***Minutes of the:*** ***47th meeting of Ecma TC39***

***in:*** ***Redmond, WA, USA***

***on:*** ***28-30 July 2015***

# 1 Opening, welcome and roll call

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

**Mr. Neumann** has welcomed the delegates at the Microsoft Campus in Redmond, WA, USA.

Companies / organizations in attendance:

Mozilla, Google, Microsoft, Intel, jQuery, Facebook, Netflix, Indiana University, PayPal, Yahoo!, INRIA, Shape Security, Airbnb

## 1.2 Introduction of attendees

Istvan Sebestyen – Ecma International

John Neumann – Chair

Jordan Harband (invited expert – Airbnb)

Caridy Patino – Yahoo

Allen Wirfs-Brock – Mozilla

Yehuda Katz – jQuery

Rick Waldron – jQuery

Mike Pennisi – Boucoup (invited expert)

Domenic Denicola – Google

Adam Klein – Google

Brian Terlson – Microsoft

Ron Buckton – Microsoft

Sebastian Markbage – Facebook

Dave Herman – Mozilla

Brendan Eich

Mark Miller – Google

Jafar Husain – Netflix

Sam Tobin-Hochstadt – Indiana University

Daniel Ehrenberg – Google

Peter Jensen – Intel

Dan Gohman – Mozilla

Michael Ficarra (invited expert – Shape Security

Waldemar Horwat – Google

Chip Morningstar – PayPal

Eric Ferriauolo – Yahoo!

Paul Leathers – Microsoft

Allen Smitt – INRIA

Jonathan Turner – Microsoft

Mohamed Hegazi – Microsoft

Abhijith Chatra – Microsoft

Tom Care – Microsoft

Akrosh Gandhi – Microsoft

Jonathan Sampson – Microsoft

Ben Newman (part-time, on phone)

### 1.3 Host facilities, local logistics

On behalf of Microsoft **Brian Terlson** welcomed the delegates and explained the logistics.

## 2 Adoption of the agenda (2015/037-Rev1)

The agenda was approved as posted on the github:

# Agenda for the: 47th 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 (TC39/2015/037)
3. Approval of the minutes from May 2015 (TC39/2015/035)
4. Report from the Ecma Secretariat

i Status of ISO/IEC Fast Track of ECMA-262 Ed.6, ECMA-402 Ed.2 and ECMA-404

ii Report from the GA and CC

5. ECMA-262, Editorship
6. ECMA-262 7th Edition, 2016

i Advance Array.prototype.includes to stage 3 (Domenic Denicola)

ii Introduce (and maybe advance?) RegExp.escape proposal (Domenic Denicola)

iii. Introduce promise rejection tracking proposal (Domenic Denicola)

iv. Advance Async Functions to Stage 2 & discuss open questions (Brian Terlson)

v. Proposed Changes to Observable API (https://github.com/zenparsing/es-observable/tree/zenlike) (jhusain)

vi. BindingRestElement should allow a BindingPattern ala AssignmentRestElement (Brian Terlson)

vii. new & GeneratorFunction (Brian Terlson)

viii. SIMD.js: Start the process to move towards Stage 3 (spec, presentation) (Dan Gohman, John Mccutchan, Peter Jensen, Daniel Ehrenberg)

ix. Reconsidering the Number.prototype-as-an-ordinary-object change (Daniel Ehrenberg)

x. Advance Exponentiation Operator to stage 3 (Rick Waldron)

xi. The scope of "use strict" with respect to destructuring in parameter lists (Andreas Rossberg)

xii. String.prototype.split, its limit argument, and ToLength vs ToUint32 (gist) (Adam Klein)

xiii. Advance Rest/Spread Properties to Stage 2 (proposal w/ spec) (Sebastian Markbage)

xiv. Spec defacto String.prototype.trimLeft/trimRight (es-discuss / proposal) (Sebastian Markbage)

7. Test262 Updates (Brian Terlson, Mike Pennisi)
8. ECMA-402 3nd Edition, 2016

i. Intl.NumberFormat round option

a. https://github.com/rxaviers/ecma402-number-format-round-option

ii. Migration to ecmarkup system

a. https://tc39.github.io/ecma402/
b. https://github.com/tc39/ecma402/tree/master/spec

9. Tooling Updates (Brian Terlson)
10. Timeboxed process-document discussion, to settle things once and for all
11. Date and place of the next meeting(s)

September 22 - 24, 2015 (Portland, OR - jQuery)
November 17 - 19, 2015 (San Jose - Paypal)

12. Closure

## 3 Approval of minutes from May 2015 (2015/035)

The minutes were approved without modification.

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

For details please see Annex 1 in the Technical Notes.

## 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

**Mr. Sebestyen** reported that the GA has approved that the "experimental" TC39 IPR policy options (RF Patent and Software Copyright Policy) become "final" and "Ecma-wide" policies.

## 7 Date and place of the next meeting

I. September 22 - 24, 2015 (Portland, OR - jQuery)

II. November 17 - 19, 2015 (San Jose - PayPal)

# 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, **Microsoft** for the organization of the meeting and the excellent meeting facilities and dinner. Many thanks in particular to **Mr. Terlson**. Many thanks also to **Microsoft** and **Ecma International** for the social events.

Annex 1

Technical Notes

# July 28 2015 Meeting Notes

Allen Wirfs-Brock (AWB), Sebastian Markbage (SM), Jafar Husain (JH), Eric Farriauolo (EF), Caridy Patino (CP), Waldemar Horwat (WH), István Sebestyén (IS), Mark Miller (MM), Adam Klein (AK), Michael Ficarra (MF), Peter Jensen (PJ), Domenic Denicola (DD), Jordan Harband (JHD), Jonathan Turner (JT), Paul Leathers (PL), Chip Morningstar (CM), Vladimir Matveev (VM), Ron Buckton (MS), Brian Terlson (BT), Alan Schmitt (AS), Ben Newman (BN), Mohamed Hegazy (MH), Abhijith Chatra (AC), Tom Care (TC), John Neumann (JN), Dave Herman (DH), Brendan Eich (BE), Rick Waldron (RW), Mike Pennisi (MP)

## Introduction

BT: (logistics)

AWB: I will chair until John N. arrives.

## Adoption of Agenda

AWB: https://github.com/tc39/agendas/blob/master/2015/07.md

YK: Propose that future agendas be organized by stage advancement

(Agreement)

IS: Need to add Ecma Secretariat items (Allen adds) like report from the GA.

### Conclusion/Resolution

- Approved

## Approval of Minutes from May 2015

AWB: Posted on Github and es-discuss

IS: Ecma is archiving (mirroring) the most significant documents from GitHub, like technical notes as well as summaries associated with the agendas. This practice should satisfy both the requirements of Ecma as an SDO as well as of Tc39.

# Report from Ecma Secretariat

# 4.i Status of ISO/IEC Fast Track of ECMA-262 Ed.6, ECMA-402 Ed.2 and ECMA-404

IS: Need an Ecma-262, Ecma-402 and Ecma-404 package "Explanatory Report"

- Next project is "linking" the three explanatory reports in one
- Then discussion on the fast track started

AWB: Can we push through our current documents, as-is? Without changes?

IS: Similar to 5.0 -> 5.1, editorial changes allowed. Any technical changes, no. Concerns about stability.

AWB: Concerns about an ISO document that says one thing, vs. the Ecma document that says another.

IS: Can disallow discrepancies, therefore it is always the policy to synchronize the standard on both sides

WH: What happens if the ISO reviewers discover technical issues?

IS: The problem is that the yearly update for JTC1 might to be too fast. We are elaborating on them. Theoretically worst case, we withdraw the "fast track"

WH: [incredulous] Withdraw instead of fixing bugs? That wouldn't be nice.

IS: (explains why fast track). We have started this about 17 years ago, as long TC39 came out with big time gaps between Edition, no problem, but the current "turbo speed" like standardization (update) may create problems in synchronization. we are discussing this new thing with ISO.

AWB: Good to communicate to ISO

- We appreciate technical feedback
- Feedback will be put into the next edition as part of the on going process

IS: Explained to Rex Jaeschke (JTC1 SC22 Chair) the new schedule. Normally ISO likes to update standards every three years, have expressed our one year schedule plans.

WH: Afraid ISO might not like "take it or leave it". In the past, we got excellent technical reviews from them and have integrated the changes into the ISO standard and to the Ecma standard.

AWB: But that would be next June

WH: That's how we've always done it. The ISO version was a year behind the ECMA version. If we don't integrate the known defect fixes, the ISO version would be multiple years behind ECMA.

AWB: Ignoring unknown issues, we have 5 or 6 technical issues that could comprise a 6.1

RW: If we have a yearly release for full edition points, perhaps we can also include a 6 month "sub release" to address minor changes to current

YK: Not unreasonable when the work is done by a group using shared tools

IS: Theoretically in Ecma we could have a 6 month correction cycle (TC52 Dart uses that). Depends on TC39 what they want.

AWB: What does the GA think about doing this process every six month?

IS: no problem there, this could be reasonable solution, for Dart it has been accepted. Of course Impose a deadline of 2 months before December GA meeting for the version to be voted upon and published.

RW: Can correspond with our September meeting.

IS: (agreed)

AWB: Trivial to make minor changes, willing to work through the first round as we transition editorship.

- Need to decide this meeting and produce a draft for approval at next meeting.
- https://bugs.ecmascript.org/describecomponents.cgi?product=ECMA-262%20Edition%206

RW: Does this need to include the ISO review comments (Waldemar's concern)

YK: Would like to come back to this after we have a more in depth discussion about the future of our tooling.

Discussion about what we produce.

AWB: We can produce errata any time, but it's not the standard

DD: If we have the base covered, avoiding shipping bugs, do we publish errata every year or every six month?

WH: It's not ok to ship a standard with known defects. We can publish standards on whatever schedule we want, but anything we publish should have all known defects fixed.

YK: Think "annual" is enough time to fix bugs

AWB: The issue is ISO. There is concern about divergence between those two documents (ISO document with changes made by their reviewers, Ecma version)

BT/IS clarifying why we need ISO version?

