword?

LH: Contextual keyword

[Discussion of syntax and no-line-terminator-here]

LH: Questions about whether 'await' should be allowed at top level. We could add to the module top level grammar, not the script grammar

WH: If it's usable in modules outside async functions, would await be a contextual keyword inside modules? It isn't right now.

MM: That would be a breaking change from ES6.

?: We could disambiguate in grammar.

WH: No. Is x = await(foo) a legacy ES6 function call or an await call?

LH: Good point, we'd need to reserve "await" in modules in ES6.

LH: Original proposal included an optional "await*" syntax that was sugar over "await Promise.all(...)". It was unrelated to "yield*" which has no meaning in async context.

MM: Using await * that is not analogous to yield * is bad, let's take it off the table.

LH: Agreed - we'll remove that from the proposal.

LH: Current proposal requires "static async" to be ordered that way for async methods.

Several: Sounds okay.

Conclusion/Resolution

- "await" needs to be reserved in module context
- Keep async arrows, and keep the conservative grammar with [no line terminator] annotations
- Tentatively believe await at top level in modules should be okay.

Closing

JN: Want to thank Mozilla for hosting us and appreciation to Ecma for dinner last night.

Conclusion/Resolution

- Unanimous agreement :)