Minutes of the: ***48th meeting of Ecma TC39***

in: ***Portland, OR, USA***

on: ***22-24 September 2015***

1 Opening, welcome and roll call

1.1 Opening of the meeting (Mr. Neumann)

Mr. Neumann has welcomed the delegates at JQuery in Portland, OR, USA.

Companies / organizations in attendance:

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	Gorkem Yakin	Microsoft	Member	
6	Jeff Morrison	Facebook	Member	
7	Lee Byron	Facebook	Member	
8	Sebastian Markbage	Facebook	Member	
9	Domenic Denicola	Google	Member	
10	John Neumann	Multiple	Member	
11	Mark Miller	Google	Member	PHONE
12	Dan Gohman	Mozilla	Member	PHONE
13	John McCutchen	Intel	Member	PHONE
14	Istvan Sebestyén	Ecma-International		PHONE
15	Jafar Husain	Netflix	Member	
16	John Harband	Airbnb	New Member	
17	Eric Ferrainola	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	Ben Smith	Google	Member	

25	Adam Klein	Google	Member
26	Allen Wirfs-Brock	Mozilla	Member
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

1.3 Host facilities, local logistics

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

2 Adoption of the agenda (2015/042-Rev1)

The agenda was approved as posted on the github:

Agenda for the: 48th 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 July 2015
4. Report from the Ecma Secretariat
    i. Report from the GA and CC
5. Proposals for Future Editions of ECMA-262
    i. ✓ Proposal: Shared memory and atomics (Lars T Hansen, Mozilla) Proposal materials on github
    ii. ✓ SIMD.js Stage 3 proposal (Daniel Ehrenberg, John McCutchan, Peter Jensen, Dan Gohman) draft specification presentation
    iii. ✓ Async Functions Stage 3 proposal (Brian Terlson) draft specification
    iv. ✓ Updates on class-properties proposal (Jeff Morrison)
    v. ✓ Updates on decorator proposal and discussion of intersection of private state and decorators (Yehuda Katz)
    vi. ✓ Proposal: call constructor (Yehuda Katz)
    vii. Proposal: auto-super in class constructors (Yehuda Katz)
    viii. Proposal: `super()` sugar in methods (Yehuda Katz)
    ix. ✓ Trailing commas in function parameter lists (Jeff Morrison)

x. ✓ Proposal: String#padLeft / String#padRight (Jordan Harband, Rick Waldron)
xi. ✓ Proposal: Object.values / Object.entries (Jordan Harband)
xii. ✓ Proposal: String#matchAll (Jordan Harband)
xiii. Updates on rest properties proposal (Sebastian Markbage)
xiv. Function.sent metaproperty - call for reviewers
xv. ✓ Exponentiation Operator update (Rick Waldron)

6. Updates on Loader (Dave Herman)
7. ✓ Meeting work sessions and plenary meetings
8. ✓ Test262 Updates
9. ✓ ECMA-402 3rd Edition, 2016
10. ✓ Tooling Updates (Brian Terlson)
11. Date and place of the next meeting(s)
    - November 17 - 19, 2015 (San Jose - Paypal)
    - January 26 - 28, 2016 (San Francisco - Salesforce
    - March 29-31, 2016 (San Francisco - Mozilla)
    - May 24 - 26, 2016 (Europe or San Francisco/Google)
    - July 26 - 28, 2016 (Redmond - Microsoft)
    - September 27 - 29, 2016 (Los Gatos - Netflix)
    - November 29 - 30 - December 1, 2016 (Menlo Park - Facebook)
12. Closure

3 Approval of minutes from July 2015 (2015/040)

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 regarding the ISO/IEC JTC 1 fast-track of ES6 the current status of the discussions with the ISO Secretariat is the following:

1) We talk here about 3 standards (ECMA-262 "ECMAScript" - the "problem case"; ECMA-402 "Internationalization API" - which is a short standard but it is very right now linked together with a ECMA-262 Edition; 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, whatever time it takes.

2) ECMA-262 6th Edition has been approved by Ecma on June 17. It has been to large extent already implemented in Browsers and in other applications. It is an extremely popular standard, in the first months we had about 12,000 downloads, now it is back to monthly 6,000 downloads, and the HTML version has been assessed around 100,000 times so far. The earlier version of ECMAScript was downloaded constantly about 2,500 times per months. Ecma has a yearly total downloads of 90,000-100,000 standards / TR per year. So this gives an impression how important ECMAScript for Ecma is.

3) The volume of ES6 is about 600 pages vis-à-vis the earlier edition 5.1 which is 250 pages. So while the standard is very popular and people are happy, we obviously are getting a large number of bugs and corrections that we must be corrected before we even might think about a fast-track "synchronization". The feeling is that this situation will continue for the next 12 months or so, but it has to be carefully watched when we get to a "final" ES6. One will see.

4) In addition we have the ambitious, yet untested plan of TC39 to come out each year with new added functions, so the plan is that the "ECMAScript 2016" will contain the bug-fixes mentioned above plus some new features which became finished by a "freezing" deadline, which should be rather fast (early next year or so latest) if we want to approve it by the June 2016 GA. We do not know yet if in practice that plan will actually work or not. We will see...Maybe not, then automatically we have to shoot for a longer cycle, or may be yes...

5) So, if yes, then probably the best strategy is, not to "synchronize" each and every ECMAScript Edition, maybe we will only synchronize every other... We will see. But what would be important is a clear communication what Ecma Edition is synchronized with JTC 1 (and what is the corresponding Edition there) and what not, and why we are proceeding like that. Ecma has a similar procedure with the DART Scripting Language (ECMA-408). That standard is updated and approved in Ecma every 6 months, but the "Dart community" produces between the 6 months interval additional 2-3 versions. In our publication we clearly make then references which Ecma-Edition corresponds to which "Dart community" version.

6) Ecma and ISO/IEC JTC 1 will discuss the situation again at the JTC 1 Plenary meeting in Beijing at the end of October 2015.

7 Date and place of the next meetings

- November 17 - 19, 2015 (San Jose - Paypal)
- January 26 - 28, 2016 (San Francisco - Salesforce)
- March 29-31, 2016 (San Francisco - Google)
- May 24 - 26, 2016 (Europe)
- July 26 - 28, 2016 (Redmond - Microsoft)
- September 27 - 29, 2016 (Los Gatos - Netflix)
- November 29 - 30 - December 1, 2016 (Menlo Park - Facebook)

Conclusion/Resolution

- Need to confirm Munich in May (Google)
- Rick, Dave will figure out March meeting host

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, **Yehuda Katz** (**JQuery/Tilde)** for the organization of the meeting and the excellent meeting facilities. Many thanks in particular to **Mr. Dave Herman (Mozilla**) and **Ecma International** for the social event.

Annex 1

Technical Notes

Sept 22 2015 Meeting Notes

Allen Wirfs-Brock (AWB), Sebastian Markbage (SM), Jafar Husain (JH), Eric Farriauolo (EF), Caridy Patino (CP), Mark Miller (MM), 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), Rick Waldron (RW), Yehuda Katz (YK), Jeff Morrison (JM), Lee Byron (LB), Daniel Ehrenberg (DE), Ben Smith (BS), Lars Hansen (LH), Nagy Hostafa (NH), Michael Saboff (MS), John Buchanan (JB), Gorkem Yakin (GY), Stefan Penner (SP)

On the phone: Mark Miller (MM), Dan Gohmann (DG), John McCutchan (JMC)

(Need attendance list)

1. Opening Items

JN: (Introduction)

JN: Agenda is https://github.com/tc39/agendas/blob/master/2015/09.md

YK: Post ES6, do we want to break up days into plenary and presentation.

AWB: An agenda item?

BE/YK: Add to agenda

Suggest:

- morning plenary groups
- afternoon discussion

LS: (facilities)

Adoption of Agenda

AWB: Future agendas should avoid being specific about the version

BE: Helpful for me to know which features are on track

Conclusion/Resolution

- Change agenda "Proposal for future editions of ECMA-262"
- Agenda approved

4. Secretariat Report

JN: 1 o'clock on Thursday

5.1 Shared memory and atomics

(Lars T Hansen, Mozilla)

https://github.com/lars-t-hansen/ecmascript_sharedmem

http://lars-t-hansen.github.io/ecmascript_sharedmem/shmem.html

Need slides

LH: Work in progress at Mozilla, Google, etc.

Use Cases

asm.js

- pthreads in translated C/C++ code
- Support for safe threaded languages

"Plain" ES - Shareed state and multicore computation - Fast communcation through shared memory

Use cases conflict:

- asm.js has flat memory, no gc, string types
- plain ES is object based GC'd weak types

Compromise?

...

Approach

Provide low-level facilities - SharedArrayBuffer + TypedArray - Atomic Operations - Concurrent agents (introducing) - Agent sleep/wakeup operations

Build Higher Level facilities

- Locals ... (need slides)

API: Shared Memory

A new data type:

js var sab = new SharedArrayBuffer(size); (need slides)

Views on Shared Memory

js var sab = new SharedArrayBuffer(size); var ia = new Int32Array(sab); var fa = new Float64Array(sab, 8, 10);

(need slides)

API: Atomic Operations

js ...

API: Agent Sleep and Wakeup

Modeled on the Linux "futex" (fast user space mutex)

js Atomics.futexWait(i32c, loc, expect, timeout); Atomics.futexWake(i32c, loc, expect, timeout);

- Minimal assumptions, very flexible (need slides)

Example: mutex lock()

```js Lock.

get code from slide

```

Discussion re: scheduling and execution.

DH: tasks can be scheduled, but not executed

CM: Understanding there would never be shared state threads in JS. This appears to be that.

(Discuss after presentation, Moving on)

LH:

Agent Model

- Need a model for concurrency in ES
- Define concurrency in termsn of agents

- Define agents in terms of ES6 jobs
- Give jobs a forward progress guarantee

Agent Mapping

- In a browser, an agent could be a web worker
- SAB sharing is by postMessage,
- WebWorker semantics need work, nearly useless today.
- In a non-browser setting (SpiderMonkey shell)
- concurrent thread, separate global environment
- mailbox mechanism for sharing memory

AWB: Can imagine request for generalized model

DD: A different group could approach to define generic

AWB: Dont want a mechanism that will only work in a browser.

LH: (Confirms)

Implementation Concerns

- Trick to block on the main thread in browsers?
- Subcontracting, main browser thread actus on behalf of worker
- Possible to deadlock if main thread is waiting
- Make workers not truly concurrent
- Where's the bug?
- Subcontracting for UI and other things are also problematic

DD: An early version of this proposal said that these APIs were not on the main thread

YK: Scary to have locks on the main thread

LH: Not that scary, it's no different than a loop

YK: Async functions are the solution that has evolved

AWB: Are the workers multiplexed?

LH: If JS on main thread creates a worker and waits on the location, it will hang because the worker doesnt get started.