BT: Why?

WH: There are benefits to ISO publication

AWB: There are organizations that may require

IS: Public procurement, if it's an Ecma standard, maybe problematic. ISO is in the "highest level" of international standard category.

BT: Are there concrete examples of using ISO specification over Ecma?

(Spending too much time on something we don't know the value of)

DD: Propose that we don't do this process unless an external request for it is made.

IS: We are already for 17 years in this process, and it is expected that we continue for future editions as well. So this is automatic and not exteranl request is needed. How to explain why ES 1, 2, 3, 5 & 5.1 are ISO standards, but 6 and 7 etc. not and what if Ecma is already e.g. at 7 and ISO still on 5.1?.

Can we be forced to do this?

IS: Yes, the current rule for synchronization "forces" it.

YK: (clarify) Once we issue *any* ISO spec, we're required to maintain?

Several "yes"

WH: We don't want to destroy the relationship between Ecma and ISO

STH: How does is hurt to not create ES 6.1?

YK: ISO was ok with a two year wait for 5 to 5.1, should be ok with a year between 6 and 7

WH: But much longer since 5.1

AWB:

# - Errata?

Discusson re: schedule, responsibilities, etc.

AWB: We will submit to ISO, with exact same text (subject to ISO naming, etc). But the text will include any known errors.

WH: No, they have to be fixed. I'm not ok with shipping standard with defects known to us.

DD: We agreed to fast track at the last meeting and Waldemar is now disagreeing?

WH: Incorrect

DD: You said you won't allow us to fast track with defects

WH: No, we *must* fix defects

STH: Why does fast track require fixing errata?

WH: Not a lot of work

DD: You're saying that you no longer agree to fast track

WH: We will get comments from ISO reviewers, these will identify defects, these need to be fixed. It is our job to fix

CM: Exactly the proposal: take the existing text and put it in the pipeline and publish the errata later.

YK: Treat ISO as an LTS?

WH: That's *what* it is

DD: Can we have members of the committee that are responsible for incorporating the errata, as a separate task.

AWB: That's the editor's job. Incorporating errata is not a lot of work.

RW: This is exactly my suggestion, but with a schedule requirement.

Discussion about who and how.

AK/JT: What is the cut off for known defects?

DD: Suggest that we let whoever volunteers be responsible for making those decisions

YK: With better tooling, we could really just do this as branches in a repository where all work is tracked.

AWB: Don't think that any editorial difficulty. Bigger isssue: what are we going to do with ISO on a regular basis? Yearly ISO and it's the same document?

BT: Agreed.

AWB: Ideally we'd have "fast track authority", submit the Ecma document and they either take it or leave it.

YK: Can we revisit this after we've read the agenda items? (Specifically item 5)

AWB: That item is Editorship?

YK: That seems to determine tooling.

JN: Assignment of Editorship role.

BT/AWB: Yearly release, give them to ISO without change, accept them as is. Take their changes and roll them into next year's edition.

WH: This is agreed to an incorrect assumption. All ISO comments must be addressed.

IS: That's correct. That's required by the ISO process.

BT: If Istvan can convince ISO to accept yearly releases "as-is", are you ok with this?

WH: If ISO can accept as-is, then let's discuss this again, but I don't think they can

IS: I told ISO at the last meeting that we are moving much faster and that we need to create a new way to submit. This is a way to handle that and I can present it to them as a solution. Normally, they don't update until every 3 years. I pointed out the issue with their timeline and they agreed. This is a possible solution and I will work with them to sort out this problem.

WH: Are they ok with not making comments?

IS: From some members, but not all, awaiting final word.

WH: Not ok with making decision until we know where they stand.

## Conclusion/Resolution

- Yearly release, given to ISO without change, to accept as is.
- No sub releases
- Publish errata as follow up
- Take their reivew comments/changes and integrate them into next year's edition.

- Istvan will present this to ISO as a solution

# Propose Life Membership for Allen Wirfs-Brock

(Rick Waldron)

RW: As we have for Brendan, I'd like to propose inviting Allen Wirfs-Brock as a lifetime member of TC39

IS: Such a category does not exists officially at Ecma, only some freedom for the Secretary General to allow participation as invited expert. He will bring this to Ecma Management for seeking agreement. This is a special case, and I am confident.

## Conclusion/Resolution

- Istvan to get approval

# 5 ECMA-262, Editorship

JN: Need to appoint a new editor, only one volunteer has come forward and that's Brian Terlson.

- Motion to appoint Brian

YK/RW: Seconded

JN: Discussion?

WH: Do you see your approach as different from Allen?

BT: Only in that I'm a different person, but generally the same. I'm not Allen, but I will do my best.

AWB: Appropriate to also appoint a manager for the content that we're producing.

## Conclusion/Resolution

- Brian Terlson is new editor of Ecma-262
- Unanimous consent

# Discussion of the Manager Role

AWB: Useful to have some web content which will need a manager

Discussion, re: content manager

"web master"

IS: Need procedures to archive the github material

AWB: Unless someone wants to volunteer now, then let's resolve to appoint someone by the next meeting.

JN: Allen please generate a "job description"

## Conclusion/Resolution

- Allen will produce a "job description"
    - Coordinate with Brian
- Submit to member companies for review
- Member appointed to the position at next meeting

# Exponentiation Operator

(The room is very excited about technical discussion finally starting.)

RW: SpiderMonkey is ready; V8 is ready; spec material is ready https://rwaldron.github.io/exponentiation-operator/; it meets the requirements for stage 3

WH: This is incomplete; it is missing details on the lexical grammar.

RW: Agreed; I can address that issue before the end of this session. Perhaps we can revisit in the coming days.

DH: Let's avoid a precedent that if anyone finds any minor hole, they can stop advancement.

WH: This is an issue of completeness

DH/BT: The resolution of this problem is obvious and trivial to implementors

AWB: We need to improve the process document to include expectations for reviewership

DD: The process document is now published as an HTML document in an public repository on GitHub.com. I have migrated it from Google Docs with no changes to content. I have opened two issues to discuss details that have historically vexed this committee.

WH: I have no doubt that you can address this by the end of this meeting.

RW: I am literally done.

BT: We have agreed that technical issues should not block advancement to Stage 3. Does anyone object to advancement?

AWB/DH: We have technical questions, but they have no bearing on advancement to Stage 3.

(break for lunch)

### Conclusion/Resolution

- Approved for stage 3

## 6.2 RegExp.escape

(Domenic Denicola)

https://github.com/benjamingr/RegExp.escape

Slides

DD: (presents slides)

YK: Don't make user think about escaping.

AWB: Generate a complete regexp from string. Append cases

WH: What does it mean to create a regexp from a string? (w/r to Proposal Ruled Out)

AWB: Regexp construction that will exactly match

YK: A simple example: a lexer, the lexemes provided as a string

DD:

SyntaxCharacter Proposal

- Escapes string just enough to be used as a RegExp

AWB: Used as argument to the RegExp constructor

DD: yes

Safe Wth Extra Escape Set

- Escapes everything the SyntaxCharacter proposal does
- Escapes - addl. for the context sentive inside-character-class matching
- Escapes hex numeric literals (0-9a-f) at the start of the string in order to
- Less readable output but safer

WH: Escaping `d` by replacing it with `\d` completely changes its meaning. The latter matches any digit but not the letter 'd'.
YK: Issue is `RegExp.escape(...)` what?
WH: Second use case (https://github.com/benjamingr/RegExp.escape/blob/master/EscapedChars.md, `0-9a-fA-F`) is not useful. There is no good reason for someone to concatenate the erroneously short (three digits instead of four) `\u004` to user-defined input. The first use case there is also not common.
MF: Should be a `RegExp.compose`, not use `+` to concatenate strings for `RegExp()`

```
new RegExp("\\u004")
```

AWB: illegal in the grammar, but allowed in the annex extentions (annex b grammar allows < 4 characters following the`\\u`)
WH: Why would you do this on purpose?

WH: How do you escape these things? `\d`, or `\0` which does not mean `0`
MM: Issue about fragment hazards, not using + for composition. I actually think that a template string tag is the solution

DD: Extended "Safe" Proposal, Allen had concerns?

AWB: ES5 takes efforts to roundtrip the string returned for a regex has forward slashes

WH: RegExp's toSource creates a string that, when eval'ed, produces a valid RegExp

DD: No change to output of `RegExp.prototype.toString`
DD: https://github.com/benjamingr/RegExp.escape/issues/4 explains why not a template string tag, but reviewing now, the arguments are not strong

MM: You get the escape functionality in terms of the template string tag.

MF: Concerns about unnecessary escaping

DD:

WH: Readability costs. Don't want RegExp.escape to turn '7' into '\u0037'.

MM: template string doesn't have those costs

DD: cannot compose a variable number of strings together with template string tag

YK: Assumes certain amount of "static-ness".

... Would like to see an injection scenario that seems realistic.

DD: it's possible, it can happen. Reasonable for a TC created solution to address all the cases. All we need to do is address all of the cases presented, at the beginning of the string

AWB: Why do we thing there is one `escape` function that covers all cases?
WH: Whoever is writing the things that are not escapes, limits themselves to a subset of "well behaved regexp". In a well-behaved regexp:

- don't end it with a single \
- don't end it with \u002
- don't end it with \1
- don't end it with (?
- etc.

DD: What if you want to match backreference \1

WH: If you want to do it just before a concatenation point, write it as (?:\1). A trailing backreference is such a rare case, this is a reasonable constraint to do to simplify readability of all escaped strings that start with digits.

MM: Object to template string solution?

WH: No. My argument is orthogonal to whether we use template strings or RegExp.escape. Object to escaping individual non-punctuation characters a, b, c, d, e, f, 0, 1, 2, 3, 7...

DD: Objection to any proposal but the SyntaxCharacter Proposal?

WH: No. Fine with other punctuation-escaping-only variants such as also escaping '/' if we choose to do so.

YK: Does SyntaxCharacter escape -?
DD: No

YK: That's a mistake

AWB: SyntaxCharacter has specific meaning

DD: Current thing on table: SyntaxCharacter and hyphen

- Advocate for whitespace?

