Minutes of the: *41st meeting of Ecma TC39*

in: *Redmond, WA, USA*

on: *29-31 July 2014*

1 Opening, welcome and roll call

1.1 Opening of the meeting (Mr. Neumann)

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

Companies / organizations in attendance:

Mozilla, Google, Microsoft, Intel, eBay, jQuery, Yahoo!, Netflix, Facebook, Indiana University

1.2 Introduction of attendees

John Neumann – Ecma International

Istvan Sebestyen – Ecma International

Jafar Husain – Netflix

Erik Arvidsson – Google

Mark Miller – Google

John McCutchan – Google

Domenic Denicola – Google

Dimitry Lomov – Google

Erik Toth – Paypal / eBay,

Allen Wirfs-Brock – Mozilla

Nicholas Matsakis – Mozilla

Sam Tobin-Hochstad – Indiana University (invited expert)

Waldemar Horwat – Google

Eric Ferraiuolo – Yahoo!

Caridy Patino – Yahoo!

Rick Waldron – jQuery

Yehuda Katz – jQuery

Dave Herman – Mozilla

Brendan Eich (invited expert)

Jeff Morrison – Facebook

Sebastian Markbage – Facebook

Brian Terlson – Microsoft

Paul Leathers – Microsoft

Abhijith Chatra – Microsoft

Jonathan Turner – Microsoft

John-David Datton – Microsoft

Matthew Podwysocki – Microsoft

Gaurov Seth – Microsoft

Ben Newman – Facebook (part time, over the phone)

Peter Jensen – Intel,

Jaswanth Sreeram – Intel

1.3 Host facilities, local logistics

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

1.4 List of Ecma documents considered

Ecma/TC39/2014/023	Minutes of the 40th meeting of TC39, Menlo Park, June 2014
Ecma/TC39/2014/024	Venue for the 41st meeting of TC39, Redmond, July 2014 (Rev. 1)
Ecma/TC39/2014/025	Software submitter contribution form filled in by Samuel Mikes
Ecma/TC39/2014/026	Draft IETF JSON i-JSON 02, July 2014
Ecma/TC39/2014/027	Responses to the Ecma Contribution License Agreement (CLA)
Ecma/TC39/2014/028	Agenda for the 41st meeting of TC39, Redmond, July 2014 (Rev. 1)
Ecma/TC39/2014/029	Ecma International External Liaison Report for 2013-2014
Ecma/TC39/2014/030	TC39 RF TG form signed by Meteor Development Group
Ecma/TC39/2014/031	Draft Standard ECMA-262 6th edition, Rev. 26, July 18
Ecma/TC39/2014/032	Presentation on Instantiation Reform
Ecma/TC39/2014/033	Presentation "Bringing SIMD-128 to JavaScript"
Ecma/TC39/2014/034	Presentation "ES6 Opt-out Review Window #1"
Ecma/TC39/2014/035	Presentation "Unscopables"
Ecma/TC39/2014/036	Presentation "Should we eliminate early errors on parameter/let declaration name conflicts ?"
Ecma/TC39/2014/037	Presentation "ES6 Rev26 status"

2 Adoption of the agenda (2014/028-Rev1)

Agenda for the: 41st meeting of Ecma TC39

```
in: Redmond, Washington, USA
on: 29 - 31 July 2014
TIME: 10:00 till 17:00 PST on 29th and 30th of July 2014
      10:00 till 16:00 PST on 31st of July 2014
LOCATION:
    Microsoft Building 31 room 3055
    3730 163rd Avenue NE
```

```
    Redmond, WA 98052

CONTACT:
    Brian Terlson
    brian.terlson@microsoft.com
    7634983633
```

Please register before 22nd of July 2014.

1. Opening, welcome and roll call
   i. Opening of the meeting (Mr. Neumann)
   ii. Introduction of attendees
   iii. Host facilities, local logistics
2. Approval of the agenda (2014/028 rev. 1)
3. Approval of the minutes from June 2014 (2014/023)
4. ECMA-262 6th Edition (2014/031)
   i. Review Latest Draft (Allen)
   ii. Annex B: labelled FunctionDeclations (Allen)
      a. eg, foo: function f() {}
   iii. Function parameter/let declaration name conflict rules (Allen)
   see http://esdiscuss.org/topic/uniform-block-scoping
   iv. Review @@create design rationale and possible alternatives (Allen)
   v. Import-into-namespace syntax (Dave)
   vi. Unscopables (Arv)
   See https://bugs.ecmascript.org/show_bug.cgi?id=1908#c14 and subsequent comments for background on latest issues.
   vii. Consider if Object.assign should silently ignore null/undefined sources (Sebastian)
   viii. Arguments/caller poisoning on new syntactic forms - Arrows, Generators (Brian)
5. ECMA-262 7th Edition
   i. Math.TAU (Brendan)
      a. https://gist.github.com/rwaldron/233fd8f5aa440c94e6e9
   ii. SIMD.JS (Peter Jensen(Intel) and John McCutchan(Google))
      a. https://peterjensen.github.io/html5-simd/html5-simd.html (placeholder link)
   iii. Precision of Math trig functions (Dave)

a. https://twitter.com/BrendanEich/status/486615926020644866

iv. Flush demorm to zero (Allen)

a. http://esdiscuss.org/topic/float-denormal-issue-in-javascript-processor-node-in-web-audio-api

b. https://bugzilla.mozilla.org/show_bug.cgi?id=1027624#c30

v. Object Rest Destructuring and Spread Properties (Sebastian)

a. https://gist.github.com/sebmarkbage/aa849c7973cb4452c547

vi. NoSuchProperty affordance to throw on typo'ed property name or confused reference base (Brendan)

a. http://esdiscuss.org/topic/my-ecmascript-7-wishlist

b. https://twitter.com/BrendanEich/status/475067783282057216

6. Test 262 Status (2014/027)

6.1 Responses to the Ecma Contribution License Agreement (CLA) (2014/027)

7. Report from the Ecma Secretariat

7.1 Draft IETF JSON i-JSON 02, July 2014 (2014/026)

7.2 Ecma International External Liaison Report for 2013-2014 (2014/029)

7.3 Minutes of the 107th General Assembly, Heidelberg, June 2014 (GA/2014/078)

8. Date and place of the next meeting(s)

Approved:

i. September 23 - 25, 2014 (Boston, MA - Bocoup)

ii. November 18 - 20, 2014 (San Jose - PayPal)

Suggested:

iii. January 27-29, 2015 (San Francisco, Mozilla)

iv. March 24-26, 2015 (Sunnyvale, Yahoo)

v. May 27-29, 2015 (Menlo Park, Facebook)

vi. July 28-30, 2015 (Redmond, Microsoft)

vii. Sept. 22-24, 2015 (Boston, Boucoup)

viii. Nov. 17-19, 2015 (Paypal, San Jose)

9. Closure

The agenda was approved.

3 Approval of the minutes from the June 2014 Meeting (2014/023)

Ecma/TC39/2014/023, the minutes of the 40th meeting of TC39, Menlo Park, June 2014 were approved without modification.

4 On the work of the TC39 RF (Royalty Free) Task Force – "Opt-out" of ECMA-262 (TC39/2014/031, Ecma/TC39/2014/034)

As requested by the Ecma GA in December 2013 in Las Vegas since the January 2014 TC39 meeting the TC39 RF TG is operational.