AWB: Semantically observable?

LH: Can observe remotely (a corner case)

Memory Model (1)

- Atomics in the program are totally oridered
- Conventional "happens-before" relation on events:
- Program order (intra-agent)
- Atomic-write, atomic-read (intra-agent)
- futexWake called -> futexWait returns (intra-agent)
- postMessage -> event callback (intra-agent)
- transitivity, irreflexivity

Discussion: clarification of consistency model

Memory Model (2)

- Reads only see writes that happen before them (and only the last of those writes)
- Unordered accesses where at least one... (need slides)

Memory Model (3)

- Races are safe, programs don't blow up
- Races are unpredictable, a race poisons the memory by writing garbage
- A race affacts... (need slides)

DH: Say there is a non deterministic collection of memory bits in location

YK: What happens when you read the garbage?

LH: Currently will get the garbage

Memory Model (4)

Complications:

- Aliased arrays and cells thar are not exclusively atomic
- Weakly ordered memory (ARM, MIPS, Power)

C11, C++11 have similar issues, mildly unsound?

Java Better? But complex and subtle

DH: if mem model says data structure, behavior localized to data structure. integration with rest of language impacts jit compilers, spec semantics (how deep). Not sure what the biggest risk is... limited to array buffer and only in specific case? Invariants in loads and stores

LH: optimization to load value from typed array, use it for something, eg. function call. If not affected, can reload.

DH: Already know typed array, can assume not shared array buffer. Cases where I *don't* know the backing store, nervous

AWB: Typed array accesses with brackets

Limit concerns to optimizations re: Typed Array cases.

DH: Concerns: How much does this leap into the current spec. How much does it affect the lower level operations?

AWB: Typed Array access get turned into operations on their bufffer. Buffer operations become polymorphic (unshared, shared). A common operation that occurs, with slightly different behavior.

Other Memory Model Issues

- No "relaxed" atomics, we problam ant them but they are complicated
- No "acquire-release" atomics, ditto though weaker use case
- Shared memory can be used to implement high-precision timers (you just count), which can be used to simplif cache sniffing attacks
- Slight security concern
- Misc minor issues, see github repo

BE: Like mips load, link (re: "acquire-release")

LH: (state of "acquire-release" on current platforms)

Status

- Spec is stable, memory model is being refined
- Firefox Nightly since Q1 2015, demo level code for asm.js, plain JS is running
- Google have committed publicaly, at least in part (need slides)

CM: This is terrifying. Can see cases for use in libraries for communicating internally, but as soon as this is exposed to random web developers... The nightmare is people that don't know how to code in this paradigm. Likely a majority. Now can write highly insecure programs, introducing the problems that JS rid us of

YK: What about other high level languages, Ruby? It exposes pthreads. Then developers write better higher level abstractions

CM: Nothing to protect from authors of page and script running in it.

YK: Concerns is third party scripts?

CM: Yes.

YK: Could mitigate

DD: sandbox?

CM: Could design the things you might make with this, then codify into language...

YK: Verification?

CM: No, if the facilities don't exist, then safe.

BE: Agree, most web developers should not be tempted, even able to write shared memory programs. We need to talk about what goes wrong. Cannot have shared memory without races? We have demand for this.

Why Not Just for asm.js?

- Some tasks are best/only done in JS callouts
- I/O
- Thread Management
- Runtime tasks in general
- JS has data structures, easier to program
- The callout needs at least some shmem access

LH: need to satisfy both use cases

DE: Any program can already get into an infinite loop, unresponsive.

CM:

BE: If we don't do something like this, there will be a bigger gap for webassembly stuff.

YK: Shared memory is not inherently unsafe, only in raw form. We can provide safe access.

(Mark joins on phone)

YK: Need to more concretely address the constraints

MM: Will read emails from waldemar:

- Introducing a high bandwidth side-channel
- Even if only computation with coarse granularity, the ability to access shared memory seems fatal.

LH: Only fatal if you can prove it's not already possible

MM: No way to do high-bandwidth side channels, such as cache attacks

MM: Timing attack is: shared array buffer with self.

BS: Issue is using the shared array buffer as a timer, you just read it and allows cache attacks

MM: If attacker can create two threads that can communicate, you can create high resolution timer attacks

BE: There's been demonstrations of not-as-high resolution timer attacks that already exist. This is not the *first* high resolution timer attack.

MM: Waldemar believes this is a bad vulnerability

BE: http://www.theregister.co.uk/2013/08/05/html5_timing_attacks/

DE: Any further conserns?

MM: Delayed weak refs because of

- JS has become mostly deterministic
- Plat
- bits in nan are platform dependency, not generatl non-determinism
- for-in iteration order is platform dependency, not generatl non-determinism

MM: On whatwg, proposed System object. System is not distinct from the ES built-ins. Grants

MM: perhaps the non-determinism comes from creating the Worker in the first place? Thinking out loud

Where does one get access to the SharedArrayBuffer constructor

Either from global or imported from module.

AWB: How do you gain access to an already existing SAB created in another Worker?

MM: Creating an SAB alone is safe.

BE: The

(http://www.theregister.co.uk/2015/04/21/cache_creeps_can_spy_on_web_histories_for_80_of_net_users/) / http://arxiv.org/pdf/1502.07373v2.pdf

LH: the resolution of performance.now has been reduced to help prevent the caching attack

DD: not concerned about the paternalistic argument, rather the actual security concerns?

YK/CM: Concerned about the programmer shooting themselves in the foot

MM: Concerned about security attack, more than programmer

CM: concerned about DOS attacks from third parties ...concerns are directly related

YK: Right way: expose async futexWait

LH: Originally had an async futexWait

CM: Proposing non-blocking-but-blocking?

YK: No, want to express sequencing in a local way. This will be done with async function … want same solution, async futexWait

MM: Q: motivation for SAB proposal: conventional c/c++ pthreads code can be compiled through asm.js can still fnction?

LH: That's one of the case

YK: Shared Memory is a useful primitive.

MM: Want to separate:

1. Compile old legacy pthread code

2. Make good use of hardware parallelism

BE: legacy?

DH: Tried to build higher level primitives to give deterministic parallelusm to JS. Challenging. Meanwhile the extensible web strategy proves every time. Expose better, high performance, multi-core parallelism primitives.

YK: Should give 0.01% more credit (like authors of gzip, etc)

LH: first thing I did when I finished the prototype

MM: fan of the Rust/Pony approach, low-level, universal. Arrived independently at a similar result.

YK: Easily imagine a Rust-like model, b/w dynamic checks

DH: Rust has Shared Memory concurrency, static type system and library design that statically guarantees ...best of performance and programming model. Can't be added directly to JS, maybe with APIs and DSLs that have some speed tradeoffs, don't know what will win in the end

BE: MM mentioned Pony, which has similar design

BE/DH: Cannot evolve JS into Rust.

MM: not what I'm proposing. I'm proposing investigating building on TypedArray safely parallel memory system with ownership transfer of region, capitalizing on Rust/Pony development

DE: how do we solve the compile from C++ problem?

BE: circles back to earlier point: don't have time to find SAB alternatives, without exposing shared memory needs of C++. If we do nothing, then other "bad stuff" will happen. If we do this, then other people can experiment to find a better solution

MM: The potential show stopper: the side channel. I want to see some real analysis of this danger and why we should be confident that the danger is minimal, before we proceed.

LH: we have security people working on that, we can try to expedite. - danger is that page load, you think its safe. Allows to fork off a worker with shared memory, and there's the attack.

YK: Might want restrictions on the main thread, so ads can't own your site

LH: is this appropriate for the language or the browser embedding?

LH: whole thing with service workers/shared workers is tricky.

YK: Thing is (oksoclap disconnected on me, notes lost)

LH: fundamental constraints with blocking on the main thread? Or can the browser implementations work around this.

YK: is this a fundamental C++ problem? - fetch in browser: fork off a thread, do the work, wait for response.

LH: depends on which web APIs are avai - want to update webgl

"shared" or "transferrable" canvas

DH: Concern, if growing API, implementation burden to find ways to share with workers? maybe WebIDL?

AK: More clarification on urgency?

BE: if we do nothing, it is a bigger jump to WebAssembly, or somebody else will do it anway to compete. Can't polyfill Unreal engine 4.

DH: Jeopardize adoption of web assembly.

AK: Is this going to be a problem with wasm features for time to come?

BE: maybe in the future JS will not be serving the cross-compilation master, but not right now

YK: constraingts: what dynamic checks do you need to make SAB work?

BE: if we have web assembly in all the browsers, why would we need these in JS?

MM: Memory model discussion?

LH: we talked about that.

BE: If we say out of bounds "no and forever", something else will happen (see: flash, java). We need to work towards something.

YK: concrete example: VLC in the browser, don't want to write C, write JS, chunk up the screen

AK: Can't use argument of "needs" c/c++ pthreads.

LH: (pre lunch summary)

- This feature has cost

- sophisticated shared memory given to those that don't know how to use it

YK: better to allow the ecosystem to help us design the higher level sync operations than try to design ourselves

LH/MM: discussion of burden of proof of timing channel attacks

BE: hard to choose between higher-level and lower-level primitives

YK: quarantining helps that problem. Have to deal with timing channel attack.

CM: not sure quarantining is sufficient, what are the modes of vulnerability. Ecosystem of mutually distrustful components, run the risk of someone not paying attention, leaving the barn doors open

YK: orthogonal to the primitive, access to the primitive

AK: depends on the primitive

YK: actually, yeah.

(Lunch)

BT: True that impl. shared array buffers in wasm is easier than normal JS?

LH: Different, not easy ...Optimizing generics atomics, backend implementation

BT: SAB in wasm have same security concerns?

LH: Yes.

BT: Need a sense, wasm is a ways out.

BE: something will happen if we don't act

LH: thought about having it only in asm.js, not sufficient. runtime and IO services need accesses to shared memory and atomics. some of it could be moved into asm.js, but not all of it. to do so asm.js would need access to DOM APIs.

DH: Value? Same potential security concerns. Same access from JS. Also brought in requirement to standard another thing. If done in asm, then need to standardize all of asm.

BT: what about WebAsm only? I don't think polyfills are critical. asm.js is great because it is a subset, sound great as a marketing story, but haven't seen a real example

DH: Important that it was a subset, was adopted.

BT: asm.js compiled stuff doesn't work when asm.js is turned off. Games will not run in wasm

YK: gzip.js?

BE: you're saying that things don't work well outside of optimized asm.js?

BT: wasm polyfill has no value to game developers. Unreal Engine running against this polyfill will not work.

DH: asm.js exists today! Polyfill WebAssembly to asm.js, it will run well.

BT: asm.js apps running without asm, don't run.

DH: two different stories: asm.js as a subset of JS is to break out of a deadlock. Disagreement between browsers about how to port C++ apps to the web. Competitive pressure created excitement. Kickstarting the process.

