**Minutes of the:** 49th meeting of Ecma TC39
**in:** San Jose, CA, USA
**on:** 17-19 November 2015

# 1 Opening, welcome and roll call

Ecma/TC39/2015/046 Venue for the 49th meeting of TC39, San Jose, November 2015

## 1.1 Opening of the meeting (Mr. Neumann)

**Mr. Neumann** has welcomed the delegates at PayPal in San Jose, CA, USA.

Companies / organizations in attendance:

Apple, Mozilla, Google, Microsoft, Intel, jQuery, Facebook, Netflix, PayPal, Yahoo!, Shape Security, Airbnb, SalesForce

## 1.2 Introduction of attendees

| | | | |
|---|---|---|---|
| 1 | Michael Saboff | Apple | Member |
| 2 | Caridy Patino | SalesForce | Guest |
| 3 | John Buchanan | SalesForce | Guest |
| 4 | Brian Terlson | Microsoft | Member |
| 5 | Paul Leathers | | |
| 6 | Jeff Morrison | Facebook | Member |
| 7 | Lee Byron | Facebook | Member |
| 8 | Sebastian Markbage | Facebook | Member |
| 9 | Birn Branner | | |
| 10 | John Neumann | Multiple | Member |
| 11 | Mark Miller | Google | Member |
| 12 | Georg Neis | | |
| 13 | Arnoud J Le Hors | | |
| 14 | Istvan Sebestyén | Ecma-International | PHONE |
| 15 | Jafar Husain | Netflix | Member |
| 16 | Jordan Harband | Airbnb | New Member |
| 17 | Eric Ferraiuolo | Yahoo | Member |
| 18 | Daniel Ehrenberg | Google | Member |
| 19 | Michael Ficarra | Shape Security | Pending Member |
| 20 | Chip Morningstar | Paypal | Member |
| 21 | Nagy Mostafa | Intel | Member |
| 22 | Peter Jensen | Intel | Member |
| 23 | Lars Hansen | Mozilla | Member |

| | | | |
|---|---|---|---|
| 24 | Jacob Groundwater | Google | Member |
| 25 | Adam Klein | Google | Member |
| 26 | Allen Wirfs-Brock | Self | Invited expert (phone) |
| 27 | Rick Waldron | Jquery/Bocoup | Member |
| 28 | Brendan Eich | Self | Invited Expert |
| 29 | Dave Herman | Mozilla | Member |
| 30 | Yahuda Katz | Jquery/Tilde | Member |
| 31 | Stefan Penner | Yahoo | Member |
| 32 | Shelby Hubick | | |
| 33 | Sebastian McKenzie | | |
| 34 | Kevin Smith | | |
| 35 | Waldemar Horwat | Google | Member |
| 36 | Zibi Braniecki | | |

### 1.3 Host facilities, local logistics

On behalf of PayPal Chip Morningstar welcomed the delegates and explained the logistics.

## 2 Adoption of the agenda (2015/048-Rev1)

Ecma/TC39/2015/048 Agenda for the 49th meeting of TC39, San Jose, November 2015 (Rev. 1) was posted in the TC39 documentation.

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

# Agenda for the: 49th meeting of Ecma TC39

1. Opening, welcome and roll call
    i. Opening of the meeting (Mr. Neumann)
    ii. Introduction of attendees
    iii. Host facilities, local logistics
2. Adoption of the agenda
3. Approval of the minutes from September 2015
4. Report from the Ecma Secretariat
5. Proposals for Future Editions of ECMA-262
    i. Promise rejection tracking events (Domenic Denicola - Google)
    ii. Advance Array.prototype.includes to stage 4 (Domenic Denicola - Google)
    iii. Reconsider .return() feature in generators and iterators (Daniel Ehrenberg - Google) slides

iv. Should destructuring declarations without bindings throw (Brian Terlson - Microsoft)
v. Update on Object.observe proposal (Adam Klein)
vi. Advance Object.values/Object.entries proposal to stage 3 (Jordan Harband)
vii. Advance String#{padLeft,padRight} proposal to stage 2 (3 if possible) (Jordan Harband)
viii. Advance Observable proposal to stage 2 (Jafar Husain)
ix. Discuss web compatibility considerations for Annex B 3.3 sloppy-mode block-scoped function hoisting (Daniel Ehrenberg - Google) slides
x. Simplification of ES2015 RegExp Semantics (Brian Terlson - Microsoft, Daniel Ehrenberg - Google) slides
xi. Stage 0 approval for the RegExp Buffet Menu (Brian Terlson - Microsoft, Gorkem Yakin - Microsoft, Nozomu Katō):
    a. Negative & Positive look-behind
    b. Named capture groups
    c. Comments
    d. Free-spacing
    e. Mode specifiers
    f. Unicode Properties (eg. WHITE_SPACE, ID_START, etc.)
xii. Stage 3 approval for function.sent metaproperty (Brian Terlson - Microsoft, Allen Wirfs-Brock)
xiii. Stage 1 approval for private state (Yehuda Katz - jQuery, Allen Wirfs-Brock)
xiv. Stage 2 approval for class and property decorators (Yehuda Katz - jQuery, Brian Terlson, Microsoft)
xv. GitHub Proposal Process Discussion: When to migrate to official repo?
xvi. Proxy [[Enumerate]] overconstrains implementations and can produce non-string keys (Brian Terlson - Microsoft, Andreas Rossberg - Google)
xvii. Simplify semantics of TypedArray base constructor to harmonise with proxies and optimisations (Andreas Rossberg, Adam Klein, Mark Miller - Google)
xviii. Improving consistency of `@@species`. 1 2 (Kevin Smith)
    a. @@Species Design FAQ by Allen Wirfs-Brock
xix. Error.isError (Jordan Harband)
xx. System.global (Jordan Harband)
xxi. Trailing commas w/ functions -> Stage3 (Jeff Morrison)

6. Test262 Updates
7. ECMA-402 3rd Edition, 2016
    i. Update on 3rd edition (Caridy Patiño)
    ii. Proposal to expose existing abstracts operations (Zibi Braniecki)
    iii. Intl.PluralRules (Caridy Patiño & Eric Ferraiuolo)

| Dates | Location | Host |
|---|---|---|
| 2016-01-26 to 2016-01-28 | San Francisco, CA | Salesforce |
| 2016-03-29 to 2016-03-31 | San Francisco, CA | Mozilla |
| 2016-05-24 to 2016-05-26 | Munich, DEU | Google |
| 2016-07-26 to 2016-07-28 | Remond, WA | Microsoft |
| 2016-09-27 to 2016-09-29 | Los Gatos, CA | Netflix |
| 2016-11-29 to 2016-12-01 | Menlo Park, CA | Facebook |

    iv. Intl.RelativeTimeFormat (Caridy Patiño & Eric Ferraiuolo)

8. Date and place of the next meetings

9. Closure

## 3 Approval of minutes from September 2015 (2015/045)

Ecma/TC39/2015/045 Minutes of the 48th meeting of TC39, Portland, September 2015.

The minutes were approved without modification.

## 4 Status of “ES6 Suite” submission for fast-track to ISO/IEC JTC 1

Ecma/TC39/2015/049 Status of ECMAScript fast-track to JTC 1

**Mr. Sebestyen** reported that regarding the ISO/IEC JTC 1 fast-track of ES6 the current status of the discussions with the ISO Secretariat is the following:

Ecma Secretariat and ISO/IEC JTC 1 has discussed the situation again at the JTC 1 Plenary meeting in Beijing at the end of October 2015.

ECMA-404 "JSON" - which has been taken out of the old "ECMA-262 as standalone standard , only a few pages, and it has not been changed for many-many years, and no intention to

change it ever... So very stable. That one we agreed can be fast tracked to JTC 1 as a normal fast-track.

The plan is that the new Edition of IS 16262 will be a 2 page standard that normatively references ECMA-262 and ECMA-402 without a “year reference”. This means that always the latest version of ECMA-262 and ECMA-402 is being referenced to. In such a manner ECMA-262 and ECMA-402 will not be fast-tracked anymore to JTC 1.

This solution solves the problem of the yearly updates of ECMA-262 and ECMA-402 which could not be effectively synchronized with relevant JTC 1 standards.

# 5 ES7 and Test262 Discussions

Most time was spent to progress ES7 related topics.

For details please see Annex 1 in the Technical Notes.

# 6 Report from the Secretariat

Ecma/TC39/2015/047 TC39 chairman's report to the CC, October 2015

This was presented to the CC and discussed in details.

# 7 Date and place of the next meetings

See the table above in the Agenda.

# 8 Closure

**Mr. Neumann** thanked the TC39 meeting participants for their hard work. Many thanks for the technical note taker **Mr. Waldron** in Annex 1.

Many thanks to the host, PayPal and **Chip Morningstar** for the organization of the meeting and the excellent meeting facilities. Many thanks in particular to the hosts for the social event.

Annex 1

Technical Notes

# November 17th 2015 Meeting Notes

Jafar Husain (JH), Eric Farriauolo (EF), Caridy Patino (CP), Adam Klein (AK), Michael Ficarra (MF), Peter Jensen (PJ), Domenic Denicola (DD), Jordan Harband (JHD), Chip Morningstar (CM), Brian Terlson (BT), John Neumann (JN), Dave Herman (DH), Brendan Eich (BE), Yehuda Katz (YK), Jeff Morrison (JM), Lee Byron (LB), Daniel Ehrenberg (DE), Lars Hansen (LH), Nagy Hostafa (NH), Michael Saboff (MS), John Buchanan (JB), Stefan Penner (SP), Sebastian McKenzie (SMK), Waldemar Horwat (WH), Mark Miller (MM), Paul Leathers (PL)

## Async functions (Stage 4 Process Discussion) (BT)

YK: can we order the agenda better? Lets make a template and follow it next time

BT: potential ES2016 items first

BT: async function process discussion, they have been stage 3 for 2 months, implemented in edge (soon FF), implemented in babel

YK: babel version i

DD: does babel pass test262

BT: no test262 yet

YK: thats a blocker

DE: Is babel 100% spec compliant, we should wait until we have 2 100% spec implementations

YK: does transpiling to generators make the edge cases easier to deal with?

SM: I believe so

YK: It seems like edge cases are always going to happen

DE: if we wait for FF, we will get more implementor related input

YK: we need to figure out this rule, babel “loose” mode clearly doesn’t count. Does babel trying for high fedility count?

DE: browser implementations will likely have different criteria

YK: high fedility simulation should be sufficent

BT: edge is basically desuguring to promises + generators

AK: two implementations that pass the test, should be sufficient. (tests implied as test262)

BT: I believe we as a group should be able to deem that the current landscape is or is not sufficient.

YK: Are we holding babel to a higher standard then browser?

DD: there is a difference between bugs and missing features

DE: we should wait until 2 implementations pass the tests

DD: test should be approved by reviewers.

BT: mainline tests must pass

SP: In summary, tests approved by reviewers must have 2 implementations that pass.

BT: should we update the process document, the champion comes with a set of tests for stage 4, the group uses this to +1/-1

YK: source2source should be considered sufficient for “implementations’

AK: stage 3 is somewhat problematic, implementations must put out there neck early.

SK: sounds like a good filter, if no implementation wants to stick out their neck maybe the feature doesn’t have value

DE: if there are tests, that we agree are good, any implementation should be sufficient.

BT: if I come back tomorrow with mainline tests, that reviewers agree on, is babel + chakra sufficient?

everyone: yup

BT: babel should be considered an implementation.