According to 2014/013: "At the request of the TC39 RF TG, TC39 met on 4/9/2014 and voted unanimously to designate the draft of ECMA-262 (ES6) reviewed at its next meeting (May 2014) as the draft for Opt-out possibility. The Out-out 60 days period will be ending in approximately end of July, 2014. This will meet the requirements for TC39 action to provide an opt-out period prior to the approval of the next version of ECMA-262 (ES6)."

However, this plan was postponed. TC39 has decided at the last TC39 meeting in June 2014 that ECMA-262 Edition 6 should be approved only in June 2015 (so 6 months later as planned). Therefore also the "opt-out" as described above has been postponed to the July 2014 meeting.

Mr. Wirfs-Brock presented the opt-out plans (Ecma/TC39/2014/034 Presentation "ES6 Opt-out Review Window #1"):

The plan is that everything in ECMA-262 so far, plus the new draft until July 2014 should be subject of an opt-out. After the opt-out period if no one opts-out the group will consider the latest draft version of ES6 as RF. Before August 10, 2014 the Ecma Secretariat will formally notify the members of the TC39 RFTG of the start and end of the opt out period per the procedures identified in TC39 operating rules.

The 1st Opt-out window will be open between Aug. 11- October 11, 2014. During this time RFTG members can notify the Ecma Secretariat if they want to out for certain parts of the TC39/2014/031 (ECMA-262 draft of July 2014) from RF to RAND or no-license mode.

The 2nd Opt-out window will be opened between March 11 – May 18, 2015.

The above action and schedule has been approved by TC39.

5 For details of the technical discussions see Annex 1

6 Status Reports

6.1 Report from Geneva

6.1.1 Brief report about the work of the IPR Group on 3rd party text- and software contributions

Mr. Sebestyen said that The new webpage for submitting 3rd party test and software contributions have been put on the Ecma website. However, internal discussions have revealed that for external ES 6 tests this is too heavy, and the GitHub portal operated by **Brian Terlson** should be used.

It is understood that the Test262 Technical Report is neither a real Ecma Standard nor an Ecma Technical Report, but a different category of product: The various tests are sent to the Test Editor (Brian) who decides what test should go into the Test262 package. The base set of tests are then approved by TC39 and the Ecma GA. However the subsequent tests that are coming continuously and added by Brian to the test suite are not approved by TC39 and the GA anymore. This procedure should be discussed by the CC, and maybe a new Ecma product should be defined. What appears to be logical that afterwards e.g. in a yearly interval TC39 and the GA approves the latest version of Test262.

Regarding IPRs Test262 seems to be uncritical, therefore it should be discussed by the CC too if for such a product the External IPR policy can be simplified (e.g. no real signature needed to submit a contribution). External contributors have started to submit tests to GitHub. See table TC39/2014/027.

For other external, more substantive contributions the details of the work sharing between the Ecma secretariat and GitHub should studied. E.g. that the Ecma secretariat receives the submission, checks if the package is complete and also if the administrative part is correctly completed, while TC39 takes care of the technical content (incl. the software modules).

6.2 Json

Mr. Sebestyen reported that the IAB (and through that IETF) has established formal liaison status with Ecma and TC39. This has been approved by the IAB and now formally communicated to Ecma. **Matt Miller** from Cisco has been approved as liaison manager between IETF and Ecma TC39. The current plans are that IETF is entitled to send liaison representative to Ecma TC39 meetings (who can contribute to Ecma work according to Ecma and TC39 rules), and Ecma TC39 can also send liaison representative to IETF meetings (who can contribute to IETF work according to IETF rules). TC39 has appointed **John Neumann** as TC39 liaison representative. This has been communicated to IETF / IAB.

It was noted that currently the Ecma JSON work and standard is stable and no changes are expected soon. The IETF draft I-JSON Format document was distributed to TC39 for comments (TC39/2014/026), if any.

6.3 ITU-T SG16

Mr. Sebestyen reported that the ITU-T SG16 has been working some time on a new multimedia system and terminal called ITU-T H.325. In the current documents it is documented that for communication protocol they want to use JSON and they are also interested in a fast ECMAScript that can be used to download and perform any sort of media codec (both standardized and private) It is suggested that closer liaison between the groups and organizations should be established on those subjects This liaison was discussed and endorsed at the GA meeting in Heidelberg. TC39 has appointed **John Neumann** as TC39 liaison representative. This has been communicated to ITU-T SG16.

6.4 JTC 1/SC 22

TC39/2014/029 is the liaison report prepared by **Rex Jaeschke** to the upcoming JTC 1/SC 22 Plenary meeting. SC 22 is the JTC 1 place where ECMA-262 has been fast-tracked. The report has been prepared by Rex based on the TC39 reports and 029 is just for information to TC39.

6.5 Ecma Recognition Award

As proposed by TC39 the Ecma GA has awarded **Brendan Eich** and **Waldemar Horwat** for their outstanding contributions to TC39 and Ecma over many-many years with the Ecma Recognition Award. **Mr. Sebestyen** handed over the awards to **Mr. Eich** and **Mr. Horwat** at the Redmond meeting.

6.6 Meteor Development Group joined Ecma TC39.

Mr. Sebestyen has informed TC39 that recently the Meteor Development Group from San Francisco has officially joined TC39 and the TC39 RFTG.

7 Date and place of the next meeting(s)

Schedule 2014 meetings:

- September 23 - 25, 2014 (Boston, MA - BoCoup)
- November 18 - 20, 2014 (San Jose, CA - PayPal)

Schedule 2015 meetings (suggested but not yet approved):

- January 27-29, 2015 (San Francisco, Mozilla)
- March 24-26, 2015 (Sunnyvale, Yahoo)
- May 27-29, 2015 (Menlo Park, Facebook)
- July 28-30, 2015 (Redmond, Microsoft)
- Sept. 22-24, 2015 (Boston, Boucoup)
- Nov. 17-19, 2015 (Paypal, San Jose)

8 Any other business

TC39 members have expressed their wish to have a dedicated “break-out” time during TC39 meeting. It has been agreed that before the meeting the planned “break-out” subjects should be communicated to **Mr. Neumann** and the **Secretariat**. At the next meeting when approving the agenda it will be decided which break-out subjects will be discussed. It needs to be discussed how many parallel break-out sessions should be performed and how the reporting to TC39 should be done.

9 Closure

Mr. Neumann thanked **Microsoft** for hosting the meeting in Redmond and hosting the dinner on Tuesday, the TC39 participants their hard work, and **Ecma International** for holding the social event dinner Wednesday evening. Special thanks goes to **Rick Waldron** to take the technical notes of the meeting.

Annex 1

 # July 29 2014 Meeting Notes

Brian Terlson (BT), Dmitry Lomov (DL), Waldemar Horwat (WH), Allen Wirfs-Brock (AWB), John Neumann (JN), Rick Waldron (RW), Eric Ferraiuolo (EF), Jafar Husain (JH), Jeff Morrison (JM), Mark Honenberg (MH), Caridy Patino (CP), Sebastian Markbage (SM), Istvan Sebestyen (IS), Erik Arvidsson (EA), Brendan Eich (BE), Mark Miller (MM), Sam Tobin-Hochstadt (STH), Domenic Denicola (DD), Peter Jensen (PJ), John McCutchan (JMC), Paul Leathers (PL), Eric Toth (ET), Abhijith Chatra (AC), Jaswanth Sreeram (JS), Yehuda Katz (YK), Dave Herman (DH), Brendan Eich (BE), John-David Dalton (JDD)

Introduction

JN: (Welcome and host details)

Introductions.

