Minutes of the: ***55th meeting of Ecma TC39***

in: ***Menlo Park, CA, USA***

on: ***29 Nov. – 1st Dec. 2016***

1 Opening, welcome and roll call

1.1 Opening of the meeting (Mr. Wirfs-Brock)

Due to the absence of **Mr. Neumann (TC39 Chair) Mr. A. Wirfs-Brock** has welcomed the delegates at Facebook in Menlo Park, CA, USA. He has replaced **Mr. Neumann** at this meeting.

Companies / organizations in attendance:

Apple, Mozilla, Google, Microsoft, JS Foundation, Facebook, Netflix, PayPal, Shape Security, Airbnb, Tilde, Intel, Salesforce, Bocoup, Yahoo!, Imperial College London, Stanford University (Guest), Brave (Guest)

1.2 Introduction of attendees

1	Michael Saboff	Apple	Member
2	Istvan Sebestyen	Ecma	Secretariat (part-time, by phone)
3	Allen Wirfs-Brock	Ecma	Secretariat
4	Brian Terlson	Microsoft	Member
5	Domenic Denicola	Google	Member
6	Jeff Morrison	Facebook	Member
7	Brendan Eich	Brave	Invited Expert
8	Thomas Wood	Imp. College Lond.	Guest
9	Mark Miller	Google	Member
10	Jafar Husain	Netflix	Member
11	Jordan Harband	Airbnb	Member
12	Chip Morningstar		Invited Expert
13	Daniel Ehrenberg	Google	Member
14	Michael Ficarra	Shape Security	Member
15	Yehuda Katz	Tilde	Member
16	Kent C Dodds	PayPal	Member
17	Dave Herman	Mozilla	Member
18	Chris Hyle	Apple	Member
19	Waldemar Horwat	Google	Member
20	Shu-yu Guo	Mozilla	Member
21	If Bastien	Apple	Member

22	Leonardo Balter	JS Foundation	Member
23	Alex Russell	Google	Member
24	Brad Nelsen	Google	Member
25	Kevin Gibbons	Shape Security	Member
26	Rick Waldron	Boucoup	Member
27	Juan Topazo	Yahoo!	Member
28	Diego Ferraro Val	Salesforce	Member
29	Christian Mattarei	Stanford University	Guest
30	Peter Jensen	Intel	Member
31	Tim Disney	Shape Security	Member
32	James Kyle	Facebook	Member
33	Caridy Patino	Salesforce	Member
34	Eric Ferraiuolo	Yahoo	Member
35	Sebastian Markbage	Facebook	Member
36	Adam Klein	Google	Member

1.3 Host facilities, local logistics

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

2 Adoption of the agenda (2016/046-Rev1)

Ecma/TC39/2016/046-Rev1 Agenda for the 55th meeting of TC39, Menlo Park, November 2016 was posted in the TC39 documentation.

The final agenda was approved as posted on the GitHub as reprinted below:

Agenda for the 55th meeting of Ecma TC39

- **Host:** Facebook, Menlo Park, CA
- **Dates:** *Tuesday*, 29 November 2016 to *Thursday*, 01 December 2016
- **Times:**
  - 10:00 to 17:00 PDT on 29 and 30 November 2016
  - 10:00 to 16:00 PDT on 01 December 2016
- **Location:** 1 Hacker Way (Building 20), Menlo Park, CA 94025
- **Dinner**
  - November 30 2016
  - TBD
- **Contacts:**
  - Jeff Morrison (jeffmo@fb.com)
  - Sebastian Markbåge (sema@fb.com)

Logistics

Any other logistics required to participate in the meeting

Closer to the meeting there will be a link to video-conference in for non-local attendees. If you intend to call in to the meeting, please ensure you coordinate a direct contect *at* the meeting in case there are issues with the VC system, etc.

Registration

Doodle

Agenda items

Secretriats Note: Those items that are striked through have been finished at the meeting.

1.
    i.
    ii.
    iii.
2.
3.
4.
5.
6.
    i.
    ii.
7.
8.
9.
10.
11.
    i.
        a.
    ii.
        a.
        b.
    iii.
    iv.
    v.
12.
    i.
        a.
        b.
    ii. 30 Minute Items

a.
b.
c.
d.
e.
f.

iii.
iv.

a.
b.
c.

13.

i.
ii. Stage 0+ proposals looking to advance

a.
b.

iii.
iv.
v.
vi.
vii.
viii. Low-priority: do we anticipate using the term `mixin`? (Domenic Denicola)

14. Overflow from timeboxed discussion items (in insertion order)
    i. `Promise.resolve`'s `constructor` check: useful elsewhere, or "we have regrets"? (Jordan Harband)
    ii.
15. Closure

Agenda Topic Rules

1. Proposals looking to advance must be added to the agenda along with necessary review materials 7 days prior to the first day of the meeting.
2. Timeboxed topics may be 15, 30, 45, or 60 minutes in length.

Schedule constraints

Dates and locations of future meetings

Dates	Location
2017-01-?? to 2017-01-??	TBD

Documents discussed:

Ecma/TC39/2016/042 TC39 chairman's report to the ExeCom, September 2016

Ecma/TC39/2016/043 Final draft ECMA TR/104 2nd edition, September 2016

Ecma/TC39/2016/044 Minutes of the 54th meeting of TC39, Los Gatos, September 2016

Ecma/TC39/2016/045 Venue for the 55th meeting of TC39, Menlo Park, November 2016

Ecma/TC39/2016/046 Agenda for the 55th meeting of TC39, Menlo Park, November 2016 (Rev. 1)

Ecma/TC39/2016/047 Responses to the Ecma Contribution License Agreement (CLA), 1 December 2016

Ecma/TC39/2016/048 GitHub archives, 1 December 2016

3 Approval of minutes from September 2016 (2016/044)

Ecma/TC39/2016/044 Minutes of the 54th meeting of TC39, Los Gatos, September 2016 were approved without modification.

4 Status of "ES6 Suite" submission for fast-track to ISO/IEC JTC 1 and IT issues

On behalf of the Ecma Secretariat **Mr. Sebestyen (only part-time, on the phone)** and **Mr. Wirfs-Brock** attended the TC39 meeting. He explained:

4.1 ISO/IEC JTC 1 fast-track

ECMAScript Suite (ECMA-414) will be fast-tracked to JTC 1. ISO editors said to the first fast tracked version that TC39 have to "beef up" the normative text, just a list of normative and informative references are not enough. The 2nd Edition submitted to the September TC39 meeting took that into account.

Ecma/TC39/2016/039 1st draft Standard ECMA-414 2nd edition, September 2016 (Rev. 2) has been approved by TC39. **The 2nd Edition will be submitted for approval to the December 2016 GA. The fast-track will be submitted after the GA.**

ECMA-404 (JSON) has also been submitted to ISO for fast-track (the 2013 edition). The DIS ballot is going on and it will finish in December 2016. When there will be a similar IETF Standard (not FRC) we will issue a new ECMA-404 Edition when we will take up that and get it synchronized with ECMA-404.

4.2 ECMA-262/402 2016 editions status

Nothing new on that.

4.3 TC39 IT issues

Ecma/TC39/2016/048 GitHub archives, 1 December 2016

Since the technical work of TC39 currently is carried out on GitHub the Ecma Secretariat's policy is to make in regular periods a complete snap-shot of the TC39 GitHub activities and storages. The above is the 3rd such snap-shot. The plan is at least to make such snap-shots before each TC39 meeting.

5 ES7, ES8 and Test262 discussions

Ecma/TC39/2016/043 Final draft ECMA TR/104 2nd edition, September 2016 has been prepared and approved. This is basically an update of the short document in TR/104. The main update is that Test262 now includes all tests for all ECMASCript components and the link appoints to GitHub where the tests are stored.

ECMA TR/104 has been submitted for GA approval at the December 2016 GA.

6 Observation from the Secretariat

TC39 has been growing during the last 2 years to a size where the old working style meets its limits. Therefore we are looking at our working procedures and practice very carefully what could be improved due to the changed situation:

In the TC39 meeting we reviewed the issue of note taking and other managerial aspects.(including new chairman appointment). Reflection of those discussions it can be found in the technical notes attached. The current solution with multiple note-takers looks promising. See Technical Notes of the meeting.

Generally it has been decided to carry out such discussions on the GitHub between the meetings, in order not to take away too much time in the face-to-face meetings from the technical discussions.

7 Date and place of the next meetings

At the November 2016 meeting TC39 has decided on the following TC39 meeting schedule.

TC39 has decided not to hold a European meeting in 2017. The reason for it is that most TC39 members are from the US, and an overseas travel is always more difficult from the time and expense point of view. The suggestion is to hold European meetings only every 2nd year. It was pointed out that the free slot should rather be used for a US East Coast meeting.

- January: Salesforce San Francisco, Jan 24-26

- March: Paypal, San Jose, March 21-23

- May: Google, New York, May 23-25

- July: Microsoft, Redmond, July 25-27

- September: Bocoup, Boston, Sept 26-28

- November: AirBnB, Bay Area, Nov 28-30 (tentative)

Secretariats note: The January 2017 meeting host has been changed. It is PayPal. The issue is that for TC39 one needs a meeting room for between 35-45 people, and that is not easy to provide.

8 Any other business

None

9 Closure

Mr. Wirfs-Brock thanked the TC39 meeting participants for their hard work. TC39 thanked **Mr. Wirfs-Brock** for chairing this TC39 meeting.

Many thanks to the host, **Facebook** for the organization of the meeting and the excellent meeting facilities. Many thanks in particular to the host and Ecma International for the social event.

Annex 1

November 29 2016 Meeting Notes

Allen Wirfs-Brock (AWB), Waldemar Horwat (WH), Jordan Harband (JHD), Brian Terlson (BT), Adam Klein (AK), Thomas Wood (TW), Mark Miller (MM), Jeff Morrison (JM), Chip Morningstar (CM), Dave Herman (DH), Yehuda Katz (YK), Leo Balter (LB), Sebastian Markbåge (SM), Kent C. Dodds (KCD), Kevin Gibbons (KG), Tim Disney (TD), Peter Jensen (PJ), Juan Dopazo (JDO), Domenic Denicola (DD), Daniel Ehrenberg (DE), Shu-yu Guo (SYG), JF Bastien (JFB), Keith Miller (KM), Michael Saboff (MS), Chris Hyle (CH), Alex Russell (AR), Brendan Eich (BE), Caridy Patino (CP), Diego Ferreiro Val (DFV), James Kyle (JK), Eric Ferraiuolo (EF), Mathias Bynens (MB) Istvan Sebestyen (IS)

Introduction

Discussion with Istvan, dinner on Wednesday, etc.

WH: ECMA member site is down and email is bouncing. How do we vote on standards at next week's GA when we can't get them and don't even know where the GA is?

IS: Will send out packet tomorrow.

IS: Correct, for more than a week we are experiencing major problems with our internal IT system, we have no email and access to the internal servers. The major HP server that we are using internally collapsed within an hour last Monday and 3 Harddisk, the mainboard, another board had to be replaced. This took a week for our external IT caretaker and HP. Now starting from Thursday morning at least the Email is back, and the file servers should also be back this afternoon. Sorry for the problems, this was the biggest IT problem I had in the Ecma office.

AWB is acting chair, as John Neumann is missing.

3 Adoption of the agenda

- [Agenda](https://github.com/tc39/agendas/blob/master/2016/11.md)

Conclusion/Resolution

- Adopted.

4 Approval of the minutes from last meeting

- [Agenda](https://github.com/tc39/agendas/blob/master/2016/11.md)

AWB: Can't get the minutes due to ECMA web site being down, so this item is deferred until January.

IS: Correct, I will try to trace it and distribute it per Email. Maybe you can approve it at this meeting. Basically I did prepare the minutes as usual and published it about 3-4 weeks after the September 2016 meeting.

Conclusion/Resolution

- Adopt at next meeting.

5 Report from ECMA Secretariat

Scheduled for tomorrow in conversation with Istvan

6 Administrative matters

AWB: Proposed dates for next year's meeting on the agenda. Which organizations may want to host? We need the January meeting scheduled! Schedule for 40-50 as a safe upper bound.

- Checking on dates for next meeting:

- - Adam @ Google SF
- - Domenic @ Google NY
- - Shu @ Mozilla MV
- - Juan @ Yahoo (?)
-

__should we write this in the notes? maybe better not to publicly shame people here to host__

IS: In my opinion there is no problem for discussion this openly. Hosting of TC39 meetings are not easy. And it is always appreciated if someone can take such a burden. But also understood if someone can not take it. Whatever help from Ecma is required we try to fulfill. For Ecma it is a great help if one of the TC39 members can host the meetings. An alternative would be to go into a hotel, but that would be quite expensive since especially in Silicon Valley many Tc39 members would not stay in the hotel, but come from home. If we had the meeting e.g. in Switzerland, then everybody would need a hotel room, and we could go for a "conference package" solution, which of course for TC39 participants also not inexpensive as such meeting would include travel and hotel costs.

7 ECMA262 Status Updates

Spec Changes

- Async Functions merged
- Trailing Function Commas merged
- Better rules for NewTarget and Super in eval.

