Minutes of the: *40th meeting of Ecma TC39*

in: *Menlo Park, CA, USA*

on: *4-6 June 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:

Mozilla, Google, Microsoft, Intel, IBM, eBay, jQuery, Yahoo!, Netflix, Facebook

1.2 Introduction of attendees

John Neumann – Ecma International

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

Jafar Husain – Netflix

Matt Sweeney – (Netflix),

Erik Arvidsson – Google

Mark Miller – Google

Erik Toth – Paypal / eBay,

Allen Wirfs-Brock – Mozilla

Simon Kaegi – IBM

Arnoud Le Hors – IBM

Dave Herman – Mozilla

Waldemar Horwat – Google

Eric Ferraiuolo – Yahoo!

Caridy Patino – Yahoo!

Reid Burke – Yahoo!

Rick Waldron – jQuery (over the phone)

Yehuda Katz – jQuery

Dave Herman – Mozilla

Brendan Eich (invited expert)

Jeff Morrison – Facebook

Sebastian Markbage – Facebook

Guy Bedford (Facebook guest),

Brian Terlson – Microsoft

Ben Newman – Facebook

Alex Russel – Google, part-time

Tatiana Shpeisman – Intel

Peter Jensen – Intel,

Jaswanth Sreeram -

1.3 Host facilities, local logistics

On behalf of Facebook **Jeff Morisson** welcomed the delegates and explained the logistics.

1.4 List of Ecma documents considered

Ecma/TC39/2014/012	Draft Standard ECMA-262 6th edition, Rev. 23, April 5
Ecma/TC39/2014/013	Minutes of the 39th meeting of TC39, San Francisco, April 2014
Ecma/TC39/2014/014	Venue for the 40th meeting of TC39, Menlo Park, June 2014
Ecma/TC39/2014/015	TC39 process description by Mr. Rafael Weinstein
Ecma/TC39/2014/016	Agenda for the 40th of TC39, Menlo Park, June 2014 (Rev. 1)
Ecma/TC39/2014/017	Call for Volunteers for Liaison Manager to Ecma TC39
Ecma/TC39/2014/018	Decorators slides, June 2014
Ecma/TC39/2014/019	Async Generators slides, June 2014
Ecma/TC39/2014/020	JSDoc, JSIDL, Code Editors slides, June 2014
Ecma/TC39/2014/021	"A better future for comprehensions" slides, June 2014
Ecma/TC39/2014/022	"Closing iterators" slides, June 2014

2 Adoption of the agenda (2014/016-Rev1)

Agenda for the: 40th meeting of Ecma TC39

```
in: Menlo Park, California, USA
on: 04 - 06 June 2014
TIME: 10:00 till 17:00 PST on 04th and 05th of June 2014
      10:00 till 16:00 PST on 06th of June 2014
LOCATION:
    Facebook
    1601 Willow Rd,
    Menlo Park, CA 94025
    USA

CONTACT:
    Jeff Morrison <jeffmo@fb.com>
```

Please register before 28th of May 2014.

1. Opening, welcome and roll call

i. Opening of the meeting (Mr. Neumann)

ii. Introduction of attendees
iii. Host facilities, local logistics
iv. Dinner - 630 PM - Gravity
v. 544 Emerson St - Palo Alto

2. Approval of the agenda (2014/00?)
3. Approval of the minutes from April 2014 (2014/0??)
4. ECMA-262 6th Edition

i. Review Latest Draft (Allen)
ii. Schedule Review (Allen)
iii. `arguments` and `caller` poisoning & static class methods of same name (Brian)
iv. `return` on generator/iterator type. (Dave)
v. Comprehensions and future-proofing. (Dave)
vi. Reserving additional words (`yield` and `async` in particular) in modules. (Dave)
vii. Removal of Realms API from ES6 (Alex, Mark)
viii. Arraybufffer neutering (Allen - Mozilla)
ix. Other open technical isssues

5. ECMA-262 7th Edition

i. Asynchronous Generators
ii. Functional record extension and row capture: https://gist.github.com/sebmarkbage/aa849c7973cb4452c547

6. Test 262 Status
7. HTML integration (?)

i. `<script type=module>`: https://www.w3.org/Bugs/Public/show_bug.cgi?id=25868
ii. Event loop: https://github.com/domenic/promises-unwrapping/issues/108
iii. HTML imports et al: http://esdiscuss.org/topic/integrating-the-webs-dependency-systems

8. JSdoc, etc. (Simon - IBM)
9. Report from the Ecma Secretariat (2014 ? GA Report)
10. Date and place of the next meeting(s)

i. July 29 - 31, 2014 (Redmond, WA - Microsoft)
ii. September 23 - 25, 2014 (Boston, MA - Bocoup)
iii. November 18 - 20, 2014 (San Jose - PayPal)

11. Closure

The agenda as posted on Github above was approved.

3 Approval of the minutes from the April 2014 Meeting (2014/013)

Ecma/TC39/2014/013, the minutes of the 39th meeting of TC39, San Francisco, April 2014 were 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.

According to 2014/013: ”At the request of the TC39 RF TG, TC39 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 TC39 of the start and end of the opt out period per the procedures identified in TC39 operating rules.”

This plan did not come through:

TC39 has decided that ECMA-262 Edition 6 should be approved only in June 2014 (so 6 months later). Too much work which is not possible to finish by December 2014. Therefore also the “opt-out” as described above has been postponed.

It is still planned that after ES6 the ECMAScript standard will have yearly editions, including new add-ons.

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 was under letter ballot in Ecma and will end in May, and it has been approved. The new webpage for submitting 3rd party test and software contributions have been put on the Ecma website. ES 6 tests can be submitted both by Ecma members and non-members. The submission process still has some technical problems, so it is the best also to involve the Secretariat via Email, in order to make it sure that the submission get to TC39. Details of the work sharing is fixed in such a way that the Ecma secretariat receives the submission, checks if the package is complete and also if the administrative part is correctly completed, while TC39 takes care of the technical content (incl. the software modules).