BE: (shows zombie chicken killing example in browser) compiled to asm.js, running in browser. (He's playing this next to me and I'm really more interested in that than the current discussion. - RW)

DH: WebAssembly is in a different place -- not about disagreement between browser vendors, there is agreement. The polyfill here is about making sure that browsers that haven't yet implemented features can still run, and people can ship functionality.

BT: wasm polyfilled ontop of asm will be a useable experience?

BE: Yes, I'm playing right now.

BT: the polyfill burden for asm.js is too much, you can probably get better performance without asm.js. Kind of a side tangent

DH: this is super important! Need to keep our eye on the ball for polyfill for WebAssembly. Still have a couple of years before wasm, maybe more years before all browsers have the functionality, maybe timeline is off, but the polyfill story helps mitigate the risk of shipping wasm features.

BT: if I'm Epic, I need two codebases to support asm.js and non-asm.js browsers

YK: long tail of things that benefit from asm.js, but still work without

BE: spoke with Apple about this, they think they don't need to specialize for asm.js, can generate good code without detecting "use asm"

BT: Why polyfill?

BE: Can run code sooner than later.

BT: Browsers that don't support asm or wasm will be gone soon?

BE: That support asm, but don't support wasm.

YK:

BT: I think I can understand the value of the polyfill

BE: But do we want to add it to JavaScript for all time? Maybe that's not the right argument, what about Rust to JS, Pony to JS, etc.

BT: But you can still do that from wasm, right?

LH: we need to make sure we support both use cases, compile to asm.js and data parallel in JS. Much more useful to have both than not

BT: I agree we shouldn't be telling JavaScript devs that they can't use these features, but do we save anything but focusing only on asm.js? It sounds like no

LH: I don't think the extra burden to the JS dev is very big

AWB: It doesn't seem like it would be optimized very well if we force JS devs to communicate through asm.js for this feature

BT: If there is strong motivation to add for all time, then I'm fine. As long as not *just* for a polyfill.

DH: I see this as a building block for the long term

BE: we need to reach consensus, but we don't have to say this is going to be in JS forever just yet

LH: Request Stage 1, with a list of things to investigate.

MM: w/r to SAB, discussed extensively that burden of proof to show cache sniffing vulnerability is on the champion

... memory model is not contained. This will bleed into anything that touches the SAB. We need to try to find a way to contain it.

LH: sounds reasonable

AWB: Object to Stage 1?

MM: no problem moving to stage 1

YK/SM: What restrictions do we want, who do they work? Difference between global and local.

CM: All concerns are stated.

Conclusion/Resolution

The committee agrees that those wishing to advance shared array buffers must explicitly address the following before further advance is considered:

1. To what degree might shared array buffers exacerbate the side channel problem that web browsers suffer from?
2. The issue is the potential increase in vulnerability over the status quo.
3. How bad might this new side channel be?
4. How can we be confident that it is not worse than that?
5. How bad is the status quo?
6. How much less bad could the status quo implementation be without breaking the web?

Cite: https://github.com/lars-t-hansen/ecmascript_sharedmem/issues/1

[Waldemar: I couldn't attend the meeting because of an injury accident but was asked to share my concerns about how bad this can be. Here's a paper demonstrating how one AWS virtual machine has been able to practically break 2048-bit RSA by snooping into a different virtual machine using the same kind of shared cache timing attack. These were both running on unmodified public AWS, and much of the challenge was figuring out when the attacker was co-located with the victim since AWS runs a lot of other users' stuff. This attack would be far easier in shared-memory ECMAScript, where you have a much better idea of what else is running on the browser and the machine (at least in part because you can trigger it via other APIs).

https://eprint.iacr.org/2015/898.pdf]

1. To what degree can normal JavaScript code be insulated from the complexity of modern memory models?
2. For code that does not itself touch a shared array buffer, but for example, merely calls something that does, it is not reasonable to disrupt the programmer's understanding of the JavaScript program in terms of naive sequential consistency.
3. Of course, one can do that at the price of fine-grain synchronization and/or avoiding all interesting compiler reordering like common subexpression elimination. But such approaches are likely to kill the performance that motivates this proposal in the first place.

5.2 SIMD.js Stage 3 proposal

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

DE: (update)

(Copy from slide)

AWB: do you mean they are not canonicalized in a SIMD.js operation or when you extract? lumping load/store w/ extracting

DE: canonicalized at extraction - Interfacing between SIMD and TypedArrays, never creating a JS scalar

AWB: don't think you need to say this

DE: consensus among implementors was to specify.

AWB: I don't think this is observable. It is no more observable than normal number arithmetic, you can tell with two values but not necessarily that a third is the same

DE: change makes spec more strict.

DH: observable.

AWB: TypedArray, have a signaling NaN. SIMD signals on signalling NaN, if the SIMD unit processes, you must get exception thrown.

DE: Yes, I think that's what should happen, but we also removed the feature in another case. It's not really in use

AWB: Read from TypedArray a f64, allow that to continue being a signalling NaN.

DE: We were trying to make the change on the writing side, but we changed to not care about that

AWB: concern that implementation will have to change to allow for signalling NaN. Saying that the implementation cannot change that to a non-signalling NaN.

AWB: Do any known implementations treat signaling as non-signaling?

PJ: x86 doesn't, it just uses the default control register

DE: We don't allow people to turn that on from JS anyway

DE: Suggested to make lane an argument to load/store

BE: case wants constant. (lane argument)

(Copy from slide "changes")

DH: (question about partial staging?)

BE: Seems better than going back to zero

DE: Diff in how hardware works, between ARM and x86. Proposal to have opaque type for this case

BE: Don't know how to fix this without support from hardware?

DG: We have a proposal that works, but we don't have enough confidence. Current proposal doesn't use an intermediate type

(Copy from slide "questiosn raised by reviewers")

- SIMD as a built in module (now, global object)?
- Methods on wrappers (now, static functions)?
- Class hierarchy to reduce duplication?
- Equality semantics?
- Motivation for certain operations?

DH: agree, modules should not be a blocking depedency

AWB: Possible to define as module, but also not be a blocking dependency ...Issue is the *size* (500 intrinsics) relative to the actual use.

DE: down to 300 now, signed vs. unsigned added an additional 100 or so

AWB: Didn't have to.

DE: targeting asm style compilers and normal JS JITs

AWB: One possibility: build heirarchy. eg. All the integer types can share an add method. You need to check the arguments anyway.

BE: this is class-side inheritance

AWB: confirmed. ...Type check -> type dispatch. ...add method can check type of value

BE: you're saying that implementations need to optimize down to functions with typed arguments

Discussion re: overall design.

AWB: remaining concern: huge API surface area

DE: Still don't think it's a major concern.

whiteboard...

js SIMD.Int32x4.fromInt16x8Bits();

Could be:

js SIMD.Int32x4.fromBits();

Because type check must be done *anyway*.

JMC: Doing that loses the static knowability

DE: Big difference between optimizing for a completely unknown case and just deoptimizing for disallowed types.

JMC: in terms of RTTI you don't lose anything, but in static case you do

static types give us hints, otherwise we...

AWB: a hint is just a hint, still have to handle the general case

DE: we can propagate the type infomration in asm.js

AWB: this is a general type inference propagation problem

BE: Full JITs do that, but asm.js doesn't. This is not a simple API

DE: Not usable by people who don't know what type they're working with ahead of time

BE: SIMD programmers must now the exact types, they'll write by hand. Nobody is calling for the frombits case except you (AWB). JMC, do you agree?

JMC: people will write this code by hand. people know the types, every SIMD program is completely typed. Nobody cares for a generic conversion, they know the types, it's been propagated the whole way through

AWB: That's generally true for most code that people write in most application domains. You just don't redundantly annotate the operations with the types

BE: I don't think frombits is a user concern

DH: This is a JavaScript DSL

RW: when you have uglifiers, like closure, it will alias builtins

PJ: that's OK, the declaration needs to be in the same context. if you look at asm.js code, that's how it works currently

RW: my question was about the static knowability, WRT to JMC's point

DH: (explains how optimizer can determine that builtin is aliased)

BE: seems like the remaining complaint is too many methods

AWB: is there some fundamental API design that can reduce the number of methods?

DH: I don't think counting the number of methods will lead to better design.

... The overall problem domain is affected by the combinatorics. It's a large domain.

JMC: you're asking is there some alternate API design we could have? We have as a group collectively tried to accomplish this, and we don't think there is.

BE: signed vs. unsigned operators

JMC: feedback we got is that the API would be better with explicit types. It goes against another constraint, lets have fewer functions.

AWB: my argument is have multiple types, but share functions between those types

DE: Doesn't work in asm

AWB: short term trade off

BE: Explicit type instructions that map to hardware instructions ...SIMD is *only* about hardware instructions

DE: equality semantics... should -0 === 0 wanted to generalize to value types, is this what we want? should it be SameValue?

AWB: if you have Float32(0, 0, 0, 0) === Float32(-0, -0, -0, -0), how should those be compared?

DE: we don't think this will be useful for devs at all, it just needs to be defined. Simplest definition I could think of, and implementable

AWB: === gets applied to each of the elements?

DE: Correct.

AWB: Same issues that come up with scalar numbers in JS

positive and negative zero are equal, but NaNs aren't necessarily

DE: Could be that === is considered a "legacy"

AWB: I don't have the solution...

MM: Find deep equality with SameValue attractive

AWB: possibly, but I'm concerned about maps and sets. using a map for memoization, key is a SIMD vector. you want to memoize float32x4 differently with +0 vs -0

js let m = new Map(); m.set(Float32(0, 0, 0, 0), 1); m.get(Float32(-0, -0, -0, -0)); // 1?

DE: Don't think anyone will do that.

MM: memoization is a strong argument

DE: I don't think memoization is useful for SIMD vectors

BE: used for constants.

MM: memoization is broken if you don't distinguish +0 and -0, e.g. dividing by this produces positive or negative inf ... Any NaN contained, must be not equal.

DE: you already have to do that with floats.

MM: not if we say that === is non-reflixive with the values

DE: Impemented in some browsers, seem convenient to us.

MM: strong preference. when it is applied to anything other than direct application to a scalar, mathematical equivalence class must be reflexive

Discussion of === and == semantics.

DE: I wouldn't want to make different behavior for ==

AK: This is more appropriate for Brian to deal with; DE said it's not important to SIMD.

DH: Is problematic for me that NaN !== NaN, except for when it's in a SIMD value, eg. Float32(NaN, 0, 0, 0) === Float32(NaN, 0, 0, 0) ... Only reasonable semantics: component-wise ===, respect NaN

BE: it seems like we'll break with IEEE754

AWB: it seems logical that === should propagate out through the components for value types

BE: Do SIMD programmers want all NaNs to not equal?