- - _Please file a bug if you run into weird things like this!_

- Export specifier is refactored to use a grammar parameter with far fewer static semantics.
- Buckets of editorial fixes as usual.
- JSON is no longer characterized as a subset of ECMAScript (due to handling of U+2028/U+2029 LINE/PARAGRAPH SEP)

- - Maybe we could fix this, by making those ES line breaks as well! It may break compatibility, though. Leave for a concrete proposal.

Tooling Changes

- New layout (got rid of dark theme :()
- Responsiveness++
- Better search. Key to efficient usage: / toggle, make use of casing.
- Pinned clauses - replaces a thumb in the spec.
- Keep in mind that CSS and JS changed. If you are hotlinking ECMA262's stylesheets your spec is broken. Don't do this.

End Game

- NOW - last opportunity for stage 4 consensus on large features.

- - global
- - SharedArray buffers
- - a couple 402 features, possibly

DD: Layering changes for the web?
BT: Need help prioritizing them

- Jan - last opportunity for stage 4 consensus on small features.
- Candidate Draft starts Feb 1, features frozen.
- RF Opt-out Period - Feb 1, 2017 to April 1, 2017. Or as early as possible

AWB: Ideally, members can review before sending to Ecma--some time before the next meeting. But it should be OK to discuss needs-consensus PRs in the January meeting in time for that opt-out period.

- June - ES2017 ratified!

AWB: Sounds like the big undetermined part is SharedArrayBuffer. If it goes to Stage 4 for ES2017, it would have to be decided at this meeting.
SG: We could get to Stage 3 at this meeting; I will present on the memory model at this meeting. Test262 tests are still needed, but we already have implementations.
BT: The integration work could be done before we get to Stage 4
DD: The test262 tests will be technically difficult to write
AWB: If we have a committment from implementers for tests by June, maybe we could go forward with it.
SG: How about the HTML integration?
DD: This is in progress, but it is also out of scope for this committee

8 ECMA402 Status Updates

DE: not much has changed since the last meeting; one small bug fix. There are a few editorial open issues, for which patches are welcome. We have two stage 3 features, plural rules and number format format-to-parts, where implementation work is ongoing.

AWB: the endgame schedule for 402 needs to be the same as for 262.

DE: yes. Note that those two stage 3 features already have test262 tests. One feature that did make it in is DateTimeFormat formatToParts. There was this change I got consensus on a while ago about resolving the constructor semantics (which changed from v1 to v2), but we're still waiting on infrastructure work to put that into the spec. I hope it can get into v4.

DE: (clarifies that the infrastructure missing is the spec infrastructure for noting something as normative-optional but putting it inline instead of in an Annex)

BT: it's OK to just use some nice HTML that makes things look good. It's not as important that it be machine-readable. You can just do something for now and we'll work on the infrastructure later; I honestly believe the inlining of Annex B into the spec is the future of 262, so we'll definitely use that in the future.

DE: OK, so if I do this some time over the next two months will that be in time?

AWB: ye

DE: PRs to get into ES2018:
  - https://github.com/tc39/ecma402/pull/84
  - https://github.com/tc39/ecma402/pull/114
Possible proposals at Stage 3 which may reach Stage 4 by January and be integrated:
  - https://github.com/tc39/ecma402/issues/30
  - https://github.com/tc39/proposal-intl-plural-rules

9 ECMA404 and ECMA414 Updates

no update

13.i Needs-consensus PRs

BT: Layering fixes, DE's item on the agenda later, (discussion of unifying Array and String lengths---let's leave it open until later)

11.i.a RegExp s/dotAll flag proposal

(Brian Terlson)

- [Proposal](https://github.com/mathiasbynens/es-regexp-dotall-flag)

BT: proposal is basically complete; really simple. It's just the additional s flag, plus the change to the flags portion of the regexp so you can query whether dotall is used. It's just a way to tell the regexp engine to tell dot to match everything instead of almost everything.

YK: even with m?

BT: yes. That just changes the behavior of $ and ^.

YK: I see. In Ruby m is basically s.

BT: Also this will make . match astral characters even for non-Unicode regexps

MB: correction: no, it won't

DE: and by "matches" you mean matches a single code unit, not the code point like the Unicode mode does

AWB: To clarify, we need to only match code units here, not code points.

WH: . has always matched all non-line-terminator characters. In non-Unicode mode it doesn't care if those happen to be surrogates or not. The mention of astral plane characters in the proposal seems to just be confusion.

(debate on whether the current spec actually does match astral code units or not)

MB: To clarify: the proposal states that with `u`, `.` already matches astral characters but not line terminators (i.e. `u` already solves one issue with `.`). By adding `s` it matches line terminators as well. `s` doesn't have

any special effect related to astral symbols.

BT: regardless, not important for stage 1. The majority use case is making dot match line breaks.

DE: V8 is fine with it.

BT: other languages have it. It's easy for Chakra to implement. Spec changes are very minimal.

(debate over whether "s" is the right name. It is, simply because every other language does it that way.)

Conclusion/Resolution

- Stage 1
- Once Brian figures out why he put the thing about surrogate pairs in there and removes or clarifies it, it will come back for stage 2

11.ii.b Promise.try

(Jordan Harband)

- [proposal](https://github.com/ljharb/proposal-promise-try)

JHD: This could start off a chain of calls to .then. To run the code on the same tick and return a Promise, so it's different from Promise.resolve().then().

DIscussion about what the semantics of the job loop are

JHD: This avoids creating an immediately invoked async function, avoids the Promise constructor, etc.

DD: This is used all over the place in userspace. It always runs synchronously. It traps any exceptions and turns them into exceptions, critically.

MM: Is this ambiguous? Will it always run synchronously, or sometimes?

JHD: Always.

MM: This is important that it's done like this.

JHD: Virtually any Promise library has this feature.

DD: Example for motivation: If you have a function that returns a Promise, it's not supposed to throw. Wrapping the function body in a Promise.try() will achieve the right result of converting the throw into a rejected Promise.

```js
function foo(relativeURL) {
  const absoluteURL = new URL(relativeURL, someBaseURL).href;

  return fetch(absoluteURL);
}

foo("http:0"); // throws!! oops

function foo(relativeURL) {
  return Promise.try(() => {
    const absoluteURL = new URL(relativeURL, someBaseURL).href;
```

```
    return fetch(absoluteURL);
  });
}

function foo(relativeURL) {
  return (async () => {
    const absoluteURL = new URL(relativeURL, someBaseURL).href;

    return fetch(absoluteURL);
  })();
}

async function foo(relativeURL) {
  const absoluteURL = new URL(relativeURL, someBaseURL).href;

  return fetch(absoluteURL);
}

function foo(relativeURL) {
  return async do { // ?!?!?!
    var absoluteURL = new URL(relativeURL, someBaseURL).href;

    fetch(absoluteURL);
  };
}
```

YK: Given that we have async functions, isn't that more ergonomic than Promise.try, which is only a half-solution?

DD: I think in all real examples, you should just use a normal async function; you don't need an immediately invoked async function.

JHD: This is addressing the case where as a user of a function, the original author of the function made the unfortunate mistake
of not ensuring that the promise-returning function will never throw.

JHD: Some use cases may want to work with Promises and not return a Promise. Promise.try may be more ergonomic than an immediately invoked async function here.

KG: Or returning an array of Promises, etc.

JHD: Async function is not always what you want to do. This is why I'm working on methods like this one, and finally.

YK: Wouldn't this be hard to teach? Not present in RSVP

AWB: Idioms are things to learn. This seems learnable.

BT: Async functions existing aren't a reason to not extend Promises.

JHD: This provides something analogous to async functions, which is synchronous execution up to the first await.

WH: I'm skeptical about adding an additional way to do it, especially since it's not shorter in terms of number of characters.

JHD: Shorter in cognitive load. I've heard from users that Promise.try and Promise.finally are the only reason for continuing to use Bluebird.

BT: I've heard that too.

MM: Cognitive overhead goes both ways. Adding features and diversity of idoms increases the congitive overhead

WH: People read others' code in addition to writing it. We already have this feature in terms of async functions. It'd increase cognitive overhead to also add an equivalent Promise.try to do the same thing.

BT: Async functions happen to do some similar things, but the intent is different. Are the only concerns that this is worth paying the complexity cost?

(What about async do?)

JHD: Let's decouple these. Promise.try and Promise.finally give us most of what we need.

YK: Why not publish libraries for just these extensions?

JHD: I have published a polyfill, but developers seem to like to get one thing that gets all the things.

MM: I overall feel this doesn't pull its weight, but it still meets the criteria for stage 1.

BT: would any promise API rise to a level of usefulness that you would be happy with in the presence of async function?

YK: Promise.any

MM: Promise.post/send/get/etc. for promise pipelining

BT: but what about things that you could implement with async functions?

MM: I'm not making a blanket statement, but async functions raise the bar very high; the case would have to be strong and it would have to add a lot of value compared to async functions.

JHD: Regardless of transpilers, there is a barrier to adopting new features. Async functions may be harder to adopt than this library feature. We don't use async functions at AirBnB because we don't want the regenerator dependency.

AR: It's good to have various features that people can adopt incrementally based on a common intelligible base, such as Promises and async functions

JHD: new Promise is unintelligible

YK: When I have a project using async functions, it's hard to find a point where I don't want to just use them all the time. And the congitive overhead is far lower.

Timebox BEEP

JHD: Stage 1?

WH: I don't object to Stage 1, but I am uncertain about utility

Stage 1.

JHD: Doesn't sound like we're at Stage 2 because of concerns about utility. What would it take for

convincing?

MM: Argument for: the proposal: Analogous to syntax, and would be expected for orthogonality, so does not increase cognitive burden but rather decreases it.

BT: Let's look at the codebases of people who use try, the people who say they're not ready for standard Promises as a result.

Conclusion/Resolution

- Stage 1
- Not ready for Stage 2, pending more evidence of motivation.

12.i.a Promise.resolve constructor check

(Jordan Harband)

JHD: `Promise.resolve` checks the constructor property and returns it straight away if so. Should new things like `Promise.prototype.finally` do this as well? Or do we regret it?

DD: Whenever trying to coerce something to a Promise--we do this in several places: Promise.resolve and async functions. Upcoming, `Promise.prototype.finally` and async iteration. What should we do? Promise.resolve checks `IsPromise && .constructor === Promise`. Async functions always wrap in a new Promise. What should we do for new features?

AWB: This is all about subclassing. If you subclass Promise, then the constructor check should make things distinct.

MM: The brand check is critical. If you say `Promise.resolve(P)` where P is a genuine Promise, that you get P back. I could entertain eliminating the constructor, but we need the brand check.

AK: What about always wrapping?

MM: A coercion which always wraps has overhead, and we want to avoid that expense.

YK: Whether we remove the constructor check seems like whether we consider subclassing an important feature.

DD: Or, subclasses could do a little more work to make their own resolve method.

AWB: If you want a Promise, you say Promise.resolve, you don't say SubPromise.resolve. We currently, with the constructor check, guarantee that we get a direct instance of Promise.

JHD: I can buy that if you do Foo.resolve, you want a Foo.

DE: Shouldn't a subclass be OK, with substitutability of subclass instances?

AWB: `finally` should use `@@species`, similar to `then`

DD: V8 has a spec compliance issue where, in async functions, we brand check and avoid creating the extra Promise. This is a spec violation, and fixing the violation has potential performance cost. I think we should do a brand check and just use it when it passes.

MM: I was surprised that await always wraps. My memory was that we decided that await would do the equivalent of `Promise.resolve()`, implying that if it is a genuine Promise, it does not wrap. Having them not agree is just creating complexity and not equivalences, and it does confuse people.

JHD: The constructor check shouldn't be needed for async functions, just the brand check

DD: But the `.then` method is always called, when we wrap the Promise and then it gets resolved by calling the `.then` later in the resolution process. We want to have a primitive version of coercion which is common.

WH: Like MM, I want all of the places where we coerce things to promises (including Promise.resolve) to have consistent semantics. A constructor check doesn't make sense in some of those, so none of them should do a constructor check. They should only do a brand check.

AWB: If we brand check, allow subclasses, and then use the primitive `then`, the whole purpose of the subclass may be to override `then`, and we wouldn't be calling that new `then`.

DD: We shouldn't contort everything and make it slower to override then.

YK: I used to support Promise subclasses, and combinators don't work well with subclassing, so it's bad practice.

DD: Promise subclassing does not work; splits the ecosystem.

BT: We should either come to consensus that we don't want Promise subclassing, and not support it, or if we don't, then we have the responsibility to support it well.

Conclusion//Resolution

- Continue discussion later (timeboxed item)

12.i.b ArrayIterator tweak for detached TypedArrays

(Daniel Ehrenberg)

DE: _shows the PR_

DE: For TypedArrays we want to be consistent, same for `length` checks.

BT: We could refactor the iterators to be different if we can get away with changing the same-valueness between TA iterator and Array iterator.

DE: Could do.

WH: If you detach the typed array upon access of the last element, the next `length` access doesn't throw even though it should.

Would it break things to have typed arrays just use the length property?

Why do we not use length for typed arrays? _who is sitting next to @msaboff?_

AWB: Not sure about historical reasons for doing it this way.