6.1.2 Ecma Text copyright suitability for TC39

Nothing new on this issue since the San Francisco meeting.

6.2 Json

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 and now formally communicated to Ecma. 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). The persons of the liaison officers are still under search. TC39 should let the Ecma Secretariat known who will be the TC39 Liaison to IAB / IETF.

6.3 ITU-T SG16

Mr. Sebestyen reported that the ITU-T SG16 has been working some time on a new multimedia system and terminal called ITU-T H.325. In the current documents it is documented that for communication protocol they want to use JSON and they are also interested in a fast ECMAScript that can be used to download and perform any sort of media codec (both standardized and private) It is suggested that closer liaison between the groups and organizations should be established on those subjects This liaison should be discussed at the upcoming GA meeting in Heidelberg and the upcoming ITU-T SG16 meeting in Sapporo, JP.

7 Date and place of the next meeting(s)

Schedule 2014 meetings:

- 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

None.

9 Closure

Mr. Neumann thanked **Facebook** for hosting the meeting in Menlo Park, the TC39 participants their hard work, and **Ecma International** for holding the social event dinner Wednesday evening. Special thanks goes to **Ben Newmann** to take the technical notes of the meeting. Thanks to **Rick Waldron** for the remote help.

Annex 1

From: Rick Waldron [waldron.rick@gmail.com]

Sent: Friday, June 13, 2014 4:50 PM

To: Patrick Charollais; Istvan Sebestyen; Allen Wirfs-Brock

Subject: TC39 Meeting Notes from June 4, 5, 6

Here are the complete meeting notes for the 40th TC39 meeting on June 4, 5, 6

 # June 4 2014 Meeting Notes

Brian Terleson (BT), Dmitry Lomov (DL), Waldemar Horwat (WH), Allen Wirfs-Brock (AWB), John Neumann (JN), Rick Waldron (RW), Eric Ferraiuolo (EF), Jafar Husain (JH), Jeff Morrison (JM), Mark Honenberg (MH), Caridy Patino (CP), Yehuda Katz (YK), Niko Matsakis (NM), Ben Newman (BN), Filip Pizlo (FP), Sebastian Markbage (SM), Rafael Weinstein (RWS), Jaswanth Sreeram (JS), Alex Russell (AR), Istvan Sebestyen (IS), Simon Kaegi (SK), Arnoud Le Hors (ALH), Reid Burke (RB), Erik Arvidsson (EA), Brendan Eich (BE), Mark Miller (MM), Peter Jensen (PJ)

Adoption of the agenda

https://github.com/tc39/agendas/blob/master/2014/06.md

AWB: Adding section 8: Other ES6 Technical Issues

AWB: Adding "Report from ECMA secretariat" to HTML integration section

SK: Add an agenda item to discuss JSDoc and its use in editors

AWB: add "ArrayBuffer neutering" to ES6 agenda items

Conclusion/Resolution

- Agenda Approved
- Minutes from April 2014 approved

Scheduling of TC39 meeting

JN: Meeting schedule changes are problematic

- In november, we'll select for next year.
- The current meeting dates need to be approved now and committed to

YK: Didn't weigh scheduling concerns of champions vs. non-champions well

AWB: Hard to weigh those concerns until we know what will be talked about

Next:

- 29-31 July 2014 at Microsoft in Redmond, WA (USA)
- 23-25 September 2014 at Bocoup in Boston, MA (USA)
- 18-20 November 2014 at PayPal in San Jose, CA (USA)

(posted: http://www.ecma-international.org/memento/TC39-M.htm)

AWB: Should set agendas based on schedule, rather than other way around

JN: I will propose meeting dates in September for your consideration

Agenda for this meeting

DH: Talk about generators tomorrow (Thursday), to accommodate Brendan Eich and potentially Andy Wingo

4.2 Schedule Review

AWB: Goal was to present ES6 to ECMA general assembly for approval later this year, meaning we need to approve finished draft at September meeting, meaning we need a finished draft at July meeting. I don't see any way we can have even an approximately finished draft by July. Also starting to get good implementer feedback which is causing spec churn, but there are pieces missing review. If we ratified what we have now we'd be missing serious implementation feedback.

AWB: Propose we slip our publication target by 6 months. Spec will still be feature frozen. We're just fixing bugs and accepting implementer feedback.

Concern 1: is this opening the door to feature creep? Important it doesn't do that.

Concern 2: Will this simply delay implementer work by 6 months?

Concern 3: Perception of committee failure?

DH: We should move forward on the work and message that we're doing so. Spec is not what drives the work, spec is the final capture of a stable consensus. I'm OK with the spec slipping.

WH: Don't want to repeat E4X (ECMAScript-for-XML) experience of shipping buggy spec, and later abandoning it.

?: Hazy distinction between calling something a bug vs. a revisited conclusion.

WH: The issues with E4X were clearly bugs.

AWB: This committee can tell the difference between fixing bugs and revisiting conclusions.

AWB: Focus energy on ES7 [if you want to revisit conclusions]!

AWB: Other factor playing into spec review is test262, to which we've just gotten approval to accept contributions

JN: Can we approve the change of dates (presenting finished draft to TC39 committee in November rather than July)?

AWB: Everything we want to include is in the spec at some level of quality, it's just a matter of ensuring quality.

YK: If you're an implementer, you shouldn't treat this as a slippage.

AWB: Should we start naming spec editions using years rather than an incrementing number (ES2015 vs. ES6)?

AWB: Let's not throw this in with messaging about the schedule postponement.