Agenda: https://github.com/tc39/agendas/blob/master/2014/07.md

JN: Agenda approval?

Approved.

JN: Minutes from June 2014 approval?

Approved.

4.1 Review Latest Draft

(Allen Wirfs-Brock)

rev26-summary.pdf

AWB:

Slide 1

- "Task" => "Job"

- Generator oject return method/for-of/in loops use return method on generatorsGetMethod, now treats null and indefined equiv as meaning no method available

- Eliminated the ability of Proxy handlers to extend the set of property descriptor attributes thet expose vis [[GetOwnProperty]]

- Added invariant checks for Proxy [[OwnPropertyNames]] internal method

- Added an informative generator function based definiton for ordinary object [[Enumerate]]

- Another round of updates to 9.2.13 FunctionDeclarationInstantiation to fix various scoping bugs.

- Eliminated duplicate property name restrictions ob object literals and class definitions

- Revisited @@unscopabe support in Object Environment Records

RW: Clarification about #3

MM: Prevent the ability to lie about decriptor values; prevent leak

"time of check to time of use" (ToCToU) vulnerability

(re: #5)

WH: Informative implementation may cause readers to incorrectly assume that anything that doesn't conform to to that informative implementation is wrong. In particular, worried about users assuming that implementation behaviors that the informative implementation doesn't do can't happen.

(discussion re: normative vs informative prose in spec)

WH: "Note" sections?

AWB: Sections marked "Note" are informative, making them normative would be redundant

WH: Too many "Note" sections appear to be normative rephrasings of other normative text, which we labelled as informative only because we're afraid of redundancy. Then when we have one which describes something that behaves quite differently from the normative text, it can be misleading.

Slide 2

- For-of now throws if iterable value is null or undefined (also reverted comprehension to throwing for that case)

- `Date.prototype.toString` now uses NaN as its time value when applied to an object without a [[DateValue]]

- Function poison pill caller and arguments properties are now configurable

General concern about whether you could take the sloppy-mode and add it to a strict-mode function via reflective APIs.

Turns out that the sloppy mode behavior is implemented in all current engines as a magic value property, so this will not be possible (phew!).

- `await` is a FutureReservedWord when parsing and the syntactic grammar goal symbol is Module

- Better integration of `Object.prototype.toLocaleString` and `String.prototype.toLocaleString` with ECMA-402

- Added `name` property for bound functions in `Function.prototype.bind`. Fixed bugs in generating length property in `Function.prototype.bind`

- Tweaked Script GlobalDeclarationInstantiations to deal with error situations that could arise from misusing proxies for the global object.

- Changed handling of NaN from a sort comparefn to match web reality (http://bugs.ecmascript.org/show_bug.cgi?id=2978)

MM: (re: #6) What is the bound name?

AWB: "bound ..."

(per previous resolution—find and link)

AWB: (re: #8, sort behavior when comparison function returns NaN) the change adds:

"If v is NaN, then return +0. "

WH: Note that that bug contains other examples where sort would still be inconsistent.

WH: The issue here is that +∞ - +∞ is NaN, which means that using a-b as a sort function allows you to compare +∞ with -∞, but +∞ is not equal to itself.

WH: I wrote this wording in ES3 and I'm fine with this change. It will fix the behavior of sorting arrays containing ±∞ but not NaN's. With NaN's you'll still get an inconsistent sort order (because the ordering relation is not transitive) and hence implementation-defined behavior.

MM: Worried that implementation-defined behavior could do bad things such as violate memory safety. Should state that it doesn't.

MM: (filing bug to make language more specific to avoid memory safety violations)

AWB: There was never a concern about violating memory safety because we don't define memory safety

WH: I tried to formalize it in ES3 but it was too much trouble. I wanted to state that the sort always produces a permutation, but that doesn't apply to sorting oddball input objects containing things such as holes with prototype properties showing through, read-only properties, getters/setters, proxies, ... Can't write "sort doesn't violate memory safety" without formalizing what memory safety is.

MM: Will write concrete suggestion for handling and submit as part of bug

https://bugs.ecmascript.org/show_bug.cgi?id=3089

Slide 3

- Updated Symbol conversions:

  - `aSym == "not a symbol"` produces false.

  - `var s = Symbol(); s == Object(s)` produces true.

  - `"foo" + aSymbol` or `aSymbol + "foo"` throws TypeError.

  - Symbol `@@toPrimitive` returns the wrapped symbol value.

  - `ToNumber(aSymbol)` throws.

- Spread now works on strings `var codeUnits = [..."this is a string"]`

- `yield *` now works with strings: `function * getchars(str) {yield * str}`

- Annex B support for function declarations in IfStatementClauses

- Annex B (and 13.12) support for legacy labelled FunctionDeclarations

- Updated Annex C (strict mode summary) WRT ES6 changes and extensions

EA: (re: #2) Removed the Object check?

BE: Andreas didn't want values on the right of a destructuring (number, etc)

EA: Definitely want to spread strings

BE: Agreed, we should revisit.

(Added: https://github.com/tc39/agendas/commit/370e3029d01659620e0ca03bf370eb5beefca45e)

Re: #4, the resolution: https://github.com/rwaldron/tc39-notes/blob/master/es6/2014-06/jun-6.md#block-scoping-issues

BE:

```js
(function f() {
  console.log(g);
  L: function g() {};
})();
```

DH: The grammar: inside a LabelledStatement, can't start with "function"

BE/AWB: Clarification of Statement and Declaration

DH: Suggest: we can deal with this post-ES6. It's a useless thing that happens to parse and not worth our immediate attention.

AWB: We need to decide if this is a function declaration, does it hoist?

DH: But won't affect the web, existing can't be relied on. The existing work has been done, but no additional work

WH: Treat this the same as function declarations inside of statements: ie. `if (true) function f() {}`. Do we allow `while (true) function f() {}` ?

YK/BE: Let's take this offline.

WH: Let's keep it as it is in ES5

AWB: Whoever maintains web specs...?

BE: Not all browsers do the same:

(results of above code)

- SpiderMonkey: ReferenceError

- V8: function g() {}

- JSC: function g() {}

- Chakra: function g() {}

AWB: Spec is up to date, without the modules work.

4.6 Unscopables

(Erik Arvidsson)

es6-unscopables.pdf

(https://bugs.ecmascript.org/show_bug.cgi?id=1908#c14)

Object instead of Array

```js
Array.prototype[Symbol.unscopables] = {
  ...
};
```

(with null prototype)

Walk The [[Prototype]] Chain

For

- HasBinding

- GetBindingValue

But nor for:

- SetMutableBinding

AWB: essentially replicating the prototype lookup algorithm in two additional places. Realized a third.

EA: ...

Setter Issue

SetMutableBinding ignores @@unscopables so we can get a mismatch:

```js
with (object) {
  x = 1;
  assert(x === 1); // can fail
}
```

YK: Needs to be written?

AWB: More that wasn't considered. Proxy issues in bug (above)

EA: The problems arise when your prototype has getter or setter or proxy. The result of HasBinding can return true for a property further down the prototype chain, but then Set got invoked using a setter that was black listed in HasBinding.