YK: Why?

DD: eval

Discussion re: template string availability

DD: Proposal coming from people that need to ship code to non-edge browsers

YK: Use today? Library code.

DD: Has been a request for a long time, give what's requested even if unsafe? Give them what they want plus things that make it safe? or make a new thing from template strings.

Discussion, re: issues with template strings in this use case

YK: Maybe template string API is flawed and needs work to be dynamic

MM: Agreed with need.

CM: this output from one piece of code to another

STH: The spec can say "the output looks like literally the input"

YK: People will then rely on it

STH: State the goal, then go back and specify to reach the goal.

DD: We've done this, and the constraints are illustratedhttps://github.com/benjamingr/RegExp.escape/blob/master/EscapedChars.md

MM: What is the motivation for readable output?

BE/YK: Debugging

MM: Ok, then issue is debugging vs creating less bugs?

DD: Then escape everything

YK/MM: Agreed

DD: Any objections to escape everything?

- There's been a lot of work so far
- Can we continue on this path?
- Preference for template strings?

MM: I think the template string proposal is better in all dimensions

YK: Maybe do both?

- Safe version of `RegExp.escape`
- Template String solution

AWB: Stage 1? Will the champions know that this could change from RegExp.escape to template strings?

DD: Want to say that `RegExp.escape` can move forward for now
YK: ?

MM: template strings handle all of the malformed fragment cases

- Small list completely safe?

AWB: and is 100% convinced is safe?

MM: eval thing is safer than

DD: "only beginning of string"

WH: What mark is suggesting conflicts with what I was saying

YK: What is the constraint?

WH: Readability

Ok to escape everything, but don't have to

MM: Rather not inro hazards, ok with either solution that doesn't

DD: Ok to advance is escape everuthing?

WH: if we don't, escape everything, unnecessary to escape "d" at the beginning of string

DD: This has been identified as a security issue

WH: it does't buy anything and there are others that aren't covered.

DD: Can you give example

WH:

```
new RegExp("\\" + RegExp.escape("x"))
```

MM: "\" is a regexp fragment.

WH: Yes (in this case), but in practice so what? There's nothing in the concatenation example that would ensure you only concatenate complete fragments.

MM: Right, this is still a security issue. Unresolvable by escaping due to the odd-even problem.

DD: Ok, the point was to handle all cases

YK: Non capturing parens solution

```
new RegExp("\\\\(?:x)")
```

WH: Stick to well-behaved

Each case fails where another does not. Template strings address all issues, but fails for not being dynamic

MM: Would Benjamin be interested in pursuing the template string solution.

DD: Will bring this to Benjamin

?: Need to work to resolve the caveats in RegExp.escape.

WH: The caveats of protecting against all possible prefixes are inherent and irremovable. I want RegExp.escape to move forward despite those. It's safe for concatenating arbitrary escaped user input with well-behaved regexp fragments.

MM: I'm not ok

### Conclusion/Resolution

- dynamic solution that has caveats
- caveats cannot be eliminated
- not confident that all issues can be addressed
- want to see template string solution explored

## 6.i Advance Array.prototype.includes to stage 3

(Domenic Denicola)

https://github.com/tc39/Array.prototype.includes

DD: Walking through https://github.com/tc39/Array.prototype.includes/issues/12

### Conclusion/Resolution

- Approved Stage 3

## Introduce promise rejection tracking proposal

(Domenic Denicola)

https://github.com/domenic/unhandled-rejections-browser-spec#changes-to-ecmascript

DD: (introduction)

YK: (summarizing) Some want to be notified when a Promise is rejected without an error handler

- Will be used as if any promse without an immediate handler to treat as a fatal error

MM: prefer this to solution had consider

YK: Easily be used incorrectly

BE: Regardless of our intentions, it will be misused

- What do you do with the promise?

DD: Use to correlate

- dedupe on client

Discussion, re: potential hazards

YK: People often try to deal with promise debugging incorrectly

MM: This provides diagnotistic tools that allow for better debugging

YK:

MM: Instead of using the turn boundary, await gc, but gc may never happen

- Intead of calling something to notify, a query API: did a promise get rejected? provide reason

DD: (refocusing) All I'm hoping for is the addition of these non-observable steps to the specification (showinghttps://github.com/domenic/unhandled-rejections-browser-spec#promise--executor- )

AWB: Nothing there that an implemenation couldn't do in terms of the current spec, it's totally allowed.

DD: You'd have to say "insert steps here"

AWB: When putting in the hooks, we more than say "this is allowed", we're saying "this is supported"

BE: This is more like a debugger API

Agree

YK/MM: An innocent refactoring to register a rejection handler

MM: The utility of having a diagnostic outweighs the noise of introducing

DD: There is no delay, occurs immediately, at the earliest step possible

MM: If I call Promise.reject?

DD: Will trigger the HostPromiseRejectionTracker

YK: Persuaded by Mark's argument

Discussion, re: .done

AWB: Different promise's can have different handlers? No

AWB: no reason an implementation needs to define HostPromiseRejectionTracker, so a default handler should be specified.

DD: Didn't see something like that

YK: Move to stage 2, because already implemented.

Discussion, re: stage process? Yes.

### Conclusion/Resolution

- Specify a default handler
- Filed: https://github.com/domenic/unhandled-rejections-browser-spec/issues/14
- Non-normative note: "If operation is "handle", it must not hold a reference to promise in a way that would interfere with garbage collection."
- Filed: https://github.com/domenic/unhandled-rejections-browser-spec/issues/15
- Approved for stage 2

## 9 Tooling Updates

(Brian Terlson)

Slides

BT:

Grammarkdown (Grmd)

- http://github.com/rbuckton/grammarkdown
- Plain text syntax for specifying ECMAScript grammar
- Support for multiple emitters—MD, HTML, Emu
- Example

AsyncArrowFunction[In, Yield] :: `async` [no LineTerminator here]
AsyncArrowBindingIdentifier[?Yield] [no LineTerminator here] `=>` AsyncConciseBosy[?In]
#1
MM: Can I use this standalone?

BT: Sure, but it's just part of Emu

Ecmarkup (Emu)

- Emu-xref for cross-references
- Emu-production references
- Emu-syntax element supports Grammarkdown
- In use by SIMD, Intl 2.0, Async Function and others
- Michael Dyck now maintaining es-spec-html, working on high-fidelity emu output
- Next steps:
- Unclock authoring of new specs and porting of 262
- New EMD processing model
- Continue improving authoring tools
- Emd 3.0

AWB: note that the Python program that Jason wrote has evolved

Ecmarkdown (Emd)

- Working towards 3.0 release
- Hand written parser
- Support for parsing entre documents (not just single paragraph/list or fragment)
- Support for more Markdown-like syntax
- Bullet lists
- Backslash escape
- Next steps:
- Unblock authoring of new specs and porting exsting spec
- Syntax for links
- Smart quotes

# 6.6 BindingRestElement should allow a BindingPattern ala AssignmentRestElement

(Brian Terlson)

Slides

BT: AssignmentRestElement allows destructuring, BindingRestElement throws.
Destructure rest on assignment, but not on binding

- Historic accident
- Bug filed on error thrown by having an array lutera after a rest
- https://bugs.ecmascript.org/show_bug.cgi?id=3361

Possible Avenues

1. Keep as is
2. Remove support for a bunding pattern in a rest element
3. Align AssignmentPattern with BindingPattern to allow further destructuring of the rest element

BindingRestElement[Yield] : ... BindingIdentifier[?Yield] ... BindingPattern[?Yield]
Plus necessary statc and runtime semantics to align with BindingPattern

MM: Made and fixed this error in http://research.google.com/pubs/pub43462.html

- Any reason to not do the general orthogonal thing?

BT: No

BT: Babel already treats the same

### Conclusion/Resolution

- Align AssignmentPattern with BindingPattern to allow further destructuring of the rest element

## 6.7 new & GeneratorFunction

(Brian Terlson)

Slides

BT:

Current Semantics:

- [[call]] `this` set normally by caller

- [[construct]] `this` is a dynamic ReferenceError

Issues:

- Potentially confusing
- new doesn't do anything useful

Possible Solution:

- GenerationFunction doesn't implement constructor, so new throws
- `this` is not a dynamic ReferenceError, but instead refers to the GeneratorFunction instance (align with current implementations: Babel, V8, SM)

AWB: Doesn't act like derived constructor, does ave a prototype property that's used to initialize instances

BT: No case where a subclass will always throw

AWB: Body of function is not run as part of the constructor. The body is not a constructor body in the typical sense

BT: Breaks down the `this` and `new` behavior, b/c can't use `this` with `new`
AWB: The body isn't run as part of the constructor, it's run as "next"

DH: A generator used with new is weird. (giving an example)

```
function * Thing(x) {
  this.x = x
}

Thing.prototype.sayHi = function() {};

let thing = new Thing("tc", 39);

thing.x // undefined
thing.y // undefined
thing.next();

thing.x // "tc"
thing.y // undefined
thing.next();

thing.x // "tc"
thing.y // 39
```

DH: Why would want to pause before fully constructing the object?

- Introduces a non-orthogonality that doesn't pay for itself

AWB: Generator method

```
let obj = {
  *foo() {
    yield this;
  }
};

let iter = new obj.foo();
```

RW: Non-generator method would throw a TypeError when `new ...`
MM/DH: (discussing special cases)

BT: Potential argument for why we should allow `this` inside generator, observables, where you might initialize this before the first yield (by calling `next` before handing out your iterator)
AWB: this binding is actually set up when the iterator is created

MM: A generator method, that does not mention super, has a call behaviour that sets this from it's caller the way that a generator function does

AWB: imagine "MyCollection"

```
class MyCollection {
  constructor() {
    // ...
  }
  * values() {
    let k = 0;
    let length = this.length;
    while (k < length) {
      yield this[k++];
    }
  }
}

let collection = new MyCollection();

let values = collection.values();
```

```
class MyCollection {
  constructor() {
  //
  }
}
```

```
let collection = new MyCollection();

collection.values = function * () {
  console.log(this);
};
```

DH: Have a construct trap, normal new object passed in. Generator function is a generator implementation of a function.