YK: The fact that people are used to ES5/ES6/etc. will make the change important as a messaging technique.

BE: Let's sleep on it.

Conclusion/Resolution

- Postponement of presentation of final ES6 draft agreed upon
- Switch to yearly naming convention to be slept on

Update on changes to spec since last meeting

AWB: See "Draft Rev 25" section of http://wiki.ecmascript.org/doku.php?id=harmony:specification_drafts

AWB: Object.prototype.toString.call(proxy) will be either [object Object] or [object Function] depending on whether the proxy is callable, so that there is no way to tell that an object is a Proxy

MM: Have to choose between use cases, and being able to hide Proxy-ness is the better choice

AWB: Already considered/dropped isProxy, so using O.p.toString as a back-door would have been inconsistent with that decision

AWB: Another Proxy issue MM and I talked about: custom property descriptor attributes on Proxies, specifically adding additional attributes to the descriptor that the Proxy could recognize.

```js
Object.defineProperty(obj, propName, {
  value: whatever,
  customAttribute: attrValue
});

// Should the customAttribute be preserved here?
Object.getOwnPropertyDescriptor(obj, propName).customAttribute
```

MM: Decision was to disallow/ignore custom attributes. Might add an additional object metadata API in future versions of the language, but it will be separate from property descriptors.

AWB: for (let x in x) -- "bombs" because referring to x before temporal dead zone ends.

```js
let o;
for (let x in (o={ f(){ return x }, x=42 }) {
    o[x](); // gives TDZ error
}
```

MM: In for(;;) {} loop, can say for(let x = expr;...) -- that expression iterates the zeroth item of x?

DH: Yes

Conclusion/Resolution

- Leave as is currently spec'd

4.3 `arguments` and `caller` poisoning & static class methods of same name

BT: Should we keep doing the poisoning?

MM: We poison to make stack-walking code fail noisily, early

AWB: In the long run, is more confusion likely to reside in not being able to have static methods called "arguments" or "caller," or not being able to get away with stack walking?

YK: Having strict mode code dynamically allow static methods named "caller" and "arguments" when non-strict mode code forbids them seems strange for strict mode (which usually only forbids things).

BT: If we can't get rid of arguments and caller can we make the properties configurable?

MM: I no longer object to making .caller and .arguments configurable, so that you can make static methods with those names.

Conclusion/Resolution:

- Make the .caller and .arguments configurable properties for all functions (except sloppy mode functions), so that they can be overwritten.

4.4 `return` on generator/iterator types

Conclusion/Resolution

- Postpone until tomorrow morning at ~10:30, after Istvan's talk at 10:15.

4.6 Reserving additional words (`yield`, `async`, and `await` in particular)

YK: Want to support async functions in top-level scripts.

AWB: Is a module with no imports essentially a top-level script?

AWB: We've gotten away with introducing new contextual keywords without reserving them.

MM: Seems useful for code that uses `async` in an arguably legal context [e.g., as a variable name] to be able to continue doing so.

YK: There's a cost to something being legal in a top-level script but not legal in a module.

MM: `yield` and `await` are already reserved (i.e. can't be variable names) in generators and async functions, respectively

DH: think about async modules as an argument in favor of reserving `await` at the top level of modules

BE: strict mode already reserves `yield` and `async`

MM: And modules are strict!

Conclusion/Resolution

- Reserve only `await` as a keyword in modules.
- `async` and `module` work as contextual keywords because of their position (which can't conflict with variable names).
- `yield` is reserved in strict mode, so doesn't need additional reservation.

4.7 Removal of Realms API from ES6 (postponement to ES7)

MM: No serious draft implementations of Realms API, even by implementors who have implemented module loaders.

MM: And it's the most security-sensitive part of the module loading spec.

MM: Can we have modules and module loaders without Realms?

DH: Yes, when I proposed I mentioned that modules can come without Realms if necessary.

DH: Let me state my point: I don't want to slow down Realms being implemented, and I worry if we remove them from the spec implementors won't have anything to go on when implementing modules or experimenting with Realms

AWB: Can we defer to July after Dave and I have had enough time to work through modules.

DH: I do want us to get them right, but I don't want to see it continually kicked down the road

AR: Want to be able to use Realms to specify (and build polyfills for) e.g. same-origin `<iframe>`s.

MM: We want to explain the existing behavior of same-origin `<iframe>`s in terms of Realms, and we can't do that until we specify the Realm API

MM: Don't defer specification until ES7 now, but let implementors start implementing the Realms API, and if we end up deferring it, no one will be surprised.

DH: Talk to Domenic about spear-heading a polyfill?

YK: Not as much interest in Realms in the Node.js community.

DH: Polyfill hard to do for the parts of the API that are the most interesting.

AR: Hixie and/or Anne VK (or Boris Z) specify these kinds of things for the HTML spec, so maybe we should have a small meeting with them.

Conclusion/Resolution

- Postpone decision to defer Realms to ES7 until next meeting.
- This does NOT amount to postponing to ES7!
- Bear in mind that it is separable from module loading, so we can still go either way (ES6 or ES7).

4.8 ArrayBuffer neutering

AWB: What did we decide this morning? (MM wasn't here then)

DH: Planning to have a discussion with implementors to decide what makes the most sense (throwing an exception on property access, or making .length be 0)

MM: Feel strongly there should be an isNeutered test operation, so you don't have to use a try-catch to test neuteredness (if we go with the exception throwing behavior)

WH: Why should the length return 0 instead of throwing on a neutered object?

DH: Lets you have a fast case in element lookup that checks for a 0 length.

WH: That doesn't answer the question. You can internally have a 0 length on neutered objects for fast lookup but throw if someone calls the length method.