AWB: Proposal is, do what we've done and leave setting as is.

Only apply unscopable at the local level, don't walk the prototype chain

Any binding resolution operation, on a `with` env record:

1. looks up unscopables, doesn't matter where in the prototype

2. checks the `name` against unscopables (has, not hasOwn)

3. if found continue up to the next level

Only applies to "with environments"

STH: Should unscopables affect things in the prototype chain

WH: Does it apply to both reads and writes?

AWB: Yes

STH: Looks only at the object with unscopables as own property?

AWB: No, it's on the prototype, to not own

STH: Should this apply to all spec algorithms?

YK: Everyone agrees?

STH: AWB's proposal says no

AWB: Only object environment records for `with`

STH: it _should_ apply to SetMutableBinding?

AWB: Yes.

EA: The reason for not doing [[Get]] was because you might have instance properties on the object

YK: Can link the unscopables

EA: Agreed. And don't do hasOwn

STH: can break existing programs that use instance properties

BT/AWB: Discussion about compatibility.

EA: What about Globals & unscopables? The global object is an ObjectEnvironment too. Do we plan on adding unscopables?

Generally, no.

YK: Are we sure there is no case to use unscopables on the global object

RW: Could we specify no unscopables on global now and relax it later?

EA: for ES7

AWB: If this only applies to `with` environments, then that's not part of the global object

MM: Unless you do: `with(global object) {...}`

Confirm.

AWB: Is this a function of `with` or the environment?

Conclusion/Resolution

- @@unscopables only works inside of `with` object environment records, not global object environment records.

- Revert to the previous algorithm:

  1. looks up unscopables, doesn't matter where in the prototype

  2. checks the `name` against unscopables (HasProperty, not HasOwnProperty)

  3. if found continue up to the next scope level

4.8 Consider if Object.assign should silently ignore null/undefined sources

(Sebastian Markbage)

(Request slides)

SM:

```js

Object.assign({}, undefined);

```

This throws, but propose that it shouldn't.

SM/AWB: `Object.keys` was relaxed

```js

Object.assign(undefined, {});

```

This should still throw

DH: undefined and null are probably treated the same by ==

Do we want to treat null and undefined the same? Probably not.

DD: The mental model should be for-of + default arguments, not for-in

MM: use case for tolerating the null is in JSON data. JSON has no way to represent undefined, except for null

JH: or omission

SM: Covered existing libraries to use `Object.assign`, feedback almost always included the undefined case.

JM: Did you distinguish null and undefined?

SM: No

YK: We should distinguish or we have two nulls

Conclusion/Resolution

- do not throw on undefined

- will throw on null

Short discussion about making generator.return() throw a special exception.

DH: Want to bring up Andy Wingo's preference (discussed on es-discuss) for modeling return() as an exception rather than a return.

General opposition.

Conclusion/resolution

- keep as is: return() method produces a return control flow, not an exception

##

AWB: In the process of for-of, if a throw occurred, does that turn into a throw to the iterator?

NO.

Yield *

AWB: Does an internal throw

When a `generator.throw()` is called and the generator has a yield* the spec currently calls `throw` in the yield* expression

DH: Call return() on the outer generator, delegates calling return() to the delegated `yield *`

BE:

```js
function* g() {
  yield* h();
  console.log(42);
```

```
}

function* h(bound) {

  try {

    for (let i of range(bound)) {

      yield i;

    }

  } finally {

    console.log("h returned");

  }

}

let it = g();

it.next();  // returns {value: 0, done: false}

it.throw(); //
```

AWB: If `it.return()` we would send a return to the `h` instance.

Confirm.

AWB: if `it.throw()` do we send a return to the `h` instance?

MM: we would do a `throw`?

AWB: we wouldn't. Think of the `yield*` as a for-of. h() doesn't know what its client is.

The problem:

the resumption from the yield is throw, back in the `yield*`, what to call on `h()`

DH: Propagate, that's the point of `yield*`, it should behave as if the inner generator is inline and anything it does propagates.

AWB: My mental model of `yield*` is that it expands to a for-of { ... yield ... }

MM: You should not think of it that way.

DH: You should not base your mental model off of an expansion; you should base it off of what `yield*` is meant to be used for.

The desugaring into for-of is not at all straightforward.

AWB: the desugaring in the algorithms in the spec is not actually that complex...

DH: the way to think about this is to directly inline the body of `h` into the body of `g`, not as a generator equivalent. This is the generator analogue of beta-equivalence.

AWB: why don't you do that for any function then?

DH: well, if we had TCP, we would have beta-equivalence.

YK: (Saw that one coming...)

DH: the important refactoring property to have is that you can extract out some generator logic into a helper function and then call it with `yield*`. Ben Newman was talking about a similar/related thing. It is very important that the throw to the outer generator get delegated as a throw through the inner generator.

MM: What is the model that the user has: who is the `throw` complaining to?

AWB: And who has control? Is the generator calling out and getting something back or into another generator.

DH: for-of and `yield*` diff roles: for-of is a consumer and generator stops there. `yield*` is consuming and producing by composition.

JH: Two models: consuming and emitting. `yield*` is a stream fusion,

STH: `yield*` compensates for the shallowness of yield

DH: It allows composition of generators

JS: You could expand these to for-of

YK: Those that work in C# find this to be a natural way to think of this, but others may not

JS: My concerns are speed

DD: Think about how you could desugar and see where it falls down

YK: disagreement

DH: Fall over in more cases

MM: Would it be plausible to have the `throw` propagate to the inner generator as well?

DH: check out [PEP-380](http://legacy.python.org/dev/peps/pep-0380/#formal-semantics). The desugaring is mind-bogglingly complex, but the refactoring principle is very straightforward. Refactoring should not introduce corner cases where it behaves differently.

MM: consensus that yield* delegates throw?

JS: no objection

Conclusion/Resolution

- `yield*` delegates `next()`, `return()` and `throw()`

- for-of propagates abrupt completion outward, calls the iterator's `return()`

AWB: Another question... when we do the `return`, that may return a value and currently throwing that value away. There is no way to override the normal loop termination.

DH: If we do a throw that causes us to call `return()` on a generator, and that returns a value, the value is dropped on the floor, which is consistent.

AWB: If the `return` call on the iterator is an abrupt return (normally means an exception)...

MM: The call to _return_ itself completes abruptly?

AWB: Yes

MM: it's "finally-like", that supersedes

AWB: In one sense, a dropped exception

BE: Let's have a smaller group with the champions look at the final specific details.

4.11 Consider adding "attribute event handlers" to ANNEX B

(Allen Wirfs-Brock)

AWB: Add to Annex B the semantics of defining an attribute handler so that the HTML spec can get out of the business of spec'ing a distinct form of ES function.

MM: An internal function for other specs?

AWB: If you're implementing a browser, you'd follow this specification.

YK: Isn't this just `with`?

DD: No, there is more there (see http://kangax.github.io/domlint/#5 for details)

EA: Why can't this be in the HTML living spec?

AWB: That's the problem, Hixie is using internal APIs, in some cases incorrectly.

MM: Is this for ES6?

BE: Not important enough for ES6

AWB: not a lot of work.

No support for ES6

DD: I don't think we should push for ES6

RW: Last meeting pushed back 6 months, this isn't that valuable.