DE: These would be observable changes, although implementations don't currently implement the spec.

AK: Is anyone against adding an extra throw-if-detached line?

AWB: I am. Current spec has a problem. And, solution is to use variant A of the proposal.

BT: Can we just make [[TypedArrayLength]] 0?

DE: Possible...

Conclusion//Resolution

- Consensus on taking the PR (aka throwing on detached typed array during iteration via ValidateTypedArray)
- consensus example [found here](https://github.com/tc39/ecma262/issues/713#issuecomment-255878360) (the first option, "a")

```js
8. If a has a [[TypedArrayName]] internal slot, then
  a. Perform ? ValidateTypedArray(a).
  b. Let len be a.[[ArrayLength]].
9. Else,
  a. Let len be ? ToLength(? Get(a, "length")).
```

12.ii.b Module Namespace Objects: Various Oddities

(Adam Klein)

- [Presentation](https://docs.google.com/presentation/d/1FgSbTHoLlDsigEHBd_qufJMPBNXCwbIw2kQsR-B0HPA/edit?usp=sharing)

AK: Module namespace exotic objects are very exotic. Mostly immutable from the outside, null prototype, special @@iterator

Slide: module @@iterator

AK: The @@iterator is weird. Proposal possibilities:
  - Remove @@iterator; you can use Object.keys() for this
  - Make @@iterator more like other iterators and freeze the function

BT: You can for-of a Map...

DD: This is incongruous, since other object literals don't have an @@iterator like this

DH: @@iterator is for iterating over collections, and modules are not conceptualized as a collection

BT: I see that argument

AWB: Map is a completely different thing; this is a namespace object.

BT: Point is, if it makes sense to iterate most of the time using keys, values, or entries, add an @@iterator to do that. Otherwise, don't. In this case keys doesn't make much sense.

DH: I can't remember the original motivation. Maybe we can find it in the bug database.

AWB: Say you've imported a namespace object. You've imported *. So you want to look at the names of the object, and you iterate over it.

AK: I would say this is not a very common operation

AWB: Arguably, this is a meta-level operation, a reflective operation, so explicit Object.keys makes sense

DH: This is in the power tools section, so we shouldn't be doing crazy extra work for the ergonomics

YK: but import * is not a power tool

DH: But looking at is as this fancy object is

JFB: We are talking about exposing WASM modules as ES module exotic objects. For dynamic linking, we may want to process all of it. You can do it with either feature, though.

AWB: The only other argument is that the @@iterator doesn't need to create this intermediate array, so it may be more directly implementable.

AK: Creating the JS functions for the @@iterator has its own overhead.

BT: I am worried that we don't have a good understanding of the motivation for creating this feature

AWB: For iterating over the objects without creating the intermediate array. But anyway, let's remove it

Slide: @@toStringTag

AK: @@toStringTag: [[Configurable]] is true, but [[Set]] and [[DefineOwnProperty]] fail. This is inconsistent.

AWB: But this is a standardized, built-in method, and we generally allow monkey-patching of those.

JM: Isn't this for polyfilling the language? But in this case, the module is something the user created not the language.

YK: And there's a null prototype, so we can't even do a real polyfill

AK: But the module doesn't get to choose.

AWB: So I think it should be configurable and mutable.

JHD: Could be useful for polyfilling if the thing that produced the module namespace object were wrapped and amended it

AWB: You don't want this to be a communications channel. If Mark were here, he'd agree. However, there's no essential invariant that configurable means it really is configurable. Anyway, it's OK to make it non-configurable--better internal consistency.

Slide: SetPrototypeOf

AK: SetPrototypeOf to null should return true

BT: This is just following the precedent of defineProperty on non-writable, non-configurable, same-valued sets.

(agreement)

Conclusion/Resolution

- Consensus on removing @@iterator from module namespace exotic objects
- For local consistency, make @@toStringTag non-configurable
- SetPrototypeOf should return true for null

12.ii.d Proposal to reform the spec to include default export in export * from 'module'

(Caridy Patiño)

- (Presentation:)

CP: `export * from 'module'` does not export default export - seems confusing

CP: You may want to reexport, and that's broken. A common pattern is `module.exports = require('...');` (The example in the presentation has some issues: `default` is a keyword and can't be imported without getting renamed)

DH: I strongly object to this proposal

CP: The main proposal is remove the special restriction on default, and this creates a named binding when imported, rather than being filtered out?

DH: What happens when there are multiple export defaults?

AK: Just like currently, if there are conflicts, then they would lead to an error when importing through the conflict

YK: Seems bad that this is an error in linking rather than statically

AK: That's a different discussion

DH: I think the idea is to just treat it like everything else. But this isn't the programmer intuition. The programmer intuition for default export is that this is the default anonymous export. This happens to be implemented through a special exported name called default, but nevertheless, the idea is that this is the anonymous thing.

JM: I see that a lot of people get confused about default exports, but in my experience, I can clear it up by explaining that default is just another named export.

DH: Maybe understanding the mechanism helps advanced users, but the programmer intuition is more basic.

JM: The concept is that default is the main export, among a number of other subordinate named exports.

DH: That's understanding the mechanism, but when we design policies, they have to be designed around the programmer purpose. What are the intuitions about what it means. What I am reexporting from another module, this is something that I decide. When I'm composing utilities, I decide what the default is, and don't want to just inherit it.

YK: The intuition part is true, but the default of one will be different from the default of another.

JM: What about the scenario like lodash where you want to combine a bunch of libraries . If you're wrapping another module, and adding a couple things, where the default is forwarded, is more plausible. The language already exposes `import { default as alias }` -- the conceptual module is reasonable, but it's at a high level. There's a difference between suggesting (with nice syntax) and imposing constraints (the behavior that the proposal is reverting) it's different, making an inconsistency with the rest of the scenarios.

JHD: Why does import * bring in "default"?

YK: Because it has no cost

DH: Because it's not putting them in a scope, it's just putting them in an object

JHD: If you're arguing that the "default as named export" is supposed to be "hidden" from the conceptual model of a developer, then it seems like exposing it in `import *` is weird, when you can't `export *` and get the same behavior. I don't think that discrepancy is worth changing what `export *` does, but should we change `import *`?

YK, DH: Too late

DH: Two use cases for export *: Composing multiple modules, and decorating a module. Filtering default is necessary for the composition use case.

AK: That's not true, you can fix the ambiguity by explicitly exporting the default that you want.

JM: Anyway, you have that problem when combining modules that have the same name, resolvable in the same way

DH: There's an asymmetry--explicit name clashes are more explicit, but the concept of a default is more that it's per-module and not the category of things that would be overlapping.

JM: But, in the wrapping use case, forwarding with export * has the same thought process, but it's a scenario where you'd really prefer default to be included. Default should be handled the same way for exporting default twice as for exporting names multiple times.

AWB: Default has to always be explicit

JM: You combine named modules because you know they are bags of named exports. If you do that with wrapping, it's with the understanding that it has a default, and exporting that. I agree that default is special, and thinking of it as named is not important here.

KG: This would be a change to existing semantics, potentially breaking. Regardless of whether it's a good idea, could we get away wtih it?

DE: In the transpiler universe?

KG: Yes

DH: It would cause something which was an error to stop being an error.

AK: The only thing the proposal does is remove the special handling for export *

YK: What about the ambiguity?

CP: That's a property fo how modules work already. This is not a breaking change, and details are in the README: It's causing additional things to be exported, and you only encounter the errors if you actually use this new export which is ambiguous. From static semantics, you only take programs which were illegal and making them legal. Explicit defaults override anyway, so their semantics don't change.

DH: Only if you actually iterate over the module namespace objects would the behavior of existing programs change.

AK: Sounds like there's not consensus to move forward with this change. It was specifically designed to work like this

DH, YK: Right

CP: Do we want to be able to reexport more flexibly, with the second part of the proposal?

(Sounds like not)

AK: What's the really bad thing that happens if someone exports default, and then imports default

DH: The bad thing is when you don't want default at all.

YK: If you export it by accident, people may depend on it, and you can't remove it later.

Conclusion/Resolution

- No consensus on this change; the current state is as designed.

12.iv.a import()

(Domenic Denicola)

DD: AWB's issue: import as a unary operator rather than a pseudo-function as currently proposed. https://github.com/tc39/proposal-dynamic-import/issues/23

KG: Would it be an actual function?

DD: No, needs to be a function-like form.
Pros for function-like:
- Future extensibility (second argument)
- Ambiguity with 'import x' form in modules. You could use this from a script, and you might expect that you can write 'import x' in a script. But that would not be the same--it would create a Promise and then throw it away

AWB: You could require it to be parenthesized when it's in statement position, with the same logic that prohibits a function expression there

DD: But, at that point, if you need parens, you might as well have them after the import, rather than before

DH: If there's a lookahead restriction on expressions...

DD, WH: There's no lookahead involved here; this is just what grammars do, and by the time you get to the reduce part of the grammar you know which form you have

- Parens are usually required anyway, e.g., if you'll then the result
- If we were to do `await import x`, then it would be impossible because we've already used the unary operator
- Another possibility (though AWB expressed dislike) is local import being hoisted, which this would disallow

AWB: Pros for unary operator, with a low precedence like yield:
- We are looking at something which is not a call, it's really based on conceptual state

BT: This is a little more like super(), which uses contextual information.

DH: But you could conceptually think of this as a function call. We have both forms, function-like and operator-ilke. Also, eval.

BT: If eval were a keyword, we'd maybe do it more as this pseudo-function thing

AWB: Actually I think we should do it as a unary operator in this case

DH: People might not actually think that hard about not needing parens for a prefix operator. So there might not be a strong conceptual distinction between unary operators and pseudo-functions

JHD: Douglas Crockford has in the past advocated using typeof with parens

BT: I claim that the mental model of import as a special function is simpler and more intelligible than

operators.

WH: The only place the function mental model breaks down is if users try to assign without calling `x = import;` (but fails early like super)

AWB: I am concerned about things like the spread operator not working.

WH: But you can't spread into the parenthesized expression in an if statement either.

YK: Initially I was sympathetic to operator, but now I am thinking more like, the analogy to require is nice. So I support a function form. require is similarly magical about how it gets your context.

BT: Could we make import really a function at all?
(Seems hard, and no one is really advocating that)

BE: Back in the day, I considered making eval a special form. Regrets!

DH: The parens distinguish this form from the import statement, which is important. It disambiguates much better.

```js
import { toString };  // With AWB's proposal, would import "foo"!
function toString() {
    return "foo";
}

(import "foo").then(...);  // JHD: Looks like something very weird
import("foo").then(...);  // JHD: Looks like normal JS
  // BE: In AWB's proposal actually imports the result of evaluating the expression `("foo").then(...)`.
```

DD: We could allow spread and have it throw an error if it is dynamically not one argument

(Discussion about whether it should be OK to throw on the wrong number of arguments)

AK: The DOM often throws on the wrong number of arguments.

DD: We should either do either fully static or fully dynamic argument number checks

AWB: For extensibility, what if we define a second argument which is an options bag

DH: Seems like predicting the future in a possibly fallable way

DE: Concretely, a hash for SRI or a nonce for CORS may take this second argument position

DH: In the past, implementations which gave an interpretation for the second argument for eval, and this poisoned the well so we couldn't add our own meaning for a second argument.

WH: If we don't support spread now, we can always extend the syntax compatibly in the future.

AWB: My concern is teaching and learnability. Anyway, the precedence issue seems like the fatal thing that makes DD's proposal the winner

JHD: People would only run into this if an error occurs and it's hard to explain

DH: However, the browser error messages may not be great. I don't want to have the mixed static/dynamic enforcement.

KCD: I'm sure people will teach that import is a function, FWIW

KG: But that ship has sailed; it will not be possible to treat it as a value.

JHD: People will accept this limitation

AWB: It'll be unfortunate if people teaching have to talk about how things don't all work the same, and we really muddy the issues about what function values are.

JHD: There's already a list of things

DD: Membrane penetration issue https://github.com/tc39/proposal-dynamic-import/issues/26 : MM said he would not block Stage 3, as he is unable to find an attack

AWB: If you have a membrane, importing a module dynamically could introduce a side-channel because someone outside the membrane and inside importing the same module, any mutable state exposed (e.g., exported function that you can attach properties to) can be used to move information across the membrane

DD: Mark already has a pre-compiler. And this isn't any worse than manipulating the DOM with [script type=module]. Mark's system depends on restricting eval and parsing and transforming the code there. So it seems like there's no reason to hold things up.

DD: https://github.com/tc39/proposal-dynamic-import/issues/27 "revised proposal still violates run-to-completion execution semantics". The spec mechanics give host environments the ability to violate run-to-completion semantics by calling back into ES when it completes.

AWB: Abstract operations aren't there to say, hosts can call them at any time and destroy all the invariants.

DD: We don't have spec mechanics for any of that right now.

AWB: Hosts don't just use the spec as an API, they extend it

BE: We don't want to monkey-patch the spec

DD: The Web host embedding environment calls Object.prototype.toString() all over the place, and we don't have anything explicitly in ES prohibiting it from being called in the wrong place. This seems like the same problem.

AWB: "Either immediately or at some future time"

DD: This is required for Node's ability to integrate require() with ES modules well. That's what the "immediately or" part is for; we need it for Node integration.