KS: stages are meant to signal churn risk, the only problem with babel is less likely to give churn feedback

YK: churn feedback?

KS: 80% implemented, 20% missing, will babel give feedback?

YK: babel does provide this feature.

DD: can we accept 2 user-land transpilers

YK: browsers could also implement as source2source step.

YK: proxies are a good example, likely input from babel isn't as valuable. As it must make all . operations slow. Which wouldn't be acceptable for a runtime. and also why babel doesn't offer proxies.

YK: so only cpp compilers qualify?

BT: we can look at it case by case, and deem if it is sufficient or not. Some features are likely fine as source2source and others are not.

YK: DD you are right to be worried

DD: thanks you, we can move on.

BT: time for a regexp talk ?

## Conclusion/Resolution

None yet.

# Object.values/entries

JHD: OK for stage 3? It got signoff … [general consensus]

### Conclusion/Resolution

Object.values/entries approved for stage 3

# String.pad{Left,Right}

JHD: Concerns raised on es-discuss: 1) No grapheme handling (response: then other existing things should be changed, and nothing cares about it. Every language follows native string support in its padding.)

… [General consensus that that's OK]

WH: I see the choices as either doing this simple thing (measuring code units) or doing something really, really complicated that works correctly on graphemes. The simple thing is useful in practice, while implementing grapheme measurement would be too complicated and take us a long time. So I support this.

JHD: 2) Naming: we sometimes use left/right, sometimes start/end, this one feels right. The language already equates left/start/index-0 and right/end/last index. In this context, RTL doesn't apply…

DD: How about padStart, padEnd?

DH: Well, it's very entrenched

JHD: BiDi is really complicated

DD: Eurocentric

DH: How hard will we get trolled? Being willfuly different…

YK: people wouldn't expect padLeft, if the language has padLeft and it does something different. That is wrong. If we have other methods do the same thing, we are good.

JHD: Only remaining objection remaining is the name

padStart only makes sense if its RTL aware YK: saying padLeft, and meaning visual right is clearly bad.

YK: any objections start/End ?

CM: Remaining parallel with trimLeft/trimRight, established names?

DD: Add trimStart, trimEnd and rename to padStart, padEnd as part of annex B.

### Conclusion/Resolution

Rename to `padStart`/`padEnd`, update trimLeft/trimRight proposal to also include trimStart/trimEnd. padStart/padEnd approved for Stage 3.

## Array.prototype.includes

DD: CC + FF in betas

DD: test262 works

DD: Safari doesn't have TypeArray.prototype.includes

DD: Stage 4?

DD: should it be in unscopables?

BT: if it not in unscopeables it may be ok?

BT: real world with no breaks, should be good.

DD: lets make it unscopable

DD: stage 4?

YK: yup

### Conclusion/Resolution

Move to stage 4 (tomorrow when it's added to @@unscopables)

# function.sent

BT: We have not gotten any feedback at all, not from implementors, no implementations, no Babel

YK: That's scary; we need some feedback

DE: Do we have reviewers?

BT: It's so tiny! It should be OK to add it. It would be almost impossible to design it wrong.

JM: But, does anybody need it?

https://github.com/allenwb/ESideas/blob/master/Generator%20metaproperty.md

SP: Let's defer until we have a strong advocate with use cases

BT: There are use cases for it, it's just that no one's used it, because it's not implemented anywhere

YK: Is stage 2, only about completing the spec language? dave's tweets say userland experiment

AK: The thing we're trying to avoid is adding something to the language that won't be useful for anyone

BT: Implementing something at Stage 2 carries quite a bit of risk

## Conclusion/Resolution

- Designated reviewers: DE and DD
- Side process conversation: AK, YK, BT

# Object.observe update

AK: Object.observe is going to be deprecated in Chrome. I'd like to formally withdraw it from the stage process.

YK: I already submitted a PR to remove it from tc39/ecma262/README.md!

AK: I haven't gotten very much negative feedback

YK: Maybe the framework wars will be settled and we can revisit this question

## Conclusion/Resolution

Object.observe is withdrawn

## Should destructuring declarations without bindings throw?

```
let { } = obj;
let { foo: {} } = obj;
```

BT: who is in favor if this being an error?

BT: this may actually be programmer intent

BT: it may be expecting side-affects, exhausting an iterator etc.

YK: decomposing (commenting out large chunks of a destructing statement)

BT: code-gen may not be as ergonomic

YK: code-gen supporting this is simple enough, that isn't a good reason

BT: no binding identifers in any pattern or sub-pattern, should error.

YK: it has to be recursively defined.

DH: refactoring transformations breaking down, due to this restriction is is unfortunate

DH: without a good argument, I error on the ergomoics wins composition over the error.

WH: This is like trying to delete the number zero from the number line. It's cognitively simpler to keep it than to avoid it. {} and [] are legitimate objects and shouldn't be a special case.

DH: it is the base-case of a recursive definition.

YK: As I write code, I often write this, expecting to fill it in after. Only to find a parse/syntax error.

DH: patterns are defined to symmetry of the structure, in JS we can define 0 or more for [] and {}.

BT: is there strong support to make this an error:

YK: what was the exact error

BT: A user used a : instead of = when attempting to set a default during function arg destructuring.

YK: This seems like a good use-case for a linter.

WH: We get these kinds of errors anyway with users putting object literals into contexts where ES parses them as blocks.

?: There is a use case for users whose style guides do not allow holes in arrays

BT: current semantic remains?

everyone: agrees

### Conclusion/Resolution

Current semantics stand

## legacy function hoisting semantics in sloppy mode (DE)

https://github.com/tc39/ecma262/pull/175

DE: self defining functions, GWT generates this whenever there is a static block

BT: how many sites?

DE: GWT is hundreds of thousands, FB uses etc. Quanitify the impact is hard to do.

YK: BT how many sites did edge break

BT: none

DE: there are several things together that cause this issue.

DE: try { } catch around all code, function hoisting out of try block

DE: summarize, the problem is: 1. sloppy mode block scope function (in a try block likely) 2. self defining function 3. the self defined function is called multiple times 4. the function is not idempotent.

DE: Google inbox broke.

DE: several work-arounds, but require changing existing code.

YK: IE 11 has this in prod, how have they not recieved bug reports?

DE: I have a proposal that may work, you may not like it

1. TL;DR making sloppy mode function declaration “host’ to a var outside the block
2. would not change strict mode WH: If the outer block contains a binding with the same name as the function being defined in the inner block, will this proposal break that code?

DE: Yes.

YK: Intersection semantics, we will only support use-cases that work cross platform

BT: We did aggregate large amount of data, and did not find this case.

DE: Team working on Google inbox doesn’t know which way to fix inbox breakage because what’s broken by the hoisting semantics is undefined.

WH: The ES2015 spec is well defined. There is nothing undefined here about how to fix Google inbox to be compatible with it.

YK: we knew this is a composition breaker

DD: 3 browsers support, 1 browser does not. Should we see if people update?

YK: it doesn’t seem like a widespread problem, inbox is an issue but it may not be a bigger issue.

BT: + 2 years of IE11 and edge (tens of millions of users)

DD: the mobile web, isn't represented here.

PL: We researched the public web, looking for this. We found some issues, solve them, came to consensus

WH: the issue is, this fix breaks other things

DE: what does it break

BE: we don't know

WH: the future is bigger then the past, lets not sacrifice the future for the now.

WH: The scary part is that under this proposal the resolution of an identifier in the outer scope will change depending on whether the code is strict or not. The same code will work in both strict and non-strict mode but do completely different things. That's terrifying because it's easy to unintentionally move a function into strict mode or, conversely, get the use strict declaration wrong and not have it be strict even though you intended it to be.

WH: this is a foot canon, not a foot gun.

AK: todays practical semantics are undefined, do to current state of implementations

KS: can you get more data, what is the "true' impact

YK: FF can do implement it, inbox needs to fix

AK: inbox fixing it is trivial

WH: if we take out block scope from sloppy, we would have to from strict.

WH: lexical semantic differences, now would critically be different:

DE: that is already the case.

WH: This is a completely different order of magnitude. This strict-vs-non-strict scoping incompatibility would be far more common than some existing obscure differences in eval hoisting.

PL: It’s true, different browsers have different audiences. I wish we made some changes in the past.

YK: we knew we would break stuff here, we agreed we would only support already cross platform sites. IE11/edge shipping two years ago, should be evidence enough.

YK: we intentionally changed semantics, at that time there was strong consensus.

DE: I feel I don’t have strong consensus, it seems like I may need more information.

DD: lets make sure we converge on two semantics

BT: lets be clear EDGE/IE11 semantics are standard thing

AK: the group seems to want more data

YK: So a google only optional spec doc?

BT: So far, inbox only broken. Fixing inbox, and implementing the spec will help uncover is the has a larger impact to the chrome specific users.

BT: Chrome should make an effort to ship the standardized semantics; don’t assume the standard is broken.

BT: in the past, IE discovered some issues. We took the approach of fixing all affected sites.

AK: long term this seems ok, trade-offs to make around release schedules etc. Its not a goal to have a compat mode.

MF: is it possible to isolate this scenario more, targeting this exact semantics

DE: in addition to confusion for implementers, that will likely make it hard for users to infer the correct behavior. Complexity is future hostile.

BT: it seems like even with more evidence, we would still have other, independent reasons not to make this change, such as the future is bigger than the past argument.

YK: I am surprised IE + edge this.

DE: do we have evidence this is common or not

MM: this is just strange

MM: future is strict, which doesn't have this problem

WH: That doesn't follow. Repeats point about script authors often believing they're in strict mode when they're not or vice versa, which is an issue if the two modes silently resolve function identifiers to different things.

DE: i don't want sloppy mode to become overly complicated by adding a 3rd case.

BE: once we made our call (assuming we did due deligence) we should risk chasing our tale by endlessly adjusting the spec…

BT: we should be prescriptive. GWT should fix its emit. Inbox should fix its app. Chrome should attempt to ship standard semantics.

### Conclusion/Resolution

No consensus on changing the existing Annex B semantics.

## RegExp simplication semantics (DE)

A few extension points for RegExp subclassing.

- Symbol.{search,replace,match,split}
- get RegExp#unicode/multiline/etc
- RegExp#exec - an easy single override point proposal: fewer extension points.

DE: today, the cost is both for implementators (no one implements correctly), and user extension

…some talk about, why is this costly for the implementation…

WH: I don't buy the implementation costs, but I do support this proposal for user ergonomics. It makes it much clearer to users what they should implement if they want to subclass RegExp and reduces the opportunities for getting a performance surprise if they override just one thing.

BE: it was added late

YK: it was maybe finalized late, but around for years

DE: it was added with a lack of implementation input

JHD: i tried to implement on the weekend, and I dont see added value for all these overriding points.

YK: alan has a motivation, and he is not present

DD: I think I know, (related to promises), out of the box it should be very easy to create a fully functional subclass. There are some inconsistencies, alan said he would tweak these things. His goal was to make this super easy to make subclasses. Dans argument is, a library could handle the “super easy part’ putting less weight on the spec.

BE: lets have symbol names for this, alan tried his best. But without implementor feedback, we it may not have been ideal. Since its not implemented, we have the flexibility to change.