- On us to prove/disprove value

BN: If trying to discourage using new, this may already be the case, since this.next(), this.throw(), this.return() will all throw if called from within the generator, because the generator is already executing

AWB: Cannot re-enter

```
function * Thing(x, y) {
  this.x = x  // current ES6: throw here.
  yield;
  this.y = y;
}

Thing.prototype.sayHi = function() {};

let thing = new Thing("tc", 39);

thing.x // undefined
thing.y // undefined
thing.next(); // current ES6: ReferenceError on `this`

thing.x // "tc"
thing.y // undefined
thing.next();

thing.x // "tc"
thing.y // 39
```

(Note that SpiderMonkey doesn't throw at either of the places called out above)

DH: The * is a way of saying that you're providing a way to the precise thing you would do by hand, but without the long form hand written

RW: Then generator method should throw TypeError when called with new

DH: Agreed

MM: What is the instance created from whehn a generator method is called with new?

AWB: instance of Generator

MM:

```
let coll = {*values() {}};
let it = coll.values();
it instanceof coll.values; // true
```

MM: Unseen: allocatiing a new generator instance. Contains an implicit `new` and implcit constructor body.
DH: * imputes an implicit `new` (in both the method and constructor)
MM: Acting like a function that's calling an implicit constructor

BT: it's a factory

DH: Simpler: "* is never new-able"

Alternative 2: a function * has

MM:

```
let coll = { *values() { this; } }
coll.values(); // ?

coll.items = function * () { this; };
coll.items(); // ?
```

DH:

| S | I | Code |
|---|---|---|
| ? | X | `{ *foo() {} }` (no [[construct]]) |
| X | X | `function * () {}` (no [[construct]]) |
| ? | ? | `function * () { this }` is exactly like `function() { this }` |

AK (offline): Further spec reading suggests that GeneratorMethods need to have their [[Construct]] removed just like GeneratorFunctions (they both pass "generator" to FunctionInitialize, which is where the decision to set up [[Construct]] is made).

### Conclusion/Resolution

- Maintain spec that cannot construct generator methods (or change spec to make them non-constructible, see above)
- Spec change: generator function do not have a [[construct]] trap, so `new` throws
- Call a generator, `this` should be set by the caller, as is normal (global or undefined)
- File Errata

## 6.12 String.prototype.split, its limit argument, and ToLength vs ToUint32

(Adam Klein)

https://gist.github.com/ajklein/335e0f948c500a0c25dc

AK: Issues:

- ES5 (15.5.4.14.5): If limit is undefined, let lim = 2**32-1; else let lim = ToUint32(limit).
- ES6 (21.1.3.17.8): If limit is undefined, let lim = 2**53-1; else let lim = ToLength(limit).

Problems:

- Return value is an Array, so a limit greater than 2**32-1 would result in a "malformed" array (one with elements past the end of the array). Iteration over the return value will skip all such elements.
- Behavior changes for negative limit: ToUint32 transforms -1 to 232-1; ToLength instead transforms -1 to 0.

AWB: One of the discussions was to change arrays to be larger

AK: That would make sense

Discussion about changes

- break web?

- but this will never likely be hit by any real world code
- clamping is valued

Proposal: Revert this spec change. Existing implementations still match ES5, and the old behavior still makes sense (even with ES6's longer String length limit).

WH: The spec needs a geocities-style "Under Construction" animated GIF here ;). This was the first phase of a series of changes attempting to move the spec to eliminate the 2^32 array length limit, with the changes spread over time.

## Conclusion/Resolution

- Revert this spec change. Existing implementations still match ES5, and the old behavior still makes sense (even with ES6's longer String length limit).
- Filed https://bugs.ecmascript.org/show_bug.cgi?id=4432

# July 29 2015 Meeting Notes

Allen Wirfs-Brock (AWB), Sebastian Markbage (SM), Jafar Husain (JH), Eric Farriauolo (EF), Caridy Patino (CP), Waldemar Horwat (WH), Istvan Sebastian (IS), Mark Miller (MM), Adam Klein (AK), Michael Ficarra (MF), Peter Jensen (PJ), Domenic Denicola (DD), Jordan Harband (JHD), Jonathan Turner (JT), Paul Leathers (PL), Chip Morningstar (CM), Vladimir Matveev (VM), Ron Buckton (MS), Brian Terlson (BT), Alan Schmitt (AS), Ben Newman (BN), Mohamed Hegazy (MH), Abhijith Chatra (AC), Tom Care (TC), John Neumann (JN), Dave Herman (DH), Brendan Eich (BE), Daniel Ehrenberg (DE), Dan Gohman (DG), Andreas Rossberg (ARB), Rick Waldron (RW), Mike Pennisi (MP), Akrosh Gandhi (AG), Jonathan Sampson (JS)

## 6.11 The scope of "use strict" with respect to destructuring in parameter lists

(Andreas Rossberg, on phone)

Slides

ARB: Strictness Scoping

Issues:

- VMs need to do parsing & static checks in single pass
- w/o building an AST (lazy compilation)
- Backtracking is not an option (at least not for V8)

DH: clarify backtracking (2 possibilities?)

ARB: rewinding token stream

ARB: easy in ES5 because only additional check is duplicate parameter names

```
"use sloppy";
function f(x, x) { "use strict"; }
```

Difficult in ES6, b/c default parameters

```
"use sloppy";
function f(g = (o) => {with (o) {}}) { "use strict"; }
```

ARB: also cannot determine scope due to interactions between Annex B and strict mode:

```
"use sloppy";
function f(g = function(h) {
  { function h() {} }  return h;
}) {
  "use strict";
}
```

WH: The issue in this example is hoisting and variable binding, which is more than just error checking. Function h here is nested inside a local block, which means that the 'h' in the return statement refers to different bindings in strict vs. non-strict mode. But you don't know that strict mode is in effect until encountering the 'use strict' later!

DH: Don't know yet whether this is an error until later. Previous examples: Store a "dirty bit". This example: have to have a data structure that will hoist in 1 of 2 ways if it turns out to be strict mode. Implementable without backtracking.

WH: Note that a single state bit is insufficient. You can have these things nested arbitrarily deep, leading to a potential exponential explosion of states.

ARB:

Much More Difficult in ES6

- The directive can affect arbitrary code
- Nested arbitrarily deep
- Would need ot defer any sort of mode-specific decisions in the parser for code that occurs in parameters
- But with arrow functions, we do not even know (in time) that we are in a parameter list

```
"use sloppy";
let f = (g = () => { /* Are we a parameter? Do we have to defer? */ with (o) {} }) => {
"use strict"; }
```

ARB:

BE: The arrows are parsed with a cover grammar

- When initially implementing strict mode, SpiderMonkey had to implement token stream backtracking to account for the function body "use strict" prologue affecting to the left

ARB:

Categories of mode specific logic

1. Mode-specifc errors: (eg. with, delete, for-in, octals, let, variable name validity, parameter conflicts) -> Easy to defer, at least in principle, but may have measurable cost
2. Special handling of eval (scoping, variable modes) -> Not an issue
3. Actual divergence in parsing/scoping (Annex B function scoping, yield?) -> Affect downstream decisions, transitively defer

DH: yield doesn't have contextual differences in strict mode, in generator functions

AWB: restricted in strict mode (non-generator) functions

- other differences?

DH: this-binding

YK: to be clear, these disadvantages only apply to sloppy mode programs; modules and strict mode programs are not affected

ARB: 3 modes, effectively:

- sloppy
- strict
- "don't know"

DH: (discussing potential for performance regression)

- Any parenthesis expression could potentially end up in slow mode

ARB: Agree

AWB: Don't see this as a problem for JITs

BE: (asks implementors what they're doing to solve)

- SpiderMonkey uses token stream rewinding and it covers all the cases

PL: We haven't gotten this far

ARB: This has been brought up with the team before, and token stream rewinding has been off the table

- Don't want to do token stream rewinding in V8, various issues (e.g. with Blink interaction?)

BE: If it can be done and satisfies performance constraints, then engines should consider

ARB: Compiling functions separately, ie. if function in parameters, compile sep. More than just rewind, would have to record all the functions, may be done with them, may not be done with them

MF: Can we see a proposal/solution?

ARB: Make it an error to have a "use strict" directive in a function with a non-simple parameter list.

YK: Should implement ES6 as written while this is resolved.

- Don't block implementing default parameters on this
- Just a suggestion

BE: "more of a guideline than rule"

AWB: You could make the restriction more specific: only disallow functions that declare "use strict" and contain a function in their formals list

ARB: Could be that only error when "use strict" inside function would change the mode, as the outer mode might already be strict

YK: refactoring hazard.

BE: Jason Orendorff says "works for me"

AC: don't like this suggestion, I don't want to impose an error because an implementor couldn't make it work

Discussion, re: hardship of implementation

YK: Don't like that end developers have to know why this won't work, b/c implementing hard.

BE: implementors are considered second to developers

AWB: previously, sloppy mode was restricted to simple parameter lists.

BE: No micro modes: no runtime semantic differences due to

AC: Chakra implements with rewinding

YK: objection: semantically this is the wrong thing

DD: but we're moving to all-strict world, so making the mixed mode cases more painful might be ok

DH: This is such an edge case that I don't tihnk there is an adoption risk.

- "use strict" retroactively providing subtle differences in the parameter list is a gotcha
- preventing diversions in the semantics to close window of confusion

BE: Suggestion fixes the right-to-left violation

DH: Could enumerate all those cases and make errors, but much harder.

- Fixes, but means new rule

YK: Poisioning `function () { "use strict" }`
BE: use top level "use strict"

YK: Not realistic

BE: Andreas reduced list to parameter lists that contain:

- functions
- arrows
- future "do" expressions

Discussion re: "use strict" expectations

YK: Prefer the stricter rule

DH: appear to have consensus on Andreas' solution

AWB: What exactly do we mean by "a function with a 'use strict'; prologue? Are we only talking about "strict transitions" (i.e. functions that declare "use strict" that are in non-strict contexts) or do we mean "use strict" in *any* functions.