DH: Let's think about using a different metaphor/terminology that isn't vaguely sexual (i.e. something other than "neutered")

MM: Not just the API (e.g. isNeutered) but also the spec language

DH: Probably not super-high demand for isNeutered because you usually know that you've used up an array and thrown it away. Code that needs to check for neuteredness (because the object might not have been neutered) is probably broken/confused/unlikely.

BE: Should isNeutered be a static method on Reflect or an instance method on ArrayBuffer?

DH: ArrayBuffer instance method

AWB: Is this feature-creep, though?

Conclusion/Resolution

- Don't add isNeutered yet, and expect clients use try-catch when accessing properties to determine status.
- Also remember to change the name. "Released"? "Vacated"?

6 Test-262 Status

BT: Have the CLA as a PDF document, now need to tell everyone with pull requests to go to the website, download the PDF, sign it, and send it to us.

AWB: How can we streamline that process?

BT: Continuing to have that conversation with Istvan and Norbert.

AWB: Where do contributors send the CLA?

BT: That could be clearer.

BT: Now have support for Promises in the test harness, but it's not clear how to keep promises

BN: Couldn't the test harness call promise.done(...)?

YK: That's doable, as is having the test harness check whether the promise is resolved or rejected.

EA: Traceur uses standard mocha/chai/tdd testing library.

BT: Willing to give up purity in the tests for the sake of coverage (overlap between test suites is fine, e.g. if both Traceur and Regenerator contribute their test suites).

AWB: Should we have a notion of a test suite rating, indicating whether they are "spec-level" or less formal than that.

BT: Don't want to discourage people from submitting less formal tests.

MM: Are we running tests that should succeed in both strict and sloppy modes in both modes?

BT: I'll open an issue on https://github.com/tc39/test262 for that.

EA: Tests don't closely follow the spec, and don't tend to be written as test262 tests from the start, just because it's easier.

JM: Not clear how to run tests against new implementations.

YK: Transpiler projects (Traceur, esnext) should make an effort to make this easier.

BT: How did test suite review work for ES5?

AWB: I did a lot of it when I was at Microsoft.

MM: Want to do commit-then-review for tests.

BT: Yes, because then it's easier to weed out tests because (correct) implementations fail them.

MM: Who would sign up to write tests for a chapter of the spec?

AWB: Brian should create twitter.com/testecmascript to evangelize the testing message.

BT: Need to reflect the depth of coverage as well as breadth, and that's hard to capture in a list.

JM: People love progress bars, gamification?

BN: What language features can be used in tests?

BT: Philosophy is to use only ES5 outside the specific ES6 features under test.

Conclusion/Resolution

- No spec decisions to be made here, just a status report.

Object.observe status

Want a way to express narrower interest via the listening API (for performance) rather than filtering in the callback, e.g. listening for changes to just one property, rather than all changes to the object.

DH: Is the `Object.observe` protocol extensible enough to support Map and Set modifications?

YK: `Object.observe` is meant to work out of the box for data property mutations, but you can also emit a synthetic change record for object types like Map and Set

Conclusion/Resolution

- YK to draft proposal for changes to the spec and present it to champions.

IO Streams

https://gist.github.com/annevk/3db3fbda2b95e5ae9427

AR: Domenic's streams repository represents a consensus among a relatively large number of people, and a small minority outside of that favors the Rx style [Observables?].

DH: Re: asynchronous iteration, we've laid the groundwork for an iteration protocol that can be both synchronous and asynchronous. Expect async/await to be very popular, building on Promises.

DH: We have single-valued synchronous functions (normal functions), multi-valued synchronous functions (generators), single-valued async functions (async/await), but no multi-valued async functions. JH thinking about this in terms of Observables, from a C# perspective/precedent.

AWB: Re: error types and detection, IO APIs tend to have web platform dependencies, like DOMError/DOMException, which we can't invoke in the ES7(?) spec.

YK: Difficult to reliably subclass ES Errors in general, regardless of whether developers will pay attention to Error type distinctions.

MM: `class MyError extends Error {}` gives a distinct new error type, right?

AWB: Have to do it right [e.g. invoke super() in constructor to record stack trace?], but not impossible.

MM: Wasn't there a proposal to introduce typed catch clauses?

BE: catch guards were implemented in SpiderMonkey a long time ago.

YK: That would help with reporting errors to the user.

BE: Really?

BE: Termination-style exception handling won out over resumption-style a long time ago, and that was probably a failure.

BE, YK, MM: Discussion about whether rejected Promises support interpreting exceptions contextually in a way that allows helpful errors to be reported to the user.

MM: Example: IndexOutOfBounds exception can bubble up from calls to a lower-level table access abstraction, but be caught by the client of a higher-level table access abstraction, and the only thing the client can conclude is that her input parameter was out of bounds, when that isn't necessarily true or helpful.

BE: requestAutocomplete example: how to Promise-ify the response (as recommended by Domenic: http://lists.whatwg.org/htdig.cgi/whatwg-whatwg.org/2014-April/254143.html)? Do you need distinct Error types in order to respond differently to different kinds of rejection?

MM: That distinction wouldn't tell you which stage of a .then chain produced the error, necessarily, unless you were careful about encoding that information in the value of the Error.

BE: DOMException vs. other exception types doesn't seem like a helpful distinction.

AWB: Should we try to replace WebIDL? (fourth bullet point from the gist above)

DH: Browser implementors love WebIDL, so anything that replaces it has to be as convenient as that. YK's idea: the new interface description language would prepend Legacy to existing WebIDL types, but still support them.

MM: What about a design language that compiles to WebIDL?

DH: Problem: people explicitly argue against better interface design because it's not convenient/expressible in WebIDL.

MM: Right, the path of least resistance in WebIDL is not good JavaScript.

DH, AR: TypeScript seemed like a way to define signatures of APIs, but was solving a different problem.