BT: alan is available right now via skype, lets pull him in

WH: The bigger override kernel is helpful with efficiency. With exec overridable, if a user overrides exec then searching must necessarily call exec on each position of the string. With the bigger kernel of overridable methods, searching could use a Boyer-Moore algorithm instead.

BE: should we defer this [the override features of ES2015] or take it out.

BE: we screwed up, it needed more implementor feedback and further iteration.

MM: lets defer this conversation for tomorrow, when Alan can call it successfully.

### Conclusion/Resolution

defer until we can talk to Allen tomorrow

## Remove generator `.return`? (DE)

- overview of `.return`
- pros of `.return`
    - signal to iterators/generators when resource is “dropped on the floor”
    - reify abrupt completions, generators as a sink for observables, etc
- ???
- `.return` in the iteration protocol
- why reconsider `.return` for resources?
    - resource allocation mostly now for async i/o and promises
    - try/finally pattern predominates for freeing resources

?, WH, MM: comments that this pushes the burden to the consumer. Consumer would need to wrap each for-of loop inside a try-finally that explicity releases the resource.

YK: example code in an abstraction?

DE: no, this is imagined direct user code

DE: Generator `return` is idempotent
[Example of speculative auto-disposal syntax: finally let …

WH: That approach (auto-disposal syntax as a replacement for `.return()`) either makes garbage collection visible or fails to work for the iteration use case. If you rely on the finally-let to clean up inside a generator, then either you get to run code when the generator is garbage collected or you don’t get notified when the user breaks out of a for-of loop that invoked the generator.
[discussion about how to do this compatibly in the future if we were to take it out.]

DH: Need to beware of implementations attaching other meanings to `.return()`
DH: Could work around the conflicts by using a symbol in the future.

MM: That doesn’t work. The problem is that an implementation could reify an iterator, run a for-of loop on it partway, break out of the loop, and later continue iterating through it. Such usage would incompatibly break if we were to remove and later re-add a return cleanup mechanism. That was (and still is) the argument why we couldn’t postpone this in ES2015.

BE: I don’t think we could remove `.return()`.
YK: Why is it bad to auto-clean-up?

DE: Cost of try-catch

YK: Expect try-catch to be irreducibly expensive in implementations?

DH: Want sync and async to be as close to each other as is practical.

YK: Want try-finally to always run finally inside a generator if the generator is used in the common case of using it in a for-of loop. Yes, there are other cases where generators are used in other ways, but this invariant should hold at least for the for-of loop.

DH: Generators won't be used for async code (in favor of async functions) and generators will be used primarily for for-of loop.

DE: Make users explicitly clean up after for-of loops.

DH: for-of is the only construct in the language that implicitly creates one of these iterators out of the iterable. That's why it's different. There is no way to get a hold of one of these to explicitly dispose it.

DH: We had and resolved those debates before. We shouldn't be revisiting this and focus on things we didn't already discuss when debating ES2015.

DD: we can seperate iterator and retainment.

MM: combining open a file, and iteration causes the conflation. Preventing a handle to the file from being available.

YK:

DE: if we encourage, generators shouldn't own the resource instead the resource is passed to the generator.

DH: having syntax for disposal is then misleading

BE: I don't think we can walk back from this

BE: We didn't add `.return()` without use cases. Don't discount those. One initial usecase for implicit `return` is yielding from the try.

DH: Don’t want to relitigate the use of try-finally in the language. Disagree with the claim that it’s ok that finally wouldn’t work in generators.

MM: Allow redundant `.return()` calls, which are ignored. This way a user can manually iterate and wrap the iteration inside a try-finally that unconditionally calls `.return()` on the iterable from the finally.

## RegExp Buffet

BT: if we have multiple proposals, I would propose we attempt to advance them together. I am hoping to get an idea from the group, what the initial RegExp proposal should include.

- look behinds
- Named Capture groups
- Comments
- Free-spacing
- mode modifiers
- Unicode Categories, Blocks, and Scripts

BT: look behind, is like look ahead but looks behind… `(?<=pattern)` or `(?<!pattern)`
BT: both positive and negative variants

BT: most potent when considering replacements.

BT: do we follow perl style static lookbehinds, or c# style quantifiers.

BT: dynamic quantifiers \w capture groups can yield to some unexpected gotchas

YK: ruby has a pretty big RegExp buffet, and does not support this feature

BT: does ruby support lookahead?

YK: will investigate… yes

YK: lookbehinds run backwards, which makes it somewhat confusing.

BT: my mind is poisoned, as its to comfortable with implementation details

WH: Note that the fixed-length and variable-length lookbehind variants are incompatible with each other. They'll differ on what gets captured by `(?<=(.){3})`
BT: this feature has been brought up before, MS is working with Nozomu Katō to make this a reality. Is there general interest?

… [positive thoughts]

DE: how do you plan to handle backreferences \w look behinds

WH: What's the problem? This is no harder than in lookaheads.

BT: Numbering of capture groups is the ordering of the opening parentheses, regardless of whether they are in a lookbehind or not.

WH: Strongly support this, in particular the variable length lookbehinds variant of the proposal.

WH: backtracking behavior should behave to match forward captures. ECMA ones do, perl does not.

BT: any other items on look behinds?

BT: stage 0?

… [yup]

BT: FYI: twitter poll indicated, people are upset about this…

BT: Next feature, Named Capture groups `(?<name>)`…
BT: result.matches { name: value, otherName: value} vs merging with result.

WH: Would prefer that named captures capture only using named properties but not also duplicate them under both numbers and names.

YK: May be easier to refactor because adding a name doesn't shift numbers of following captures.

WH: That's nothing new. You do this all the time in existig regexps when inserting a new capture into the middle of the regexp.

WH: Easier to refactor with named captures that do not alias to numeric properties and introduce new numbers. Can insert and delete those without affecting numbering of other captures. That's the same simplicity and advantage that ?: provides.

MM: what can appear in the `<>`
BT: JS identifier

BT: we can discuss tightening

MM: what about numeric identifier?

BT: we must disallow numeric names and `length`; we should probably put the named captures on a .matches property of the result
BT: What syntax to use to backreference to named capture groups?

C#: `\k<name>`, perl: `(?P=name)`
MF: why `\k` over `\<`
WH: \ is an identity escape for punctuation symbols. We had reserved it for letters, but engines were excessively permissive, so \ followed by letters other than n, w, etc., might be used in the wild.

YK: we should likely not move away from existing syntax

YK: looks like `/\k<>/` currently escapes `\k`
BT: this would be a compat issue then

WH: It may be surmountable. We had the analogous compat situation when we first defined \u.

BT: We have significant data on regexp, we can run analysis

SP: it sounds like we should base the choice on your data analysis.

MF: unicode?

DE: unicode regexp might have different performance properties

BT: don't think thats true for us (MS)

M: neither for us (Apple)

DE: oh, ok.

WH: Named backreferences and unicode are orthogonal concepts. Unicode regexps are not a substitute for traditional regexps; both are useful.

MF: would you be ok with back-reference and named not being combined?

WH: that would be unfriendly to users

BT: Named replacement syntax in the replacement strings? `"${name}"` or `"\\g<name>"`
Various: Do it!

BT: tweeted about this, majority of users were sad.

BT: I tweeted an example of destructuring and named capturing groups, its now my most retweeted tweet.

BT: Next feature, Comments & Free-spacing syntax & related semantics:

BT: The difficulty is how to do this without modifying the /regexp/ literal.

BT: can we do this in a backwards compatible way?

MM: Template string tag is the right approach to handle this. Something we can ship. This avoids the parsing problems because template string literals' lexing wouldn't need to change.

MM: lots of issues with parsing related to RegExp, this may keep it from growing in complexity.

DH: we are already in a situation, lexing is already complicated.

WH: Extensions that overcomplicate lexing, which this would do, are dangerous. If the a web page goes through a white-lister that lexes it one way and then run through an engine that lexes it in another way, it can sneak things past the while-lister.

BT: can someone more familiar with the lexing speak to this complexity?

WH: It’s undecidable in Perl. It guesses what you mean.

YK: can you explain

…

SP: it seems like we can explore this further as a templateString, if it really feels poor we can explore grammar based variant.

BT: Comments: `/(?# this is a comment)a*/`
BT: Annoying to use

WH: At least it’s not problematic from a lexing point of view, as long as comments can’t contain newlines, slashes, closing parentheses, or such.

BT: stage 0?

M: lets pick this up tomorrow

BT: preview for tomorrow: - mode modifiers - unicode categories - blocks & scripts

BT previews us some syntax for tomorrow.

# November 18th 2015 Meeting Notes

Jafar Husain (JH), Eric Farriauolo (EF), Caridy Patino (CP), Adam Klein (AK), Michael Ficarra (MF), Peter Jensen (PJ), Domenic Denicola (DD), Jordan Harband (JHD), Chip Morningstar (CM), Brian Terlson (BT), John Neumann (JN), Dave Herman (DH), Brendan Eich (BE), Yehuda Katz (YK), Jeff Morrison (JM), Lee Byron (LB), Daniel Ehrenberg (DE), Lars Hansen (LH), Nagy Hostafa (NH), Michael Saboff (MS), John Buchanan (JB), Stefan Penner (SP), Waldemar Horwat (WH), Mark Miller (MM), Paul Leathers (PL), Georg Neis (GN), Sebastian Markbage (SM)

## RegExp Buffet (BT)

something about composed RegExp

YK: composing regexp does not have a algebraic decomposition

DD: composing multiple interpolated strings for RegExp

WH: What?

YK: composing regexp fragments on the fly

WH: That's only useful for structured composition. Unstructured composition (string concatenation) is too hard to reason about

YK: not wanting this, seems like we are saying creating RegExp on the fly

DH: this is just abstraction and composition, historically RegExp have been poor at this.

WH: Whats YK's point, why is that useful?

YK: imagine a complex RegExp where deriving char classes was complex, splitting this into multiple functions and composing makes sense.

WH: We are talking past each other

WH: two questions: 1. Do we want RegExp templates? 2. Do we want to do it structured, or completely freeform?

WH: I believe the freeform way is not useful or needed

WH: Composition should work via the structured way

YK: I may want to do this via template strings

WH: The structured way already handle this

M: provides example, RegExp for different locale phone numbers. Interpolation would nicely do the trick.

YK: i don't understand how Mike Samuals solution

WH: Provides structured example: `a${foo}*`

- In structured substitution this refers to zero or more foo sub-regexps.
- In unstructured substitution this repeats the last atom inside foo, or possibly the 'a' if foo is empty, or possibly the * is a literal if foo ends with a backslash. Very nasty. Too difficult to compose or reason about.

YK: ok, it seems like these are two worlds.

SP: M's example would be a good example of interpolation

WH: Structured handles this

YK: at what ergomic cost

WH: Example of how to do this is already on the github.

BT: Should we defer free-spacing for further library exploration?

YK: what are people expect from interpolation is not structured.

SP: it seems like we want both, structured and interpolated. It seems like we need to further explore this, before free spacing is explored

WH: We do not want unstructured. If for some bizarre reason you want to do unstructured regexp concatenation, just use + string concatenation and pass the result to the RegExp constructor. Done. We don't need a new and different way of doing that niche case.

BT: yes, we will need to explore

M: what about minifiers

DD: today, minifiers will not touch template strings today

BT: they could become smarter

BT: someone should write this

… istvan's update …

WH: So ISO would make a tiny "pointer standard" that points to whatever the latest ECMA ECMAScript standard is?

IS: Yes.

WH: What would the ISO rules be on the references we can make from ECMA 262?