MM: Functions that have a "use strict" know that they are strict, whether they are "pasted" or typed

- Taking the strict function out of strict context may cause errors as a result of code execution that previously expected strict context, which is a refactoring hazard

MP: Another (less difficult/serious) case where strict-function-rewinding is relevant is in the case of named function expressions. e.g.

```
(function static() { "use strict"; });
```

Ref: https://bugs.ecmascript.org/show_bug.cgi?id=4243

Discussing implemention strategies

- checking for duplicates
- eval, arguments
- yield, let

BE/AWB: existing semantics imply some right-to-left checking

BE: Non-simple is:

- default
- destructuring
- rest

(Anything that is *not* a list of identifiers)

YK: Need to acknowledge that we'll need to tell people to *not* "use strict" in functions, if they use non-simple parameters.

Acknowledged.

The solution is just use top level "use strict"

DH: We'll fail to explain this at the level of enumerating ES6 features they can or can't use in this case. The outcome is something like "just use top-level strict, or iife strict"

- Would like to revisit

YK: Not just this edge case: ES6 and strict mode has created this weird window.

AK: Functions in the parameter list might become strict if the function is "use strict"

DH: `this` behaviour changes
YK: Ultimately breaking "use strict" for function

AWB: Any function whose mode (lost track)

Discussion about ES6 feature adoption.

RW: disagreed with error when "use strict" occurs locally, but outer mode is already strict (ie. no mode change)

Discussion about developer expectation.

AWB: there is another alternative spec function that is more restrictive: ContainsExpression

RW demures ("You might say I am not willing to die on this hill.")

### Conclusion/Resolution

- Make it an error to have a "use strict" directive in a function with a non-simple parameter list.
- Early error
- No matter what mode you were already in
- When people want to use local "use strict", doing it b/c they want to know that this is always strict, no matter where it ends up.
    - Applies to all kinds of function/generator syntax
- IsSimpleParameterList http://www.ecma-international.org/ecma-262/6.0/#sec-function-definitions-static-semantics-issimpleparameterlist

## 6.9 Reconsidering the Number.prototype-as-an-ordinary-object change

(Daniel Ehrenberg)

DE: This change breaks the web (specifically: mootools calls `Number.prototype.valueOf(Number.prototype)`)
https://esdiscuss.org/topic/number-prototype-not-being-an-instance-breaks-the-web-toohttps://github.com/mootools/mootools-core/issues/2711

Solution: roll back the change.

BE: Number, String, Boolean should be

MM: To be clear: Date.prototype will be an object, not Date; RegExp.protototype will be an object, not RegExp. Every built-in constructor introduced after ES5 will have a plain object prototype.

MM: Prefer we do Number, Boolean, String together

### Conclusion/Resolution

- Boolean.prototype, Number.prototype, String.prototype rolled back
- File a spec bug?

# 6.12 Spec defacto String.prototype.trimLeft/trimRight

(Sebastian Markbage)

https://github.com/sebmarkbage/ecmascript-string-left-right-trim

SM: Already shipping in all major engines. Controversy: what to call it?

JHD: in JavaScript, left-to-rightness of strings is a rendering artifact. The conceptual "first" character of any string, whether LTR or RTL, is index 0, and the "last" is index length minus 1. Thus, in JS, left/start/index zero are the same, and right/end/index length minus one are the same.

DH: These are extremely common in programming languages

Discussing semantics of "left", "right" and "start", "end"

DH: There is precedence for JavaScript interpreting the word "right" to mean "the end of a sequence", from ES5:`Array.prototype.reduceRight`

## Conclusion/Resolution

- Approved for Stage 1?

# REVISIT: 6.11 The scope of "use strict" with respect to destructuring in parameter lists

DH: The only thing that "use strict" can cause (in ES5) is an error (to the left)

- Our resolution was to say, "the use strict prologue in functions is now defunct"
- An alternative, w/ better outcome: revise "use strict" to no longer affect *anything* to the left.
- A back-compat change, because it only allows things that were previously disallowed.
- no semantic difference

AWB: There are implementation semantics, w/r to parameters

- extra scope contour for eval context
- duplicate parameters

WH: How does it affect hoisting checks?

AWB: Hoisting rules in annex b don't apply if there is decl of same name (need to double check)

MM: How did we resolve names introduced in the function head vs names introduced in the function body, w/r let/const

AWB: disallowed

Discussion, re: strict in blocks?

AWB: need to check this against the existing function declaration instantiation, re: strict/non-strict checks

STH: Expressions in parameter list are not in strict mode.

- Enter strict mode due to prologue at beginning of function, does put parameter list expressions in strict mode

YK: Bigger picture: ES5 introduced the concept of a strict function.

- The proposal before is honest about admitting that's not a thing
- Sam's proposal ignores that and introduce's a new thing

Discussion, re: expectations of strict mode in functions

AWB: parameter with an initializer that uses `this`

- Assuming that's called with nothing in that position, what is `this`?
- If strict, `this` is undefined
- If sloppy, `this` is global object

```
"use sloppy";
function f(foo = this) {
  "use strict";
  return foo;
}
f();
// what is the value of this expression? the global object? or `undefined`?
// w/r to Sam's proposal (eliminating right-to-left strict-ness effect)
```

YK:

With Sam's proposa, this would return `[undefined, Global]`, despite expecting `[undefined, undefined]`

```
(function(foo = this) {
  "use strict";
  return [this, foo];
})();


DH: In ES5, there was a fundamental confusion about "use strict" in functions. This was excacerbated by es6. In the fullness of time, TC39 is saying, the directive prologue should be used:

- Global "use strict" (being careful of script concatenation)
- Top-level IIFE "use strict"
- module code

MM: We should be explicit w/r to "top level strict mode", in that we actually mean "top level iife with strict mode"—to avoid the concatenation problem.

Advice: "Just use modules".
```

```
#### Conclusion/Resolution

- The previous consensus remains



## REVISIT: 6.7 new & GeneratorFunction

AWB: (answers to questions from yesterday)

- Generator functions are new-able
- Generator methods are new-able
- Implicit body


```js
function * g() { this.foo }

x = {
  *f() { this.foo }
};

new f(); // ok
new x.f(); // ok
```

Relevant SpiderMonkey bug: https://bugzilla.mozilla.org/show_bug.cgi?id=1166950

DH: Bare generator function call behaviour is correct?

AWB: Confirm

Updated:

| S | I | Code |
|---|---|---|
| X | X | `{ *foo() {} }` (no [[construct]]) |
| X | X | `function * () {}` (no [[construct]]) |
| ? | ? | `function * () { this }` is exactly like `function() { this }` |

AWB: Concern about removing new:

- Implemented to allow it without throwing
- No TDZ on `this`

Acceptable risk?

BT: Implementors on both V8 and SpiderMonkey have shown interest in making generator not newable
```

### Conclusion/Resolution

- Spec change: generator functions and methods do not have a [[construct]] trap, so `new` throws

## Process Document discussion (to settle things once and for all)

(Domenic Denicola)

DD: (Showing https://tc39.github.io/process-document/)

Reviewed:

- https://github.com/tc39/process-document/pull/1/
- https://github.com/tc39/process-document/pull/2/

AWB: w/r to reviewers, committee should assign reviewers.

DD/RW: reviewers attached at Stage 1. Stage 2 entrance requires review

RW: Propose: At Stage 0, reviewers get "attached", but non-binding. (basically, it's a "champion group")

... Next thing...

## What is an "implementation"?

Discussion re: Stage 4

- Important to get feedback from engaged implementors

RW: Concerns about flag/no flag

DD: Must be unflagged

RW: Disagree, will result in stonewalling from implementors

DH: Transpilers should count

Discussion, re: experiences had with other new features.

"Significant in-the-field experience with shipping implementations, such as that provided by independent VMs"

Stage 3: "Will require feedback from implementations and users"

### Conclusion/Resolution

- need links to Domenic's commits

## 6.8 SIMD.js: Start the process to move towards Stage 3

(Dan Gohman, John Mccutchan, Peter Jensen, Daniel Ehrenberg)

Slides

DE: (introducing topic)

- Boolean vectors Previously, the result of SIMD.Float32x4.greaterThan was a SIMD.Int32x4 vector, with -1/0 for true/false Now, new boolean vector types, e.g. SIMD.Bool32x4, represent boolean results and values Used for select and logic operations More efficiently implementable on some architectures Not simply Boolx4 because the registers in implementations may be represented differently based on width.

- Unsigned operations

Unsigned comparisons Unsigned saturating add/sub Unsigned extractLane No changes needed for constructor, replaceLane because coercion will wrap unsigned values to the appropriate signed value No separate unsigned type

- Postponed features

Float64x2--we couldn't find an important use case with improved performance Int64x2--Not needed due to boolean vectors, and really not needed because Float64x2 is out selectBits--minimal utility due to select, and efficiently implementable in terms of other boolean operations

- sumOfAbsoluteDifferences replacement

widenedAbsoluteDifference, unsignedHorizontalSum, absoluteDifference Seeking implementation feedback: applications and benchmarks Replaces sumOfAbsoluteDifferences (slow on ARM)

- Other spec changes

Homoiconic toString() `SIMD.Float32x4(1, 2, 3, 4).toString()` => `"SIMD.Float32x4(1, 2, 3, 4)"` Shift operations max out at 0/-1, rather than wrapping around Ops like reciprocalApproximation are loosely specified, like Math.sin Removed operations on DataView--TypedArray ops suffice Operations on subnormals may flush to 0, unlike ES scalars Various minor spec bug fixes
AWB: w/r toString. The approach shown is a new precedent

DD: Similar to Symbol

AWB: Will this be the new way

BE: value types with literal form should convert, but this isn't value types. I like it.

AWB: Should other new things have this toString() output? Map & Set?

(moving on)

- Strong type checks on lanes

Lanes are required to be Int32s and not implicitly coerced

(See 5.1.2, http://littledan.github.io/simd.html )

AWB: for WebIDL, we advised to use normal ES coercions

DE: Considered specified getX, getY, etc.