DH: Need a way to express what kind of implicit conversions are applied to passed-in values (something that TypeScript doesn't have).

YK: Also want to be able to express APIs in terms of function/method overloading (different behaviors for different input types), which is more like TypeScript than WebIDL.

AWB: If no work happens to build a better IDL, we'll be stuck with the status quo.

YK: Want to be able to describe `Promise<T>` result types as such, rather than `{ then: ???, catch: ??? }`

SK: Willing to start working on a new IDL design, with help.

DH: Want to capture duality between Array.isArray arrays and array-like objects, and instanceof-Promise objects vs. { then: Function } objects.

SK: Can we improve whatever was lacking about TypeScript?

AR, YK: TypeScript types don't mean quite what you think they mean (Number, String).

AR: Union types not supportable in TS today, so you can't express some important overloadings.

Conclusion/Resolution

- As JN points out, we really have to get in contact with the W3C folks and make sure they're sympathetic, and we also really have to commit meaningfully to working on a new IDL, neither of which seems like something this committee can decide unilaterally.
- Worth exploring, but no commitment at this date.

Web APIs that could move to ES(7+)

Proposals from AnneVK: https://gist.github.com/annevk/6bfa782752dde6acb379

MM: URL and fetch() together sort of imply that HTTP (vs html/dom/etc) is moving from browser into the lang -- and that's...

MM: Because http is being used in non-browsers, that's interesting

DH: Node isn't waiting on APIs for TC39 (it does a lot of its own things)

DH: Receptive to idea to shift some of these into "the lang", but unclear who to work with on some of these

BE: When you say "the lang", you mean separate spec

agreement

BE: If self-hosted, how much does it need to be anything more than code on GitHub

YK: Engines can do opts that can't be done in userland

Conclusion/Resolution

- General openness to moving these APIs into ES7+, but no firm commitment yet on any single item.

Expanded Conclusion/Resolution for the ArrayBuffer neutering discussion:

Conclusion/Resolution

- Don't add isNeutered yet, and expect clients use try-catch when accessing properties to determine status.
- Also remember to change the name. "Released"? "Vacated"?
- Any attempt to access (read or write) binary data elements of an ArrayBuffer that has been "neutered" will throw a TypeError exception.
- Accessing the byteLength property of an ArrayBuffer that has been "neutered" will throw TypeError exception.
- Have not yet decided what happens to the the "length", "byteOffset", and "byteLength" properties of a TypedArray whose underlying ArrayBuffer gets neutered.
- Keep the behavior that out of bounds reads of a TypedArray (whose buffer has not been neutered) returns undefined (MM: or throws, in strict mode) and that out of bounds write are no-ops (MM: throws in strict mode).

TC39 recognizes that the above are breaking changes relative to the Khronos spec. but we believe that the behavior of silently treating neutered buffers as 0-length buffers is seriously flawed and would set a terrible precedent for any future "transferable" data types.

 # June 5 2014 Meeting Notes

Brian Terleson (BT), Dmitry Lomov (DL), Waldemar Horwat (WH), Allen Wirfs-Brock (AWB), John Neumann (JN), Rick Waldron (RW), Eric Ferraiuolo (EF), Jafar Husain (JH), Jeff Morrison (JM), Mark Honenberg (MH), Caridy Patino (CP), Yehuda Katz (YK), Niko Matsakis (NM), Ben Newman (BN), Filip Pizlo (FP), Sebastian Markbage (SM), Rafael Weinstein (RWS), Jaswanth Sreeram (JS), Alex Russell

(AR), Istvan Sebestyen (IS), Simon Kaegi (SK), Arnoud Le Hors (ALH), Reid Burke (RB), Erik Arvidsson (EA), Brendan Eich (BE), Mark Miller (MM), Peter Jensen (PJ)

quiet typing

(Agenda item 5.2 added)

(Istvan on the phone discussing non-member CLA process)

AR: It sounds like we're making the process difficult for non-members simply because it's currently difficult for members. Can we change the goal to just make the process simpler for everyone?

IS: We just need the process to be the same for members and non-members. That is the current process -- we need the pdf form

AR: Can you take it back to the assembly(?) as a request from this TC that we'd like to simplify the process for signing the CLA to a click-through form for everyone [members and non-members]

IS: Yes, I can do that

discussion

Conclusion/Resolution

- Allen will continue to work with Istvan on this
- Not clear how to proceed concretely other than having members continue to push for this
- Current process includes printing pdf, signing it, scanning it

JSDoc*, JSIDL*, Code Editors

(Ask for slides)

SK: Type inferencing is hard

SK: JSDoc*, we need help from the user.

SK: Standardize JSDoc?

SK: JSDoc Proposal

AR: We are the only group that can add syntax so we should not put semantics in the comments.

AR: Interested in designing a type system

DH: I'm skeptic that we can design a type system for ES.

DH: Describes kinds of type systems: sound, unsound, guards

WH: I tried it for ES4. Came to the conclusion that a complex static type system would not work for ES's plethora of array-like things. On the other hand, more primitive type systems for things such as strings, numbers, classes are eminently doable and useful.

WH: I had a proposal for guards. These are more like a max min solution.

DH: Guards are too dynamic.

WH: In order for a type system to be useful, it must be enforceable in some form in order to catch erroneous or out-of-date type annotations.

DH: It is about as much a green field as defining a new language. It can mean so many different things.

JM: Would like to explore types for documentation at least.

DH: If there is no standard there will continue to be a proliferation of tools.

WH: We know this is a large space to explore. That does not mean that we should not explore it.

YK: If we make this a priority it will take years and years of discussion, dropping everything else.

BT: We can manage doing more than one thing at the same time.

AWB: This discussion feels like the early discussions of class syntax. Yet we were able to make progress.