[confusion; no answer after repeated questions] WH: What would happen to ISO 16262? Would the new ISO pointer spec replace it?

IS: The pointer spec would be a new ISO standard. ISO 16262 would then be withdrawn.

WH: What about the internationalization library (ECMA 402)? Would it have its own ISO pointer spec?

IS: Just one ISO standard would refer to both the latest ECMA 262 and the latest ECMA 402.

WH: How would contributions of ISO members interact with ECMA patent policies?

BT: They'd sign the contributor agreement for any nontrivial contributions.

WH: Are ISO members representing companies? Themselves as individuals? Entire countries?

IS: Countries.

WH: So how does the country of Japan sign an ECMA royalty-free patent agreement?

BT: They'd sign the agreements as individuals or companies per normal process.

JN: It will be hard to get the country representatives to sign anything like an ECMA patent policy.

BT: It would be the individual or company who owns the intellectual property

.....

BT: mode modifiers - syntax & semantics

BT: related to have local case insensitivity.

YK: multiline may be useful, one can imagine several such scenarios … heredoc

BT: pearl regexp, has (?=m…) which limits what can be put inside.

WH: I'd prefer that this be lexically scoped as well.

WH: I'm ok with it for the i and m flags. I'm definitely not ok with it for the x and u flags.

YK: what about U

BT: likely can't do, as it changes the lexer.

DE: what does G mean for a range

YK: Some flags don't work contextually, and because of this should we invent something new? That would seem unfortunate

BT: interpolation + regexp helpers with …

YK: this clears up my composition/algebra question from earlier

DD: this is compelling to me, it enables further composition

WH: Scoping: If mode switches are block scoped, template substitutions work ok. If mode switches take effect until they're turned off, then you get trouble with mode switches leaking out of inner substitutions: `abc${foo}*def` where foo is /xy(?i)z/ would then turn on the i flag for the def in the outer pattern, which is bad.
WH: But the block scoped one wouldn't be an issue.

?: What if foo is /xyz/i ?

WH: That would turn on the i flag for just the xyz and none of the abcdef in the outer pattern. Works as expected.

SP: it seems like the structured composition of regexp should handle this, Sub RegExp get there modifiers they need.

YK: ruby has support for this

BT: ruby has good RegExp

Yk: yes oniguruma is itself a substantial project

…

BT: without this feature it becomes difficult to substitute

WH: A more fun question is what if the template is `abc${foo}*def` (without a g modifier) and foo is /xyz/g ? Such a flag juxtaposition would be meaningless.
MF: we only want I M X

YK: this is what ruby does

M:..

BT: It sounds like we want this, it may also help us figure out interpolation/composition

WH: Definitely don't want switchable x flags. Can construct all kinds of lexical trouble with it.

BT: composing with X is unknown, we may need to defer it.

BT: mode modifier makes it easier to compose

WH: so only the M and I flags for now

BT: yes

YK: sounds like an open question if we work on X

# Unicode++ - Syntax & Semantics (BT)

BT: unicode spec defines, many things (block scripts…)

BT: an example of a block is latin/arabic etc.

BT: \u allows for more ergonomic RegExp when dealing with unicode chars..

YK: unicode adding stuff, will cause enumerated charsets in RegExp to break

BT: when naming a block, perl allows InArabic and IsArabic, C# does IsGreek, C# seems better

MF: blocks and scripts can conflict

YK: ruby has is “arabic”

YK: ruby \p === script \P is negated

DE: its possible ruby doesn’t support blocks

YK: its possible

BT: implementation concern, loading block/scripts may cause excess memory pressure.

BT: i dont have a sense for how much.

SP: is the memory pressure fatal, is it impossible to pay it pay as you go?

BT: we would love to implement it that way, but that is work that must be done.

BT: I don’t think this is fatal, but we should consider it

WH: how many?

BT: ~20 scripts, ~30 blocks, ~60Catogories

MF: could we use FooScript, instead if IsFoo in InFoo

BT: there is both a arabic script and arabic block

DE: yes

BT: we should avoid was C# does

DE: unicode would in theory prevent a naming collision here (script/arabic)

YK: … ruby only has scripts

SP: can we clarify category

DD: editorial information

BT: category data is in all the unicode data tables

DE: sounds like a quirk of the written spec, likely “Weak language”

BT: should we choose script vs blocks

BT: blocks contain slots of future usage

BT: script has no future slots

DE: its cheaper to check if a char is in a block, then in a script

YK: we should likely investigate

DD: deciding factor is, will implementors carry the burdon

BT: so you would like to have both, assuming implementers accept the burdon

DD: yes

DE: the Intl Object already brings this along. Can we take that into account?

YK: what is the usability with only blocks

BT: You can’t write a RegExp in a future proof way

YK: are there strings that contain things in them, and now they cannot be matched against

BT: if my design is to match only arabic characters, without blocks is tricky

BT: e.g. does a user, writing a unicode aware RegExp do we want future characters to be taken into account

YK: in what case would this be highly important

DE: what doesC# do

BT: it has arabic === script, inArabic == block

DE: this seems most appropriate

DE: expose expert feature of blocks, but encourage scripts

DD: there are things in scripts that aren’t in blocks

DD: seems to be missing symbols

BT: category + script should cover this

AK: it sounds like for stage 0, we should defer to C#, and gain more context over time,

DD: We should consider being a subset of C#

YK: It is not obvious that the C# choice appropriate for us, our startup constraints are pretty high.

BT: In reality, there should not be startup cost (for chakra), unless such a RegExp is used.

BT: the first time we load Intl, it startup time takes a hit

YK: I am happy with stage 0, i think there are design choices that are not just implementation

WH: Given how few scripts there are and given that a script is mostly a small number of consecutive ranges, estimate that script tables are a few kilobytes. Doesn't seem like a lot to obsess over trying to optimize.

WH: A different issue not raised yet: Some unicode characters have the "inherited" script setting, which means that they're chameleons: their script is inherited from the script of nearby characters in whatever string they're embedded in. How would regular expressions deal with those?

DE: from combing marks

YK: seems reasonable

BT: how will that handle the

… missed some stuff

BT: how do other RegExp engines handle this

MF: should this be allowed inside char classes

BT: I would like to, it seems handy

MK: small syntax suggestion \p matches a p according to spec. Implementations agree on that, we are not aware of usage… \u does not have this, can we change \p to \u.

DD: it does mean literal u in non-unicode RegExp

… music from next door disrupts flow …

BT: precedence wins, unless \p is not compatible with the web.

YK: both are fine, \u is extremely nice. \u flag already means unicode

BT: so you hope for a compat issue

YK: confirm

Wasn't there another proposal is `\uUNICODE_CODE`
WH: Are the \p category or block names case-sensitive?

BT: Yes.

WH: What about multiword script names? Spaces between words?

BT: No

YK: in general RegExp has lots of divergent, would people expect foriegn RegExp work

BT: C# has a Ecma mode

DD: we should be careful to not be trolled

AK: stage 0 likely doesn't need this level of detail yet

MF: we should make sure its possible

BT: I would like to write this up, so more context is better

WH: We are talking about the substance.

WH: The place this discussion loses value is when we start debating things in the abstract, such as should we be doing the same thing as Python/Ruby, etc., without that debate being informed by what those languages are actually doing, their advantages/disadvantages/lessons, etc.

BT: ok, i guess in-order to break the deadlock. It would be useful regardless, what the compat story with escapes is.

BT: im going to proceed with \p, and see how the compat story shakes out

BT: this does not mean we can't change

BT: we have much data, that can be analized.

BT: we should do the Other RegExp item, because Alan will call in. With slides

… stepped out … ambient music from next door

# AWB: Summarizes some OO Concepts

BT: we accept the shared vocabulary

AWB: RegExp has an abstract base and a concerete base

AWB: the public interface that it exposes, exec/split etc… essentially all the methods on the interface. No methods to provide a subclass interface, it is important \w match & replace that any class that provides those methods be a subclass of a RegExp Object. The String object was restructed in es6, so it works with any object with that implement. RegExp has a subclass interface, and it is exec. This is very intentional, 1 kernel method is required and a subclass works. By implementing this one concrete method, the abstract algos will work.

AWB: exec shows up in many places, public interface, key kernel method for subclass interface, and depends on the internal matcher algorithim.

WH: How does an implementation, that does boyer moore searching work in this case.

AWB: Let me rephrase, what if a subclass (or baseclass) wants to change or refine its search algorithim

WH: or what if the built-in wants to change/mature the algorithim

AWB: Whatever algorithim an implementation wants to use, the algo must confirm to the observable behavior defined in the spec.

WH: the spec may be over constraining this, which is what we are looking at

AWB: we should then look at the search algorithim and see where that is

AWB: the actual algo defined, are essentially the same algo in ES5 spec

DE: While that true, before ES6 I believe it was not observable to skip indexes. Unfortunately, now it is observable.

AK: I believe there are some changes we could make, that would loosen this, and likely improve some performance. There is a bug in the spec that causes multiple lookups of exec in a loop; those should be factored outside the loop.

YK: Can I ask a question about observability

YK: Are your worried that subclass should be able to participate in boyer moor algo

DE: Lets here AWB full summary

AWB: Extension strategy, someone could implement a total replacement of the public interface using whatever algo they want. As long as the public interface is implemented correctly, it should work. This is possible, but likely more work then its worth. What is likely common, is a subclass that provides some minor extensions, and defers the vast complexity to the ancetors. For example, an exec method that logs. Or a scheme for memoization Relatively simple things, some construction time modifications. Etc… Which all should utilize the built-in matcher algo, calling super to exec. All the abstract algorithims should work correctly, and all the abstract algorithims and concrete interfaces should work.

AWB: Another subclass, a more ambitious one which extends the built in algorithim. LIkely requiring exec and constructor overriding. You may need to override some of the other public interface.

AWB: These where extension styles that we took into account when designed, any questions?

WH: lets go back to the boyer moor algo example, if an implementation wants to use boyer moor it would be observable

AWB: in earlier additions of ES, the algos where concrete(not abstract) and they called specific internal algorithims. In ES6, this was exposed as exec. In the past, you have exploited the knowledge to accomplish an optimization.

AWB: Is that right?

WH: yes, my comment is exec is too small of a kernal

AWB: You can still do that. (explains an possible solution)

WH: You will end up falling off a performance cliff, when following the second extension strategy on your slides

AWB: Thats fine, i don’t expect such subclasses to have the same performance characteristics. If they want the performance, they can re-implement

DE: that sounds like a big task for a subclass

AWB: that is unfortunate, that subclasses cannot benifit from existing platform optimizations

DE: to give more context. JHD is working on the ES6 shim and ran into some issues

DD: multi inheritance

YK: decorators etc..

DE: RegExp > RegExpShim > UserLandRegExp subclass

AWB: let me move, on i speak to something related

AWB: I had heard there was misunderstanding of the original design. I want to be sure we have a common design

AWB: The boyer moor example doesn't invalidate my current thinking.

AWB: Lets talk about what is an extension point, search/replace/match/split i dont consider those extension points. They are just methods that *may* have alt implementatinos, they are just refinements of the kernel

AWB: I suspect no-one is expecting those.

AWB: @@ infront of the name, is abit distracting to me they are just public interfaces.