DE: Could we separate advancing this proposal from getting the job queue mechanism in order? We seem to be in agreement on all the substantial issues here, and the job mechanism was already present.

DD: I'm not comfortable with advancing this with broken semantics

AWB: Let's start by using the mechanisms that are there, and use them, or come up with new mechanisms and use those, rather than this vague prose?

DE: Let's not be so insistent on one form or the other. We have other forms which are unimplementable, like the lack of describing resource restrictions.

BE: If we ignore feedback like Domenic's and implementers ignore the spec text, we risk the actual spec being a dead letter.

YK: We just need to say that the jobs happen separately in order, with run-to-completion.

BT: I want to avoid spending a lot of time writing spec text for the job queue mechanics, when we could instead use general prose which preserves the run-to-completion semantics. This would be in accordance with Domenic's PR https://github.com/tc39/ecma262/pull/735

DH: It may be very important to Node to do this synchronously (as Domenic was attempting to do, but which AWB's semantics object to)

DE: Could we decouple this into another proposal?

YK: Interoperability concern: top-level evaluation in modules imported in Node may see the side effects synchronously.

DH: Maybe Node could delay things to a new turn, but we need to talk to them. Also, it violates expectations to have something that returns a Promise execute everything immediately.

YK: Node will still have require; people can still use it if they want synchronous.

AK: Is the major problem here that we don't have Node folks here?

DH: Yes

DD: Seems like it should be possible to move forward to remove the synchronous possibility, and then have the discussion with Node and work on patching the whole system, as part of Caridy's proposal which patches the module system all over.

JHD: I don't understand why this is a problem for Node. You could begin the require call later, when called from Node, so it should be possible to be always asynchronous. Stage 3 seems early to constrain these things; seems like this is about implementer feedback.

AWB: My understanding is that Stage 3 means we have really decided on semantics. Among ourselves, we can decide that it should be asynchronous.

JHD: You can be synchronous or asynchronous without violating run to completion. I would predict that Node implementation feedback won't require synchronous, and Stage 3 is where we get their feedback.

DH: I am not OK with synchronous semantics without them being in the room to explore the options.

AR: I don't understand the discomfort with getting implementation feedback. Can't we have the conversation with them?

JHD: Bradley Farias can call in tomorrow.

Conclusion/Resolution

- Stick with import()'s function-like form, with no spread, no commas, just an assignment expression; can't be extended with extra arguments (Chapter 16) https://github.com/tc39/proposal-dynamic-import/issues/23#issuecomment-263742551
- Open issues which block Stage 3:

- - Interoperability with Node if modules execute synchronously
- - Spec mechanics for ensuring that we are preserving run-to-completion if we allow synchronous execution in Node, asynchronous execution in the browser.

- To resume with a call from Bradley Farias.

November 30 2016 Meeting Notes

Allen Wirfs-Brock (AWB), Waldemar Horwat (WH), Jordan Harband (JHD), Thomas Wood (TW), Brian Terlson (BT), Michael Ficarra (MF), Adam Klein (AK), Jeff Morrison (JM), Chip Morningstar (CM), Dave Herman (DH), Yehuda Katz (YK), Leo Balter (LB), Sebastian Markbåge (SM), Kent C. Dodds (KCD), Kevin Gibbons (KG), Tim Disney (TD), Peter Jensen (PJ), Juan Dopazo (JDO), Domenic Denicola (DD), Daniel Ehrenberg (DE), Shu-yu Guo (SYG), JF Bastien (JFB), Keith Miller (KM), Michael Saboff (MS), Chris Hyle (CH), Alex Russell (AR), Brendan Eich (BE), Caridy Patino (CP), Diego Ferreiro Val (DFV), James Kyle (JK), Eric Ferraiuolo (EF), Mathias Bynens (MB), Istvan Sebestyen (IS), Mark Miller (MM), Cristian Mattarei (CMI), Brad Nelson (BNN), Jafar Husain (JH)

5 Report from the Ecma Secretariat

(Istvan Sebestyen)

IS: ...

(Istvan was speaking remotely, over a broken audio connection, so I have no idea what he was saying. Please fill in)
(Link to slides please)

IS: The slides shown have been distributed via Github and the Ecma Tc39 reflector 1 hour before the meeting. He said that 1 TC39 Standard and 1 TC39 TR will be approved (hopefully) by the Ecma General Assembly next week. He said that the JSON fast-track in JTC1 DIS ballot will end soon in December. WE have received comments from the Japanese NB, and Allen Wirfs-Brock looked at them and we took up contacts with the JApanse NB in order to eliminate the problems. According to Allen not really a big problem. Then he talked about the necessity to have a new TC39 Chair / Vice Chair selected. Until a solution is found, for a few meeting Allen Wirfs-Brock on behalf of the Secretariat will jump in. Unfortuantely, he is only willing to that that short-term.

(Discussion about new chair person)

(In response to question from DE)
AWB: Chairperson should probably not have major interest in the specification and have motivation to block any particular proposal, but be technically skilled to direct the meeting. The vice-chairperson would have less of a problem with such conflict of interest.

IS: This is the ideal situation of course. In practice it is often the case that a chair person has to take position on behalf of his company on certain topics. What is important that also makes it sure when he speaks as Chair and when on behalf of his organization. Often, if he presents something on behalf of his organization he may step down as Chair and ask someone else to sit in as long as his proposal is discussed.

AWB: The dilemma is, why would someone with no interest in the specification but technically savvy with it want to come to the meetings to chair?

WH: That dilemma makes the requirements too difficult to fulfill.

BT: Don't want to do the prior ad-hoc funding arrangement. Instead, fund chair person via ECMA dues or a TC39 surcharge?

IS: ECMA's job is to provide the technical secretary, not chair person. Willing to discuss other alternatives. Ecma TC39 is too important for Ecma not to try to find a workable solution. We have to be flexible.

AWB: How do we manage a standards process with such large meetings? How do we compare to other large language committees, a new chairperson may wish to look and bring forward proposals.

AWB: No desire to continue long-term, even if funding issues were resolved. I've been doing this for 10 years, an expanded chairperson role would consume too much time. I might welcome a vice-chair position to help. But I want to move on to other things.

YK: Thanks to DE for improvements to meeting process, has assisted chair.

YK: FInding a good chair is not an emergency, in case we end up with a bad chair.

AWB: We need a collective sense of urgency here.
IS: What is also important that TC39 members think about what kind of chair / vice chair (that position is also open...) they would like to see for the next period. In the past 6 years we had a chair who was not an ECMAScript expert, so really neutral, but with the current new requirements maybe a chair with some strategic guidance would be good. This is really something TC39 should think about and decide.

6.i Meeting scheduling

DFV: Salesforce willing to host Jan, but NDA issue needs to be sorted.

IS: This is the first time in my Ecma history that I am faced with an NDA issue. here Generally our meetings are Ecma meetings (hosted by somebody, but not somebody's meeting) that needs no NDA.
Usually if such issues emerge, meeting participants can be requested e.g. not to walk around in the meeting bulding, but just to stay in the meeting room or the next lavatory.
He offered to discuss the matter with Salesforce when he is back from the GA in December.

JM: Checklist of organisational points for meetings should be drafted.

IS: That can be done.

AWB: If salesforce are unable to resolve NDA issue are people still be willing to attend?

MM: Previously, I was threatened with physical eviction from building for not signing NDA, caved in and signed.

JH: Lawyers claim NDAs hard to enforce, recommend never signing.

AWB: Not a meeting with the host company, independent.

JM: NDA to protect host company from secret contents of building leaking, not meeting.

BT: Interpretation of NDAs often badly worded that also would cover meeting contents.

MF: Would need advance notice to run NDA past my company's lawyers.

YK: I often sign them, (to MM) do you often have this objection to NDAs?

MM: If visiting the company to meet directly with the company, it is because they're offering something, quid pro quo tradeoff.
Taking on an intellectual property obligation as a part of an open standards committee meeting, it's not appropriate. If I wander around and see something secret, if the NDA was sufficianely narrowly prescribed to prevent me leaking something from I've seen wandering around, I'd be happy with that. NDA needs to be narrow for the company not to reveal secrets to me as well, to not burden me with a secret I did not want to hold.

WH: NDA at Yahoo prevented laptop or cellphone being brought into the room.

MM: ECMA is trying to be very careful about IP and patents. I'm not personally so careful outside of committee activities. It is important that we stay clear of these conflicts for ECMA in our role at meetings.

BT: No NDA was signed at Redmond.

AWB: Microsoft have sanitary spaces to hold non-NDA meetings.

JH: I could be fired for not consulting lawyers before signing an NDA on behalf of my company.

YK: We should ask for NDA allowance before meeting, if sufficient time.

CP: January would be hard, but could try for Salesforce meeting.

SYG: Mountain View Mozilla space has max of 32-34 people. There is also a v large open room, but next to kitchen, people come and go, not suitable. SF Mozilla room would take less than 30 people, previously had 40 people sign up there, not sure how many showed up. SF room was poor shape.

AK: Google SF still awaiting response for Jan.

BT: Mozilla is a fallback, should wait for Google SF to respond.

AWB: 24 hours at least to wait for response. Could do conditional - tentative Salesforce, but fallback to another. Would have to be resolved quickly. Revisit this again tomorrow.

AWB: Has anybody reviewed strawman meeting dates for rest of year? Any conflicts?

DD: Does not conflict with JSConf EU. Can't guarantee no conflict with confs in Sept/Nov.

YK: Conflict in Mar 28-29 with Emberconf.

AWB: Please bring up issues asap.

AWB: Would Mar 21-23 work?

YK: ok

March meeting changed to 21-23

DD: Jewish holidays surround Sep meeting.

AWB: Resume discussion tomorrow. Ideally cities, at least hosts to be determined.

Conclusion/Resolution

- March 21, 22, 23

4 Approval minutes from last meeting

IS has uploaded to Reflector.

WH: Not ready to approve yet. Will read overnight.

13.ii.a SharedArrayBuffer

(Shu-Yu Guo)

- [Proposal](https://tc39.github.io/ecmascript_sharedmem/shmem.html)
- [Slides](http://tc39.github.io/ecmascript_sharedmem/presentation-nov-2016.pdf)

SYG: Goal: Advance SAB to Stage 3. We have already agreed on agents and the API, and are at Stage "2.95". Since then, we revised the memory model to fix a bug from WH.

SYG: The memory model should answer questions such as, is the following optimization valid? Probably not; it would be weird if this ever printed 0.

Should we allow this optimisation?

```js
let x = U[0];
if (x)
  print(x);
```

To:

```js
if (U[0])
  print(U[0]);
```

YK: How should this code be interpreted?

(Committee: JS, or pseudocode, or imagine it's typed; whatever)

SYG: What about

```js
while (U8[0] == 42);

===>

let c = U8[0] == 42;
while (c);
```

AWB: No

WH: The answer is yes.

AWB: (surprised) Why?

WH: The consensus of programming languages is that this optimization is allowed. If you were to set the answer to "no", you'd lose too many opportunities for simple optimizations needed to get good performance.

SYG: OTOH, the same thing with atomics would be disallowed.

SYG: Independent reads/independent writes in parallel threads *don't* have to be ordered in a consistent way as viewed by other threads. It is allowed to be acausal, and this happens on ARM and Power. Not a simple interleaving of instructions!

WH: Memory model defines which paradoxes are allowed for two reasons:
- Model what the hardware is doing
- Provide a model for what compiler optimizations are valid

SYG: Complicated math, but helps implementations decide what to do

SYG: Memory model could have a lot of undefined behavior, but this tries to be fully defined, to provide interoperability and security. Also need to work with WebAssembly with a common model.

MS: Will WebAssembly use this?

JFB: Yes

SYG: Goal: Strong enough for programmers to reason

- - sequential consistency for data race free programs

SYG: Weak enough for hardware and compiler reality.

DH: Sequential consistency is already hard to use, and you need to build abstractions to make it more usable

(discussion about sequential consistency and data races)

WH: You can trivially turn any program into a sequentially consistent, data-race-free program by turning all reads and writes into atomic reads and writes. That does not make it correct!

SYG: Implementer intuition: non-atomics compiled to bare loads and stores, atomics compiled to atomic instructions or fences. For optimizations, atomics carved in stone, reads and writes stable (no rematerialization of reads, or observable changes to writes), don't remove writes. Semantically, you can reduce the size of the semantic space, but not enlarge it.

SYG: No rematerialization. Can't turn one read into multiple reads [see first example above].

WH: Note that the converse is often allowed. It's often fine to turn multiple reads done by a thread into one read.

SYG: Notions of atomicity: access atomicity (can't be interrupted--easy part), copy atomicity (when accesses are visible to other cores--ordering/timing is the hard part)

DH: Multicore processors are just distributed databases and as bad as webapps.

SYG: Memory model describes all this with a bunch of math, written in the spec, very different from the rest of the spec language. Our atomics are the same as C++ memory_consistent_atomics, or LLVM SequentiallyConsistent. Non-atomics are between non-atomics and C++ memory_order_relaxed, or between LLVM non-atomics and Unordered--a bit more strict than the total non-atomics.