DH: TypeScript is the strongest proposal at this point. Guards have serious problems for structural types. A sound type system (for JS) is still an academic research topic without a solution in sight.

YK: TS type system is incompatible with ES6.

YK: TS is a fork of ES5.

WH: These problems of incompatible forking are exactly why we need to take over this work.

f

AR: I want this for ES7

AWB: We haven't yet started to prioritize our next work items.

DH: The cost is very very high and IMO there are more important thing to work on than types.

WH: IMO documentable types are one of the highest priorities for us now. There is a lot of external interest as evidenced by the various external efforts.

BN: Strict scrutiny (e.g. from DH) can and should come before the proposal comes before TC39.

SK: Back to slides... Not proposing a type system. Implementors are interested in standardizing JSDoc..

SK: JS IDL?

SK: Type definition...

Conclusion/Resolution

- Stage 0 to work on JSDoc

Closing Iterators

(Dave presenting, send slides!)

Slides: https://speakerdeck.com/dherman/closing-iterators

Notes that key stakeholders have signed off on this (Andy Wingo, Luke Hoban, ... others)

DH talking about breaking out of loops and calling .return() when an abrupt exit occurs in the loop

WH: Does a yield inside a for loop call the return method in the normal case?

BN: No.

WH: What if that yield then gets killed by a break in the caller?

BN: Show example.

WH:

```javascript
function * a() {
  for (let x of y) {
    ...
    yield ...
    ...
  }
}

for (let b of a()) {
    break;
}
```

** Discussing: How should abrupts inside a yield* get bubbled? **

BN: Any yield on which the generator (or any nested delegate generators) are currently suspended effectively become returns, so the return value "bubbles" all the way out to become the result of the outermost generator.

WH: (back to his code example) What does a's return method do?

MM: The break in the for loop calls the generator's return method -- causes the yield to return

DH: Any abrupt (but only abrupt completion) of a loop [calls .return() method?]

** continues through slides **

DH: Next issue: What if iterator, when given opportunity to stop, decides to keep going?

(see slides for details)

```
function* g() {
    yield;
}
g = g();
```

g.next();
g.return(42).value === 42;

WH: where does the 42 passed to g's return method go?
BN: It returns out of g with 42.
WH: Is there any way for g to get at the value 42?
DH: No.

Brendan's Example
```js
function* gen() {
    try {
        yield 1;
    } finally {
        yield 2;
    }
}

for (let i of gen()) {
    break;
}

// The for-of loop unrolls to this:
var g = gen();
g.next(); // { value: 1, done: false }
g.return(42); // { value: 2, done: false }
// Not executed because of the break from the loop:
g.next(); // { value: undefined, done: true }
```

AWB: Important to remember that none of the costs (e.g. checking for a "return()" method) happen on "normal" completion of the loop -- only abrupt exits from the loop

DH: Let's talk bikeshedding: return() vs close()

BN: close() doesn't suggest that you can pass an argument

MM: .throw() is reflecting the throw construct within the generator, thus .return() makes sense

AWB: [strawman idea?] What about forbidding yield in try blocks?

RW: (silently, because I'm muted) Disagree with restricting use of `yield`

RW: Since we have an additional six months of "quality control" time with the spec, should we consider dealing with this directly as part of the "quality control" effort? Opposed to limitations on yield in try/catch/finally.

AWB: that could be applied to any subject that comes up.

DH: forbidding yield in try blocks is a non-starter because there would then be no way to ever do any kind of cleanup on a yield using finally

BE: Python 2.5(?) added similar restrictions and then later relaxed them, which proves that there are use cases for yield in try-finally.

AWB: You'd have to structure your loop with an outer try-catch, yield takes away guarantee that finally would run

DH: Understand this is intended as temporary (i.e. to avoid adding complexity to generators last minute), but worst thing would be if we landed on this as a permanent semantic

DH: Agreement is that this is right semantics, there's a question as to whether we can get there in time

AWB: I worry about the precedent of adding changes like this now

BN: I will be happy to review the changes to chapter 25 (Generators and Promises) that are necessary to capture these changes.

AWB: Still worried because of the precedent this sets.

Conclusion/Resolution

- Ratifying DH's proposal ("proposal" to be clarified), with the fallback (in case we run out of time, or decide to prioritize other work) of forbidding yield in try-finally blocks and making abrupt exits from for-of loops put the iterated-over generator into the GeneratorComplete state (which is a semantic change from leaving it in the GeneratorSuspendedYield state).

Generator comprehensions (slides plz)

Slides: https://speakerdeck.com/dherman/a-better-future-for-comprehensions

** Basically DH is proposing deferring comprehensions to ES7 with some minor changes to future-proof **

let a = for (x of a1) for (y of a2) if (y > x) {x,y};

WH: What type does this return?

?: It depends on the type of a1.

WH: Generating an array in the proposed world is ugly. If you want an array, you now need to convert the result of the comprehension to an array.

BE: The basis case doesn't work in the new design. To create a generator, you need to start with a generator. Awkward to write the first generator.

BE: problem is that a1 is evaluated eagerly rather than lazily

AWB: Where would we put Iterator.prototype in the Generator-related prototype object graph?

AWB: Hardest thing about this change is LAYING OUT THE DIAGRAM ALL OVER AGAIN: https://people.mozilla.org/~jorendorff/es6-draft.html#sec-generatorfunction-objects

BE: Inserting a new object without populating it with any methods invites breaking changes when we do populate it with methods like .flatMap.

JH: Really need to go with type-directed comprehensions (letting the iterable object determine the type of the comprehension) rather than the current syntax-directed syntax: (...) and [...].

WH: Worried that we'll rat-hole on the new proposal, which is pie-in-the-sky with a number of identified deficiencies. The current proposal is solid. The perfect is the enemy of the good here.