DE: should i explain why i wrote the slide that way

AWB: yes

DE: another way the spec could habe been written

AWB: it wasn't written another way

DE: well, another can currently implement the. `String.prototype.replace` could have provided this.
AWB: i intentionally did not follow that design path, to leave open the design to allow subclasses to extend and optimize (how we spoke about here)

AWB: if the entire abstract algorithim is in strings, we have coupled it to the class hierarchy rather then the interace. Which is just bad design

DE: i see you point

AWB: bug but easy to fix, the internal slots that are used to store the flag are part of the concrete built-in implementations The actual matcher algo, is only intended to access the internal slots. Part of what is going on here, in previous spec, these flags where readOnly nonconfigurable own instance property. Unfortunately, the annex B compile method specifying them the way it did violated this. As compile could change them…

AWB: we had to add the accessor methods, to RegExp to explain the observable semantics across compile calls. Another way to look at it, was accessors are part of the public interface… I believe i classified them as part of the public interface. So those can be overriden by subclasses for there own purposes. When you get down to matches they should use the internal slot. So simple bug, we can fix that.

DE: if we where to make a tweak with flags, `RegExp.prototype[@@split]` uses the flags accessor, instead of looking at individual flags, which is not great…
AWB: we can look at that later

AWB: when I look a the code, i can see the source of the the bug, The ES5 code did a get, and it wasn't changed. So faulty refactoring. Not a big deal.

AWB: calling RegExpBuiltinExec directly would require too much work from subclassers

DE: I see your point

AWB: regardless of what you did, your saying that the methodof the regex is not extensible. You might be able to provide an alternative implementation that is more extensive, but the object we are used to is not.

WH: I withdraw my concerns because the @@ algorithms are basically thin wrappers over exec. Modifying exec is sufficient to do Boyer-Moore.

DE: there are additional performance considerations

AWB: I know a better way, we can talk about it.

AWB: it is very important, we didn't intend to make change to the existing algorithims. My assertion is with some relatively simple guards, you can't continue to use the existing implementations. The guards should be similar or the same to existing guards. e.g. is the prototype a built-in or subclass, these seems reasonable. If there are unintentional spec changes that prevent this, we should correct.

DE: you could imagine, users mutating the builtin RegExp (Adding props etc), for these use-cases more detailed guards are required. Cross platform guards may differ, making it hard for users to get good performance

BE: DE is saying, there "may" be issues, i want to know what concrete is hard for V8 to do. I would like to hear from other implementors aswell

AWB: exactly my question, it doesn't sounds dissimilar to other tricks

DE: we have tricks similar to this in Arrays, we could potentially do them here, or something similar to that.

DE: this wouldn't be an absolute blocker, it wouldn't make it extremely bad

BE: what about apple

M: ya

DE: concerns with overall system complexity

M: another engineer has implemented this in JSC, he had to put some checks in to see or not if these are overriden. 1 guard upfront. The above flag issue would be good. He couldn't find any big issues, other then extremely targetted benchmarks (single character matches etc) where it was demonstratably slower.

BE: any chakra experience

BT: I spoke to [a developer] about it. We were concerned that the exec function may make us recompile patterns in some cases. It could be addressed by making a new kernel method exec (maybe symbol) that takes lastIndex and flags as a param. split has to make a copy of the regexp to work around that.

BT: we haven't really dug in yet.

AWB: when looking at it from kernal method extensiblity. It didn't really add any additional flexibilty. It seemed to add more complexity and an additional level of indirection.

AWB: I am still inclined to prefer the latest design.

JHD: doing what you suggested (lastIndex related global), would help the matchAll proposal.

…

AWB: (couldn't understand)

DE: I wanted to say something else, although others can v8 will likely have to rely on the more brittle guard.

AWB: (couldn't understand)

DE: it would be nice to cleanup.

further changes on slide deck

AWB: my bias is that, this complexity should be absorbed by the runtime. It shouldn't be pushed to ES consumers

AWB: I don't know the details of your framework. This ES6 feature has been described using OO best practices, the runtimes should be able to reasonably implement.

YK: I wanted to provide some histroical context from ruby. I also believe Smalltalk is similar, subclassing built-ins. Most people in ruby feel this was a mistake, kernel methods would have been more handy.

YK: For example Rails re-implements Hash to support string/symbol interopt, this ended up with 200 loc etc.

YK: it is true, userland can do the work, but system + userland get out of sync, and it is unfortunate

YK: rubinious even added a hook hash.store to deal with the issue.

AWB: the original small talk implementations were the result of people not yet being aware of these problems. Lots of coupling

AWB: …there is a right way, and a wrong way.

AWB: my conclusion, I don't think we need a re-design. There are some small fixes and tweaks.

JHD: Symbol.exec, i would gladly work on it with you AWB

BE: A functionally pure kernel would be great.

AWB: one reason i didn't go that way, is due to some hesitation do to adding additional @@ methods.

## Conclusion/Resolution

- no major change
- DE/AWB will make minor changes + tweaks
- AWB/JHD to investigate Symbol.exec + a pure functional exec kernel

DE: TypedArray proxy issue (from Andreas Rossberg)

AWB: What if we just made an internal algorithm to do the construction, and call it from each TypedArray constructor directly, rather than having a super constructor and proto chain walk?

DE, AK, BT, MM: Perfect!

Resolution: Rephrase spec, a pull request for this is welcome

AK: while we have you, im curious re: typedArrays and stuff that changed during ES6

AK: imagine Reflect.constructor on a built-in like number, passing your own new target. There is this problem, that some of the built-in constructors have side-affects before they pull the prototype off the new topic. This yields to some wierdness

AK: Will file a separate issue for this concern

BT: other issues for AWB while we have him?

KS:

### Conclusion/Resolution

- %TypedArray% constructor is directly called rather than having a super constructor and a proto chain walk

## Improving consistency of @@species (Kevin Smith)

KS: on map and set, they are not used in the spec themselves, But subclasses should be able to override

KS: usage patterns, instance methods, that want to return related subclass.

KS: Promise.all and Promise.race are the only ones that use this

YK: race/all are combinators Array.from is a custom constructors:

KS: it would be good to define what the usage patterns are

AWB: ..

DD: I disagree, resolve/all/race is more “casting” and species should not be used.

AWB: I agree the return value should be using species, as the user is specifying.

DD: the argument values can be anything, and they are to be cast to the RacePromise

AWB: but why

DD: an example …Could be… CancellablePromise.race(arrayOfMiscPromises) likely wants cast its input.

YK: those examples are dubious

SP: lets us InstrumentedPromise as an example

DD: ok lets use that one, InstrumentedPromise.all(mixOfPromises) casts all inputs to InstrumentPromises

DD: auto-casting is the intent.

AWB: that works great for the case, where you want to cast

AWB: but it breaks the case where you don't.

DD: I believe it was a mistake, likely my own mis-understanding. I don't believe it makes sense.

DE: zepto took a literal array that and swaped out its constructor. If that was new'd it wouldn't work. `Array['@@species']`create, has two checks. this.constructor if thats not an object, it goes back to the default `ArrayCreate['@@species']` of fallback. The critical original issues (based on the notes) was WebCompat
YK: @@species has another precednt

DD: A solution was needed, this pattern appealed to others. I believe it was misused, and misunderstood.

DD: it expressed a use-case for arrays, that are hypothetically use-full.

AWB: avoid nodelist instead get an array out of it.

KS: I would prefer, i would not want to argue removing it. I would like understand how it ought to be used.

YK: why is it on map/set

AWB: someone argued it should be for the future, for example adding filter/map

AWB: if we wanted to provide filter to math, a subclass to math. This would allow them to change species

YK: we should add it on-demand.

DD: we should move away from @@species on the promise constructor

DD: we should discover a future strategy. `Function.prototype['@@species'] = this`
AWB: not sure why we didn't think about this sooner

DH: whats the best next step for a change like that

YK/DD: PR

DH: should I file a bug if i don't have time

DD/KS: bug

YK: should we remove it from map

DD: lets remove it from everything, and put it on `Function.prototype['@@species'] = this;`
AWB: it was added for array instance extensibility.

DD: that would break zepto

DE: `Object.__proto__` is `Function.prototype`, which have a species returning this, which breaks the web.
KS: it seems like the cascade of missing species will break things.

DE: we could erase species create, is the species of the constructor object, if it is fallback. Then species does what I wants to do.

YK: I believe `Function.prototype` behind right about `Object.__proto__` is unfortunate.
DD: i see two paths forward,

1. investigate `Function.prototype['@@species']`
2. Map and Set shouldn't have it, we should add it on-demand.

DD: we can work out the details offline

AWB: I know from email threads, we should factor in the DOM design.

DD: many scenarios to flesh out.

AWB: API level support would be interesting

everyone: himm?

AWB: for example, filter some collection, give the power to the caller.

DD: shifting the burden on the user seems unfortunate.

YK: filter species feels good, map feels back. mapping a nodelist, getting back a nodelist is funky

DD: ya for this DOM api, it doesn't seem good

DD: it feels good for like, OrderedCollection from small talk

DE: this pattern works in several scenarios

BT: summarize?

DD: i see two paths forward, 1. investigate `Function.prototype['@@species']` 2. Map and Set shouldn't have it, we should add it on-demand, and do no carry it forward.

## Conclusion/Resolution

- remove `@@species` handing from `Promise.all`/`Promise.race`
- no decisions made on other `@@species` handling
- investigate `Function.prototype[Symbol.species]`

BT: discussion on various GH threads, this should be easy.

BT: the problem is that, proxy enumerate trap iterator is allowed to return w/e it wants.

YK: symbols are can be enumerable but not as for in key

BT: currently it can return anything

BT: first proposal 1. error for non string return 2. coerce to string

BT: proposes errata thrown error if non-strings are returned

MM: is there a precedent for this?

BT: Other places where invariants are violated, we throw

DD: can non descriptors be returned from the descriptor

MM: no, we fixed that.

DD: promise rejection handlers

…

MM: there is another defactor standard for JS (both node and browser) and sort fits in the same general area of purpose is console

MM: console is interesting, the writing to console is exposed to JS, but the visibility is platform specific.

MM: the kind of diagnostics seem to related

DD: this is also meant as a way for programmers to report back to the server (for example)

MM: the console can show the traceback

MM: this feature is used as a programmer to find bugs, it is diagnostic. And should be gather only by code that has authority.

DD: a good debugging/diagnostics proposal sounds reasonable in the future

BT: improving the layering of HTML + ecma, to improve the boundaries are (internal slots, and abstract algorithims)

BT: how much scrutiney should this sort of work get?

DD: these sorts of changes already exist

MM: I would prefer such things brought to the group.

YK: it seems like we need some healthy interop between the various groups, so if webcrypto wants to add some concepts. It should be part of some inter group discussion

MM: should it go through the group before it can advance outside

MM: clearly the web will advance without

MM: being notified early is important

YK: I agree

BT: should likely be an editor discretion. With github, changes are public and watchable.

YK: a small process may solve, for example. The editor of a given group tags changes as potentially relevant.

BT: bi-weekly changelog, with highlights called out. Grouped sections, to draw attention to various interested parties.

### Conclusion/Resolution

- promise rejection hooks are in
- editor may apply own discretion on further “implementation hook” proposals

## Proposal Repos, and where they live + editor update

BT: for people not watching the spec, let me show you what we have.

BT: we have a permanently bleeding edge spec at http://tc39.github.io/ecma262/

BT: more people using it the more bugs we can fix

BT: we have a nice fuzzy searching table of contents

BT: find all references (when clicking on various identifiers)

BT: …

SP: We can tweet every release

DD: who controls @tc39

JHD: I believe i gave that to DH

DH: I believe i have it

MM: what happened to the wiki, we should get it back

YK: finding historical references etc.