Conclusion/Resolution

- Scheduled for ES7 Annex B

4.9 Arguments/caller poisoning on new syntactic forms - Arrows, Generators

(Brian Terlson)

BT: All function-like things agree on having arguments object

DH: What's wrong with having the poison properties?

BT: The motivation for having those properties may not apply to those new syntactic forms.

MM: Keep them and they are there and configurable, or mandate w/o caller and arguments properties

EA: Too much weight on edge cases

MM: Born without extra properties would be fine

AWB: Do we even need poison pills?

BT: Can we get rid of it?

AWB: Can't add properties called "caller" and "arguments" to strict mode functions

MM: New forms, the properties are absent.

AWB: Are these properties implemented as own properties or inherited?

MM: And we agreed that `Function.prototype` remains a function

Conclusion/Resolution

- Get rid of all poisoned caller and arguments, except for the poisoned caller and arguments on Function.prototype

- All functions born of non-legacy function syntactic forms do not have caller and arguments properties

4.10 Signaling stability of existing features

(Domenic Denicola and Yehuda Katz)

YK: Problem: ES6 signaling is too fuzzy. ES6 is a monolithic thing. Three stages:

1. Seeking the happy-path semantics

2. Find the happy-path semantics

3. Finalize edge cases, done.

Need to

AWB: What are we doing that sends the wrong message?

YK: eg. when we said were pushing for a 6 month extension, people assume this means all features are unstable

RW: (relaying additional experience re: above)

DD: Proposed stages:

1. Locked

2. Stable

3. Almost Stable

4. Stabilizing

5.

(need to fill in descriptions from proposal document)

AWB: The problem is that some things that are "locked" become "unstable"

JM: It's possible to be "unstable" until spec is published

STH: And publication isn't even the end either.

WH: Any time someone proposes something like this, I want to ask if this would've correctly predicted the results had we done it some time ago. For example, had we done this, say, in January then comprehensions would have been in the Locked stage, but then we took them out.

WH: Math functions are listed in the Locked stage in your proposal but at the same time we have important discussions at this meeting about their precision.

?: Math function precision could be a different feature.

WH: That's weasel wording — when you want to change some aspect of a feature, you just move the goalposts to make that aspect a separate feature.

DD: "we're" not good a the PR of specification churn

MM: Not sure what this proposal is really addressing. The community has a way exxagerated sense of instability, and over-reacts to any change. So what?

JM: Won't implement modules at FB because of churn.

AWB: Does this change the model for ES7?

DD/YK: No.

AWB: So this is for the next 5 months of ES6?

MM: Not enough community feedback because the feedback is limited to only those that are willing to accept churn?

YK: Yes

DH: Priorities: getting feedback for ES6 is low, because it's too late in the game. Focus feedback priority on ES7. Despite the inclusion of more practitioners in the TC, there are still broad misunderstandings about TC39 and ES6.

DD: The perception is that ES6 is the new ES4, except that we all know this isn't true.

AWB: Two things... concerns about how you're defining these stages. Who is going to do this work? I don't want to say 5 months from now that the spec is "unstable" in its entirety.

Mixed discussion about implementor opinion of feature.

AWB: We don't want uninformed feedback that we have to filter

DH: It's really bad to not talk to the community, because people think the worst.

YK: A vast majority of ES6 is stable

MM: How we should be messaging as individuals. TC39 should not be spending time

PL: This is all too hard to quantify and assess because change _will_ happen.

DH: Stability chart is

Conclusion/Resolution

- Individual evangelism, feedback and outreach

Postpone Realm API to ES7

MM: Can we?

DH: I'm ok with this, but don't want to be in a situation where we're permanently postponed while waiting for a security review. Let's reach out to other security reviewers.

DD: I can implement a draft implementation in node for the purpose of review.

AWB: The modules spec depends on realms

DH: Only the ability specify the Realm in user code needs to be removed.

MM: let's pull Realm from ES6, if there are issues we can address them.

AWB: The Realm API cleaned up how you go about eval'ing things.

DH: This clean ups can stay as-is, now ready for the reflection to come in ES7

AWB: Not going to have anyway for user code to eval in Loader.

DH: w/o Realm no ability to virtualize eval. Doesn't effect utility of Loader. The specification is detailed and complete, should continue moving forward.

MM: And any issues that are encountered can be addressed.

DH/YK: Agreement that test implementation in node is ideal (vs. browser)

Discussion re: security issues created by implementations in browsers.

MM: The security implications and risks are greater for Realm because this _is_ the sandbox api.

DH: Agree.

Conclusion/Resolution

- Realm postponed to ES7

Revisit Object.assign()

JDD: The issue currently: if we allow `undefined` then `null` is the only value not allowed. I don't see anything distinguishing.

It's strange that `null` is singled out like this. When `null` is used correctly, it makes sense here.

DD: Then the argument is that it should also throw

JDD: No, it shouldn't throw.

YK: Should throw on numbers, booleans, etc.

JDD: Should affect `Object.keys` as well

YK: Doesn't have to

JDD: There shouldn't be special casing for null and undefined

DD: `undefined` triggers the default parameter, `null` doesn't.

YK: The mental model is: `undefined` is missing, `null` is not

AWB: Mentions the relaxation of rules for `Object.keys`

YK: We should enforce the difference between `null` and `undefined`

SB: (details about a study in FB code re: how `null` and `undefined` are being used)

DH: We need to decide whether there is a useful programming model for these cases: `null` and `undefined`

JDD: I think the boolean, number, string values are a side effect because they are just treated as empty. Propose to treat _both_ `null` and `undefined` the same way.

JM: Sounds like a better argument against boolean, number, string.

AWB: (example of a number object to be extended)

SB: The differnce is target vs. source, `null` and `undefined` throw for target.

Mixed Discussion

DH: To avoid rehashing, guiding principle:

- `null` represents the no-object object, just like NaN represents the no-number number

- `undefined` represents the no-value value

Conclusion/Resolution

- Overriding previous resolution:

  - `Object.assign` does not throw on `null` or `undefined`

- Adhere to the guiding principle stated above

Test 262 Update

(Brian Terlson)

BT: CLA is now online, fully electronic. Lots of contributions, specifically awesome help from Sam Mikes.

- Improvements to the test harness

- Repeat contributors

- Converting ES5 to ES6

- Converting Promise test inbound

- Massive refactoring commit

Discussion about Promise testing

JN: Work with Istvan to write a press release for this?

BT: Yes.

DD: Node runner?

BT: MS has been using a node runner internally, I've pulled out the useful pieces and pushed to github: https://github.com/bterlson/test262-harness

Conclusion/Resolution

- announcement effort

 # July 30 2014 Meeting Notes

Brian Terlson (BT), Dmitry Lomov (DL), Waldemar Horwat (WH), Allen Wirfs-Brock (AWB), John Neumann (JN), Rick Waldron (RW), Eric Ferraiuolo (EF), Jafar Husain (JH), Jeff Morrison (JM), Mark Honenberg (MH), Caridy Patino (CP), Sebastian Markbage (SM), Istvan Sebestyen (IS), Erik Arvidsson (EA), Brendan Eich (BE), Mark Miller (MM), Sam Tobin-Hochstadt (STH), Domenic Denicola (DD), Peter Jensen (PJ), John McCutchan (JMC), Paul Leathers (PL), Eric Toth (ET), Abhijith Chatra (AC), Jaswanth Sreeram (JS), Yehuda Katz (YK), Dave Herman (DH), Brendan Eich (BE),

RFTG Admin: ES6 Opt-out period.