Conclusion/Resolution

- Defer comprehensions from ES6 to ES7 to work on the more general type-directed, method-based approach to supporting comprehensions for arbitrary iterable types (arrays, generators, streams, parallel arrays, etc.).

7.1 `<script type=module>` status update (from DH)

DH: Would really rather have `<module>import { foo } from "bar"; ...</module>`, which is like `<script>`` but async, strict mode, has its own top-level scope, and can import declaratively (using ES6 module import syntax) from other (named) modules.

DH: `<module name="qux">` creates race conditions with HTML imports (part of WebComponents).

YK: People who saw named HTML module tags though you should mix html imports w named module imports

YK: When you have packaging solution (SPDY, etc), you no longer need named modules

MM: `<script type="module">` would inherit the special termination rules of </script>, whereas old browsers might not handle <module> the same way, since that tag name doesn't mean anything special in old browsers

AR: `<script type="module">` means the browser won't even try to parse it as JS, which is what we want [so that we can execute the script contents as a module, via some sort of polyfill]

DH: `<script type="worker">` might also need to have the `<script type="module">` semantics, and type= attribute syntax makes it hard to mix and match those attributes; maybe <script worker module> would be better? (i.e. the type attribute values become optional value-less attribute names)

DH: The difference between `<script type="module">` and <module> is that as long as there's … you always have the option of writing `<script>System.import("main.js")</script>`

TODO: Get DH to clarify this point when we edit the notes.

MM: The `<module>` tag still has HTML misparsing consequences that <script> wouldn't have.

YK: The success of <template> proves that we can introduce a new `<script>`-like `<module>` tag.

DH: Need to create ways of doing this with pure JS, like `<script>System.import("main.js")</script>` and then add new HTML sugar later, like `<script type="module">`.

AR: [note taker (BN) may be misinterpreting] The JS API remains important even when we have HTML sugar.

Conclusion/Resolution

- Adopt `<script type="module">` and expect transitional code to do `<script>System.import("main.js")</script>`.

7.3 HTML Imports

BE: Recast as `<script type="import">` rather than `<link rel="import">`?

Conclusion/Resolution

- Consensus not reached.

7.2 Event loop

The issue is interleaving Promise response tasks with other tasks.

MM: Let's just not use the term "task" because it's confusing with other tasks (microtasks, etc).

Brainstorming names: job, tick (seems too close to process.nextTick), slice, turn, chore (lots of enthusiasm for chore!), quest (fits with realms!), schlep

Conclusion/Resolution:

- Consensus not reached.

Two or Three-day TC39 meetings in the future?

AR: Breakout sessions the third day?

EA: Can notes be taken in those breakout sessions?

MM: Breakout sessions for the rest of today?

Modules breakout session:

Notes: https://gist.github.com/ericf/a7b40bf8324cd1f5dc73

 # June 6 2014 Meeting Notes

Brian Terleson (BT), Dmitry Lomov (DL), Waldemar Horwat (WH), Allen Wirfs-Brock (AWB), John Neumann (JN), Rick Waldron (RW), Eric Ferraiuolo (EF), Jafar Husain (JH), Jeff Morrison (JM), Mark Honenberg (MH), Caridy Patino (CP), Yehuda Katz (YK), Niko Matsakis (NM), Ben Newman (BN), Filip Pizlo (FP), Sebastian Markbage (SM), Rafael Weinstein (RWS), Jaswanth Sreeram (JS), Alex Russell (AR), Istvan Sebestyen (IS), Simon Kaegi (SK), Arnoud Le Hors (ALH), Reid Burke (RB), Erik Arvidsson (EA), Brendan Eich (BE), Mark Miller (MM)

Follow up on deferring comprehensions

DH: IMHO Andy Wingo's email made my case stronger since it added another dimension that we need to take into account.

AWB: Let's leave it in until next meeting and try to resolve this before then.

DH: This is basically a go/no go question.

AWB: No spec changes will be done until after we have consensus.

DH: There is no rush to get it out of the spec.

Conclusion/Resolution

- Resolve this online
- Make a go/no go at the next face to face meeting.

Block scoping issues

BT: In sloppy mode

```js
foo();  // maybe?
if (test) function foo() {};
foo();  // yes
```

AWB: Wanted to hold the line and not cover this.

BT: One option is to treat the FunctionDeclaration in if as if there was a Block around it.

AWB: The changes in the grammar to support this would be nasty.

WH: Don't want to pollute strict or non-browser-compatibility mode with this stuff.

AWB: This won't affect strict mode.

WH: Then it shouldn't be in the main grammar. Leave it in Annex B.

BT: Can we hand wave this in Annex B? Can we just use prose to say that it behavase as if there was a block around it?

BE: Without grammar we cannot validate

WH: I'm not going to validate the grammar in Annex B. I already know there's ugliness there.

Conclusion/Resolution

- Substatement function declarations will be added to Annex B unless we can convince ourselves it's safe to remove.
- Allen will investigate how to specify in Annex B (including whethe we need to specify the grammar or hand-wave).

Initializer in for-in

AWB: Got feedback that there are 11 sites that use the code pattern.

WAB: https://bugzilla.mozilla.org/show_bug.cgi?id=748550

Conclusion/Resolution

- Have an engine try to remove support and report back. Otherwise it will have to be added back to the spec.

Rest properties and spread properties (Sebastian Markbåge)

Presentation notes: https://gist.github.com/sebmarkbage/aa849c7973cb4452c547

SM: Rest properties

```js
let {x, y, ...z} = {x: 1, y: 2, a: 3, b: 4};
x;  // 1
y;  // 2
z;  // {a: 2, b: 4}
```

SM: Spread properties

```js
let n = {x, y, ...z};
n;  // {x: 1, y: 2, a: 3, b: 4}
```

SM: The [[Prototype]] will be %ObjectPrototype%

MM: It should not use the original prototype since it does not have the same properties

WH: What about about non enumerable own properties

SM: Only enumerable are copied.