DH: I will try to convince mozilla ops to get it back up

BT: I tried once

DH: I will ask them

…

BT: I will provide the bi-weekly callouts, but wont come to each meeting with the delta

YK: It is reasonable for others to keep up to date online, and be prepared for the next release

BT: it would be nice if everything under the tc39 org is going to be archived in some way that ecma likes

BT: stage 1 approval (entry critera) that the repo be on the ecma

BT: but stage 0 is lost

SP: we can move the repo

BT: great, but moving the GH pages redirection seems to fail post move

JHD: creating a new repo on the old location allows a manual gh-pages redirect, but breaks the automatic repo redirect, so please don't do that

YK: I will ask if there is a good reason for that.

BT: many have stage 1+ that are not on tc39 org yet

BT: it seems like there are several steps, may be labourious.

BT: lets talk to istvan and see if we can give everyone owners

BT: stage 1+ email me, and we will work on the repo transfer.

SP: editor doing this, acts as a good filter.

DE: Proxy Implementation for in issues

### Conclusion/Resolution

- any stage 1 or above proposal repos must be transferred to the TC39 org, as a stage 1 entry requirement

## JEFMO: trailing commas

JM: stage 3?

AK: does this cause problems with arrow functions

BT: it may increase the complexity of parsing

BT: i believe the spider monkey folks had thoughts, but I don't know why

DH: does you spec include handling of sequence expression grammar

JM: no

DH: then it doesn't seem good

JM: spec doesn't have trailing commas in arrow function

…

JM: ok I will add arrows (pending stage 3)

DD: stage 3 tomorrow?

JM: yes

### Conclusion/Resolution

- hopefully stage 3 tomorrow after arrows are added

## Proxy [[Enumerate]] ocerconstrains implementations (AK)

https://github.com/tc39/ecma262/issues/161

AK: proxies have an enumerate trap, the worry is (from us implementors) must call next at specific times. Which causes some concerns, it seems like something is underspecified…

DH:

AK: we want to leave it up to the implementation to eagerly fetch the keys, regardless of proxy

BT: To summarize: if i am enumerating a proxy, we cannot pre-collect the keys because the call to the next is observable.

BT: spreading before the loop, may have issues

JHD: proxies with infinite enumeration wouldn't work then?

JHD: something like an iterator that iterators for 5 minutes and stops. (laziness)

BT: …

YK: Can we move from loosening, to changing to the usage to what the implementations want.

YK: for example, changing the spec ahead of time

YK: proxies can observe, IE has some behavior, people get used to it. If collect seems like the right thing, we should move that way.

DE: lots of cross platform differences are already observable here, because for in is under specified.

BT: YK I believe that will be safe from a compat standpoint

BT: I would be surprised if the enumerate trap is being used

YK: i believe a delegating exotic will want it.

BT: only the for in code

DD: for app code, unlikely

YK: agree

YK: its the copying protocol

BT: it seems unlikely that is is an issue, we can safely make the change

BT: If not, maybe we can leave it under specified?

YK: I feel MM should care we shouldn't under specify

MM: we should have deterministic specs, remember our target audience is many web programmers for many websites. Reproducible behavior is important for this environment

### Conclusion/Resolution

- specify that [[Enumerate]] spreads before entering the loop https://github.com/tc39/ecma262/issues/161#issuecomment-157910543
- the committee would not agree to underspecified behavior
- there is a compat risk for Chakra but the assumption is that it's not a problem until there's data saying so

## Function.sent (BT)

BT: its in babel

DE: did you add internal slots

### Conclusion/Resolution

deferred till tomorrow…

## Async Await

BT: I did not finish the tests, but noticed some troubling things. IE ships AsyncFunction, I believe we should actually not do this

SP: im curious why

BT: unsure, MM?

MM: these is no reason to give it a global name

DD: TC39 believes the global isn't a mess already

MM: well, because of this, we should take extra care. We should not contribute to the problem

MM: we discussed that modules will be a mechanism for us to prevent additional polution.

YK: yes its a risk, modules are a way out of this.

MM: GeneratorFunction is not shipped, SES also wants to splice it out. Lets stay with the precedent

MM: ES5 added a new global called JSON, this causes grief. Facebook had such a global,

DD: Ok

BT: does the fact that Edge not ship `%AsyncFunction.prototype%` at all
(that `Object.getPrototypeOf(async function () {}) === Function.prototype` and should not) mean we need to wait on stage 4?
(lots of discussion about whether global topology should block stage 4)

…

MM: for in order is different we might revert

AK: is this demonstrating a problem with that staging process

YK: yes

AK: im talking on behave a process

YK: there is a well known process, feature flags. They have a cost, but the benefit improves the integration process

MM: what is it

YK: isolate chunks of code (markup) that is isolated, interim work can take advantage of this.

AK: the spec is too big

YK: what will happen in practice, if AsyncFunction is present. Related work will be able to take into account. There is cost associated for sure.

MM: there is an existing cost today, FF has an additional cost.

AK: …

BT: I asked the question, because I want to decide how to allocate my time tonight. I didn't want to allocate the time, if the already presented issues blocked anyways.

AK: I'll take your word the "cost" is trivial

BT: As an implementor, I need to frontload the work that the group feels appropriate

DE: it would be optimal to have high quality tests, if there are issues / failing tests we can and judge the risk associated.

… deciding core semantics … BT: 95% confidence interval on "core concepts" or tests related to non-trivial changes. Or issues unrelated to performance/stability.

MM: populate visible primordials must be populated

BT: this is a hard conversion to have

BT: i will report back after tonights

DE: various contextual s keywords in edge cases, may have unforeseen complexity. I would like to have this considered a core semantics.

SP: risk of shipping, with bugs vs risk of lacking feedback from shipping

BT: its a risk forsure

AK: That is a good reason why multiple implementations are good, as it they will hope to have overlapping bugs

## Conclusion/Resolution

- do not add GeneratorFunction or AsyncFunction constructors to the global object

# November 19th 2015 Meeting Notes

Jafar Husain (JH), Eric Farriauolo (EF), Caridy Patino (CP), Michael Ficarra (MF), Peter Jensen (PJ), Domenic Denicola (DD), Jordan Harband (JHD), Chip Morningstar (CM), Brian Terlson (BT), John Neumann (JN), Dave Herman (DH), Brendan Eich (BE), Yehuda Katz (YK), Jeff Morrison (JM), Lee Byron (LB), Daniel Ehrenberg (DE), Lars Hansen (LH), Nagy Hostafa (NH), Michael Saboff (MS), John Buchanan (JB), Stefan Penner (SP), Sebastian McKenzie (SMK), Waldemar Horwat (WH), Mark Miller (MM), Paul Leathers (PL), Georg Neis (GN), Sebastian Markbage (SM), Zibi Braniecki (ZB)

YK: i wont be giving my full presentation this time, I will provide a short update.

YK: i plan to work with somepeople like dan, and evolve it

YK: There was some concerns with adding free floating APIs for decorators, what we are going with (right now) is a mirror that decorators have access to, during the time that they are run.

YK: this based on input from several different parties, type checkers implementors etc.

DE: KS on private state

YK: KS has been working on, private methods/functions

## Observables (JH)

JH: proposals are at a reasonable mature state now, lots of iteration and evolution.

BT hand delivers coffee to JH

JH: current state of the proposal, is largely unchained since last time

JH: we have moved away from the generator interface, partly do to ergonomics

JH: return /complete confusion

JH: composing generators + observables leads to some issues, so we took an adaptive path

JH: a similar APi, with some changes

JH: sync subscribe and sync unsubscribe, to prevent

DD: Symbol.observable that does return this, caught me off

YK: is the start method related to priming, or is unrelated

JH: unrelated, start method drives the subscription

JH: DD

DD: there is no more Symbol.observe

JH: oh, thank you

WH: What are the question marks for after some of the methods in the Observer interface?

JH: Those are optional

WH: But the ones without question marks are optional too.

JH: [Removes question marks from slide]

WH: After an Observer receives an error or complete call, can it receive any other call?

JH: No

WH: What can happen while Observable.subscribe is running? Can the observation run and complete?

JH: Yes

MF: doesn’t a subscription unsubscribe need to take some…

DD: no it doesn’t atleast not in the readonme

MF: what if it wasn’t successful

JH: we will talk about that shortly

JH: …

JH: reviews existing behavior (if anyone caught that, jump in)

…

WH: What is Observable.forEach?

DD: what is forEach, it is an attempt to make it ergonomimc

SP: is forEach being async, a hazard

DD: I think, this is a change we will be seeing more.

DD: an async taxonomy would be unfortunate

DH: same name, doesn't mean its the same interface.

SP: so this is the start

YK: start with promises, API evolves. This becomes async loops, which is natural.

DD: its not really clear cut

DD: one thing that catches me, where is the second thisArg to forEach.

YK: its bad in async siutations, thisArg entangles the lifetime

DD: other constructs do already

… it returns a promise

JH: yes

WH: Is it possible for a next to be called re-entrantly (i.e. an Observer receive a next call from within a next callback)?

JH: We don't want to allow that, but I don't believe the spec currently guards against this

JH: wk you brought this up yesterday with kevin

YK: yes, a loop re-entering during iteration, seems fatal

YK: in the middle of a block, jumping to the top of the block is unexpected

DD: yes in a for loop, run to completeion invariant should not be broken

YK: if the observable has no buffering, this will just happen

YK goes to write out an example.

YK: i'll assume some async for of syntax.

```
let { producer, consumer } = …;

async for  (let item of consumer) {
    producer.next(value)
}
```

YK: there is a producer side "call next" on the consumer you subscribe and receive values. This is a generic statement of async loop constructs.

YK: in the observable model, consumer is an observable. Producer is an observer.

JH provides some quick context for some confused about producer/consumer.

YK: when someone calls next on the producer, without buffering the consumer then producers, and the loop is re-entered.

MM: is the producer something that calls next? Or something that gets next get called one. Lets be careful to prevent confusion

YK: there is code that calls next, and code that recieves next.

MM: is the object

YK: lets reframe

```
async for  (let item of observable) {
    observable.next(value)
}
```

…

JH: this is the classic re-entering problem, buffering is one possible solution

YK: the problem now, is the observer has no buffer, next is sync. So calling next in the loop will either be dropped, or re-entrant

MM: this is exactly synchronous plan interference problems

YK: this is likely worse

JH: if we had syntax, we would need to schedule

WH: What do you mean by the synchronous plan interference problem?

MM: See chapter 13 of http://erights.org/talks/thesis/markm-thesis.pdf for explanation of plan interference

…

MM: any sync notification will have the plan interference issue

YK: this is more specific issue

MM: it is up to the observable, how it deals with the re-entrancy problem.

YK: unfortunately, it does not realize it is mid for of.

DH: one obvious know case, is making this an error condition.

YK: it could also be buffered

DD: I think making this an AsyncIterable solves the issue

MM: i believe it is up to the observable to make this choice

DH: doesn't there have to be an API for that then?

YK: you are correct, that is a way the observable can make the differences

YK: …

WH: going too fast to track multiple people, lets slow down.

MM: i don’t believe there is a problem the language needs to solve. It is the responsibility of the observable to deal with this case.

YK: what happens if the notification does re-enter.