(Allen Wirfs-Brock)

ecma-262-6-optout1.pdf

AWB: This is the opt-out period: Aug. 11, 2014 - Oct. 11, 2014

Final opt-out window: March 16, 2015 - May 18, 2015

Read the policy, speak with Istvan Sebestyen for further information.

The opt-out version of the spec is: ECMA-262 6th Edition, revision 26, document: tc39/2014/031

4.4 Instantiation Reform (Review @@create design rationale and possible alternatives)

(Mark Miller, Allen Wirfs-Brock, Dmitry Lomov, Tom Van Cutsem. Based on Claude Pache proposal)

instantiation-reform.pdf

AWB: Currently:

```js
new Foo(arg)
```

1. `Foo[[Construct]](arg)` ::=

  1. `let obj = Foo[@@create]()`

  2. `foo.call(obj, arg)`

The actual constructor method typically does all of the initialization and setup on the instance object.

Issues:

(DOM centric)

1. If instances aren't sufficiently initialized by @@create, then instance objects could leak (e.g. via a nefarious decoupling between the @@create allocation process and the constructor-function initialization process)

2. @@create could be called directly

DH: Do we have concrete examples of this problem?

JM: Dealing with legacy code where you want to subclass from legacy constructors, need to set up state, `this` uninitialized

AWB: Gives `Date` example, where `[[DateValue]]` isn't setup until `super()`

DH: The initialization of the internal field cannot happen until the super constructor is called to create _that_ field.

YK: This is caused by the @@create not accepting the constructor arguments.

AWB: Yes. Propose: Remove @@create, replace it with @@new, which will make the arguments available to

YK: When you subclass, it's normal to call super and pass the arguments.

AWB: @@create is a property of the constructor, not the instance.

Creating a subclass, you may want to do something with the arguments before passing to `super`

JM: or adjust some state on the subclass instance before calling `super`

AWB: There is a complication that built-ins have: different behaviour when `new`'ed or `call`'ed

No internal distinguishing mechanism. No way to indicate that a constructor was called or newed.

YK: Don't think we need to be concerned with handling

AWB: A subclass will create a "call" up to the `super`

Explanation of current spec handling.

JM: Issues: were we called, or newed. One deals with intermediary state initialization.

AWB: The issue is coupling between state and the instance.

Do we agree that there's a problem?

(no one says no)

JM: There are scenarios where a subclass wants to initialize state before calling super()

YK: It seems like a feature, not a bug

WH: What are you calling the intermediary state

YK: The fact that you can observe the creation

There has to be brand check

DH: The simplest way of saying is that all need brand checks.

YK: Can do: `foo.call(any)` and there is obviously a check there.

AWB: Internal slots are initialized in an atomic unit,

YK:

DL: TypedArrays missing creation information

AWB: You can move all the logic [to @@create or @@new or something akin], but you've just created another constructor

WH: Why not do everything from @@create in the constructor

DL/AWB: Jason Orendorff's proposal.

DH: (not-quite-right summary of Jason's proposal)

AWB: One way: reify [[Construct]] to the user

DH: When `new Foo(args)`, calls `Foo[@@new](args)` ... ?

DL: Just pass the args to `@@create` and change `@@create` to `@@new`

NM: But then the subtype must have same signature

AWB: 2 viable options for ES6

- Live with what we have, @@create is grungy, but not unsafe

- Alternative, originating with Claude Pache

Bring back to constructor with atomic invocation. I'm for this approach and it's reasonable for ES6

(Mark presenting...)

MM:

Goals

Subclass exotics

Avoid un (or partially) initialized exotics

ES5 compat (aside from "rcvr")

ES6 class compat (aside from @@create)

Reliable test for "am i called as a constructor?"

Support base-creates-proxy scenario

```js
class Derived extends Base {
  constructor() {
    // TDZ this, on "new Derived..." etc.
    super(...otherArgs); // this = what super returns
    // this is initialized.
  }
}

//
function Base(...otherArgs) {
  // implicit this = Object.create(mostDerived.prototype, {});
}
```

AWB: The `super()` call is calling the super class constructor as a _constructor_ when `new Derived()`— that's important.

WH: When constructor() is called as a function, super is called as a function too?

MM: Yes

WH: What causes the TDZ to appear? The statically visible presence of a super call in the body of the constructor?

AWB: Yes

WH: What if the super call is inside an arrow function?

BE: If `Derived` called without `new`?

AWB: `super()` is called a non-constructor.

WH: super cannot appear in a nested function?

AWB: they can appear, but... (trails off)

JM: A related use case is being able to set up stuff on the instance before calling super()

AWB: Show me code that does that, to make sure we don't break that.

BE: code that doesn't actually use super() won't break, and there is no such code yet

MM: Base is an example of a function that doesn't have a super call (because it can't). On entry, before user code, implicit init this of fresh newly created object. This is a difference from ES5. The "mostDerived" prototype ...?

AWB: this actually isn't a difference from ES5, because there is no super() in ES5

MM: you are correct

MM: how do people feel?

JM: It's not an issue with ES5 -> ES6 legacy, it's an issue with ES6 class designs that evolve over time

YK: my concern is the pedagogy of this approach.

MM: the pedagogy is as shown in this slide.

DH: No! It cannot be taught this way.

BE: let's just let Mark present.

MM:

From Claude Pache

```js
F.[[Construct]](args, rcvr)
```

- Distinguish functions-which-call-super

- Vanilla function at end of super-call-chain is base (instantiation postponed to base entry)

**Modifications to Claude's proposal **

```js

F.[[Construct]](args, rcvr)

```

- mod: Only MOP signature change

- Distinguish functions-which-call-super

- mod: call-super-as-a-function

- `super()`, but not `super.foo()`

- Vanilla function at end of super-call-chain is base (instantiation postponed to base entry)

- mod: instantiation postponed to base entry

YK: What about subclass constructors that don't include a call to `super()`.

AWB: Throw when `new`'ed

Agreement.

JM: I still have issues with state initialization

YK: Issues

BE: Concern about setting properties on the instance before `super()`

JM: Code patterns exist, they won't just go away.

AWB: Can get around it with `super.constructor()`

BE: Lose the static analysis of `super(` (right paren intentionall omitted)

MM:

[[Call]] Traps

```
```

F(...args) -> F.[[Call]](undefined, args)

Derive.[[Call]](const this, args)

  super(...other) -> super.special_name(...other)

```

WH: What is the special name?

MM/AWB/DD: (to Waldemar) This is the ES6 spec

WH: Explain?

AWB: methods that ref super are bound to an object where the super ref takes place. that binding is the current inst. two bound values object where look up starts and the method name.

MM:

**[[Construct]] Traps**

```

new F(...args) -> F.[[Construct]](args, F)

Base.[[Construct]](rcvr, args)

  entry -> const this = [[Create]](rcvr.prototype)

Derive.[[Construct]](args, rcvr)

  entry -> TDZ this

  super(...other) -> const this = super.[[Construct]](other, rcvr)

```

**Remaining Requirements**

Am I called as a constructor?

What is the original's constructor's prototype?

How do I provide alternate instance to the subclasses?

**Am I called as a constructor?**

```js

function F(...other) {

```
  let constructing = false;
  try { this; } catch(_) { constructing = true; }
  super(..);
}
```

Base instantiates proxy scenario

```js
function Base(...other) {
  return new Proxy(... this.prototype ...);
}
```

Kill two birds with "new"

```js
function Date() {
  let now = $$GetSystemTime();
  if (new*) {
    let obj = Object.create(new*.prototype);
    // obj@now = now; // private “now” state
    return obj;
  } else {
    return ToTimeString(now);
  }
}
```

MM: Proposing a new special form (shown as `new*` above) whose value is the most derived otherwise undefined.