DH: I don't know if it's that important to coerce or check.

BE: Doesn't want this to be an enumerated type. Symbols?

AWB: The advantage is that Symbol is not coercible and you only accept those that you've defined.

DE: if we had a Symbol API, we'd still want an extract lane API

STH: numbers are the right thing here

AWB: ToNumber, because that's what ES does

- Actually ToInteger
- ToLength

WH: Note that the proposal is a bit inconsistent in validating numeric inputs. For example, shifts use ToUint32 for the shift count, which turns -1 into a huge shift amount. ECMAScript's built-in shifts treat the shift amount mod 32, while the proposed ones treat it mod 2^32.

DE:

- load and store operating on any TypedArrays

load and store take TypedArrays as arguments and permit array element type mismatch with SIMD type

WH: Reading spec, cannot load and store booleans. Looks intentional.

DE: Fails on booleans, intentionally.

- Intentionally abstract data type
- Want concrete data type, call select

WH: Fix other places in the spec for booleans, such as the statement that bitwise cast can cast any SIMD type into another SIMD type.

AWB: (requests justification for extracting values of a different type than the type of the Typed Array)

DE: DataView doesn't really work, in an asm context. Operations on DataView accept endianness where this has to use native endianness to be efficient

AWB: if int8 array, extracting floar 64, does it

DE: No, it is element-aligned. The hardware supports this.

DG: (phone) confirms hardware support

AWB: If you're talking about a float32array and you're extracting int16's, that just seems like a type error.

DE: Better?

DH: Go back and change array buffer, can't do that

- ArrayBuffer was defined as opaque
- Everyone wants to change rthat now
- No real world constraint

DE: how is opacity enforced?

(no answer yet)

WH: Some ArrayBuffers are indeed opaque. Restricted exposure for security reason

DH: TypedArray views on ArrayBuffer, detached

- Allow operation on ArrayBuffer
- We'll have to break the invariant

BE/DH: what they're doing will work

WH: Not a pointless invariant. Move on.

(moved on)

Questions for Committee

- Should wrapper constructors be explicitly [[Construct]]-able, like Number, or not, like Symbol?

BE: have to call with new to get the object

AWB: Symbol is just odd because concern that `new Symbol()` would be used to get a generative object.
DD: Necessary to impose a rule for creating a Symbol

- No literal form? No wrapper

DE: if you call construct, if NewTarget undefined, construct wrapper

AWB: Minimize new patterns. Overloading constructor is not new.

AK: Why this route instead of Symbol route?

DE: Symbol is special throwing on new? But maybe Symbol is the new way

DD: If no literal, then no `new`. If a literal is added later, then re-open `new`
BE: (explains wrapper history, aggree with DD)

(moving on)

Spec Language Innovation Acceptable?

- Rest parameters
- SIMD as a spec meta-variable

AWB: There was use of rest early in ES6, but taken out

BE: parameters that were optional

AWB: implies JS level parameter list

DE: no

Discussion, re: spec mechanisms

(moving on)

RW: Recommend moving this to a separate spec, similar to Intl (Ecma-402). (Note that this was considered ideal by the champions as well, despite the opposition from other members).

DH: disagrees

WH: Also disagrees with separate spec. This defines primitives tightly bound into the core of ES, with serious interactions and precedence-setting with other ES work such as value types.

DE: Issues surrounding value type definitions, but don't want to to wait for value types. Don't want to be blocked and separate spec ensures that

(moving on to implementation status)

Firefox Implementation Status

In Firefox nightly:

- Float32x4 and Int32x4 entirely implemented and optimized in JavaScript (regular and asm.js) on x86 and x64.
- Missing boolean vectors
- Other SIMD types (Int16x8, Int8x16, Float64x2) partially implemented in the interpreter only (ergo not optimized). The newer APIs (SAD) haven't been implemented yet.
- All SIMD types are implemented as value objects, at the moment.

Microsoft Edge Implementation Status

- Majority of SIMD.. APIS supported.
- Some of the new APIS need to be implemented such as ExtractLane and ReplaceLane, and unsigned operations
- Asm.js optimization is complete (minus new api support).
- Non-asm.js optimization we plan to start soon.

V8 Implementation Status

- Intel object implementation will not be used for V8 and work has started to implement value types from scratch. Intel code generation may be rewritten to the new model.
- Bill Budge has added a Float32x4 type with correct value semantics and basic operations, without acceleration, behind a flag.

Specification Status

- SIMD.js Specification v0.7.2
- Updated polyfill and tests validate all operations, basic value semantics
- SIMD.js is ready for reviewers and and editor comments/signoff
- Hope to move to Stage 3 in the September meeting

Requesting reviewers

WH: purpose of spec, disagreement whether to support only use cases experienced or useful with a more ecmascripty orthogonal feel. For example, the spec currently can load int8, uint8, int16, uint16, int32, uint32, but it can only extract the first five of them (can't extract uint32 even though can extract int32 or uint16). Min and max work on floats but not on integers, even though there is no hardware obstacle to do so and even though there are much more complex intrinsics defined such as unsignedAbsoluteDifference.

DE: support unsigned operations on int16 and int8

BE: waldemar wants max on integers that can

DH: SIMD building clear layer to hardware

WH: want consistency:

- integer max
- uint32
- for usability, uint8 SIMD pixel values printing liked TypedArrays as 255,255,0 instead of -1,-1,0.

WH: Diverged from TypedArray,

AWB: TypedArray both have int32 and uint32, JS programmer expects that

BE: argument to be consistent

DE: TypedArray is the model going forward, with the exception of Uint8ClampedArray

AWB/BE: Yes

WH: treatment of denorms non-deterministic

DE:

- operation and flush to 0
- operation only

WH: Does the hardware flush both inputs and outputs to 0, or is there some major hardware that flashes only inputs or only outputs to zero? (an arithmetic operation might take denorms as inputs, or it could produce a denorm output even though all inputs are norms)

DE: Both inputs and outputs

WH: [later in the discussion] Intel has FTZ and DAZ bits which control these separately, so they can be independent.

(Difference between Arm and Intel)

DG: Arm 64, arm v8 corrects this

MM: this non-determinism is actually just difference between platform

WH: The SIMD spec doesn't state that. It could differ operation-by-operation.

DE: The spec can't state that.

WH: Really? Why?

Discussion re: nan encoding

DE: http://www.ecma-international.org/ecma-262/6.0/#sec-setvalueinbuffer (step 9.a)

AWB: (explaining why spec says this)

DH: we should change the def of SameValueZero to say that two different bit patterns for a NaN are not equivalent

DE: create a data property and set value to one bit pattern, non-writable, non-configurable, call Object.defineProperty with a different NaN, then do [[Get]] and you'll receive old bit pattern

DH: No mutation, but incorrectly states that change was successful

DE: spec says a platform will have one endianess or the other; this can be applied to denorms. Spec dependent.

WH: What do relational operations (.equal, .lessThan, etc.) do when they see a denorm? treat as zero?

DG: treat as zero

WH: So then what will === do? Do they equate all distinct denorms to zero (which would be really weird as ES behavior)? Are you going for the fast implementation of === or the precise implementation?

DG: === does not equate denorms to zero, even on platforms that flush denorms. It can be slower than .equal.

http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4610935

DE: One outstanding question was whether this work should be done in a separate specification.

RW: Yes, I had suggested that. But you said something about a polyfill for SIMD.js?

DE: It is incomplete

BE: It can't express value object; it can't intercept == or ===
RW: Alright, then it is no longer a suggestion of mine

Unsigned Types? Argument for them?

Consistency with TypedArray

DH: I want to know more about what SIMD use cases are, operations should map to need

WH: want unsigned types:

- consistency with typed array
- printing (want 255, not -1)

(not an explosion of the API)

DE: fewer function names, more function objects.

AWB: lot's of domains with similar conventions

- once integrated, it becomes part of the spec as a whole and will require understanding of these irregularities
- an argument *for* separate spec

DE: it's a large thing, mostly irreducible

JHD: non-specialists are going to write this stuff by hand.

(Argument to make the API more like JS)

DE: prefer to keep it as is

YK: So a specialist doesn't need the unsigned types?

- then don't worry about use by non-specialists

DH: Either accept Waldemar's argument, or state case based on historical precedence. If no consistency, then don't care.

WH: Note that I'm not alone with that argument. I don't want this to become personal.

### Conclusion/Resolution

- new throws

## - not separate spec

- spec mechansisms: Allen, Brian, Rick and Dan to work through
- Reviewers:
    - Waldemar Horwat
    - Jordan Harband
    - Brendan Eich
    - Mark Miller

# July 30 2015 Meeting Notes

Allen Wirfs-Brock (AWB), Sebastian Markbage (SM), Jafar Husain (JH), Eric Farriauolo (EF), Caridy Patino (CP), Waldemar Horwat (WH), Istvan Sebastian (IS), Mark Miller (MM), Adam Klein (AK), Michael Ficarra (MF), Peter Jensen (PJ), Domenic Denicola (DD), Jordan Harband (JHD), Jonathan Turner (JT), Paul Leathers (PL), Chip Morningstar (CM), Vladimir Matveev (VM), Ron Buckton (MS), Brian Terlson (BT), Alan Schmitt (AS), Ben Newman (BN), Mohamed Hegazy (MH), Abhijith Chatra (AC), Tom Care (TC), John Neumann (JN), Dave Herman (DH), Brendan Eich (BE), Daniel Ehrenberg (DE), Dan Gohman (DG), Andreas Rossberg (ARB), Rick Waldron (RW), Mike Pennisi (MP), Akrosh Gandhi (AG), Jonathan Sampson (JS)

## 7 Test262 Updates

(Brian Terlson, Mike Pennisi)

Slides

MP: Introduction

See slides

WH: (re: http://jugglinmike.github.io/presentations/2015/tc39-test262-update/#7 ) question, re: es6draft, ecmascript spec, before/after publication?