JH: observables should not be re-entrent, we should schedule and the problem goes away

YK: is this better solved by AsyncIterable

DD: no-one has proposed this syntax

WH: DD, what you’re saying is that this should not be used with async for?

DE: there is a larger issue, it seems that this is an interlocking issue. We must consider these units has part of a single package.

YK: DE, i agree with you

YK: MM, i think you can agree there is design needed here

MM: when you changed the syntax, I realized i had a confusion. When this is async, the problem goes away

SP: yes, then the run to completion semantics we expect remain

YK: i think DE was write, we likely want to add syntax in the future. If we get these primitives wrong, we may block that.

DD: sorry, i mistated before. I agree

DE: I agree

DE: lets have JH continue, and we can continue

JH: look, we want to look at how this works with AsyncIterator

JH: i don’t want to propose syntax for this future (now or ever)

JH: there are issues, there is no natural backpressure. Unsubscribe is the only option, but Unsub and and Sub are no the same as pause.

JH: previous iterations of the proposal had some support for this.

JH: we adjusted, because Observable feels like a much better event Target then we have today.

WH: what happes in this example?

```
async for (let item of mousemoves) {
    await somePromise;
    mousemoves:next(new MouseMove());
}
```

JH: It's bad.

WH: I can see that it's bad. But which particular bad thing happens if someone does this?

AsyncIterable has a natural way to handle this, as they are more "pull" based. Observable can't really do this.

JH: I see Observable as a better event target, no syntax and this problem isn't an issue.

YK: should this be a DOM proposal

DD: it is unclear, it isn't unclear

DH: a test, does it make sense in node?

DD: promises are needed for modules

YK: promises for modules tiped it this way.

DD: it is ok if this is the venue.

WH: So Observable should not be usable in an async for loop?

JH: Should use this instead:

```
    async function() {
        await observable.forEach(x => {console.log(x);});
    }
```

SP: forEach has the same issue as syntax for of, it is re-entrant.

JH: there are solutions, buffers.

WH: What happens if someone ignores our advice and uses Obserable in an async for loop?

JH: It wouldn't work.

WH: What about it wouldn't work? What would be different about the interfaces that would make them not fit?

JH: example code:

```
interface AsyncIterator {
    next(): Promise<IterationResult>
    throw(): Promise<IterationResult>
    return(): Promise<IterationResult>
}
```

The Promises prevent reentrancy.

WH: Good. That explains it.

…

DH: You're saying, Observable needs to be synchronous to meet most general needs, but then that causes the re-entrancy issue, inherment to synchrnous callback mechanisms. So it won't be connected to `for await`. However, it'd be possible for individual Observables to be connected to async iterators which are connected to syntax.
… [Discussion of many generic combinators to convert Observables to async iterators]

YK: Stef raised an issue with forEach EIC protocal, blessed the behaviors.

YK: there is code in the wild that does the

DE: it would be great to capture this in the spec

DH: we have traditionally avoided rationale

JH: other methods aswell? map/filter/etc

YK: no, EIC is focused on forEach (i believe)

YK: I am also disappointed in the abstract, but we actually did this.

DD: this means we cant used forEach on iterables

YK: give me a compelling reason

DD: this wont break

YK: this will break

SP: This could retain object graphs, cause different DOM code to be executed

DD: Some npm packages do use foreach in an async way already. You're only robust against types that conform to the protocol that you're implementing

YK: Using a library together with something could cause it to break

DD: Making types into observables could be a bad interaction

YK: Let's think more carefully about it

YK: we may want to consider abondoning the EIC protocal

YK: Every collection has a forEach method, which has the same signature that it gets, so that you can write code which is generic over multiple forEach implementors

WH: Define EIC.

DD: forEach((element, index, collection) => {})

DD: Why should asynchronicity be considered part of this? The index and collection can be passed in too.

CM, JM: Why not just use a different name?

DD: It would be bad for generic code if we had to use a different name.

JH: Generic code should use the iteration protocol, not .forEach, when feature-testing and doing synchronous iteration

DD: We're making a gradual transition towards iteration, so we should be OK breaking the uniformity of forEach. There's a small window for code to use generic forEach and expect it to be synchronous.

YK: How could we help figure this out?

DD: We could market it somehow and get people to switch to the iteration protocol

YK: I think Domenic's transitional story is good

JH: So, should we advance to stage 2?

YK: Dan had an objection that I agree to. We are nervous about advancing before the syntax is worked out

DD: We want to have a full story with async iterators and have a single story to present to the web before pushing this forward

YK: And maybe things won't fit together if we do it piecewise

WH: I share that concern

JH: I think this is well-thought and we do have this all figured out. Kevin has some good drafts

DD: two seperate proposal is ok, but it seems like they should be combined, advancing together as an async plural proposal

YK: async loops

DD: async plural includes the broader idea.

YK: we should be sure the picture looks cohesive

BT: what about promises + async Function

YK: they were presented much earlier

DE: the process has changed

YK: expressions are simpler

BT: likely true, im not saying we shouldn't advance them as a group

YK: DD JH and I have been working on this for some time, and it is complex. We feel without maping it out, there may be problems

YK: Several champion groups, should coalesce into 1

JH: who is working on these people

DE: KS is working on it, and it is making progress

JH: I am concered coupling this proposal

JH: so we are looking at 6 to 8 months?

YK: we haven't..

BT: We don't need async Iterable to be at stage 2, to advance observables…

DD: we are saying, they should be one proposal

WH: I want to be able to convince myself that the two will play nice together. I don't care as much how we go about doing that.

JH: i can provide an adaptation

YK: the devil is in the defaults

MF: prior art to refer to

JH: .net etc. has these two separate protocols that work well together

YK: devil is in the details, JS !== C#

DD: I believe they feel as a package, and should be presented together.

YK: coalesce into 1 champion group, members must be convinced.

DE: i don't believe they need to be 1 champion group, they concepts must be cohesive.

YK: In practice, it sounds like a similar

M: not advancing, is blocking implementation investigation.

M: we should likely advance, and block at a later time.

YK: I think you are misreading the politeness. I believe some feel the observable may need dramatic changes.

DE: I don’t see why we cant move to stage 1.

M: its a draft

YK: we do not have consensus on stage 2 entrance

M: i think it will langish.

…

JH: i believe observation as a pattern is a thing, and has a space in the spec.

MF: you care only that they overlap

YK: i believe we should avoid overlap

MF: so one should not entirely overlap

YK: I lean in the direction that it is useful

DD: i also lean in that direction, but have consumes

JH: I agree with that conclusion

### Resolution

No stage 2 for now; let’s see how async iterables turn out

## JHD Error.isError

JHD: … brand checking, regardless of toStrings output. Error (and associated subclasses) lacks any internal way. Current done via Object.prototype.toString.

JHD: Chrome/V8 may have shipped toStringTag

JHD: Cross realm errors are not currently brand checkable

DD: This is not useful. We shouldn’t be encouraging brand-check programming

WH: What about proxies? Is there any way for a proxy to proxy an Error and make it look like an Error?

YK: internal slots are cannot be trapped by a proxy so this ok.

MM: This breaks the parallelism with Array.isArray, which recursively looks underneath proxies

JHD: I’m fine with adding support for the paralleism with Array.isArray

JHD: motivating reason, determining if a given value should be wrapped or not (to promise rejection)

YK: ES5 error subclasses wont pass this.

YK: Array.isArray has motivating code

DD: We should use instanceof Error

JHD: But that doesn’t work cross-realm

DD: A cross-realm Error won’t work. Why do you want to check whether it’s a real error?

JHD: what motivated Array.isArray

DD: it shouldn’t have

YK: Cross realm is not the only issue, node ecosystem (duping in npm) has the same issue

DE: What does error give you?

JHD: Stack traces, message property, name property, and people tend to stick other properties for additional payloads

JHD: seems like there are two objections: 1. proxy support (I will make it work) 2. “I don’t believe that programming model should be encouraged”

DH: really really critical use-case for Arrays.isArray, overloading function arguments and array vs non array type requires a very clear case. I don't believe that use-case comes up for errors.

YK: JHD did provide this

JHD: User uses an `Error` sentinel value, which is similar to the function overloading use case.
DD: Rejections should be for same realm errors

DH: i suspect this may be hazardous, because of "security" and you can throw anything.

YK: strings are also errors in JS…

DH: Error is not a hard predicate for cleanly divide the universe. Because plenty of usecases where non errors are used as errors

JHD: The same is with array, objects can mimic arrays often.

DD: those in favour of brand checking where, those not did not

DD: brand checking was not intentional

MM: In ES5 brand checking was very intentional, maybe not in ES6. SES depends on on this.

YK: what about error

MM: SES does not use this

DD: it is a bad precedent to make every new type exotic, to allow brand checking. Error should not be exotic.

MM: general issue is, is there some guarantee that something is given a following brand

DD: Spec has a note, saying this was a mistake.

YK: why does this exist

DD: toString fallback

DD: We should remove it

DH: i think i have not articulated the invariants from this usecass. Array.isArray was not intended for this, but it fit an important usecase. Specifically the overloading scenarios want a strong invariant here.

DH: two array types in JS, branded arrays, and objects the obey the array interface.

JHD: and methods that rely on that interface

DH: I don't believe there is a reasonable programming model that uses overloading with error. Particularly the error wrapping case for promise rejection. I feel this is going down a poor bad.

DH: Possibly an alternative approach could exist

YK: The problem is forgibility, but if someone forges an error is doesn't seem important.

JHD: why do we have the internal slot

DD: we should not have the internal slots. Errors should not be exotic

DD: map + set make use of internal slots for unobservable

BT: slots don't make an object exotic

DD: You are correct, exotic is the wrong word.

DH: With a well motivated programming model, i could be convinced.

SB: An example would be a debugging tool. I want to be sure I do not loose this information because currently we cannot detect.

SB: it is interesting for what this means in general. Observable land may not care, but does the entity may carry information for the system. Without brand checking that may be lost

JM: …

DD: you cannot…

JHD: passing additional information between realms may not be good, but it is done.

JHD: if i find further examples, would that be sufficient?

DH: no, but it would help advance the conversation

DH: I believe a programming model could be extracted from this. We should likely not bless emergent programming models, merely because they exist.

DH: we should excercise our critical thinking, is this programming model worth standardizing.

JHD: I want to gauge if this is worth engaging further

DD: We should assume, error internal slot was a mistake.

YK: we should avoid encouraging brand checking as a pattern.

MM: SES uses instanceof Error, but not used in any security critical areas. Basically, inservice of implementing a getStack API. Case splitting between browsers, if it is an error fetch its getStack. On FF, if it is an error apply the dot stack getter property. When accessed it is wrapped in an try/catch

DD: we should standardize what exists cross platform

MM: it does not, a stack property exists, but the content of the stack is widly different, and could not be standardize without breaking

MM: Some api should exist, which extracts a spec'd stack from an error object. non errros wont carry stacks. Which implies that an error is unique.

DD: Ember.isError does not seem like the right tool

MM: Maybe System.getStack could use it.

YK: …

JHD: Regexp is the only one?

…

SM: can you explain more

SM: isn't stack an anti pattern as it is branding

DD: If it does not have internal structure (private data) branding should not be encouraged

YK: public interfaces should be truthful

SM: what about strings

DD: strings, arrays, math have internal (private state)

YK: Private state should not be taken in account when and outsider inspects.