JMC: Yes

BE: Do SIMD programmers want -0 to equal 0?

JMC: Yes

MM: Understand -0/+0, by why NaN semantics?

AWB: my only concern with the present design is WRT maps and sets. we decided to use SameValueZero

js let m = new Map(); m.set(Float32(0, 0, 0, 0), 1); m.get(Float32(-0, -0, -0, -0)); // 1?

AWB: we could define SameValueZero for vectors to not distinguish + and - zero.

DH: we want SameValueZero and === to be recursively defined

DH: nobody expects 0 and -0 to be different in a set, nor do they expect NaN to be different

RW: And has, get will behave accordingly:

```js let m = new Map(); m.set(NaN, 1); m.has(NaN); // true m.get(NaN); // 1

even though NaNs are not `===` equal.

```

DE: we assume that SIMD users are more clever and will not be confused by this?

MM: we all agree that SameValue recursively does SameValue. My preference is that all of them recur with SameValue. Next... (missed the specifics here, sorry)

... in order to implement SameValue, if x != x && y != y then return true; else ... it knows that NaN is a bizarre case, and it tests for it explicitly. It knows that 0 and -0 are weird and test for it explicitly. That coding pattern is copied a lot

BE: this won't get any better with this

MM: I'm not suggesting that SIMD.js diverge, just for value types

DE: Brian, do you have an opinion?

BT: I'm in Dave Camps

MM: I'm going to register an objection, but if no one else agrees, then I'll let all these recur on themselves.

AWB: that's how value types should work, still concerned about maps. Maybe we should revisit for maps an explicit comparison function.

- https://github.com/rwaldron/tc39-notes/blob/master/es6/2013-01/jan-31.md#mapset-comparator
- https://github.com/rwaldron/tc39-notes/blob/master/es6/2013-11/nov-20.md#reconsidering-the-map-custom-comparator-api

BE: We have unfinished business with map

DE: We have an objection registered, but consensus on the current equality spec

Discussion of min/max vs minnum maxnum. Defined by IEEE754, related to NaN behavior. minnum/maxnum will return the other value if one is a NaN

JMC: confirms that IEEE754 defines min num, max num

AWB:

discussion about including length in SIMD operations

MS: Resolution on built-in module?

DH: Need module loading completed

AWB: Don't need this

YK: Signed up for this previously, can restart work on it.

DH: Browser vendors are blocked on loader api

AWB: we should as a matter of policy, call an end to adding new globals

DD: I don't agree, the global object is where you put these things

DH: aspirational, but not there yet

AWB: Stage 3 means API is frozen.

DE: There is an outstanding API question re: {load|store}[123] (load1(...), load2(...)), that needs to be investigated. Potential performance cliff.

YK: TLDR here is if you still want to make this change, you need to signal to the implementors that this is what stage 3 means

DE: Can decide now that we're not going to do this change.

PJ: problem is that we want people to use constants for accesing the lane, but they don't have to, and that needs to work

JHD: how hard would it be to split the load/store methods into stage 2, and the others move to stage 3. Does it work without those methods?

AWB: you could code the equivalent, not as efficiently

BE: will we learn more to help us decide?

JMC: this is a relatively new suggestion. I don't think we'll learn anything new performance-wise. If compiler cannot be sure that it is a constant, it can choose not to optimize. Let's just make a call

DG: difference from extract lane: there are indexes in bounds that we can't be sure to make fast. Slow and tricky to implement

DE: behavior is strange to have operation throw if it would be slow

AWB: Still alot to work on in the spec. Is it at API freeze state or not?

BT: to the best of our knowledge, no changes.

AWB: enough thorough review of this large and complex addition to the language?

BE: SIMD seems stable. it's funky but its gone through the paces for stage 3. We wanted it for 262 because it blazes the trail for value types

AWB: more comfortable in another spec

Conclusion/Resolution

- Stage 3 Approval
- {load|store}[123]
- Consensus on existing
- Equality Semantics
- Consensus on existing

- An Objection from MM
- Work to do on Maps and Sets

10 Tooling Update

(Brian Terlson)

BT: All proposals are now using ecmarkup

- 402 Complete
- 262 In progress

CM: Need improved workflow documentation.

AWB: Concerns about general public review; diffs not ideal

YK/DD: will review visual, structural diffing tools to add to workflow

Conclusion/Resolution

- Write up workflow documentation.
- Review and recommend visual, structural diffing tools to add to workflow
- Move 262 to Ecmarkup

5.4 Updates on class-properties proposal

(Jeff Morrison)

https://github.com/jeffmo/es-class-properties

JM: (updating)

YK: Decorator interop, with this proposal, needs reflection

JM: Don't decorators operate on the instance?

YK: Unrelated, decorators are described in terms of descriptors.

MM: How do class instance properties interop with decorators?

... Revisiting the proposal semantics.

MM: Expression itself is evaluated once per instantiation?

YK/JM: Yes

AWB: How does inheritance work?

JM: Each property and expression is evaluated in top down order as a result of super() in constructor() {}

YK: Mark suggests that these go in the constructor

BE: Problematic:

```js class C { ha = eval(args) }

var args = "arguments"; var huh = new C(); console.log(huh.ha); // ? ``` http://gul.ly/4du5

JM: as written, the this binding is the object that's been instantiated.

DD: This is an issue, the lexical scope is unclear

BE: But there is a "secret" thunk scope

JM: Users have been using this in Babel without any confusion about the expression being delayed or defered.

Discussion about arguments and this
```

MM: How is this an improvement over https://github.com/jeffmo/es-class-properties/issues/2 ?

DH: (responding to alt proposal) the syntax isn't good.

- The class body is for declarative parts of the object template
- The constructor body is for imperative initialization parts

Easy to get caught up in semantics and lose sight of syntax.

MM: Disagree. Convention to follow: always begin a class by putting the constructor first and in the constructor, properties declared, then blacnk line (for organization).

RW/AWB: Where does super go?

MM: This is an open question. Depends on what you need to do with instance properties and when

Discussion comparing:

js class Point { constructor(x, y) { public this.x = x; public this.y = y; } }

vs.

```js class Point { public this.x = x; public this.y = y;

constructor(x, y) { // ... } } ```

x and y not in scope.

JM:

```js class Stuff { autoBound = () => { console.log(this); };

id = getId(); } ```

DD: This SHOULD be written in the constructor.

RW: Completely agree, and also concerned with:

```js class Stuff { a = () => { console.log(this); }; b() {

} } ```

AWB: And what is arguments in that arrow? JM: Same as arguments immediately before the class declaration:

js function foo() { arguments; // same class Bar { a = () => { arguments; // same }; } }

...

Discussion comparing the merits of Mark's proposal vs. Jeff's proposal.

MM: These declarations should result in non-configurable properties.

YK: Disagree.

Conclusion/Resolution

- Move to stage 1
- Jeff will follow up with Mark to further discuss his counter-proposal, and Allen to further discuss his thoughts on private properties
- Jeff will update proposal to store property declarations in slots (rather than container on prototype)
- Jeff will include reflection API for introspecting/reflecting declared properties
```

Sept 23 2015 Meeting Notes

Allen Wirfs-Brock (AWB), Sebastian Markbage (SM), Jafar Husain (JH), Eric Farriauolo (EF), Caridy Patino (CP), Mark Miller (MM), 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), Rick Waldron (RW), Yehuda Katz (YK), Jeff Morrison (JM), Lee Byron (LB), Daniel Ehrenberg (DE), Ben Smith (BS), Lars Hansen (LH), Nagy Hostafa (NH), Michael Saboff (MS), John Buchanan (JB), Gorkem Yakin (GY), Stefan Penner (SP)

5.5 Decorators Update

(Yehuda Katz)

https://github.com/wycats/javascript-decorators/tree/big-picture

YK: Consider this an exploration in several cross cutting features that are in development.

Starting at https://github.com/wycats/javascript-decorators/blob/big-picture/interop/reusability.md#usage-with-property-declarations-in-javascript

```js class Person { @reader _first = "Andreas"; @reader _last = "Rossberg"; }

let andreas = new Person(); andreas.first // "Andreas" andreas.last // "Rossberg" ```

The @reader decorator has created the getters for first and last. ie.

```js class Person { @reader _first = "Andreas"; @reader _last = "Rossberg"; }

Actually produces...

class Person { _first = "Andreas"; _last = "Rossberg";

get first() { return this._first; } get last() { return this._last; } } ```

Assume @reader is defined as:

```js function reader(target, descriptor) { let { enumerable, configurable, property: { name, get }, hint } = descriptor;

// extractPublicName('_first') === 'first' let publicName = extractPublicName(name() /* extract computed property */);

// define a public accessor: get first() { return this._first; } Object.defineProperty(target, publicName, { // give the public reader the same enumerability and configurability // as the property it's decorating enumerable, configurable, get: function() { return get(this, name); } }); }

function extractPublicName(name) { // Symbol(first) -> first if (typeof name === 'symbol') return String(name).slice(7, -1);

// _first -> first return name.slice(1); } ```

AWB: How is super treated, when encountered?

YK: Not yet considered, avoiding entanglement with yet to exist features, trying to stay future proof to account for them.

Moving on to: https://github.com/wycats/javascript-decorators/blob/big-picture/interop/reusability.md#basic-rules-of-decorators

DH: Decorators vs. macros: staging. Decorators good next step

YK: Consider decorators a meta programming facilities.

Basic Rules of Decorators:

- Decorators always operate on a particular syntactic element, providing a hook into the runtime semantics for that syntax.
```

- If the runtime semantics for the syntax include let x be the result of evaluating SomeExpression, that expression is passed into the decorator as a function that, when called, evaluates the expression (a "thunk").
- Decorators are not macros: they cannot introduce new bindings into the scope and cannot see any downstream syntax. They are restricted to operating on a local declaration using reflection tools.

Looking at: https://github.com/wycats/javascript-decorators/blob/big-picture/interop/reusability.md#appendix-making-propertydefinitionevaluation-decoratable

AWB: Why not capture the property key, could be computed property

YK: Don't want to make that policy decision, but can revisit.

AWB:

YK: general open question about whether a decorator function has to return a descriptor or not. alternatives like "false => cancel property creation" or "undefined" to keep going untouched, etc ...explanation of "decorate" implementation in the appendix, which is semantics, but not API.