?: es6draft ? ECMAScript 6 draft

BT: Ongoing. Could also add Chakra bugs

MF: raw flag? Does this prevent issues other than directive prologue issues?

MP/BT: "raw" means: don't touch the file, just run it. No access to assert features, useful for syntax only tests

[discussing distinction between early errors and parse errors that test262 does]

http://jugglinmike.github.io/presentations/2015/tc39-test262-update/#12

AWB: These are a specification artifact. An implementation doesn't have to do parsing and early errors in separate passes.

WH: This becomes visible when there are multiple errors, some in parsing and some early errors. Does the spec designate which error should be reported when there are multiple of these?

AWB: No. An implementation is free to report any of these errors.

AWB: Distinguishing early and parsing errors in tests is overspecifying.

(re: website improvements)

BT: current test262 site does not have ES6 collatoral.

- Implementing ToLength appears to lock up the website
- Alot of work needed to get the test262 site into a functional state
- Is the work something that Bocoup is going to do?

MP: Nearing end of time, but ideally would like to take on this work

BT: Then we need to find a "resource" for this project

- Enumerating outstanding issues with test262 website

YK: Considered running tests in iframe or worker and timing out?

BT: Harness uses iframes

Discussion of implementation details of test harness

BT: Requirement of new site: use web workers

Moving on to http://jugglinmike.github.io/presentations/2015/tc39-test262-update/#15

MP: (first example) TypedArray, 9 sets of the same tests. The tests might start in sync, but it's hard to maintain over time.

- Typed Arrays (9)
- Destructuring Assignment (3)

    - AssignmentExpression evaluation
    - ForIn/OfBodyEvaluation
        - for..in statement
        - for..of statement
- Spread Operator (4)

    - ArrayLiteral: [...x]
    - ArgumentList

        - SuperCall: super(...x)
        - CallExpression: f(...x)
        - MemberExpression: new f(...x)
- Expressions & Declarations

    - Function
    - Generator
    - Class

AWB: Similar to wanting to share code in tests, implementations will want to share code.

- Look at how an implementation and its tests go wrong?
- Tests that would catch the most common ways the implementation abstractions might go wrong

YK: Instead of testing all the things, find the things that might fail

- things that look similar, but are subtley different: test that.

MP: (gives an example of how strict and non-strict tests are automattically run, unless flagged)

WH: Value in testing exhaustively when the number of cases is reasonable. Per the presentation, there are only 18 contexts in which destructuring is used, which is reasonable for full cross-product testing.

- Re: Allen's point, there are subtle things that can go wrong. I can imagine an implementation getting destructuring right in most contexts but messing up some subtle aspect of it for, say, arrow function parameters.
- But still worthwhile to test try to be exhaustive

BT: Not suggesting that we won't test syntax in unique contexts

- More about the authoring
- For tests that cover identical syntax semantics and the only difference is syntactic context, then test once
- Why not sweet.js or an existing templating engine? We need to test syntax errors (and other negative tests)

WH: Just pick one that doesn't make you escape all of the ECMAScript syntax

Discussion, re: https://gist.github.com/jugglinmike/476bdb6dd69ffddaf9d2#syntax

AK: (concerns about useful output)

DD: most consumers clone the repo, so we need to make sure that we check the output as well

AK: I'm less concerned about running, but about the output that will inform me of the failure

BT: Don't want to check in the actual generated tests

- This proposal MUST make it easy to identify failures
- The test harness will write to disc the actual code that failed at the moment of failure
- produce, on-demand, the filled-in template

DE: For tests users, this can be automated?

BT: Yes.

## Conclusion/Resolution

- Continue exploring test generation

# Meeting Schedule

(John Neumann)

JN: will propose a 2016 schedule in september. Need to know if we'll be in Europe?

- Switzerland: Geneva, Lugano, Montreux, Lausanne (Geneva may not work: too many attendants and too expensive)
- Germany: Munich
- France
- England

Discussion planned for next meeting

- September in Portland, hosted by jQuery/Yehuda
- November in Santa Clara, hosted by PayPal

JN: Need to confirm what the 7th edition will contain

YK: Straight forward, whatever is Stage 4 goes in the spec

AWB: (enumerating general spec process timelines)

- submission
- review
- 60 day opt-out

YK: That's after January?

AWB: Approximately end of January

YK: January is submission, work backward to find deadline.

Discussion re: time needed for Brian to integrate the stage 4 features.

## Conclusion/Resolution

- Get proposals wrapped up if you want to get these features into the spec

# 6.4 Advance Async Functions to Stage 2

(Brian Terlson)

Slides

BT:

Updates from Last Time

- Complete spec available ( http://tc39.github.io/ecmascript-asyncawait/ / https://github.com/tc39/ecmascript-asyncawait )
- Removed `await *`
- Implemented in Babel
- Losts of positive feedback from the web

DH: Future proof for potential future syntax extensions

DD: Clarify not `await *`
BT: `await *` was not useful. If there is a proposal for a useful semantics, then good, but not in this proposal.
Questions

- Async Arrow Function
  - async (a, b) => await a + await b;
  - (a, b) @=> await a + await b;

DE: What's the issue with the first?

BT: `async` not a keyword and that could be a call: `async(a, b)`
MM: @ seems like a poor choice, but I don't have an alternate suggestion

AWB: only b/c "a"sync

- the `async` form defeats the conciseness

BT: I've worked out the grammar, it needs review, but it appears to work correctly

YK: Shouldn't have 3 sigils right next to eachother.

- hard to google a sigil or set of symbols
- a "grawlix" problem

BT: Opposed with the top grammar? (assuming the grammar works)

(no opposition, pending review)

DH: Would like to see more code illustrating actual use cases

DD: There's a lot of code in the wild that's been using the top form

DH: Any higher order combinator is where this will have the most use.

MM: No consensus on the second. Consensus on the first contingient on grammar validation

DH: Cannot make `async` a keyword, one of the most popular modules on npm is `async`
DD: `Promise.all` doesn't quite work because it's a combinator that takes promises, not functions that return promises
DH: The syntax that we've come up with for past features like `=>` map to an extremely high use case with hardship. Not clear to me that we have the full async callback picture
DD: should look into the ecosystem of Babel users and see where it's come up

Design Questions (continued)

- Newing an async function:
  - Error
  - Promise-for-instance

BT: Suggest "no" (no disagreement)

Design Questions (continued)

- Errors in Parameter Initialization

```
function* foo(a = (function() { throw 1; }())) {

}
```

```
let iter = foo(); // throws 1
```

BT: the equivalent in async functions:

```
async function bar(a = (function() { throw 1; }())) {

}
bar(); // as of the latest proposal, this throws 1

// Alternative semantics would have the error deferred:
bar().catch(cb => ...); // Possible semantics
```

DD: Wish this was caught for generators

AWB: We knew about it, but nothing can be done

MM: For generators, we have immediate throw

- Async function: throw in body breaks the promise
- think about *when* the throw happens
- what way is it reported?
- throw in generator body is reported with a throw
- throw in async body is reported by rejecting promise

YK: when use promise, you can catch all the errors into one place

AK: Arguing for throw immediately?

BT: Yes, found that error throwing default parameters don't happen (rare case?)

- Mark's point about multiple symmetry in play

CM: setup preparation part, consumption part. The generator setup might have error in setup, reported immediately

DD: Code will expect to receive the error via the catch path, not an immediate throw path.

- Most promise ops go through the API
- Devs can muck with the built-ins and produce errors

AWB: What if you put a proxy on an async function and the call trap throws?

MM: There's not a problem with the [[Call]] trap. If the [[Call]] trap in the Proxy throws, then you have a "throw".

AWB: Say the async function is actually a proxy wrapped around an async function?

MM: It's exactly the same hazard as if you hand someone a function that wraps an async function and the wrapping function throws.

DD: Problem is thinking of async functions as a separate category. Just a function

DH: Mark, I agree. There are errors we consider normal to happen in programs that you should deal with, and those where soemthing went badly wrong and you can't write code that has try/catch every where. A function call that throws an exception should follow the error handling model for the programming paradigm that we're in; since we're in an aync world, the programming model for handlng errors is `catch`
BT: want to be very clear about rationalization, so that same can be applied
to `async` generator.
AWB: The generator function returns its value before evaluating its body.

# - A generator is a function

MM: An async generator... I missed the rest of this

YK: Can you `yield` in a default parameter in a generator?
AWB: No, you cannot `yield` there because the object doesn't exist yet.
BT: We need to establish the rationale for the behavior of error handling in non-simple generator parameter lists

YK: Might've been a mistake?

AWB: Generators do what generator functions previously did in Firefox. We weren't sure if it should be changed. Different results:

- timing evaluation changes: happen on the first `next`, the values aren't the same values at the time of the call.

DD: Generators suspend immediately, `async` suspend on the first `await`

DH: All cases, expect that parameter default expressions execute eagerly, not that they execute at some time later

DH: Generators are all about order of execution. The fact that generator functions and async functions have different control flow expectations means that it's reasonable for their error handling behavior to differ.

Discussion about complications of the implicit `yield` in generators
DH: Don't need symmetry between generator and async

YK: Async function is just a function, but the generator is a different thing.

- When calling a generator, you're not calling a function as shown

DD: When calling a generator function, the first line of the function does not execute

DD/YK: Agreed.

YK: The consequence is that the refactoring doesn't make sense.

Design Questions Continued

- Await synchronous things synchronously
  - `await 1` does it trigger a turn of the event loop as in the desugaring?
  - allow synchronous await but ensure promise resoltion is async?
  - do neither and allow returning an already-resolved promise?

MM: describing the hazard of sometimes "sync" and sometimes "async"

??????????????Z??????????A??????????L????????????G????????????????? ??????? !????????????

:applause:

YK: The same hazard doesn't exist with await, because the callback model is changing which scope the function is called in, but doesn't exist in await.

MM: Agreed that the hazard to aync functions is smaller than that to Promises. The bugs will be both fewer and more subtle and harder to figure out.

DD: no cost to go back to the event loop

YK: not true.