SM: …

YK: this is daves point about overloading

SM: There appears to be missing mechanism to detect tag/branding. instanceof doesn't work across realm

DH: value types should be branded, userland data-types that have unambigious testable distinctions, and pattern matching those attributes is correct. I don't believe DD is saying that, but we need to think about how to distinguish the two.

SM: ad-hoc tagging seems like a common problem

DD: stringTag sounds like the feature here.

DH: does flow have ad-hoc union types.

SM: yes

DH: so similar to typed Racket

JM: yes

DH: that seems like a very natural fit for JS for this programming model. Deferring this problem to the type system.

DH: What is the flaw in that way

SM: ad-hoc and security issue. There was an issue in react. If there was internal branding this could have been avoided

SM: this is more general yes

DH: should there be a more general tagging/branding mechanism.

SP: WeakMap and proposed private state can do this.

YK: !@#$!@#$!@#%$%^& (discussing actual JS syntax)

SM: toStringTag doesn't pass between JSON,

…

JHD: I want to make sure doing more research isn't a waste of time

MM: there must be a motivating use-case, if there is no such use-case it is a waste of time.

YK: we should be sure the motivating cases are good

JHD: the risk grows the longer we wait.

DD: it sounds like more information is needed.

### Conclusion/Resolution

```
- More research
```

DE: can i propose I18n?

… [everyone yes]

## INTL

CP: todays meeting update: 1. html version of ECMA 402 2. just sent to istvan CP: same workflow for 262

CP: same tools, same flow. getting the HTML version similar (hopefully same) as 262

CP: asside from that, we fixed the tutorials, and ? syntax in the spec instead of returnIfAbropt

CP: new features

EF: first thing usage experience: 402 1.0 afew years ago, and has made its way into browsers. It is being used in many ways. Node is getting it Gecko's UI is using itself Chromes UI is using itself Library level: jQuery globalize, formatjs (suite of libs) l20n from mozilla l10ns an int.js polyfil and more We have experience at libs, and all the new web stuff at Yahoo is using this under the hood. (For safari which doesn't support it yet)

?: Firefox OS is using intl js for all the platform level stuff.

EF: any users of the intl library all have similar requests 1. plurals 2. relative Time 3. duration 4. unit 5. list 6. …

We support in polyfills, but we need more.

EF: cross implementation lack of specification for data, is tricky.

CP: we are not using es discuss for this, we are using the github repo for issues.

YK: seems good to me

BT: we are still reading ESdiscuss, but issues seem good

CP: when we want to take one of these features into consideration, we can go through esdiscuss

EF: v3 clearly wants more, but we lack information. More experimenation is good. We would like to allow userland to explore further.

EF: our abstract algorithims are commonly required for users to experiment. This also includes the corresponding data.

EF: to encouratge experimentation (to acquire more info) we want to expose some more primitives

EF: formatToParts just strings is unsufficient, order/context is hard to encode.

…: Some formats are meanlingless, but important to take into account. The construction of the datetime string is very cultural

EF: essentially it is lossy

EF: formatToParts aims to explain how format even works. It aims to provide an array of objects, with the relevant context. Allowing userland code to do its thing

YK: changing formatToParts should change format

…

EF: unfortunately .format doesn't need to be bound.

ZB: lets open issues and see if we can improve this thing.

YK: regardless it seems like it should still work.

DD: it is unclear

YK: it would be nice

DD: but we understand if the previous choice may prevent this

ZB: implementing the gecko patch, the formatToParts is much slower.

YK: sounds like the same shape as the RegExp Problem. If its overriden, take a slow path.

BT: why an array of objects

EF: order would be lost.

BT: it seems when I care about the subset, it is more complex

CP: I actually believe, it would likely be more complicated.

ZB: LTR RTL languages alsopresented some issues, this pattern worked well

EF: unless it is too expensive, having abstract operations available.

BT: in essense the abstract operations are likely just spec refactorings.

BT: in Windows this would not be straightforward, but is probably doable, using a Windows 10 API… I'll follow up to make sure.

ZB: Windows 10 certainly does things similar to this.

CP: Edge does not use the algorithm we have, just delegates to Windows for best effort?

BT: yes

BT: I agree this should be doable, and in general I'm not concerned. I don't think it constraints implementations too much. Edge might need to ask Windows for a better API.

EF: (next slide) "How: Adding to ECMA 402"

DD: is it possible to move away from the bound method pattern?

BT: no; not web compatible

EF: (next slide) "Current Status"

(Discussion of IE not using CLDR vs. everyone else using it.)

EF: we would like to advance to stage 1, we dont have spec test. But we have draft impl + gecko imp.

BT: I have to get the windows guys to sign-off to on the new API, so something concrete (spec text) would be best.

## Resolution: (for exposing abstract operations)

# Stage 1

## PluralRules

EF: apps must solves this, if they want to use I18n in the UI. This is heavily requested, we have implemented it.

ZB: it is hard to implement correctly, CLDR gives the required information

(some visual examples)

ZB: Without this, the shear amount of complexity required blocks good localized sites.

ZB: two plural forms in the same sentance explodes the complexity.

ZB: this nicely supports the simple case, and the complex case.

EF: also good for relative times "1 hour ago" "2 hours ago"

BT: is there precedent for something like this

ZB: gettext for 30 years

EF: Java's plural format is built on a lower level class that is uses for plural rules and categories (I believe)

EF: we would prefer this to be non optional

ZB: if you cannot afford to store all the data, keep one language

YK: 402 should be take it or leave it, partially support would be unfortunate

YK: all features should be implemented, languages should be based on available data.

M: What about an implementation that supports currency but not plural, since message formatting handles it

EF: It seems like it would be cheap to then provide the real thing.

M: paying for the data would be unfortunate

EF: loading it for all of english, is 800bytes or so. All languages …

SP: wouldn't this be better solved by partial locale data. Implementating a subset of features seems fatal.

BT: would like you all intl or none (or partial subset)

YK: it is possible to imagine scenarios where it is an extreme trade-off, but that isn't how the web works

BT: the thing that is concerning me is. We need to support JS on IoT devices.

YK: and your guessing what so support?

BT: we need to make a choice, based on budget avilable

ZB: I would like to point out, you cant support everything. There is always an edgecase, so we designed 402 APIs, by using fallbacks. By the end of the day, you will get currency. It may not be in chinese, but it will be A currency.

SP: does that deal with the concer?

BT: Yes it should, there is a concern though. If a platform wants to implement parts of i18n in a seperate namespace.

YK: is seems like portable code has been thought about. By allowing all data

M: but what if a given platform provides an alternative

YK: what about eval/toString. It seems like they can provide the alternative and carry on.

BT: if intl is all or nothing, some may take nothing.

ZB: like SIMD, intl should always be there.

DD: either 402 or not

YK: i don’t believe 402 should allow piecemill.

MM: what about partial data sets

EF: the APIs are built for this, as they are built to fallback.

WH: CLDR’s cross product of locales refering to other locales’ names, time zones, their currencies (including various plural inflections), etc. is enormous

ZB: we don’t ship those, in-fact we designed the API for support this.

EF: CP has written up spec text

EF: we need to deal with decimals, the spec test needs to be updated

CP: we need to figure it out yet

resolutions stage 1

# Abstract Locale Operations

EF: aspects of each of the components currently go through some abstract operations, we would like to expose them

ZB: Intl.getCanonicalLocales(locales)

ZB: naming suggestions are open.

ZB: This is useful any time we do language negotiation, this allows us to verify. Implementing this in userland is like 4000 loc, exposing it instead has a clear advantage.

DD: what are the input types

EF: same as the numberFormat

DD: eh. i guess consistency over design

EF: We should about it yesterday, but we felt that it would be simpler this way.

DD: consistency wins it for me

EF: next is a Intl.getParentLocales(locale), the naive implementation would fall short as many exceptions exist. Userland implementation would likely be non-obvious

ZB: for example serbian cyrillic and serbian latin don’t have obvious fallbacks

ZB: We don’t just provide the final solution, the proposed the higharchy of locale inheritance

ZB: We wanted to implement some userland custom code at mozilla, unfortunately we where forced to important several thousand lines of code from the internals.

BT: does this require a giant table

ZB: No, just as the rest of the model, additional data improves the results.

YK: is there any part of the spec that requires explicit data to be loaded

ZB: no

ZB: the goal is, to allow (if data available) the best possible information.

EF: you are allowed to do better

EF: Intl.resolveLocaleInfo

EF: this is an API that has a very similar signature. It will provide the best possible (based on data) resolved language.

EF: the results provides a summary of available information, this will grow.

MF: why not have in seperate functions

EF: the number of data points grows, adding 1 method per data is unfortunate.

DD: what about "current system preferences"

(discussion about user settings/preferences)

DD: This same API, could be used to get the users preferences

EF: maybe, leave out the locale argument, so the default provides this.

CP: user fingerprinting is a potential concern

ZB: this will be happening

EF: We propose stage 1

… [consensus]

ZB: our next step spec proposals and get feedback

BT: outline is good, full spec text isn’t needed right away

DD: spec text for 2

BT: i would love to give feedback before to much investment in spec test.

CP: it shouldn’t be to bad, largely this is extract existing abstract algs

### trailing , in functions arguments

JM: fixes from yesterday (cover grammar support), MF BT looked at them.

shows spec text

DH: looks good!

## resolution: advance to stage 3

### Test 262 updated

BT: not many new tests recently, except for SIMD.

BT: many open issues, but nothing worth mentioning.

BT: we dont have tests for some things, like tail calls.

YK: is it possible

BT: open question

BT: destructing needs to be done

DD: some work to share tests between destructing binding and assignment

BT: Async functions have some more tests, and we'll discuss it more in two months. Async functions will remain stage 3 in January.

### System.global (JHD)

JHD: no reliable way to get the global cross platform.

JHD: shims need it, but required using many tricks.

DD: rationale is good, more bikeshedding on the details.

SP: node-webkit is gnarly here, it belives its both node and web… many existing feature detections failed.

DH: MM has many ideas here, we should be careful to involve him

JHD: I spoke with MM, and tried to get his input

JHD: Were can we put it. MM felt on System as along it was configurable.

JHD: arguments against reifing self, may break existing code

DD: the ideal way is to reuse something exist, whoever feature detection is the tricky one.

DD: global.self maybe ww, global.global maybe node?

JM: whats wrong with with System.global

DD: its long

DD: existing names are accessors, configurable with no setters ony getters, changing that sounds dubious

JM: self is a common idiom.

YK: self is an existing trap

DD: global is my preference

JM: what about System.global

DD: I would not use it

JHD: let me continue, we can bikeshed more

JHD: it can be a windows proxy, it should be the thing new Function("this") would return.

(discussion about windows observability)

JHD: The goal is to use the existing spec to frame what is returned.

DH: CSP concerns, for example eval is prevented.

YK: sloppy CSP already gives access to the gobal

DH: are they guarding from access to the global.

MF: not really https://github.com/w3c/webappsec-csp/issues/2

JHD: in all reasonable platforms, the global is accessible. This merely provides a consistent solution.

YK: SES is ok with this, even though they provide no global access.

### Conclusion/Resolution

- stage 1

# Wrap-up

JN: thanks to paypal, for hosting the meeting lunches and breakfasts. Excellent thank you

JN: Thank you to paypal and ecma for dinner

JN: next meeting is January 25, 26, 27 in SF at salesforce

JN: In january meeting, we must wrap up the june 2016 release.