DH: question about if decorators have to be identifiers, or LHS

YK: (explaining how computed and uninitialized property decorators would work the same)

YK: static properties would be treated the same as object literal properties, as type "property" instead of type "field"

Discussion re: static @reader vs. @reader static

YK: you have to decide where to put the decorator, I've always put it to the right. Usually modifying the builtin thing

AWB: I've always thought of it as modifying the declaration

question about shadowing symbol bindings in Methods example, update function

YK: easy to create shadowing hazards with symbols and function args

(explanation of decoration on properties in object literals using shorthand syntax)

DH: basically the hint is a tag to explain to the decorator function the context being used

YK: yes, this is a motivating example

AWB: ambiguity here? can't tell whether the user wants to initialize to null vs. normal shorthand syntax

YK: it's up to the decorator to decide the semantics

YK: simplest solution is to say that you have to type @reader _first: undefined directly

YK: but my suggestion is to allow decorator to decide

(describing how the motivating example would look in MM's syntax) js public @reader this._first = first;

YK: feel less strongly about how in-constructor declaration is bad. Moving @reader to the right of public

DH: I don't think that's the issue. more important that properties are at the toplevel of class

YK: not arguing one way or the other for "initialization in constructors or not" but trying to demonstrate that that question is orthogonal to this topic

MM: accept that this is orthogonal

YK: *showing syntax alternatives for privates, not trying to bikeshed*

YK: in this hypothetical privates scenario, subclasses would not have access to superclasses' "private" fields, ie, lexical scope

YK: no reflection API to get at privates outside of lexical class body, and Proxies do not allow access to it

DH: if you want to think of it like weak map...

YK: that's not the programming model or representation. Reflect.construct is a difference

DH: doesn't affact intercession

YK: from semantics perspective, very similar

Discussion of observable differences between weakmap and this

MM: only becomes observably different in the future were it could be reified

YK: I agree. My proposal doesn't make it a weakmap, but from a programmer perspective, you could see it that way

AWB: difference: a mirror of private state needs to be presented to decorator, mirror is presented at class definition time not instantiation time…

YK: there's one difference, it's a *read only* WeakMap - you aren't allowed to set the fields

MM: about mirror: mirror is reflecting on the class...

AWB: no instance-level reflection

MM: why does the reification of the field name as part of the reification of the class need to be able to give read access to the instance value

AWB: it doesn't

YK: TL;DR new metaproperty called class.slots, effectively a limited subset of the weakmap API. only operation I care about is class.slots.for, gives dictionary with private fields

MM: some reification of the name of the private fields?

YK: just the ones that are lexically available to the class definition. Gives keys and values of properties declared

MM: why give the ability to read the instance value, but not to set the intsance value

YK: sorry, imprecise. You can set private slots, cannot replace dictionary. Think of it as non-configurable

MM: ok, good. reification is not quite a weak map, doesn't have ability to delete. Violates the fixed shape constraint

AWB: agree details need to be figured out, but big picture sounds good

YK: (describing syntax class.slotsof)

MM: class.slotsof is a special form

YK: refers to a lexical binding, not value. wanted for ergonomic reasons, if you need to do things on the class outside of the class definition lexical scope

AWB/YK: better to try to move this inside the class definition, perhaps with a static { ... } block inside the class definition

(discussion of accessing parent's slots from nested inner class)

DH: for this case, just use a local variable in parent scope, can be accessed by inner class. maybe let bindings in class definition?

YK: class.slots reification mechanism is 90% sugar

DH: (example of let binding of class.slots in class toplevel so binding can be accessed by inner class)

AWB: if you have this you have less need for static private slots

YK: static method that you invoke and delete immediately has same behavior

(discussion of private statics)

MM: thing that makes private statics different, for an instance, all contributions are collected into one instance, in static it is spread out over the prototype chain... should be able to let the prototype chain do it's work

AWB: don't think so, what if you make an instance count static. now define subclasses, don't they get their own instance count?

MM: but they can't access it

YK: (getting back to presentation)

YK: (explaining how slots.for is abstraction for get/set in decorator for public vs. private properties)

(come back to field, slot, property distinction)

BE notes: - class elements define properties of the class prototype, class statics of the constructor - JM's property declarations specify something else: instructions to run from [[Construct]] before the constructor call - this suggests using "field" as jargon for JM's public property declarations and YK's private slot declarations - field: instance property instruction specified by declarative new kind of class element, public or private - slot private field AWB recalls ARB's V8 self-hosted JS has private slots, want them to pack in instance storage

DD: cannot be allowed to run synchronous operations arbitrarily in all constructor (HTML Elements, etc.)

YK: Agree, lock down reflection APIs

AWB: Any new reflection APIs should be reviewed by Mark for security

(lunch)

YK: Summary...

- Decorators on methods
- Decorators on object literal properties

Neither rely on some other proposal

DE: previous proposal mentioned function decorators

YK: Issues

- Staging
- Where does decorator go? It's clear with class and object literal, not with function decl.

AWB: TDZ?

```js

TDZ on access vs invocation

addEventListener("error", onerror);

@metadata("foo"); function onerror(...args) {

} ```

Unclear what this does.

DH: The decorator is not wrapping, is mutating existing object.

- not calling the decorator later.
- An identity decorator should be equivalent to not having a decorator at all
```

- Becomes the one place a decorator is not wrapping.

Angular use case: want to decorate function declarations for unit testing.

DH: factor out function declaration decorators. Rather have an imperfect decorator than none at all.

YK: Not actually proposing this.

MM: Happy with this at stage 1, no controversy.

YK: Important change: If expression, thunk on it. (thunking in lieu of quoting)

AWB: is there specifically a name parameter that is that thunk?

YK: yes.

AWB: trying to understand various binding contexts where wouldnt want to evaluate ...

YK: not confident that there is no case to defer evaluation

AWB: concerned about expressions *not* being evaluated

Discussion, re: effect on computed and non-computed properties

JM: VM authors lose the ability to understand the shape?

YK: No

AWB: requires analysis at runtime, rather than compilation.

AK: might lose runtime fast path

DH: can't be any worse than existing policy of looking at the shape after the constructor is done.

AK: Once you see decorator in literal, the VM gives up until sometime later.

YK: Doesn't mean it's slow

AK: In the short term, maybe.

DH: need to udnerstand challenges:

- already ahave machinery to analyze and find optimizations
- accept that decorators may have a slow first implementation
- why can't you then immediately optimize with existing machinery?

YK: Request moving presented today and "privates" (abstractly) to stage 1. Want to think about future additions in terms of being decoratable.

DD: Not enough discussion re: decorating shorthand properties.

AWB: This is just part of feature design details.

AK: Don't understand the "bundle".

YK: Form a champion group to work forward these features as a group.

JM: Do something outside of committee.

- Private state
- Decorators
- Decoratable object literal properties

Conclusion/Resolution

- Remains at Stage 1 with all changes presented

5.3 Async Functions

(Brian Terlson)

BT: Stage 3 proposal?

- Waldemar has reviewed, changes in place.
- Yehuda and I agree on cancelation
- Believe that cancelation can be done later

YK: If you don't type return in your async function, no way to the return path. May have written some finalization, but no way to guarantee

BT: Can write async that assumes some finalization, that might not happen.

YK: In the absence of cancelation, if there are no upstream promises to reject...

DD:

```js async function foo() { setup(); await delay(5000); cleanup(); }

const p = foo();

p.cancel();

Would need to be...

async function foo() { setup();

try { await delay(5000); } finally { cleanup(); } }

const p = foo();

p.cancel(); ```

If async functions forever stay non-cancelable, no issue. If all async functions become cancelable, the hazard is introduced.

BT: async functions *must* be cancelable. We won't add them unless they are.

- fulfilled: normal completion
- rejected: throw abrupt completion
- canceled: return abrupt completion
- ???: continue/break abrupt completion

JHD: q. about synchronous "return abrupt completion"

YK: Generator, call return()

"Return Abrupt Completion" is for any kind of return from current execution

BT: case where promise returned by async function, awaiting 5 promises? A promise is canceled, reasonable to handle that promuse and move onto the next thing—otherwise have to nest 5 try/finally deep.

Discussion of return, cancelation and recovery.

AWB: A new type of completion?

YK/BT: Yes

JHD: Clarify need for cancelation?

BT: Cancelation important, widely wanted, they will come to Promises, and will not be able to use async functions for that. So needed for async.

Discussion re: promise cancelation, how to introduce it.

CM: clarification between the two main uses of a cancelation.

YK: Promises should be allowed to represent themselves as cancelable
```

CM: No disagreement.

JHD: maybe cancel is just the wrong name?

YK: cancel impacted design, want to make sure that adding cancel to async function in the future wont pown them.

BT: haven't found significant issues with cancelation for async functions

- Subtle to add to Promises, but work needed
- No obvious issue to add to async function

Hazard:

js async function foo() { setup(); await delay(5000); cleanup(); // <-- not called if canceled. }

...

DE: Changes since last update?

BT: No relevant.

AK: Async arrows?

BT: Waldemar reported no issues.

DD: Should use try/finally to guarantee cleanup. Strictly a hazard if someone starts calling cancel() on your promises.

BE: Does this push up need for finally method?

BT: Need more work there.

YK: finally is a handler, doesn't correspond to a completion record change, but a cluster of completion records

DD:

```js async function foo() { setup();

try { await promise; } catch cancel {

recover and continue?

} finally { cleanup(); } }

promise.cancel(); ```

BE: Add cancel to Promises, then ok for async/await

BT: finally is being conflated with cancelation

This is stage 0.

SP: A future where finally is important, if code written to expect, then ok.

BE: Should use try/finally, regardless of await

...

BT: Chakra has implemented async/await on Edge. SpiderMonkey may have implementation? In Babel, TypeScript, Flow.

DH: Confirmed active development in SpiderMonkey

JHD: (some q about try/finally)

SP: assume code written indefensively might fail.

More totally useful, important, completely clear and not repetitive discussion about how bad code might fuck up if promises are canceled.
```

Conclusion/Resolution

- Stage 3 Approval

ECMA 402 Update

RW: blah blah blah

AWB: A lot of proposals, what is the criteria for new additions

EF: Paving cow paths, otherwise remain in library code. Working on proposals for plural and relative-time formatting.

CP: For ES2016, we will focus on exposing low level apis for existing abstract methods. New features will probably arrive in ES2017.

DD: FirefoxOS devs discovered real needs that are being reported.

EF: Getting the locale data (e.g. CLDR data) into/accessible-by the runtimes is a focus because it avoids haivng to transfer larges amounts of data over the network. NumberFormat and DateTimeFormat have large amounts of locale data to back their APIs; looking to do something similar for plural and relative-time formatting proposals.

Conclusion/Resolution

- Send new HTML to Ecma

5.10 Proposal: String#padLeft / String#padRight

(Jordan Harband, Rick Waldron)

https://github.com/ljharb/proposal-string-pad-left-right

JHD: (run through history of proposal)

- min length vs. max length semantics? min length semantics with repeat. max length is the desired functionality.

AWB: re: min length and max length, concerning unicode characters, code points.

JHD: Issues will exist in *any* string apis. This API doesn't use code points, can be changed to do so. Length property wouldn't be useful. If filler is a surrogate pair, I can use a multiple of it's length...

- Max length is in code points?

DH: re: terminal output, do control characters count against length?

JHD: Yes, always.