The test being: reliably check if I am called as a constructor.

WH: Unless the most derived receiver is falsy. Is there a way to create such a thing?

AWB: Yes, you can invoke the reflection trap and specify a falsy value for the receiver.

MM: Modified the above example to:

```
if (new* !== void 0) ...
```

AWB: We could fix this by throwing if reflection is used to invoke a constructor with undefined as the receiver.

Reflection and Proxies

- `Reflect.construct(F, args, rcvr)` (throw on undefined)

- construct trap: `construct: function(target, args, rcvr)`

YK: How does this work in ES5 classes?

AWB:

YK: Is conditional super a hazard?

MM: Yeah

AWB: New power, new complexity

YK: Exposing something that was implicit into the main path. Calling super in a constructor conditionally?

EA: Bug, can be fixed

AWB: (re: Date example) Where it shows Object.create...

DL/AWB: If you conditionally forgot to call `super()`, [[Construct]] will have to check at exit and throw.

YK: With @@create you had to know what you were doing. With this you could tread on weird cases without knowing it.

BE: Lets park that discussion for now.

DL: The sign that TypedArray is giving us is a sign of what user code might do as well so they will have the same issue.

AWB: Better direction. Don't go another decade where implementations can have private slots, but user code cannot.

MM: The direction I've presented is what I prefer. What I'm _actually_ proposing is that we allow Allen to be the champion and work out the details remaining. Objection?

None.

BE: No objection, but I want to make sure Allen works with YK, JM and Boris Zbarsky

Conclusion/Resolution

- Agreement to MM proposal: Allen to be the champion and work out the details remaining

(This did not gain _final_ consensus, as follow up was necessary)

... On to JM objections

JM: Start with a class never meant to be subclassed. Later you want to re-use aspects of this class, but need a way to hook in to the subclass `this` before `super()` class.

DH: eg. an `initialize` method that just sets up properties and state

AWB: If it's just state that it doesn't need to know about, it doesn't matter?

If it's state that does need to know about, what the channel? Seems very tenuous at best

JM: An example, we want to re-write some of the dom before calling the parent constructor.

DL: How is dom related?

WH: Are you unable to munge parameters to constructor?

AWB: Consider a scenario where the DOM mutation is contained in a method of the super class that must be invoked, for side effect, with no dep on object instance state, but is an instance-side method. The way around is to access your prototype or original prototype and invoke the method on the instance

Discussion of legacy scenarios and validity.

AWB: More of a refactoring issue

YK/JM: Agreement that we need more real world cases.

MM: Need a very concrete example, showing: the code written that wasn't intended for subclassing and the newer code that's attempting to subclass.

YK: There are issues created by memoization

Discussion re: subclassing in general.

MM: Need to do the concrete example exercise, and before the end of this meeting.

AWB: The fallback is that we just keep what we have.

DD: Worried about @@create, that it won't be possible to subclass because there is negative feedback

MM: Break on this discussion until JM has adequate examples.

5.2 SIMD.JS

(Peter Jensen and John McCutchan)

simd-128-tc-39.pdf

Other slides: https://peterjensen.github.io/html5-simd/html5-simd.html#/

JMC: (introducing SIMD, Single Instruction Multiple Data)

Slide presentation

Proposing a Fixed 128-bit vector type as close to the metal while remaining portable

- SSE

- Neon

- Efficient scalar fallback possible

Scales with other forms of parallelism

WH: Why fixed 128, given that x86 SIMD is now up to 512-bit vectors?

DH: Plenty of real world use cases for this, video codecs, crypto, etc.

STH: Wider widths?

JMC: Yes.

AWB: Works with IBM PowerPC?

JMC: Yes, overlapping instruction sets.

Proposing, specifically:

- SIMD module
  - New "value" types
  - Composable operations
    - Arithmetic
    - Logical
    - Comparisons
    - Reordering

- Conversions

- Extension to Typed Data

- A new array type for each

float32x4, 4 IEE-754 32-bit floating point numbers

int32x4, 4 32-bit signed integers

float64x2, 2 IEE-754 64-bit floating point numbers

Float32x4Array, array of float32x4

Int32x4Array, array of int32x4

Float64x2Array, array of float64x2

Object Hierarchy

SIMD

-> int32x4

-> add, sub, ...

-> float32x4

-> add, sub, ...

-> float64x2

-> add, sub, ...

DH: Introduce new value types, but does not depend on user created value types

JMC: Examples...

```js
var a = SIMD.float32x4(1.0, 2.0, 3.0, 4.0);

var b = SIMD.float32x4.zero();
```

```

MM: Why is zero() a function instead of a constant?

JMC: It could be a constant.

... additional examples. See Slides.

STH: How much of the difference is SIMD and single precision?

JMC: I don't have numbers, but would say SIMD

MM: Do SIMD instructions preserve denormals or flush?

JMC: ARM flush to zero. SSE you can select

**Inner Loop**

JMC: All high level JS can be stripped down in the JIT

**Shuffling**

(copy from slide)

JMC: Compiles down to a single instruction

WH: There are 256 of those constants defined?

JMC: Yes.

**Branching**

(copy from slide)

WH: What data type used to represent that?

JMC: int32x4

WH: Any kind of 4-bit data type for the mask?

Q about displayed data on slide

WH: Is `select` bitwise?

JMC: Yes

WH: Cool. It's data type agnostic and lets you slice into smaller bit slices.

WH: Also, those of us who do NaN-coding will need to beware, because this can both view and manufacture arbitrary NaN's.

**How does VM optimize for SIMD**
```

(copy from slide)

**Firefox implementation Status

(see slide)

Chrome/v8 implementation status

(see slide)

YK: is Chrome interested in these patches?

JMC/PJ: They want confirmation from TC39

DL: This is v8, not chrome. v8 team is fairly conservative.

Emscripten Implementation Status

(see slide)

JMC: Much of these operations are used in real world platforms written in C++

V8 SSE Benchmarks (Early 2014)

(see slide)

MM: How can you get faster than 4x faster with 4-way SIMD?

DH: Float 32

SpiderMonkey SSE Benchmarks (Early 2014)

(see slide)

Dart VM NEON Benchmarks (Early 2014)

(see slide)

MM: Why are the relative speed ups across the vms are so different?

JMC: Different output from different code

Why Fixed Width and not Variable Width Vectors

(see slides, 1 & 2)

STH: A problem bigger than variable width vectors. If we wanted 256 bit widths, on 128 bit vector platforms, observable differences.

JMC:

WH: Why is intel building hardware with 128 bit vectors

-- Dmitry Lomov (DL) will fill in details of discussion here.