WH: memory_order_relaxed is a bit stronger

SYG: We are in new territory, in terms of the strength of guarantees for non-atomics. See spec for details, or more description in the slides.

WH: The rest of the ECMAScript standard is described prescriptively. But the memory model is a filter on all possible executions and ways it could happen. In the spec mechanism, you run the program, you return all possible bit values for every read, and the memory model later retrospectively says which executions were valid. This is different from sort, where you have choices but don't mess with causality.

DH: The rest of the spec is operational semantics, and the memory model is axiomatic semantics.

AWB: Will implementers be able to read this and follow it?

SYG: The specification is a bit inscrutible. Because it's a filter on all possible executions, you can't read it and say, "I know what I'll do now".

WH: The spec is commented reasonably well with informative notes.

WH: Give me any simple example of a program transformation. By reading and following the memory model in the spec, I can give a definitive answer about whether the optimization you want to do is allowed or not.

WH: An implementation is not required to exhibit all possible executions; that's not practical or necessary.

WH: On the other hand, some concurrency is mandated by the memory model. It is not valid for an implementation to simply sequentialize all agents, running each agent one at a time until it blocks and then switching to a different agent, as that would cause deadlocks. Last week we added liveness guarantees; without those things, the atomic busy wait example earlier in the presentation wouldn't work.

SYG: Stage 3?

BT: How many implementations do we have?

BN: Four! Chrome, Firefox, WebKit and Edge.

DH: Memory models are hard, so it's OK for us to refine this in the future. But it's great that we have a starting point with a model, so we have a place to talk about which optimizations are legal. But no way to know it's perfect.

SYG: There will be bugs, and academics will write papers on our bugs for the next 10 years

DD: We will be able to write test262 tests

WH: In prior iterations of the memory model over the last several months, I had discovered fatal counterexamples exhibiting both too much synchronization and too little. Those had required several complete rewrites of the spec to fix. I am happy with the model now.

AWB: Stage 4?

SYG: We don't have test262 tests, that's what's missing.

JHD: How can I access an implementation?

SYG: It's on in nightly in Firefox

AK: It's in about:flags in Chrome.

AWB: If it's likely to go in soon, it'd be good for the editor to get started on the integration. Should we make it conditionally Stage 4?

AK: The editor can start on the integration work before the it really reaches Stage 4; that seems unnecessary.

AWB: Who is committing to doing test262 tests?

SYG: I'll do them.

LH: As a starting point, the repository has tests, just not integrated yet.

DE: We will want the harness work done on implementations to get it to work

BT: Do command-line runners have a way to open agents? Chakra does. This will be useful for me to make an eshost mechanism.

AK: In d8, you can use new Worker()

LH: Firefox has a shell function for creating agents, and a shell function for sharing memory between agents.

Conclusion/Resolution

- Stage 3
- Intended to reach Stage 4 at the next meeting pending test262 tests; integration work can begin now.

10 Test262 update

(Leo Balter)

LB: Bocoup has a new contract with Google, so we will work on maintanence of test262, but this is a smaller contract, just a single day per week. We are looking for other partnerships to provide skilled work for the maintanence of test262 to fund its development.

KG: I made a web-based test262 runner, which we hadn't had for a while. It loads from the repository, so it should only need updates for harness changes. https://bakkot.github.io/test262-web-runner/

12.ii.a. Proposal to reform the spec to solve Node.js ecosystem compatibility breaks w/ ES Modules

(Caridy Patino)

-
[Slides](https://docs.google.com/presentation/d/1EYOysPhgjXtgmuNoZ_wUCMEIZ8GKLxJmCLeF0EvUXkc/edit#slide=id.p)
- [Proposal](https://github.com/caridy/proposal-dynamic-modules)

CP: Dynamic module reform: revisiting the ordering of module loading to incorporate Node.

CP: For review, ModuleDeclarationInstantiation first parses the modules, then, links the bindings. For Node, we introduce a new "pending" state for the bindings, based on the discussion from the last meeting.

AWB: For CommonJS modules, the problem is, you have to evaluate the body to find the exported names.

CP: AWB is saying, There is nothing in the spec that precludes the evaluation of the NCJS module

MM: What if we look at the imported names? It would be good to achieve certain invariants on the resulting combined system.

AWB: Node is another environment, so the body serves a different role, and we can do different things.

MM: worth examining cases where postponed evaluation is possible

AK: Don't want evaluation order to change when converting (eg. NCJS to ESM)

DH: (clarifying previous meeting consensus)

YK: node module objects are intrinsically "live", but the node representatives said no interop issue

DH: But do you snapshot the current bindings? Or read through?

AWB: Node export is values

DH: two decisions:

1. does lookup read from own slot
2. does lookup read through continuously?

We don't want access to trigger side effects

AWB: what happens with commonjs modules? Evaluated only once?

JHD: If they succeed, then evaluated only once.

CP:

- Does access `b` go through process of determining is NCJS is still in TDZ?

AWB: How does that work?

CP: walk through https://rawgit.com/caridy/proposal-dynamic-modules/master/index.html#sec-module-namespace-exotic-objects-get-p-receiver

AK: Step 7 is meaningless?

DH: Rather than spec details, let's step back to the big picture. Let's think about cycles. Could you tell me the story there?

CP: Cycles work fine with all ESM or all CJS, but the complexity is when it's mixed: Depending which you import first, you may get an error, without this proposal. Adding the "pending" state allows us to delay checking temporarily.

DH: The key design question is how we integrate them. For dynamic modules, validation of inputs being performed as late as possible--when evaluating a reference--would provide maximal interoperability, and then you can get earlier errors once you start using ESM.

AWB: A design goal of ESM was that, statically, bindings could be totally resolvable at link time (modulo TDZ, etc).

DH: That's the case in my proposal for `ESM<->ESM`, and for CJS, maybe a version of that could be true as well, with an extra guard.

AWB: You're suggesting that we add a different type of binding, a dynamically resolved binding. At link time, you could determine, for each import, whether it's statically resolved as an alias, or a dynamic CJS-style binding.

AK: Thinking about my implementation of modules, the spec text here is a bit incoherent--the data structures don't match up

AWB: I think that has to do with how we actually have to reflect a different kind of binding. (Question about

circularity.)

DH: Previously, we had uninitialized variables, now we also have a new class of error for things that are unbound.

AWB: Probably insignificant for users.

DH: You find out later, when running your test suite, as today in CommonJS, whereas ESM finds out earlier.

YK: What would be the behavior when there is an error?

DH: ReferenceError when you actually use the variable

YK: Now it looks like a funny feature, you can import whatever names you want, and you don't get an exception unless you use it. This may be an unintentional feature.

CP: If you change your program from CJS to ESM, then you get a static error

AK: And this is a problem, making it more difficult to upgrade

CP: There are three options for the validation:
1. Don't validate anything, just check when accessing the dynamic binding, TDZ, ReferenceError
2. Whenever an ES module is about to be evaluated, if it depends on a CJS module, then go and check that the imports resolve correctly from the exported object
3. Whenever evaluating a dynamic module, check the things that it exports *to* and validate that there are no missing things there

DH: Option 2 has the issue with mixed circularity that you may cause something to not work, but 1 and 3 make that work. The downside of 1 is that it is too permissive, which could make evolution difficult.

AK: For some examples, 3 works out just fine and does the checking, but it still has the "fully dynamic" checking semantics if the ESM executes before the CJS.

CP: What would be the error that occurs then?

AK: ReferenceError!

CP: Which importer would be blamed by the implementation?

AK: Oh yeah, it would be a fatal not executing thing.

DH: This is a quality of implementation issue.

DE: To clarify, what's the case where you'd get the fully dynamic checking? Were we considering hoisting executing the CJS modules earlier? If it's all ESM, you never see this issue where, e.g., an exported function isn't filled in yet, as it's hoisted.

AK: A design goal here is to preserve the ordering of when the modules execute. So it might just not be filled in yet. Inherently, there are going to be some issues with circularity, but Option 3 is a good option for many of them.

DH: It's great how much progress we are making on module interop

AK: It's good that we have laid out these options, but we haven't worked out all the details and motivation for Option 3 yet. For background, it must've been more than implementation concerns that drove the current static semantics, right?

DH: Principle of least power, and ability for programmers to statically reason about their code bases, e.g., with code manipulation tools. Though soundness is not necessary for tools; seems like it's fine if we introduce this dynamic stuff which is unsound.

YK: Just because we have static linking doesn't mean we can't also have dynamic linking. ESM has already been a big benefit to tools, e.g., TS, Rollup, linters.

CP: Should my proposal be considered a PR or a staged proposal?

AWB: The spec text here is not ready yet. It should be a staged proposal.

AK: I think this is ready for Stage 1. This is an important problem, but it's a big enough change that it makes sense to think of as a proposal. It changes the model of modules significantly--goes from sound to unsound. We also need a tighter loop with Node on evaluating these options--Node currently doesn't snapshot keys, so it's even more dynamic than 1.

DH: Node has already said that snapshotting is OK, so this is a big step forwards.

DE: Are we still considering all three options?

AK: Yes

Conclusion/Resolution

- Stage 1, champion Caridy Patino

12.iv.b LineTerminator normalization: implementation feedback for F.p.toString

(Michael Ficarra)
https://github.com/tc39/Function-prototype-toString-revision/issues/19

MF: Function.prototype.toString is supposed to maintain the source text, in the new version. Based on feedback from AWB, the current proposal normalizes all line terminators, consistent with template string literals. However, implementation feedback from Mozilla raised questions as to the cost of doing the normalization. Seems serious, should we reconsider normalization? Normalization is on 2.b-c of https://tc39.github.io/Function-prototype-toString-revision/#sec-function.prototype.tostring .

AWB: Template string literals normalize line endings so as to avoid changing the behavior of a program when run on different platforms.

WH: The same argument applies to Function.prototype.toString.

DE: At this point, the type of line break doesn't really differ all that much across platforms. It's analogous to how many spaces you include in your program--something that you do in an editor. It is just not so significant. Because of this, we should look to second-tier concerns, such as Mozilla's performance argument.

YK: Further, just returning [[SourceText]] would be the cleanest version.

MM: We shouldn't expose the platform, but editors often use native line endings. We should be consistent with template literals.

YK: There is a difference between the scenario for template literals and Function.prototype.toString(). For templates, you may want to split on \n or something, but toString is more about getting out the text from the file. Maybe you want to search for the text that you found within the file. You wouldn't want it to be messed with.

MM: The fact that entrenched tools like git will still transparently convert line endings on the same source on balance argues for normalization. Such conversions of source code should not cause a change in that code's runtime behavior. [gives example]

YK: I think returning the original source text is the right thing to do here.

MM: Raw template text data is fairly close to capturing the original source.

AK: Treating functions as a quoting mechanism is odd

MM: But people may still do it

YK: Our current notion is raw text

AWB: Even raw template text data normalizes line breaks.

AK: When would it cause a problem to not normalize?

MM: Say Git or somebody converts line endings, and then you execute the code on both platforms, and do Function.prototype.toString() and send it over the network. You would be able to observe it!

AK: But presumably you'd eval it at that point.

YK: You'd only observe that if you split the string on \n, which doesn't make any sense

MM: What does C++ do?

WH: Just checked the C++14 spec. It says that line breaks inside raw string literals evaluate to the same thing that "\n" would evaluate to in a regular string literal.

JFB: You don't want to follow C++ here because it allows identifiers to start with a non-breaking space; it's poorly thought-out.

WH: C++'s identifier conventions have nothing to do with this discussion. Plenty of unrelated warts exist in ECMAScript as well.

AWB: You can observe that the lengths are different.

AK: What about performance, and how this compares with existing implementations?

MM: Does this have significant cost?

AK: Certainly significant implementation burden. There are a lot of things we do with the source text; storing two copies would be worse than storing one. V8 stores the text that came off the wire.

AWB: Do you return a string which literally points at the input?

AK: Yes, it's just pointing at the original memory.

MS: JSC does the same thing.

AWB: When you originally parse the thing, you can determine whether it was already normalized.

AK: OK, so now your program takes twice the memory when you check it out the wrong way in Git?

SYG: It's not that there doesn't exist a possible fast implementation. There's enough mechanism, other than toString, that requires you to directly index the original source buffer (every engine does lazy parsing),

memory is too much of a concern to keep two copies. If you don't keep two copies, doing the conversion at Function.prototype.toString() time would be a pretty big expense.

AK: One of the reasons this proposal went through was to have a better specified thing which was efficiently implementable.

MM: Even if efficiency wasn't a goal for bringing it up, the lack of objections came from an understanding that there wouldn't be efficiency concerns. It's valid for people to object to the proposal who didn't object to it based on these considerations.

WH: Per MM's criteria on valid objections, I would object to this proposal if line ending normalization were removed, as it would fail to achieve its only goal. The goal of the proposal was interoperability, and it would fail to achieve that. The goal was not efficiency because platforms already had efficient implementations before this proposal.

YK: Why would it not be interoperable without line ending normalization?

MM: The observable changes that come when you change the source text. WH's concern is as valid as AK's.