Discussion, re: code points vs. latin centric characters

BT: This API is no different than existing APIs

JHD: really want to solve this? Change everything to use graphemes

AWB: interpret "length" as simply: "number of occurrences of your fill character"

DD: assumes the rest of your string is Emoji. Never going to get the width correct.

AWB: if it doesn't do the right thing in the real world, why?

BT: Trying to do things beyond the simplest use case is just untenable.

RW: Intl could adopt responsibility for a non-latin character set handling?

BT: Agree.

Derailed into discussion about what gets to go in the standard library and what gets defined in a standard module.

DH: (To Rick) Not cool to attack other specs when inclusion of proposal is questioned.

Note: I made a comment that if two string methods are "too much", then we should revisit SIMD for the same concerns.

Back to padLeft, padRight...

DH: Clearly important if implementers are agreeing to implement before other features.

Discussion re: stage 1 or 2?

AWB: For stage 2, controversy resolved.

JHD: Then stage 1.

Conclusion/Resolution

- Stage 1
- Reviewers to be assigned.

5.11 Proposal: Object.values / Object.entries

(Jordan Harband)

https://github.com/ljharb/proposal-object-values-entries

JHD: Need is obvious.

Question about return: iterable or array? Spec wants to be an array.

DH: Confirm, when array you get a snapshot. If an iterator, then updates would be shown, which breaks from keys

Conclusion/Resolution

- Stage 2 Approval

5.12 Proposal: String#matchAll

(Jordan Harband)

https://github.com/ljharb/String.prototype.matchAll

JHD: pass a regex, returns an iterator, each yield returns what exec would.

Notable:

- doesn't accept a string, not coercion
- always adds the "g" flag
- makes a "copy" of regexp to avoid mutating

BE: reason for not including string?

JHD: Enforces a better practice

AWB/BE: valid, but creates an inconsistency, should be updated to include string.

JHD: Will update then.

AWB: Needs to work with RegExp subclass as well.

DE: Will call RegExp.prototype.exec, can leak the RegExp.

Confirmed.

Specify as @@matchAll

AWB: Default impl of @@matchAll, see default @@match

DD: Not a lot of "library" code

BE: Won't find it there, it's generally in "open code". Steve Levithan wrote about this.

YK: Ruby has a scan method, which I use frequently.

DD: an alternate: add another flag that avoid global state

nah.

DE: Still leaks.

Can we avoid the observable "cloning"?

BE: Take back to work through this

Discussion of algorithm approaches.

YK: Missing? A thing that's like exec, but gives you back the lastIndex?

AWB: exec() could grow an additional argument for start position.

The name stinks, but nothing really better.

- matchEach? It produces an iterator... (some agreement)

nah.

Conclusion/Resolution

- Stage 1 approval
- Accept strings
- Allow RegExp subclass
- Default impl of @@matchAll, see default @@match

5.9 Trailing commas in function parameter lists

(Jeff Morrison)

JM: Updates since previous...

MF: Change arity? Object, array...

RW: No, ignored.

YK: Symmetry with arrays? Holes on the call side?

nah.

BE: (revisit C issues with trailing commas)

AWB:

Missing:

- evaluation rules
- static semantics rules

BT: Don't need to block on this, can be delivered later.

Quick run through trailing comma in parenthetical expressions

nah.

Conclusion/Resolution

- Stage 2
- Reviewers?
- Michael Ficarra
- Brian Terlson

Process Discussion

- A plenary day
- Two presentation and discussion days

Exponentiation Operator

RW: Precedence issues:

BE (On whiteboard): -x^y in math and every other language means -(x^y)

Old spec is (-x)^y

AWB: Does the potential confusion of this (whichever way it happens to be decided) suggest we should pull it out?

BE: No, math and every other programming language do it this way...

AWB: I agree if there is an exponentiation operator the precedence should be followed going back to math. Since this potential confusion has been identified, we have a perfectly good way to do this...

RW: This is a case of the new process working as it should - implementers gave feedback, proposal is now better.

YK: Arguing for -5 ** 2 being (-5) ** 2 because people will think that's how it works.

YK: I think minus is kind of an edge case and I'm happy with this either way. Is my intuition wrong? If everyone else thinks the answer of what the proposal is now, then fine.

BE: Problem is people thinking minus is part of the literal.

DH: For math it seems obvious that -5^2. But for -5 ** 2, because of the whitespace around the infix operator. Even without space, - seems to be part of the literal.

BT: Python docs says don't use space.

YK: Doesn't JS precedence win over all other precedence?

RW: Jason's intution matches that of all other programming languages that has intuition operator.

YK: This is cargo culting.

BE: Let's say that fortran did it to be mathy and everyone copied it. There's still a precedence argument.

YK: I'm a rubyist. I had to look up what is in ruby. People don't know ow it works.

DH: We are operating in the context of an industry that gets expectation from historical context or JavaScript operators?

BE: You should be parenthesizing.

RW: Coffee's ** operator matches Python and Math and Ruby.

JH: Will value types complicate this? Will people copy/paste from C programs or other things? JS is eating the world. This will cause friction if we break historical precedence.

YK: I don't copy/paste code from other languages.

BE: Options: 1) wall of confusion, do nothing. 2) math/programming language precedence. 3) Javascript/dave/unary minus binds tighter people.

RW: Mark advocates for abandon/withdraw (1)

BE: Let's vote...

1: 4, 2: 15, 3: 2 (including MM virtual vote)

DH: Let's be clear - people might copy code I guess, but effectively zero people have an intutition about this from other languages. Agree people have an itutition that ** is the exponentiation operator. But people usually try to avoid dark corners so they never develop an intuition for negative bases.

I also reject that we must do what math does.

DD: I disagree.

BE: New option: 4 - it is an error to combine ** with unary minus. Code you port doesn't have this almost for sure.

No consensus... people are leaving.

Sept 24 2015 Meeting Notes

Allen Wirfs-Brock (AWB), Sebastian Markbage (SM), Jafar Husain (JH), Eric Farriauolo (EF), Caridy Patino (CP), Mark Miller (MM), Adam Klein (AK), Michael Ficarra (MF), Peter Jensen (PJ), Jordan Harband (JHD), Chip Morningstar (CM), Brian Terlson (BT), John Neumann (JN), Dave Herman (DH), Rick Waldron (RW), Yehuda Katz (YK), Jeff Morrison (JM), Lee Byron (LB), Daniel Ehrenberg (DE), Ben Smith (BS), Lars Hansen (LH), Nagy Hostafa (NH), Michael Saboff (MS), John Buchanan (JB), Gorkem Yakin (GY), Stefan Penner (SP)

On the phone: Mark Miller (MM), Istvan Sebastyén (IS)

Exponentiation Operator

RW: BE suggested option 4 yesterday (throw when unary negative is used without parens)

(whiteboard) ```js

Ok

let z = K - x ** y;

No Ok

let z = -x ** y; // Syntax Error

Must be:

-(x ** y)

Enforced by grammar change, just a SyntaxError

```

RW: BE wrote up option 4 and a summary and emailed to esdiscuss

RW: *similar to TDZ: an error when something ambiguous happens*

RW: AWB + MM switches 1 vote to 4, RW switches 2 vote to 4.

RW: question to group: do we have consensus on stage 3 with option 4?

AK: i think this sucks for users

RW: only when the user does something ambiguous

AK: we already have Math.pow which is unambiguous. why add another way that has ambiguity concerns? it looks confusing to me as a user.

AWB: it requires that you put in parens, what a careful developer would already put in parens

RW: considering your feedback (a syntax error would be a bummer) are you willing to block consensus?

DH: I will push spidermonkey team to display a better error in this case

AK: i think it's just fine to choose one (ie option 2 or 3) and i chose option 2

DH: that's just going to be a bug farm

BT: like it is in every other language?

DH: yes.

YK: there's bugs in every other language, but we don't have to copy those bugs into JS

RW: this is sufficiently useful to the end-developer

DH: Here's another way of looking at it: there is a natural ambiguity between two historical precedents, reasonable to expect it to go one way or the other. Don't just pick one because it will
```

confuse people who expect the other. Making it an early error to avoid the ambiguity is good software engineering practice

AK: why do you think other langs (with exp operator) don't make this an error?

YK: they cargo-culted it.

AWB: some of them actually have negative numeric literals (option 3) baked in to the language.

DH: so -1 actually ends up doing the right thing...

AK: I'm not the only one who thinks this, Waldemar might think this too

RW: I don't think so; Waldemar supported requiring parens early on

DH: very recent precedent for exactly this kind of decision when andreas proposed that "use strict" implications for parameter lists were ambiguous; we decided to make all of the ambiguous cases SyntaxErrors to avoid confusing people one way or another. That was a good decision, this is similar. Let's not knowingly create an ambiguity where both are reasonable interpretations.

MM: in agreement with Dave, avoid misinterpration where the consequence is the program proceeding without diagnostic that defies user expectation

MM: ... proceeds without a diagnostic that violates user expectations.

AWB: if we're wrong about this and get tons of complaints, we can always remove the SyntaxError and select options 2 or 3 later.

DH: Oh, python has the same error... -1**2 == -1

AWB: maybe Matlab...

RW: concerns enough to block consensus?

MS: i agree with AK; i don't like option 4. we'll implement the spec but I think it's unnecessary complications given other languages.

BT: I think it is unnecessary, but I don't care

YK: Do you agree, if we pick 2 or 3 there are some users who will pick the wrong one

BT: yes, but that's true for everything that includes precedence. We don't make all things an error where precedence is ambiguous.

YK: there are different kinds of confusion, mixing operator classes is an especially bad kind of confusion

AK: use strict in parameter list is different -- had implementor concern as well (discussion of "use strict" w/ parameter list, comparison with this case)

YK: BT, your point about general operator precedence is valid. I wondre why the difference from the normal precedence... why is changing the precedence in a precedence class not a red flag?

BT: I'm not sure people talk about operators in terms of precedence classes

DH: I think people do -- I do. I don't remember factoring rules for BNF, I think of it terms of whitespace and "tightness" or "looseness"

AK: it was useful to see the thoughts yesterday. another straw poll would help me see if i'm an outlier or not.

YK: option 5 of "i don't care" please

RW/YK: let's say: 1 is "do nothing". inaction.

BT: I violently don't care

MF: let me remind the room; option 4 doesn't prevent options 2 or 3 later.

[poll] 1. Dropped 0 2. Hold @ current proposal (-5 ** 2 => 25) 3 3. Orig Proposal (-5 ** 2 => -25) 0 4. BE syntax 11 5. IDGAF 3

RW: we can still relax the syntax error

AK: rare enough case that it likely won't happen

DH: I'll prototype a good error in my own parser

Conclusion/Resolution

- Stage 3, with the provision that option 4 goes in
- http://jsbin.com/baquqokujo/1/edit?output

8. Test262 Update

(Gorkem Yakin)

GY: (General update on status of ES6/ES2015 support in test262)

BT: previously we decided it was important to have clear delineation between test artifacts at specific spec versions, but that's hard.