- hot code is always awaiting some promise, could be optimized to consider inline-able. Never true if you always have to go back to the queue

MM: If an implementation can optimize, which is very hard and hazardous

YK: I'm making a different point. If you say. "you are not allowed to go back to the event loop" and if you discover that it is synchronous, then you can optimize by in-lining the synchronous code. In an ideal world, it is "just a loop", and a loop is fast.

JH: The relative harm: cost not zero, need to weigh

MM: Experience with `asap`, cost is worth it to reduce the hazard.

- When bugs do occur, very subtle and hard for programmer to figure out

YK: Hazard in the callback case is in pieces of code that are right next to eachother that appear to [...] Let me put it another way: the hazards you are talking about are based on assumptions that I would never make when writing a program.

MM: hazard as soon as mutated state is sufficiently non-local

YK: cannot rely on what interleavings are possible

BT: hazard here: order of your next turns change. not that callback could be async, but that your callbacks mayb run out of order. What Yehuda is saying is that people don't write programs with multiple Promises where they expect the promises to resolve at specific turns of the event loop.

MM: Two different hazards:

- The order of events on the event loop is one hazard
- the isolation of side effects is job that was executed up to the await point, completes before any code of the `await`
    - All invariants storing during the job

Mark, I need you to fill this in, we're moving too fast.

YK: Transaction committed before any await is called. This exists in Ember.

- Need to see an example of the invariant Mark is describing

MM: After the await point, the callee is starting with a fresh stack.

- Event loop programming paradigm that suspends an invariant must restore the invariant before the job ends.
- Illustrates with example of doubly linked list operation

MM: Event loop concurrency requires restored invariants before the end of the event loop turn

YK: Different from what's encountered in practice

DD: What's the position?

YK: Discussing already-resolved-promises

DD: `await 1` is not the same as `await Promise.resolve(1)`

- If you have a promise, you've said that the value is async and breaking that is crazy

MM: Found with the Q.asap bug: I wrote my code, tested my code, then my code had bugs. Result of code running out of turn with test code.

YK: consistency is absorbed by syntax

MM: disagree

- The bug was directly the result of assuming the invariant will be resolved.

Discussion re: invariant restoration on event turn completion

```
// BT:
foo().then( function() {
   /* 1 */
});
/* 2 */
// Which is first
```

```
async funcion foo() {
  /* 1 */
  await x;
  /* 2 */

  // are these the same turn?
}
foo();
// HAZARD: setup state required by /* 2 */
```

Currently, `await` is always async.
JH: Concern that the notion of asynchrony bound to...?

MM:

JH: predicts that await means async to developers. await is composition.

DD: yield is composition

MM: asynchrony is mandated by the async keyword

JH: is that adequate? `async` means asynchrony, `await` does not mean asynchrony, just pause the execution.
YK: Come back to this?

Agreed. Next turn of the event loop.

JHD: Agree that `await x` always means `await Promise.resolve(x)`?
Some agreement.

YK: Doesn't fully describe the trade off

MM: Implementation prohibited from making optimization

JH: can we put together a program that illustrates the hazard?

```
// BT:
foo().then( function() {
   /* 1 */
});
/* 2 */
// Which is first

async funcion foo() {
  /* 1 */
  await x;
```

```
  /* 2 */

  // are these the same turn?
}
foo();
// HAZARD: setup state required by /* 2 */
```

YK:

```
let x;
foo().then( function(val) { x = val; /* 1 */ } );
// what is x?
```

JH: Is that an issue in asap? or were there other issues?

```
// BT & YK:
async function foo() {
  // relies on y being `1`
  await x;
  // relies on y being `2`
}

let y = 1;
foo();
y = 2;
```

MM/DD/YK: discussing the hazards above.

AK: Different conditions, different behavior.

Discussion about race conditions in JS

DH: largely around the interleaving asynchronous events

- Mark wants this to be limited

MM: confirms

YK: When type `await`, I expect unpredictability

MM: unpredictable interleaving of job

DD:

```
function dStuff() {
  if (window.state === 1) { doA(); }
  else { doB(); }
}

async function foo() {
  doStuff(); await x; doStuff();
```

```
}

window.state = 1;
foo();
window.state = 2;
```

MM: impossible to reason about a program without knowing something about what is going on.

- testing in sync case, and it works
- execute in async case, and it fails

DD: Always have a consistent state on the next turn

Discussion, restating positions back and forth.

- Promise cancellation:
  - Depends on cancellation at the promise level
  - Urge promise champions to consider ergonomics with async functions

BT: not going to specify cancellation in this proposal

MM: Not sure that base Promise is going to change?

DD: Talk offline.

- Await at top-level
  - Can you await at the top level of a module?
  - Useful, especially in Node

DH: Issues.

YK: Previously, could assume that top level ran start to finish, if many pieces have setup await

AWB: this changes the entire model of module execution

BT: modules are wrapped in async function which called immediately

MM: after top level await point, import/export, are these effected?

AWB: initialization happens in the order of import/export but before the module body starts

MM: only "when" the execution happens

AWB: This could be misunderstood

DH:

```
import ... from ...;

let m;

if (dev) {
  m = await local.import("lib-dev");
} else {
  m = await local.import("lib-production");
}

export let x = 17;
```

DH: Requires module be async.

Unresolvable if node remains sync at the top level.

(break)

DH: we need to do more work offline.

BT: is top level await worth pursuing?

DH: Addresses a definite need, eg. dynamic module loading

BT: Flesh it out?

DH: No, this needs to be designed in concert with loader. Suggest decoupling to unblock async/await

BT: Confirms.

- Waldemar's grammar concern?

WH: (presenting now)

```
async(a=await/
```

- According to cover grammar, this is a division symbol
- Is await a keyword or identifier here?

BT: parsed like a keyword, but there is a static semantics that says any occurence of await expression is an error.

```
async(a=await/x, b=c/i) => b;
```

- Parse cover grammar, don't know that you're in a arrow function. This lets await sneak through as an identifier
- When you do the reparse, `/x, b=c/i` is treated as a regexp

MF: The reparse says it has to match this additional grammar, same set of tokens

WH: It's not clear in the spec.

BT: Yes, the await would appear as an identifier until you apply the cover grammar.

MF: (reads from ECMAScript 2015 specification, section 12.2.1.1

MF: if await is always an await expression, this would fail to reparse, resulting in a SyntaxError

BT: Correct, expected.

BT/WH to have offline discussion re: grammar.

BT: Those that know execution model should review this.

## Conclusion/Resolution

- `async (a, b) => await a + await b;` contingent on grammar validation
- `async` functions: no construct slot
- errors occuring in parameter initialization take the catch path
- Approved for Stage 2
- Reviewers
    - Waldemar Horwat
    - Jafar Husain
    - Yehuda Katz

# 6.5 Proposed Changes to Observable API

(Jafar Husain)

https://github.com/zenparsing/es-observable/tree/zenlike

Slides

JH:

Conclusions after 45 Issues and Impl Work

- Generator should not be used Sink
- Sync Subscription Affirmed Necessary
- Can implement EventTarget using Observable

Using Generators as Sinks

Issues:

- Type Fudging (would like implicit {done: true})
- Priming Problem
  - issue with implicit yield
  - considered a decorator, but likely a footgun
- Return invoked on unsubscribe
  - mistake because it conflates two concepts: the need to free scarce resources and the need to signal iteration has completed
  - the equivalent to breaking out of a for loop,

YK: Why does it end up not being important to generators? (dispose being separated from return)

YK: Straight forward to add a `dispose()` method to Generator instance objects http://www.ecma-international.org/ecma-262/6.0/#sec-properties-of-generator-prototype
JH: Believe adding `dispose()` is good

DD: And/or remove `return()`. To remove the try/finally that's implicitly added around a for-of loop
Discussion, re: try/finally

JH: even if we fix the return/dispose conflation, the priming problem is still a compelling reason not to use generators as sinks

Using Generators as Sinks (continued)

Issues: ...

- Multiple unsub methods

Proposed Solution: Observer Type

(get bullets from slide)

Advantages of Observer Type

(get bullets from slide)

Proposed Observable Type Changes

(get code from slide)

DD: Strange to hide a method by using well-known symbol

- if sync subscription is allowed, just make it easy

WH: What does Symbol.observer return? (Not referring to the symbol, but to the method to which this is the well-known symbol-as-computed property name)

JH: Nothing

DD: What happened to the flush that we discussed?

JH: In progress

MM: What is the `Object?` in `Observer` type?
JH: Allowed to return something

(Should be "any")

MM: Why not just support the Observer (in `subscribe` method)

- Prefer two separate methods

JH: previously had a forEach that returns a promise, still exists, just not shown

Discussing a swath of code on jsbin

Sync Subscription Affirmed Necessary

- Necessary to build EventTarget using Observable (get rest of bullets from slide)

Jafar moves through slides too fast.

DD: Would prefer something simpler than Symbol.observer, instead just call it something obvious

MM: Name it such a way that the name is clear to the user (no suggestion)

Discussion, re: await observable.

All scalar combinators produce promises or observables

### Conclusion/Resolution

- Approved for Stage 1 (which means: nothing changed here)
- Offline discussion, re: sync subscription

## 6.13 Advance Rest/Spread Properties to Stage 2

(Sebastian Markbage)

https://github.com/sebmarkbage/ecmascript-rest-spread

SM:

Static Properties vs. Computed Properties

SM (slide): question on evaluating computed property keys multiple times on the LHS of a destructuring assignment?

SM (slide): question when a "rest" destructured object param throws (via getter, for example), should `rest` be undefined, or a partial object. or, should the getter be copied rather than evaluated

```
try {
  var throwingObj = {
    x: 1,
    y: 2,
    get z() { throw myError; },
    w: 4
  };

  var {x, ...rest} = throwingObj;

} catch (err) {
  x; // 1
  rest; // undefined (could be { y:2, w: 4 })
}
```

MM: possibilities?

SM:

- undefined
- { y: 2, w: 4 }

MF: also `{}` and `{ y: 2 }`
MM: further semantics need to be explored

## Conclusion/Resolution

- Approved for Stage 2
- Reviewers

### - Andreas Rossberg