YK: Engines like this because it relates to current practice

DE: This line breaking concern sounds outdated to me. All modern platforms can deal with all sorts of line endings. Just because a tool does something doesn't mean anything--should we try to make sure there are no observable changes when you write your text in an email and send it, with possible line breaks inserted?

MM: There's no way we could make that work, but we could make things be unobservable across the git line break change

YK: Currently, implementations are allowed to not change the line breaks, and this proposal makes them have to do that, so the status quo is better in that way.

AK: It's reasonable to find objections at this stage, as Stage 3 is when implementations are starting to look at this and go through with a finer eye.

MM: I want to move forward here; I would be OK with either way, though I'd prefer normalization.

AWB: What percentage of source text is already normalized? Maybe the cost here is not so big.

MS: That would require introducing an extra scan, or a lot of extra complexity in the scanner. When we're scanning, we're looking for other things, and just ignore whitespace.

CM: Just normalize the source text as you read it.

DD: You can't do that with HTML. HTML preserves line break types. You can observe this from the DOM.

YK: Could we defer to implementers here, given that we have two valid options?

MM: All of us have valid input

DD: I am wrong; Chip was right

BT: We use source text for all sorts of things, e.g., debugging. Maintaining the mapping between the normalized and non-normalized string would be a nightmare.

AWB: Or, you could normalize way at the beginning, so you don't need the

MS: Our parser is really performance-sensitive

KM: Recovering parser performance from async/await parsing took weeks of work

AK: A slower version of Function.prototype.toString() would be bad for users as well as developers, serving no one.

BT: I just don't see a lot of value for this. I go out of my way to avoid tools that use CRLF

SYG: My understanding was that Windows devs understand line ending difficulties, and specifically do work to counter that.

MM: I don't think that developers would do work for that

AK: Currently, they would have to, if they use Function.prototype.toString().

WH: I'd prefer to do something consistent for users rather than a premature optimization. Even if you have to make a copy of it each time.

AK: Why do you think it's not performance-critical?

WH: Why do you think it is?

DD: It's clearly used a lot in Angular

BT: How many people saw painful regressions for @@toStringTag? (All implementers raised hands.)

AK: To answer the question, I'm not likely to want to ship this with normalization.

MS: What's the compelling use case for fast, normalized strings?

AK: Storing another copy of the normalized string is a non-starter.

WH: It's clear that efficient implementations are possible. We've already demonstrated some earlier in this discussion. Arguing the details of implementation strategies is not a productive use of this meeting.

AWB: I am flabergasted if people are writing JavaScript programs which depend on efficient Function.prototype.toString()

YK: Angular 1 calls Function.prototype.toString() functions which are dependency-injected in startup.

KD: To clarify, if you care about performance, you should turn on build-time optimizations, though not everyone does.

AK: I would recommend to the champion to discuss further with Waldemar about possible changes.

Conclusion/Resolution

- There is no consensus to removing normalization, however four implementers have strong reservations about normalization.
- Champion will get usage data of CRLFs in scripts that require performant F.p.toString
- Champion will ask implementers for hard data about performance costs

13.v Progress report and request for comments on 64-bit int support

(Brendan Eich)

- [Proposal](https://github.com/BrendanEich/ecma262/tree/int64)

- [Slides](fill in)

BE: We need a new plan not based on waiting for SIMD and/or value types, but let's do a way that doesn't depend on those and just hard-code this one to start.

BE: Let's instead start with Int64/Uint64 and then extend from there if possible, by paying attention and future-proofing as we go. In the long run we ideally want to enable libraries to create their own numeric types instead of having to hard-code each one.

BT: Could we put Decimal in through this path?

BE: Maybe, or maybe as a library. The big change here is to add a second numeric type, and then go from there.

BE: The "BBQ joint meeting" and subsequent Twitter came up with a path that allows easier incremental work: no implicit coercions, including no C-like ones. Simple dispatch with just one receiver. Literals and operators are separate proposals. For now literals and operators are hard-coded to work on these types, but the way that this is done matters and is future-proof with those separate proposals, not discussed today.

BE: ToBoolean must not throw, so this proposal hard-codes zero values instead of allowing customizable behavior for ToBoolean.

DE: Why do we need Int64/Uint64 0 to be falsy?

YK: there are languages with zeroes as truthy but it does not work for JS where we have one zero that's falsy.

BE: There are some handwaves, as decimal may have many zeroes. But, generally, numeric types ("value types") would give a canonical zero value.

BE: It's important that ++ and -- be specified as they are today as expanding to, effectively, += 1/-= 1. So we also specify a distinguished value.

MM: Does that always have to be the multiplicative identity?

BE: Eh, not sure if that comes up, but it'd meet that property.

("Consequences/examples for Int64 usage" slide)

BE: The proposal here has hard-coded literals and operator overloading, just for Int64. Literals 1L/1UL, possible to use all operators like *, etc. Operators will use ToNumeric rather than ToNumber, which may return a different Number type.

("Spec.html plan of attack and status")

DD: what is an example of when ToNumeric applies?

BE: for example, Date stuff would not, but multiplication was

DD: I see, so that you could do `5L * { valueOf() { return 4L; } }`, and that would call ToNumeric first.

("Numeric Types" spec screenshot slide)

BE: unaryMinus is necessary because of -0/+0. unaryPlus is not necessary; it's always meant ToNumber, but here we change it mean ToNumeric. Separate sameValue and sameValueZero to avoid continual special-casing of +0/-0.

WH: What do you get when you divide by zero?

AWB: Are all methods mandatory?

BE: For now, all are in Int64/Uint64. Maybe in the future, these could be mapped to symbols, but not for now--all are present, and this is internal spec mechanics.

MM: because we're hard-coding, we don't need to decide on a unique NaN value?

BE: that's right, in fact there's no NaN stuff here

MM: but are you thinking in the future that, like zero and unit, they'd define a NaN value?

BE: I have not found a place in the spec that needs it, and I'd like not to if possible.

BE: In the future, maybe we could have a 'value class' syntax.

(Slide: "Future possible literal suffix support")

BE: in doing Int64/Uint64 I added early errors for out-of-range literal values; there's no good way to do that for future library-created types. I have a horrible regexp hack but it's not fun.

AWB: it seems fine to only get early errors for the built-in ones.

MF: the Uint64s can be negated?

BE: yeah sure, it's just twos-complement.

BE: No implicit conversion

WH: So you can't print them by using + to concatenate with a string?

AWB: ToString for concatenation?

BE: Yeah, sure, just not between numeric types

WH: you could also allow conversion between Int64 and Uint64; it's harmless

BE: yeah, but let's not open the door

MM: C doesn't specify that integers overflow for wrapping, just unsigned integers. I hope you specify deterministic two's complement integer overflow semantics.

WH: Yes, these should just wrap modulo 2^64.

WH: what happens when you divide by zero?

BE: I haven't specified that, but let's do throw?

WH: yeah, seems better than returning an out-of-domain value such as the Number NaN

BE: also `0L ** 0L`

WH: also `3L ** -1L`

WH: `**`, `<<`, `>>`, `>>>` should be heterogeneous operators. It doesn't make sense for them to require the left and right operands to have the same type.

BE: The shifts are heterogeneous. Note that they take Numbers rather than Int64/Uint64 as the right operand.

YK: Rust does some things differently, like suffixes and overflows, so we should consider all choices carefully

MM: sidestepping all the hard problems by requiring homogeneous operands is a great idea

MS: I'd like to see Int32, and Float32, etc, it seems like a wart on the side. It would be great if we had proper types for all of these things, which would be a better way to do things for numbers that makes sense, e.g., for asm.js.

BE: I'd like to keep it limited to just these for now. Scope creep is the enemy.

WH: Historically, this is the right move; we have over the past 16 years created many int64 proposals that got expanded in scope and then collapsed under the increased weight. On the other hand, int32 is so similar to int64 that I'd like to just do it as well.

Conclusion / Resolution

- specify overflow/underflow
- specify division by zero
- communicate with WebAssembly folks to make sure everyone's on the same page
- stage 1

12.iv.c RegExp lookbehind

(Daniel Ehrenberg)

- [proposal](https://github.com/littledan/es-regexp-lookbehind/blob/master/README.md)
- [slides](https://docs.google.com/presentation/d/1jOwKkqQGfRsPH6X9jWNqwMvRB9MbxWJ3NgD_s9jGyRk/edit#slide=id.g199589a01b_0_15)

DE: no back compat concerns, new form added:

```
(missing)
```

DE: earlier semantics was modeled after Perl (fixed-width lookbehind), but no reason not to do variable-width lookbehind

WH: Will backreferences backtrack to x a number of times if y doesn't match?

DE: will backtrack

WH: if you do forward match an assertion x, y can return to x and force x to return different captures... on lookaheads

DE: (need concrete examples to address issues)

WH: Example of what I was asking about: `/(?<=(\d+))A\1/` applied to the string "01234A2345" should succeed and match the "A234" substring with capture 1 set to "234".

Current status: (see slide)

- Implemented in V8 (behind flag)
- Have spec, doesn't merge clean, but can be updated; seems to match implementation

Stage 1?

WH: Concern: numbering of capture groups, they shouldn't be numbered backwards unless you also write the characters of the backwards assertion backwards, which you don't. I am only referring to how capture groups are numbered; the semantics and order in which they are filled (right-to-left) are good as is in the proposal.

BT: In other languages backward executing captures are still numbered forward. It makes it easy to find them by counting left parentheses.

DE: Can look more

BT: C# capture groups still numbered forward

WH: should be numbered forward

BT: Should, but confirm

MS: what about negative look ahead/look behind?
- When doesn't match
- Instead of `=`? Can do all at once

AWB: q about motivation

DE: is this feature well motivated?

Room: yes.

Conclusion / Resolution

- Group capture numbering: not backwards
- Cross language comparison of regexp look behind syntax
- Include negated look ahead and look behind?
- Stage 1
- Stage 2

12.ii.f Intl.Segmenter

(Daniel Ehrenberg)

- [proposa](https://github.com/tc39/proposal-intl-segmenter)

DE: Motivation: https://github.com/tc39/proposal-intl-segmenter#motivation

- useful for grapheme breaking
- text editor jump to next word
- line breaking
- sentence breaking
 (etc.)

 - V8 had a prefixed api

DE: Would like to standardize an Intl API that standardizes. Similar API: https://github.com/tc39/proposal-intl-segmenter#example

- possible additional functionality: jump to point? Could be unit, could be point. Start with minimal API, if users need it, address it.

DE:

Q: Should this be a built-in module?
A: Decouple from built-in modules, if this lands after built-in modules

Q: Passing in a locale for grapheme breaks?
A: Pass in locale for future proofing. (per recommendation)

Q: Hyphenation?
A: Might need it, but very complicated and not analogous. Likely a different API

DE: Thoughts?

DD: glad to see practical application feedback has been incorporated during design

AWB: is there a polyfill?

DE: yes

- Once we agree on API shape, we'll implement in V8

AWB: looking for the implementation feedback, multiple implementations

EF: Should we put this in Intl.js?

SM: A couple node modules can ship without the dictionaries. Shim on top of those—see if use case is still satisfied.

CM: Q about widespread need.

DE: in the C and C++ world, code always using ICU, in JS using node modules

CM: How much is exploratory design vs capturing practice?

DE: not much to it, this proposal captures the need, makes more idiomatic.

SM: Why was it added in V8 originally? That's a use case

DE: Don't know the entire history?

Conclusion / Resolution

- Stage 2

11.ii.a Revisit NaN Again!

-
[slides](https://docs.google.com/presentation/d/1eqimbmVpMZET_5H9NacVkXGP2WNATg8bXWi3Ky2bsGo/edit#slide=id.p)

Terminology:

- scramble: replace a nan with another arbitrary nan
- canonicalize: replace a nan with a _particular_ nan value.

DE: V8 may violate the ES standard for NaN observable behavior, but I want to argue that it's OK

From ES2015:

> SetValueInBuffer (arrayBuffer, byteIndex, type, value [, isLittleEndian])
> "An implementation must always choose the same encoding for each implementation distinguishable NaN value."

Every time you get a new number, it can be scrambled at _that_ point. Wen you put it into an ArrayBuffer, it doesn't scramble

MM: satisfies: "All operations that just move a value opaquely, without interacting with that value, none of them will enable another scramble."

DE: That was my reading

MM: This is what I was looking for, distinguishing "canonicalize" and "scramble". Dealth with information leakage, dealt with performance issues. Thank you.

- Issue: SetValueInBuffer must always provide the same scrambling

DE: No

MM: All opaque conveyence of a first class value may not rescramble

JFB: Not always true. Sometimes moving a NaN changes its value on hardware.

WH: Merely moving a NaN through a floating-point register can change it.

Discussion of non-signaling vs. signaling nans

JFB: Doesn't make sense to distinguish signaling and non-signaling nans

DE: Another possibility: change language in SetValueInBuffer to rescramble

WH: (agreement)

MM: Doesn't satisfy...

(Discussion of floating registers)

WH: I don't see what the quest to nail this down to anything more than "SetValueInBuffer stores an arbitrary NaN" achieves. Arithmetic operations, maps, etc. generate arbitrary NaNs and the same concerns apply to those.

(re-ask about risks, WH and MM?)