WH: What about accessors?

SM: Either snapshot the value or copy the accessor.

WH: How do accessors behave in existing destructuring?

AWB: They snapshot the value.

WH: So that's how accessors should behave here too.

SM: Precedence set by `Object.assign`. So that means that we would use a `Get` to get the value.

WH: Does destructuring look into prototypes? Concrete example:

`let {x, y, ...z} = {y: 2, a: 3, b: 4}` where the RHS object inherits from a prototype whose x property is 7. What is the value assigned to the x variable?

SM: Yes, it'd assign 7 to x.

WH: Then it would be bizarre for x to snapshot properties from the prototype but for z to ignore them

MM: Can you do let {x, y, ...something-more-complex-than-an-identifier}?

SM: No.

MM: Properties must be added via defineProperty, not put.

AWB, BE: Yes, that's how object literals work. Don't want to accidentally run setters in the prototype.

MM: This extends possibilities for dynamic name clash errors in strict mode.

[Discussion about whether dynamic or static name clashes in object literals should be errors or rightmost-one-wins.]

WH: Whatever we do, let's be consistent. Don't want one rule for explicitly named properties, another confusing rule for dynamically generated properties.

WH: If the spread only uses own properties than it is inconsistent with destructuring.

YK, DH: No, the problem is to figure out the key names for ... and after that we do a Get

MM: Are methods enumerable? (Yes) Then I think this is fatal

DH: We have to look at existing code and all existing code uses for-in+hasOwnProperty

WH: I want substitutability, so that I can pick of properties one by one.

BE:

```js
z = Object.create({w: 42});
obj = {x, y, ...z};
```

WH: Wants w to show up in obj.

JM, BN: If you want to pick up objects from the proto you can use `__proto__`, as in `{ x, y, ...z.__proto__, ...z }`. The other way does not hold (if we include non own by default).

MM: Very confortable with Yehuda's rational that we use own properties to find the property names and then use Get to get the value.

YK: Destructuring is sugar for Get.

WH: Wants destructuring to be sugar for HasOwn+Get as in:

```js
var {x} = obj;
=>
var x;
x = obj.hasOwnProperty('x') ? (THROW?) : obj.x;
```

WH: What is the compelling use case for having ... in patterns?

DH: Consistency with ... in spread.

WH: Doesn't necessarily follow. ... in spread constructs objects. The proposed ... in patterns deletes properties in a rather arbitrary way. As an analogy, we have + to concatenate strings but no equivalent operator to undo the effects of +.

SM: [example of passing function arguments to a wrapper]

MM: In this case one could make a use case for either removing explicitly named properties in the wrapper or leaving them in the object before passing them on.

WH: I can see a weak use case for ... in patterns, but not a strong one yet.

RW: (Arguing for not reaching into prototype in destructuring) The final agreement was that built-ins will always create non-enumerable properties, to stay consistent with the spec. User created object and class properties will remain enumerable and it will be left up to user code to decide how to handle these.

MM: (re: decorators proposal)

YK: my decorators can make individual methods or all methods on a class non-enumerable

MM: I am ok with removing the constraint that duplicate dynamic object properties throw (in strict mode) with the caveat that we also remove the same constraint for duplicate static properties.

Conclusion/Resolution

- For both strict and sloppy object literals, both computed and static properties no longer have any duplicate name checking.
- Instead, the semantics are strictly left-to-right (i.e. rightmost instance of the duplicate name wins over previous properties of the same name).

Async Generator Functions (Jafar presenting)

(Jafar to send slides)

DH, YK: Consider the cost of having two semi-compatible but different APIs, Streams and Observables.

DH Might be worth having a layered API in order to let people drop to low level for perf reasons.

YK: We should get more clarity on this.

:: Jafar explains slides ::

** Discussing .observe() / Object.observe() **

YK: Object.observe() takes a callback now, but it doesn't return anything. Could return an observable in the future.

DH: We need to resolve this soon -- having Object.observe() and Observable.prototype.observe is too confusing.

JH: I really prefer the analogy between iterators and observers (observers are inverted iterators). Very hard to think of a new term that fits better than Observable/Observe

YK: JS community has established that .observe() does one thing, and observable/observe doesn't quite fit that pre-existing convention

** discussion about naming of "Observer/Observe" and whether there are alternatives **

BE: Could do -- Iterable : iterator :: Observable : "observer"

YK: Why not subscribable/subscribe

JH: Want to get away from thinking about things in terms of pub/sub. IME with training devs, decoupling brains from pre-existing notions of pub/sub is important

JH: By pub/sub I'm referring to observable minus .return() and .throw(). "pub/sub" pre-existing notions cause ppl to have difficulty with conceptualizing how they can compose with map/filter

DH: 3 competitors: Object.observe, streams, and ?

Need to rationalize

Need Domenic, Raf, and Jafar together to work it out.

WH: What does this do inside an async function*?

```js
while (...) {
  yield line1;
  await yield line2;
}
```

Is it just a backpressure issue, or can the first yield (without the await) surprise the observer in some kind of an unexpected state?

YK: I think it's important that the first value in an observable can be published synchronously. Importantly, there are well-motivated use-cases for Observables whose

initial value is ALWAYS synchronously available (like Object.observe), but there are no well-motivated use-cases for APIs that consume promises whose values

are ALWAYS synchronously available.

Conclusion/Resolution

- Async generators proposal is now in stage 0, for consideration as part of ES7.

- Need to consider more deeply how average programmers are going to make sense of this syntax without understanding it completely.