YK: It stops making so much sense now. Can I suggest a feature flag thing? (discussion about how to handle old tests that break with a new version)

YK: at any point in time, take a snapshot of all stage 4 features and that's what we give to ECMA

AWB: strictly speaking, only yearly releases approved by GA that are final

AWB: nothing becomes stage 4 until version of the spec they're in is approved (more discussion about stage 4 vs. stage 5)

BT: I see it this way. for implementors, there is value in having tests checked in to test262 when they're working on the features, ~ stage 2/3. Only consideration here is what implemetors find useful

YK: ok stage 3 seems fine

BT: if things break, git has tools for this. going forward let's not have branches, just say test262 is living

AK: I say stage 3

(RW agree, this aligns with implementation step)

YK: I think stage 2 criteria is test262 tests. For me, as spec author this is valuable

AWB: during stage 2, I expect these to change a lot. These are part of the artifacts

DE: for the SIMD tests, we went through a lot of churn. it makes sense to have them outside of test262, now we're stage 3, we're ok with test262

YK: I think that implementors should have commit access to test262 while they're working on it

BT: difficulty mentioned by DE is real

YK: not intractable to make this frictionless

BT: I don't think this is too bad, having written thousands myself. Not frictionless, but surmountable

YK: this sounds like a real blocker. proposal: I think you should be able to put all tests for a feature in a file, people who are working on stage2 and up features should have commit access

BT/DE: we should use pull request model

AK: nice for implementor to know these are stable, stage 3 is nice

(discussing whether we need to use pull requests vs. direct commit access)

DE: problem with stage 2 is that the API is not stable, so it doesn't provide value

DH: having tests are similar to having spec, helpful to make it clear to people how the feature works

AK: we have automated process pulling test262, we don't want to have to triage unstable changes

YK: feature flags should solve this problem

BT: difficulty is that numerous different test262 harnesses, I understand Google has their own

_ general agreement that test262 tests should be stage 3 _

DE: YK and others are going to look into adding feature flags to implementors test262 harnesses

BT: difficult to test features in isolation, they are often cross-cutting

_ discussion about difficulty of implementing feature flags _

AWB: what is the plan of record for user-facing test page?

BT: someone has to fund this work. Someone in Microsoft working part-time on this, build scripts to package the tests into a site. Maybe get UI guy to spend some time on it

MF: why do we want this?

BT: useful as a marketing feature, show how accurate your browser is

AWB: original motivation: valuable to have vendor-independent neutral test, not subject to gaming

JHD: easy to test in browser with a web link, harder with a harness

AWB: if we think it's valuable to have a website where users can run tests against multiple browsers, then some members have to step up and do the work or foot the bill. Otherwise take away the website

BT: money problem right now. accomplished webdev, about a month of effort

Conclusion/Resolution

- Test262 is "living"

5.6 Proposal: call constructor

(Yehuda Katz)

YK: (from proposal)

- ES5 constructors had a dual-purpose: they got invoked both when the constructor was newed ([[Construct]]) and when it was called ([[Call]]). This made it possible to use a single constructor for both purposes, but required constructor writers to defend against consumers accidentally [[Call]]ing the constructor.
- ES6 classes do not support [[Call]]ing the constructor at all, which means that classes do not need to defend themselves against being inadvertantly [[Call]]ed.
- In ES6, if you want to implement a constructor that can be both [[Call]]ed and [[Construct]]ed, you can write the constructor as an ES5 function, and use new.target to differentiate between the two cases.

Motivating Example

The "callable constructor" pattern is very common in JavaScript itself, so I will use Date to illustrate how you can use an ES5 function to implement a reliable callable constructor in ES6.

```js

these functions are defined in the appendix

import { initializeDate, ToDateString } from './date-implementation';

export function Date(...args) { if (new.target) { // [[Construct]] branch initializeDate(this, ...args); } else { // [[Call]] branch return ToDateString(clockGetTime()); } } ```

This works fine, but it has two problems:

1. It requires the use of ES5 function as constructors. In an ideal world, new classes would be written using class syntax.
```

2. It uses a meta-property, new.target to disambiguate the two paths, but its meaning is not apparent to those not familiar with the meta-property.

This proposal proposes new syntax that allows you to express "callable constructor" in class syntax.

Semantics

- The presence of a call constructor in a class body installs the call constructor function in the [[Call]] slot of the constructed class.
- It does not affect subclasses, which means that subclasses still have a throwing [[Call]], unless they explicitly define their own call constructor (subclasses do not inherit calling behavior by default).
- As in methods, super() in a call constructor is a static error, future-proofing us for a potential context-sensitive super() proposal.

```js import { initializeDate, ToDateString } from './date-implementation'; class Date { constructor(...args) { initializeDate(super(), ...args); }

call constructor() { return ToDateString(clockGetTime()); } } ```

AWB: This is an exceptional example, most cases just defer to new, where this goes off on some other operation.

RW:

```js class A { constructor() { this.aProp = 1; } } class B extends A { constructor() { this.bProp = 1; } call constructor() { return new B(); } }

class C extends B { constructor() { this.cProp = 1; } }

let b = B();

b.aProp === 1; // true b.bProp === 1; // true

let c = C(); // throws! there is no call constructor here ```

AWB: Think about it, not as a property added to the prototype, but as an alternative to the constructor

MM: interaction with Proxy call trap?

AWB: class with a call constructor get a distinct [[Call]] that knows how to dispatch to the user defined behavior. ...no proposal yet for reflection ...no toString yet.

MM: toString the constructor, you get the entire class

JHD: currently unspecified in ES6; some engines give the "class", some give the constructor, some give a function that has no indication it was a "class"

YK: Then it would be included.

YK: *shows call constructor() {} vs () {} inside a "class"*

Mark's Alternative:

js import { initializeDate, ToDateString } from './date-implementation'; class Date { constructor(...args) { initializeDate(super(), ...args); } () { return ToDateString(clockGetTime()); } }

AWB: i don't like it because you could forget the method name

YK: also conflicting with square bracket syntax and arrow functions *lots of agreement that this is confusing*

MM: happy to drop it; glad to have a reaction.

DH: only thing that gives me pause is Stroustrup's principle: new stuff, people want long syntax; later, people want short syntax.

MF: May want a decorator augmenting a call constructor (Michael, I missed the tail end of this, can you fill in?)

YK:

MS: would this allow you to omit the constructor?

YK: class would create a default constructor, but otherwise behave like a function

*discussion about callable non-constructable classes; general consensus is that you should use a function for this*

DH: Like this b/c it matches built-ins and defines a clear separation of semantics

AWB: *discussing built-ins that create wrappers* ... Don't need to add sloppy mode semantics for built-ins, eg. SIMD

JHD: it's an axiom that Object(x) === x if it's an object, and not if it's a primitive. This must be preserved.

MM: Object.prototype.toValue, auto-wraps this.

AWB: Then unwraps

JHD: Every JS value must be wrapped when pass through Object.

AWB: Not saying "no wrapper", needs wrapper to do method lookup

DE: Sounds like two versions of ToObject?

JHD: in a sloppy mode function, if i do this.foo = "bar";, and this isn't undefined, it *always* sets the property such that this.foo can be retrieved immediately after. If this is a primitive, this isn't the case. If we change the auto-wrapping behavior of "this" in sloppy mode, for any value, then this will break existing code.

AK: Not going to add reflection here?

YK: No, but reflection is needed.

AK: Will it "grow" reflection?

AWB: We hope to advance this to ES2016

YK: Reflection is needed along side decorators and this proposal, but don't want to block this on decorators.

AWB: No reflective way to change a function with toString, but can see it's whole body. ...tweak toString, for class with a call constructor, include it in the toString representation

MM: Should be the entire class definition

DH: ??

AWB: That position is contraversial

MM: Thought we agreed?

YK: Bigger toString issue.

AWB: For 2016, toString should at least include call constructor body

MF: Want added restrictions before 2016, we can limit the reform proposal to buy time.

*This can be implemented in terms of new.target.*

AWB: Nec. in support of decorators:

- If you want decorators to be able to create call constructors, there needs to be some reflective way to install them.

AK/BT: *discussion potential implementation details*

Conclusion/Resolution

- Stage 1 Approval
- toString includes call constructor body in output
- Goal for Stage 3 next meeting.

Secretariat

IS:

- Only Ecma-404 will be fast tracked.
- Bugs in Ecma-402 need to be addressed.
- ES2016, 17, 18, etc. might cause issues?

I can't really understand most of what's being said over the polycom.

Sounds like ES6 has a lot of downloads

Meeting Schedule

JN:

- November 17 - 19, 2015 (San Jose - Paypal)
- January 26 - 28, 2016 (San Francisco - Salesforce)
- March 29-31, 2016 (San Francisco - Google)
- May 24 - 26, 2016 (Europe)
- July 26 - 28, 2016 (Redmond - Microsoft)
- September 27 - 29, 2016 (Los Gatos - Netflix)
- November 29 - 30 - December 1, 2016 (Menlo Park - Facebook)

Conclusion/Resolution

- Need to confirm Munich in May
- Rick, Dave will figure out March

5.13 Updates on rest properties proposal

(Sebastian Markbage)

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

SM:

https://github.com/sebmarkbage/ecmascript-rest-spread/commit/f4cb9c0ff9f4509854d5d30f4517a23b1a7d7e98

```js var o = Object.create({ x: 1, y: 2 }); o.z = 3;

var x, y, z;

Destructuring assignment allows nested object

({ x, ...{ y, z } } = o);

x; // 1 y; // undefined z; // 3 ```

AWB: Object pattern destructuring doesn't check enumerability, just goes by property name

DH: The ... means "I'm enumerating the names"

SM: ie. getting the keys

AK: What about in the binding case?
```

SM: Consistency. The BindingRestElement only allows an identifier.

SM: I will write a proposal to fix BindingRestElement to allow nested BindingPatterns.

Conclusion/Resolution

- No stage change

6. Updates on Loader

(Dave Herman)

https://github.com/whatwg/loader/ https://github.com/whatwg/loader/blob/master/roadmap.md

DH: *introduction*

BT: crass question: will we have something to implement soon?

DH: all you need to get started implementing is a basic understanding of what the really core name resolution is going to look like.

BT: and that's done?

DH: it's done *enough* that you could be hacking on this now. not written in the spec yet.

BT: Concerns that if spec is not written and we implement, it might be useless. Stuck in holding pattern.

DH: Should start hacking on this.

DH: we will get you stage 0 stuff written as soon as we can.

BT: (wants estimate to communicate to Edge team)

DH: We can promise name resolution drafted in the spec by the next TC39 meeting.