DE: (explaining changes that eliminate information leak)

MM: Add text to SetValueInBuffer to canonical

Conclusion / Resolution

- Change SetValueInBuffer to allow set either a bit pattern or a particular canonical value. It can either canonicalize or _not_ canonicalize.
-

- # December 1 2016 Meeting Notes

Allen Wirfs-Brock (AWB), Waldemar Horwat (WH), Jordan Harband (JHD), Thomas Wood (TW), Brian Terlson (BT), Michael Ficarra (MF), Adam Klein (AK), Jeff Morrison (JM), Chip Morningstar (CM), Dave Herman (DH), Yehuda Katz (YK), Leo Balter (LB), Sebastian Markbåge (SM), Kent C. Dodds (KCD), Kevin Gibbons (KG), Tim Disney (TD), Peter Jensen (PJ), Juan Dopazo (JDO), Domenic Denicola (DD), Daniel Ehrenberg (DE), Shu-yu Guo (SYG), JF Bastien (JFB), Keith Miller (KM), Michael Saboff (MS), Chris Hyle (CH), Alex Russell (AR), Brendan Eich (BE), Caridy Patino (CP), Diego Ferreiro Val (DFV), James Kyle (JK), Eric Ferraiuolo (EF), Mathias Bynens (MB), Istvan Sebestyen (IS), Mark Miller (MM), Cristian Mattarei (CMI), Brad Nelson (BNN), Jafar Husain (JH)

4 Approval minutes from last meeting

AWB: Approval of previous meeting minutes?

IS has distributed the minutes via TC39 Email Reflector and has also uploaded it on GitHub.

Conclusion/Resolution

- Approved

6.i Determine 2017 TC39 meeting dates, locations, and hosts.

IS: WE said that in 2017 there will be no European meeting. But if nothing else works Ecma and Switzerland are always a fall-back possibility for organization of TC39 meetings. Of course we need a few months lead-up time.
I will also bring up the issue of the TC39 at the GA next week. Maybe I can find some additional hosting help in the Bay Area. Last point (not said in the meeting): we may try to access some of our university members, Universities usually easily have sufficient large rooms, especially if we come during a time when there is no teaching.

AWB: January meeting at Salesforce.

DG: Yes

AWB: March, May, November still open.

DD: Looking into May in New York

IS: We could also have it in Europe, in Switzerland

AWB: We prevously decided not to have a Europe meeting, but we'll have two east-coast meetings. We should find the meeting place for March ahead of time

SYG: Moz can host, but don't have the space necessary for the size of this group now

AWB: Last meeting 40+

JHD: I'm looking into hosting, tentatively for November.

AWB: What about Google New York in March rather than May?

DD: Maybe

KCD: PayPal can host in March, tentatively. My manager supports it.

AWB: PayPal has hosted in the past.

Conclusion/Resolution

- January: Salesforce San Francisco, Jan 24-26
- March: Paypal San Jose, March 21-23
- May: Google New York, May 23-25
- July: Microsoft Redmond, July 25-27
- September: Bocoup Boston, Sept 26-28
- November: AirBnB Bay Area, Nov 28-30 (tentative)

12.ii.c Promise.prototype.finally

(Jordan Harband)

JHD: Looking at the spec, .catch() calls out to .then(), and .finally() should do the same thing. This removes the concern that I raised about how we should do this wrapping check--we'll just follow what .then() does.

Details in PR: https://github.com/tc39/proposal-promise-finally/pull/14#issue-192162240

Downside of finally calling into .then:

- another call added
- another place where spec created functions are exposed to user code.

AWB: Implementations can do lots of things to make the overhead of closures go away.

DE: Actually, it's hard to eliminate the allocations totally, in some cases, if it is bound to a particular thing.

AK: And we are seeing this with Promises; we are doing a lot of work to optimize, but we are still having a fight with Bluebird as they don't care about following the expensive parts of the spec.

JHD: Even if Promises call into methods, most parts of the spec call directly into internal algorithms.

AWB: We shouldn't permanently introduce inconsistencies because we are having trouble dealing with coming up with optimizations in the short term.

DD: We could further remove the way Promises call into internal methods, e.g., for catch. Maybe that would be web-compatible, even if it's a technically breaking change.

JHD: Seems like this comes down to, do we prefer the observability or the consistency?

MM: I prefer consistency. finally and catch should be consistent with each other.

AWB: This changes the requirements for subclasses, so they have to implement three methods instead of one.

YK: Let's come back at a future meeting to go for Stage 3. then doesn't seem like a good kernel method.

AWB: I'd want to understand the benchmarks better; is this all microbenchmarks?

BT: ecmarkup improved build times by 25% by using Bluebird rather than V8 native promises.

YK: The gap is partly spec compliance and partly implementation quality. Ember does get faster

MM: What are the use cases for Promise subclassing?

JHD: Probably not much usage on the web at the moment.

DD: Because we have the duck type chaining assimilation system, we don't need subclassing.

MM: Would it be web-compatible to kill Promise subclassing?

AWB: What, are we going to introduce final classes?

DD: Just not call into the kernel methods, just call the internal algorithm. It would be more like map, rather than based on kernel methods.

YK: Maybe we should make a protocol for await, to override behavior, rather than the thenable assimilation protocol. I'll come up with a more concrete proposal in the future.

AWB: You can pursue a brand checking mechanism as well, and this may lend itself to advanced JIT optimizations

Conclusion/Resolution

- Remain at Stage 2
- Solicit reviews from both spec text versions by January

12.ii.e Variation on UnambiguousJavaScriptGrammar

(Dave Herman)

DH: Goal is to get to Stage 1 and discuss further with Node, whose represenatives weren't able to make it today.

AK: Thanks for articulating your goal clearly at the beginning!

DH: Script or module?

AWB: This isn't a new question. We had the same discrimination dilemma before between scripts and CommonJS scripts, which are actually function bodies.

DH: The goal is to allow your module source file to always be interpreted as a module; this is useful on the web as well.

WH: You can trivially do that today without needing any proposal. How is this a motivation?

DH: Node's proposal: Let's mandate that you have at least one import or export statement in the module grammar. Then, zero overlap, so tools can disambiguate.

AWB: A problem with that proposal is chapter 16. An implementation is allowed to extend semantics that would make import or export statements parse in scripts.

DH: Node's proposal: If there's out-of-band data indicating whether it's a script or a module, then early error for presence/absense mismatch. If there is no out-of-band data, then decide based on presence.

DH: This violates some important constraints.

WH: When deciding based on presence, you can completely change of meaning of an entire file at the end of the file.

DH: Also `export {}` is a weird magic incantation.

DH: Don't want to mandate a bogus export even if living in a world in which there are only modules.

DH: Alternate proposal: `"use module"` at the beginning of the file. Don't mandate it, but that's a way to switch into module mode. Not required for `<script type="module">`.

WH: (in regards to using the presence of an import statement to distinguish scripts from modules) A script can never import a module?

DH: That's true now. That might not be true in the future.

(Discussion about import statements in scripts: If we allowed that, then the Node proposal would break.)

DH: I don't want to force either `export {}` or `"use module"` in an all-module ecosystem where it is indicated out-of-band. Also, refactoring-wise, bad if removing an import changed semantics significantly.

DH: So, the proposal is instead, `"use module"`, optional to force into that mode, basically the same as `export {}` would've been.

WH: Is this your proposal or Node's proposal?

DH: This is my variation to their proposal.

AWB: Why not have this happen on Node's part, now? Couldn't Node look for explicit imports or exports, and tell developers this, and implement "use module"?

DH: That's saying something about the meaning of content in JavaScript which deserves standardization.

DD: If it only means something in one embedding environment, then it can be done at their level.

JM: Maybe this should work across environments.

AK: Yes, that's why we need to discuss here, if it should work on the web too.

DH: So, the proposal is to allow the "use module" in addition to the import/export conditionality. "use module" would be an early error in a script on the web, permitted but not required in modules on the web, and in Node
it would be the conditionality. I wouldn't want "use module" to have to be sprinkled all over the top of all modules, though.

WH: This suffers from the same perils as you just listed as arguments against Node's proposal. If you use the presence of import statements, scripts will be misclassified as modules in the future when we add the ability to import into scripts. To make this work you'd need to either get rid of the import/export statement sniffing or define a "use script" in addition to "use module" to in-band force scripts to be interpreted as scripts.

AK, CP: What if we just did "use module" without the import/export conditionality?

DH: (Experimental implementation at https://github.com/dherman/esprit)

DH: Usually the mode is known upfront, but the deferred check will have some cost in the cases where it runs. Maybe the import/export conditionality will only come up in a few of the Node corner cases.

AK: So you think it's uncommon is because Node folks were receptive to the idea that it could only come up in certain circumstances? (NB: Background is that the import/export check may require parsing multiple times.)

DH: We should aim for out-of-band signalling most of the time, and in-band only for a couple remaining edge cases, and for defensive programming.

DD: The node leadership is already happy with .mjs

AWB: From Node, this isn't a problem between discriminating between modules and scripts, but modules and CommonJS modules. This is Node's problem: Node doesn't handle Scripts, only CommonJS modules. So, given a source file, Node needs to determine if this should be processed as ES or CJS, it can do its heuristics. Differences in semantics? Mostly strict mode. Does that matter much?

KG: Annex B 3.3 being excluded is big

MM: Strict arguments are very different

AWB: I think anything that Node might be confused if they take a script-like module and treat it as a CJS module.

YK: They don't think that's true empirically.

JM: We are far over timebox, let's move to another agenda item.

WH: I object to Stage 1. It would be a waste of the committee's time as it is strictly worse than the previous proposal, and it suffers from the same issues as the Node detection solution, and it doesn't solve the problem of import statements in scripts in the future.

DH: It would be great if this could reach Stage 1, as this relates to JDD&Bradley's proposal.

WH: We have to wait to see that proposal

AWB: One of the criteria for Stage 1 is that we think it's worth spending time

AK: Actually it says we "expect" to spend time on it. I would expect to spend time on it!

BT: This is a feature request from an important user of JavaScript and we shouldn't just dismiss it.

Conclusion/Resolution

- Stage 1
- The committee has deep doubts about this proposal

14.ii import() open issues and stage 3 discussion

DD: For Node, JHD reached out to Bradley Farias of Node and confirmed that we can start the import of the module asynchronously, as he had suspected. Therefore, the hook can be clarified to always be asynchronous, not synchronous or asynchronous as previously.

DD: For MM's membrane penetration concern, the plan is that we will be working towards hooking of agent hooks ("the loader spec"), though we aren't blocking on it. Current embedding environments do support hooking.

DD: (walking through https://github.com/tc39/proposal-dynamic-import/issues/26)

MM: We're expecting that we'll work towards a standardized loader spec for generalized hooking, and be happy with how embedding environments (e.g., Web and Node) do provide an API.

AWB: If need API, then say you need in spec.

MM: minimal requirement, have a standardized API now. Minimal requirement that there exists a way for this to be implemented on all runtimes, there must be a spec hook to

AWB: We can also require host environments to provide a way to expose these hooks, rather than just expect it, writing out in the spec what is required for it to be sufficient.

DD: This is novel, and I'd rather not tie my proposal to that. But it seems like a possible path. I'm expecting to get a PR from Mark to formalize whatever is required.

MM: I'd like to look at what hosts expose more closely. Is there great urgency?

DD: We'd like to implement this.

AWB: And you won'd do it if it's not Stage 3?

DD: Right

AWB: Stage 3 is about getting usage and implementation feedback, and adjusting if necessary. We are anticipating feedback from Mark

MM: In this delecate agreement, I'd like to take a look at some details before giving a final signoff.

AWB: But we're not talking about changing the normal public surface area of the feature. So there's no particular reason why it should block Stage 3.

MM: I am not expecting it to block Stage 3, but I'd like more time to look at it. But I can do that at Stage 3.

DD: Final obstacle was run-to-completion semantics, from AWB. Talking with BT, we were thinking of working this out spec mechanics during Stage 3. I have a note indicating the need for run-to-completion.

AWB: Should be a normative requirement!

DD: Done.

WH: Any remaining syntax issues from Tuesday? For the record, what was the decision?

DD: Result is that it looks like a function call.

AK: With some static restrictions.

DE: Stage 3 is good because it indicates some stability that helps implementations know they're not wasting their effort.

MM: Will this ship before the next meeting, prohibiting future changes?

AK: We'll initially develop it behind a flag, and it will go through normal Chrome shipping processes, which take some time.

Conclusion/Resolution

- Stage 3

13.ii.b Private State

(Daniel Ehrenberg)

- [proposal](https://github.com/tc39/proposal-private-fields)
- [slides](https://docs.google.com/presentation/d/1QBK8GsTYmQHJQJm_P0HuELNN4uGkm9gB7Df9g4tRK-o/edit#slide=id.g1994dec4f4_0_35)

DE: Private state to stage 2

- Syntax is same
- Follows Kevin Smith's proposal
- Class private state, not instance private

```js
class Foo {
   #x;
   #y = z;
   foo() {
      #x++;
      return this.#x;
   }
}
```

DE: Questions remain for both private state and public fields, but public fields still at stage 2

AWB/WH: They're in fact the same questions. Apply to both.

DE: (semantics)

- Only accessible from methods inside class body (“hard-private”)
- Internal slots mapping objects to field values
- Add field after super() returns, or at beginning in base class
- Intersperse initializer evaluation with field addition
- Throw on redundant property definition
- Scope of initializers as in property declarations

The intention is to match public fields, need to be interleaved, specifically the ordering of steps e.g. adding fields.

AWB: Concerns about separate proposal for public fields. Merge the proposals.

DE: The plan is to merge the specs for Stage 3

-

DE: If feedback reveals that the # syntax for private state is fatally unworkable, we wouldn't want to kill public state because of it

MM: if integration reveals issues for one and not the other? THen we can address?

AWB: Or drop entirely?

YK/AWB/DE: (dscussion of the merging concerns, but only w/ regard to Stage 3?)

AWB: Concerns about separate proposals modifying the same spec algorithms. They need to be merged when we get to that level of detail.

- in stage 2, merge must be addressed

DH: I love the smell of consensus in the afternoon ;)

DE: Should there be a way, through reflection, to access private state outside of the class? No.
- Previously, formalism based on weakmaps
- New formalism is same, but without the GC semantics

DE: Add field after super() returns, or at beginning in base class—no matter where super() is called

AWB: May be issue? Reflect.construct call is intended to be an alternative super(). Possible to move this in the construct, doesn't have to relate to token
- construct is where belongs

WH: Concerns that one shouldn't be able to use this to stick a class's private fields onto arbitrary unrelated objects

...

DE:

Interaction with other features

- Decorators -- reify a PrivateFieldIdentifier() object (https://github.com/tc39/proposal-private-fields/blob/master/DECORATORS.md)
- A decorator can provide soft-private state
- Private methods -- a clean extension with the syntax you expect (https://github.com/tc39/proposal-private-fields/blob/master/METHODS.md)
- Static private -- ditto (https://github.com/tc39/proposal-private-fields/blob/master/STATIC.md)
- Friends -- may be lexically exposed via static block (https://github.com/littledan/proposal-class-static-block)

```js
let barGetter;
class Foo {
  #bar;
  static { barGetter = instance => instance.#bar }
}
```

MM: could via decorator, declare a private field to be effectively const, initialized, not further modifiable?

DE: Not presently

YK: Not yet

(back to syntax)

MM/DE: (discussion about the dot in `o.#p`, ultimately they came around to the dot.)

DE: Why hard private?
(https://docs.google.com/presentation/d/1QBK8GsTYmQHJQJm_P0HuELNN4uGkm9gB7Df9g4tRK-o/edit#slide=id.g1994dec4f4_0_20)

WH: This relies on not being able to create fake instances of the class that has the private property. That's the concern I raised earlier.

AWB/WH: We need to explore that.

DH: possibly making a mistake by using the concise syntax for the air-tight private state?
- The question is: what do people want in the real world? Afraid we're missing that

JM: Have you seen the decorator opt-out?

DH: yes, but concerned we're making wrong decision.

DE: the developer field is split evenly, some for hard and some for soft private.

JM: The concern of hard private is that its very hard to debug
- if debugging in a debugger, you can see the hard private, but your tests cannot

WH: Same argument could be made about the other hard private facility of the language, namely locals in functions. Although some might want non-air-tight functions (for example for testing), it's not worth it to add another kind of functions to the language. Just like with private fields, a debugger will show you the values of locals in functions.

YK: Some know how to use underscore (prefix), some symbols; might use underscore as example that we did it wrong

(Discussion about all the problems of using underscore, ie. not soft-private/soft-protected)

MM: It's important for me that hard-private be ergonomic.

AWB: The key elements of the proposal are hard private and lexical scoping of private declarations. If either of those were reversed, this would turn into a radically different proposal. If we're advancing this to stage 2, it's with the understanding that these will be in the proposal.

DE/DH: (further discussion of hard private).

- Want real usability feedback

YK: (point about developer ergonomics compared to similar cases re: decorators)

AWB: Proxy issue?

DE: Resolution was: can add private state to proxy

MM: WEakMap semantics: proxy has identity and can key on proxy

MM, WH: No proxy issue.

MM: Intended to be observationally equivalent to weak maps. If there is a place that isn't, it's a bug.

DE: Can add text that explains the equivalence to weakmaps; semantics of this proposal, etc.

Internal slot-based specification mechanics

- Private State Identifier specification type
- [[PrivateStateValues]] is a List with pairs mapping Private State Identifiers to their values
- Difference from WeakMaps unobservable except for implied GC semantics

DE: Q. is this just too unergonomic? Some wanted @, not #—and very opposed to #. Some Wanted `private`.

WH: Need a sigil or equivalent. We've spent years working through attempts to define private state without any distinguishing sigils on use. None of them worked. They work fine in statically typed languages, but in dynamically typed ECMAScript were hopelessly mired in the same class of namespace shadowing problems that plagued the with statement. Consider what happens if a class defines a private field called "length" and also wants to access a length field of an unrelated object.

AWB: 6-7 years ago worked through extensive ways of doing this and all had problems.

AWB: Extensively documented in old wiki

DE: static private, methods included? Or wait?

AWB: Do the full package here.

KG: why? (as opposed to a later proposal, if confident that this will allow for a later proposal?)

AWB: cross cutting concerns, if not pinned down together, and proceed with only confidence, then discovery later will be problematic or impossible to fix.

DE: Open issue: require a "private" keyword on declarations? (i.e. `private #x;` instead of `#x;`)

DE: Pro: useful for learning the language

WH: Any syntactic reason for needing a keyword?

DE: No.

AWB: had to chose: follow separator of block or object literal, went `;`

YK: important to support commas to enumerate a list of fields:

```js
class A {
  #x, #y, #z
}
```

Annoying to write:

```js
class A {
  private x;
  private y;
  private z;
}
```

Bad: private and decorators and no comma list

WH: Whether or not we allow commas is orthogonal to this proposal.

DE: could make private keyword optional, but is weirder

YK: `private @protected foo` is something to keep in mind?

DD: Does anyone like the `private` keyword?

AWB: the sigil alone is hard to read, `public` and `private` are clear

WH: Can't mandate `public` and `private` keywords. Public methods don't use a keyword, so that ship has sailed assuming that we'll want private methods at some point. My preference is no keywords — sigil only.

KG: opposed to `private`, but those who are opposed to `#` are less opposed to `private #x`.

(discussion re: `=` or `:`)

CM: why is this even a question?

JM: There is a question on public fields: Some believe `=` implies assign/set and that `:` implies define.

KG: only relevant for public fields, evaluation order

YK: TypeScript and Babel transpile as:

```js
class A {
  x = 1;
}
```

to

```js
var A = function A() {
  _classCallCheck(this, A);

  this.x = 1;
};
```

JK: It does this by default, however when you enable "spec" mode it will switch to using a defineProp to set configurable to false

KG: concern that if not addressed now, Babel will proceed and like the "sigil swap", it will become hard to change later.

AK: valid concern, but `public` is much more widely used

DE: much more open to `private` with comma list

JK: This is a slightly different situation from the sigil swap, there is a better migration path available

WH: keyword ship has sailed, problems when we do private methods with `private` but public without `public`

(JK: We've been shipping public fields and methods in Babel without a public keyword and have not seen confusion about this. I don't see a need to match `private` and `public` if we add the `private` keyword)

AWB: This is why I proposed `own` and not

AWB: Change to IdentifierName, not IdentifierPart

WH: Yes, I raised that the last time we discussed this proposal.

JM: Why?

AWB: Don't want properties called #1, #2, #3

JM: Why?

AWB: We use IdentifierName in the grammar

WH: It seems odd to have a PrivateName be a single token. The more natural way to express it would be as two tokens, `#` followed by an IdentifierName.

KG: Won't that allow whitespace between `#` and IdentifierName?
- The hash is intended to be thought of as part of the proeprty name, ie:

```js
this.
  #foo.
  #bar;
```

(RW, LB: Weird)

AWB: No whitespace between `#` and `foo` in `#foo`, but also `foo` not IdentifierPart

WH: Want whitespace allowable on either side of #

DE: Should private state be accessible within eval?

Conclusion/Resolution

- Stage 2
- hard private, lexical scoping
- visible in direct eval
- stay in lock-step with public fields proposal for '=' and initializer scoping

- static private and private methods should be investigated as part of stage 2
- no consensus on adding 'private' keyword (current proposal does not include it)

13.vi Process proposal: require an implementation to land a normative PR to the spec

(Daniel Ehrenberg)

- [proposal](https://gist.github.com/littledan/f85d2d1f4cff44927a75afd5d30bbe2e)

WH: What are the majority of the spec pull requests?

BT: Editorial. Among the remainder, the majority are consensus items.

DE: proposes that PRs for normative changes should also have an implementation before being merged.

AWB: case by case?

MM: We can adopt this process and apply it on a case-by-case basis

AWB: what if it's a bug?

DE: Want a high bar to making normative changes to the spec to avoid bugs.

WH: It also makes a high bar to fixing bugs.

AWB: Caution against adding too much burden to the process.

BT: Don't want to inject a lot of process into simple things. Don't think adopting this proposal would change anything in practice.

YK: But it would cause more process lawyering.

Conclusion/Resolution

- Raised the issue, pay attention.

13.vii Open-ended: How can we promote diversity and inclusion in TC39

(Daniel Ehrenberg and others)

DE: How can we improve this?

AWB: member companies need to send more diverse language implementors

(general discussion wrt adopting a Code of Conduct that applies to diversity based on demographics background)

WH: Code of Conduct and meeting participation changes must be handled at the ECMA General Assembly level. Also, ECMA has a code regulating conduct.

RW: We should talk to Istvan to have ECMA adopting this Code of Conduct

(general agreement)

RW: The next step is we as delegates communicate to our companies that we should sponsor and support diversity, affecting who we bring to the meetings as well. This might be done via scholarship or otherwise.

MM: We lack geographical diversity.

WH: We've never had a meeting in the midwest. In the US we've only had meetings near ocean coasts.

RW: (explanation of counter to geographic claim, w/r to specific demographics and personal safety)

MM: Geographical location is one of the few things that this committee has direct control over

(discussion about different working and debate styles)

WH: The biggest indictment of our committee is what happened to Kevin Smith in Munich.

JK: There are groups unwilling to come, due to the nature of TC39

WH: Are they members?

JK: Certainly within member companies, but also people who don't push their companies to become members

JK: We've been talking about diversity and not directly mentioning disenfranchised groups. There are people who are less likely to come because they have been historically mistreated. I wouldn't say one diversity metric is more important than another. However, we need to consider all of these groups in context of one another. We should not hold meetings in places where disenfranchised groups would not feel safe. For example, I am gay and I ask that the committee not hold meetings in places where gay people are unsafe, there are countries where gay people are being killed.

DE: Mark makes a good point about geographic diversity; globally, many of our companies are focused on the next billion users, and we don't have much representation from developers who are targeting that, who may have different mental models. It's not just diversity of people who we bring here, but also inclusion of the people that are in the room. Our culture here can be intimidating--it is hard to give a presentation when committee members will interrupt from very early on, and this works for some cultural styles of presenters but not for others.

BT: Sounds like we are all in agreement that diversity is an important issue that we want to work on

MS: It is important for us to work on making this a less intimidating place which is more welcoming to new members.

MM: re: hostile environment issues that resulted in Kevin Smith leaving: we missed the opportunity to improve ourselves as a group after the incident in Germany

BT: Let's continue this discussion, including developing a statement from the committee that would help companies like mine and Rick's find funding to allow the participation of diverse members.

DFV: It feels like there is unnecessary tension, tension about more than technical things, and it makes me hesitant to speak much. We saw some of that today, where it was uncomfortable to speak. It feels like there is sometimes too much confrontation here.

JK: We can also work to create more resources for people to learn how to get involved.

DE: Maybe we should be fixing our culture to be more respectful and inclusive as the first step; I wouldn't want to bring a lot of new people into a hostile situation. The code of conduct could help here.

SYG: Can we really expect people to follow this?

MS: We are getting better at enforcing timeboxes, but we should consider being even more strict in enforcement--if members don't grant an extension, it should be done.

LB: Some companies may have trouble/refuse participating because they see the lack of diversity and a code of conduct on a company. I have had discouragement from colleagues personally as English is my second language--this made me feel like running away. Fortunately, my employer, Bocoup, on hearing this, put me on the test262 project and eventually sent me to the committee. I have many people from my country (Brazil) who contact me and want to get in touch about reading the spec, etc. I am just a normal person--I needed to find that, regardless of where I come from, I was able to attend. It would be very useful to have an official document from ECMA saying that ECMA and TC39 commit to social responsibility, where people can feel respected in any sort of background. Everyone should feel encouraged. I am here, I am the proof of it. I feel responsible to bring that forward to the new people.

Leo giving speech that's going to make me cry.

Conclusion/Resolution

- Consensus that diversity is important and something we'd like to address and improve
- Take initial steps to adopt a Code of Conduct and determine how to enforce

    - - Rick Waldron will contact Istvan
    - - Rick to review with Kevin Smith

- Representatives should work with their member organizations to promote more diverse representatives, via scholarship/sponsorship or diversification of member team
- Extend our charter to include social commitment to committee representative diversity.