JM: How are relative urls resolved? Relative to what? (toplevel document? importing module url?)

DH: "referrer" used for relative, "/"?

DH: *explains how "module src" will respect base url, but imports inside the module will not*

MM: Loader constructor has no concept of URLs, just name resolution.

DH: Loader class is generic. Specific default Loader has to be host specific, there is a spec.

AK: Who are the experts working on the Loader spec? Anne? Dimitri (Glazkov)?

DH: Both have reached out. Let's have a meeting with those people.

AK: Dimitri is quite busy, so...

YK: I think it's important for Anne to be involved here.

DH: script type = module should have roughly defer semantics, asynchronous but ordered.

YK: before doc ready, content loaded

DH: We have to asynchronously load the deps, do we have to complete the compile phase of first script type = module, or ... execution phase as well

YK: argument for deferring evaluation at all, first is ... if you move on to second, your script didn't get a chance to configure...

... you cannot start download script type = module imports before ...

AK: this is all about what script tags do right now

YK: you need to have one block. this block is for configuring my loader, I don't want any scripts to load before this is done. Not good to block loading modules before all scripts are loaded

DH: if semantics of type=module is only that one has to precede the compilation of the other, forces you to use legacy script.

YK: there's a worse problem, path for legacy script => legacy blocking script, or use legacy defer script, and now you have to decide order between them. Another option is loader config, that has to run before, but not other scripts

DH: a narrow case? Could have a syntax instead of reusing form?

YK: if you try to make other scripts ordered, you have to speculate whether one is a loader

DH: Another Q: Allow completed subgraphs to execute early? Hold the line on deterministic top level execution.

Issue: module system forces sequentiality

Ie. - download - get deps - download - then start compiling

Probs

- May create latency
- Could a subgraph that was completed start executing?

MM: what would be the definition of a completed subgraph that distinguishes it?

DH: for example, toplevel module main, requires foo and bar, foo requires jquery, bar requires 100 modules. While bar is downloading dependencies, foo is downloaded jquery, waiting to compile and run. In that case, schedule foo and jquery's top levels to run, more concurrent semantics, less deterministic

MM: makes sense, introduces new element. optionally breaking toplevel module execution into separate turns. multiple modules part of 1 dependency graph may be executed in separate turns

YK: middle ground: start executing subgraphs in the order they've been specified. gives you determinstic ordering

AWB: in terms of modules, there is only one syntactially type of module. two types written: script type modules, root of execution, import other things. OTOH, module-like modules have exports and imports and logically are doing useful work if people import them. I thought that module-like module would never be initialized if someone didn't import it. In theory, a script tag that names a module-like module, doesn't have to be executed unless a script-like module imports a module-like module.

DH: no interest in inline-named modules

AWB: not relevant to this discussion.

AWB: script-like module: used like a script, doesn't have exports, does have imports, written for effect

DH: either are there, b/c in a script tag, or not.

AWB: so you write a script tag, name a script-like module, that needs to execute as soon as the deps are available.

JM: has to wait for deps?

AWB: it can't run until it's script is available. Here is code I want to execute. If you name the script whose exist to export it, but nobody uses it. What does a module script tag mean, this gets to the deferred semantics. Here is the script, I want to evaluate this, even if nobody depends on it.

DH: No. Cannnot express the former in HTML. The module doesn't have name, cannot be depended on.

DH/YK: they have source but they don't have name.

AWB: Anything in a script tag is the root, execute directly. Assume ordered?

DH: *confirms*. Intentional order of execution. Code directly or indirectly depended on has to execute immediately. Does all that have to happen in a single event turn? Or across several? If the subgraph has completed, can execute?

MM: bottom-up execution constraint. clearly doesn't imply any type of suspended caller. If all module execution is executed in single turn, then starts and ends with empty stack. implementations can do whatever they feel like to do other jobs

DH: (explaining options between single job vs. multi-job loading. if multi-job, then decision between total ordering and partial ordering) 1. single job 2.a. total ordering of modules N jobs 2.b. partial ordering of modules

YK: module imports are declarative, so gives clear dependency order, effectively a set of jobs, deterministic. Another set of jobs exist...

...modules want to initialize state, using set of promises. dependency ordering that is opaque to module execution order, benefit of that is we can have synchronous 1-shot path through module execution...

YK: problem with 2.a...

JM: we've used 2.a. for a long time and it has worked well

YK: ... other things in the dependency graph is opaque, to repair that, we need toplevel await

AWB: my concern with 2a. only comes up with circular module deps, a set of dependent modules isn't fully initialized until you circle around, each module has been given a chance to initialize and execute through entire body. If you start doing other jobs, somewhere along that path, are there going to be potential observable....

DH: difference between observable and likely. We shouldn't care about reflectively being able to observe that things are complete. Ignoring the reflective stuff, just declarative, initializing these in dep order, outside of cycles you can't have a problem. Even within a cycle, early reference to something that hasn't executed yet, but that can happen synchronously too.

YK: there is a toplevel cycle problem already

MM: I think there is an adjustment to 2a and 2b to avoid cycle problem. adjustment is strongly-connected cycle is atomic, if we allow interleavings, only between strongly connected cycles.

AWB: then you'd have to identify these

...

discussion about N job module loading -> toplevel await

DH: *describing the problem with toplevel await and node.js* ... people want to use synchronous functions in module initialization, but this forks functionality between async/sync variants... ... toplevel await gives you a way to prevent module use before initialization. in node.js currently people often just race between initialization and use.

MM: question: node modules is loaded via require? If someone calls require in a later turn, the module is loaded later. [everyone confirms] so toplevel modules can already be run over multiple turns? [confirms]

YK: their concern is that introducing await for require will change the behavior of require in a confusing way

JM: But it's backward compatibile and not a breaking change, right?

no

MM: why isn't toplevel await something that can be written already?

YK: you can desugar that already, but they still don't want it

AK: They just don't want the feature.

SP: Bad experience when tried to do this the first time.

confirmed

YK: they say, once you start using async, everything leaks. And that's true

DH: sounds bad, but there is a reasonable solution. allow toplevel await, asynchronous loader spec, and node won't use it. any modules that have toplevel await will be incompatible with node

YK: syntax errors, or run to first await and done

DH: they can do whatever they want here, if it becomes popular on the web, maybe that will change their opinion

MM: If toplevel await is equivalent to something users can write anyway...

YK: I just realized that it is plausible to do what async function does, if await appears at toplevel the export is a promise

DH: Yes.

- require is zalgo-ish
- may block severely

MM: that to me, disqualifies it. But we're not talking about require, but import

DH: all node packages expect to start with toplevel synchronous main and require

MM: would expect to interface between two systems, what you obtain is a require immediately returns

DH: two things that you want for evolutionary path: require an es6 module that awaits, thing you get back is a promise rather than module. Other part is an alternative entry point for node: execution semantics at toplevel is es6 loader program, everything runs through async pipeline, with imports instead of require

YK: await, import at toplevel could trigger that

DH: maybe inferred rather than opt-in

MM: i don't see how this forces you to require as suspension point for caller

YK: browser does not want require that sometimes returns value, sometimes promise. legacy purposes, require returns value.

MM: require has to return what the required module exports

DH: could say: all ES6 modules provide a promise

- Agreement on 2A semantics

2A: Modules execute in the order that they appear textually (same order as ES6 semantics), but may run across multiple turns.

Moving on.

DH: Need stage 0 for module metadata

https://gist.github.com/dherman/c35e968991b67ae98423

MM: there's a defacto spec in JS: source maps. very specific about what type of information is mappable. I think this corresponds to filename/dirname...

(discussion of import.context) EF: why import and not export? (module would be better, but it isn't a keyword)

DH: import.context.lineno, callno?

MM: my point is: if template strings actually created a sourcemap on the template object, as part of the extended definition, you'd have all the information by that mechanism, including where the individual thing in the source ...

DH: the competition here is __filename, __dirname, etc. import.context ergonomics is pretty close, are you talking about something worse?

MM: there is an overlap of functionality with sourcemaps and template strings -- we should coordinate these.

(question about this exposing a new execution context for module system)

DH: this was always the case with modules.

MM: can't you just create an error, throw and catch it to get the information for the filename, line, etc. how is the info on import.context different?

CP: just one concern about import.context compared to import local from this module; proposal, we need a new way/channel to ask loader for such as information, before it was just a synthetic module that relying on the loader semantics to do the bindings.

extended discussion about what information would be stored on import.context -- is it source name, dir name? or is this too node-specific?

Dynamic Relative Loading

- Dynamically load a relative module when it's only needed rarely:

```js

lib/index.js

configButton.onclick = async function() { // corresponds to: import { readFile, writeFile } from 'fs'; let { readFile, writeFile } = await import.named('fs'); // corresponds to: import config from './config.js'; let config = await import('./config.js'); // ... }; ```

DH: Dynamic scope tricks you into thinking that you can abstract normally (say, by moving outside), but you can't

Need: static syntax to communicate information so that we can dfkjnsdkfbjv ;skdfjblskdjfnbvlksdjfbsdfv

*discussion about import(...) and import.named(...) being meta operators*

AWB: make it not an operator: meta property whose value is a closure

MM: could be closed over the data from import.context

*discussion about ambiguity of syntax*

DH: syntax needs work, not complete/final

MS: because dynamic, needs a new API. Don't like the properties on keywords.

DH: *re-explains everything that lead to current design*

*Discussion re: import(...*

DH: "out of band" in a subexpression and need additional data

AK: Assumed import dynamically, methods on a loader instance?

DH: Missing ability to do relative dynamic loading, needs context otherwise need to write information in your code about where the file lives.

AK: need some syntax that says "where am i?"

DH: we want the smallest number of constructs, there is benefit to users for this. However, inband vs. outband is real ergonomics issue. Ability to correspond syntatic forms between declarative and imperative forms has value

DH: Moving on. Need global APIs here, reflective API for dynically creating a module instance. I want Reflect.Module (or something living under Reflect), DD wants...

tl;dr: Spec defines a module namespace exotic object, need a way to create these. Custom loaders need to make these.

AWB: can other modules import those? [ others say yes]
```

DH: reflective vs. source text module objects...

- Module Records, not Module Namespace Exotic Object
- need something for that
- needs to go somewhere
- DD says WHATWG shouldn't be monkey patching onto Reflect object
- Loader spec lives on a bridge
- Not specific to WhatWG
- Belongs in Reflect

Agreement

MM: *talking about System*

DH: This will block stages of Loader.

AK/DH: Milestone 0, not Stage 0

DH: milestone 0 doesn't depend on this syntax, but milestone 1 does, and we don't want to hold that up, so people can start building the ecosystem

Conclusion/Resolution

- Stage 0 granted for dynamic relative loading
- Module Record thing goes in Reflect, full-stop. ("Reflect.Module", perhaps)
- Reflect.Loader, Reflect.Module