JMC: this will expose differences hardware

JMC: no implicit conversions, 1 + <float32x4> will do string concatenation

MM: why?

JMC: too much magic

DH & JMC: overloading operators is ok, no lifitng or implict conversions

WH: It's bad that you can do -<float32x4> but not 2*<float32x4> and instead have to splat the 2 into its own vector first.

JMC: like asm.js, have to be clear about what types you're operating on.

YK: Don't have to make the ergonomics good

JMC: Don't have to, they never will be.

Planned Features 1

- SIMD and value objects/types

  - float32x4 and friend will be value objects

  - overloaded operators (+, -, ..._ will be mapped to SIMD.<type>.<op> equivalents

- Additional data types (int8x16 and int16x8)

  - Looking at VP9 encode/decde for justification

AWB: The top bullet has a lot deps, but the bottom not, are these near term?

JMC: Yes

WH: Why not int64x2?

JMC: Support is not universal

MM:

- Universal across processors

- something that has compelling algorithm

Unsigned integers don't fall in the second?

WH: Why not unsigned integers?

JMC: Not widely used

WH: uint32 perhaps, but uint16 and uint8 are heavily used in graphics to represent pixel values.

JMC: tried, issues encountered

- Extracting kernels and analysing the algorithms they're using and finding the instruction set overlap

- start with smaller scope, can expand later on. can add x8, x16 later. Surveyed internal teams

- 128 SIMD and expand from there.

MM: What's being saved, given the already exposed information?

JMC: time, complexity, etc.

AWB: How would you specify "slow", "fast", etc.

DH: Leave it undefined. "High recommended if supported, etc"

AWB: worried about gaming.

DH: same

Planned Features 2

(see slide)

DH: Risk:

- Some content is written such: if optimized, do this, if not, throw an error

- Browser doesn't want to be left out, will fake the optimized flag.

YK: The only reason to do the check is if you know you have a faster scalar implementation for systems without the SIMD feature; path of least resistance is to use polyfill and do no check at all. So maybe risk is not so great.

WH: Flip side also could be an issue: Web site has code for the optimized case which is not present on common platforms, somebody changes it and doesn't test it properly, it later breaks on optimized implementations, so browsers don't want to set the optimized flag.

JMC: (confirmed awareness of gaming)

BE: Some won't fake for fear of the performance cliff. See WebGL precedents.

Discussion re: risk, generally: some risks worth taking.

WH: instead of boolean, maybe a value that indicates speed level?

AWB: Application could do a mini benchmark as a test?

Stage 1 Ready?

(see slide)

AWB: Sounds like it is ready for stage 1. Can it be its own independent standard?

WH: It creates new primitive data types. Don't want specs outside creating new types

AWB: Do you expect every runtime to implement this?

JMC: Yes. They will run to implement this!

BE: Some embedded systems have trouble ith regex and unicode, it's expect that there will be "code commerce" among distinct device classes' embedded runtimes.

MM: We need a general framework for new value types

AWB: Without the value types, it's fairly clear cut.

MM: Preserving reference identity makes it prohibitively expensive

DD: Per the ES7 model, the feature can progress without being in another spec.

Discussion of the spec process.

STH: Back to MM statement, what does typeof have to do with reference identity?

- Could be implemented by memoizing the identity, not that you'd implement that way

MM: (example of using a weakmap)

- Logically, if they're a reference type, we have to admit them to WeakMaps, if they are a value type we can reject them. I hadn't considered the memozation

AWB/DH: (clarification of coupling and timing issue)

DH: Needs to be the same semantics as value types, if we ship this sooner and learn that we made a wrong call, then we have to deal with deciding whether or not we apply the mistake or break with SIMD.

Conclusion/Resolution

- Moves to stage 1

4.3 Function parameter/let declaration name conflict rules

(Allen Wirfs-Brock)

parameter-scoping-7-14.pdf

Current spec, Controversial:

```js
function(x) {
  var x;
  let x; // early error
}
function(x) {
  let x; // early error    <--------
}
try {
} catch(x) {
  let x; // early error    <--------
}
```

AWB: Andreas wants consistent handling

DH: The mental model is that let is block-bound,

DH: `var` is "I assert there is a binding in this scope, but that can be re-asserted as much as I want". `let` is "I have one unique declaration, I don't allow redeclaration".

YK: If you say `var x = 42` half way down the function, you can use the original parameter `x` until that point. With TDZ, if you had `let x = 42` half way down, you couldn't mean anything with `x`

DD: (points about let and const protecting from mistakes)

BE: (channeling Andreas) Worried that there will errors when you want to shadow.

DH/YK: The shadowing is meaningless.

MM: I was indifferent, but side with Dave's points about refactoring

STH: Generating code via macros, introduces non-local restrictions that could break

DH: Just have a notion of parameter bindings and block bindings, distinct from the surface syntax, and latter can't shadow former; easy workaround for code generators is to add an extra pair of `{ }`.

MM: (example of user blindly changing `var` to `let`)

STH: This isn't a language issue, it's a non-semantics preserving change.

DL: (on behalf of Andreas)

For Do-Expressions:

```js

() => {} = () => do {}

```

DH: Doesn't hold b/c left is statement body, right is expression body, not equivalent.

AWB: (revisting decisions about duplicate declarations in same contour)

AWB: Need to ensure that lexical declarations are disjoint sets, there spec mechanics there.

STH: Proposing

RW: The refactoring hazard only exists for the one time the code is run after the change from `var` to `let` and the refactorer is shown the early error and immediately knows to fix the bug. Removing these errors is unfortunate

YK: It's not clear what the program does when there is no early error.

RW: What is Sam's position?

STH: Why do we have these errors? What do we gain from them?

RW: Arguably, JavaScript could use some degree of "nannying" if it has positive results.

MM: No way to explain that function declaration initializes the parameter?

BE: It doesn't. Andreas just wants `let` to behave like `var` re: redeclaration

MM: Strict programming should be understandable in terms of lexical scope.

1. Parameters and body are two scopes

  - If explain as two scopes, can't unify.

2. One scope

  - Has to be an early error.

BE: Good argument, but not sure it depends on strict.

MM: In sloppy mode, functions are crap as well.

STH: He's just trying explain the semantics of `let`, w/r to block scope alone.

MM: A `var-less` strict program should be understandable in terms of lexical scope.

BE: `var` is huge turd that should be recalled into some lesser demon's bowels.

- We want the error.

Conclusion/Resolution

- Status Quo

- DDWIDM: "Don't Do What I Didn't Mean"

4.7 Revisit Comprehension decision from last meeting.

(Allen Wirfs-Brock)

AWB: There are a lot of TC members and non-members concerned that this was not a wise decision and that we should revisit. Included link to Andy Wingo

RW: if I had been here at the last meeting I would've objected to the removal, but as I told Dave offline, I trust him and his plans for syntax unification. I just wanted to see progress in that regard.

BE: I want to say: I was the champion for years, but letting go. I want to see the comprehensions laziness addressed.

DH: I did this exercise, the sudoku solver in:

- pythonic

- linq style

- no comprehensions

https://github.com/dherman/sudoku

JH: I'd like to know if there are objections still, to deferral

AWB: Objecting to the late removal of a complete feature.

RW: Same objection, reached independantly, but again I trust Dave to see through the syntax unification.

DH: First, laziness is not a problem; you just need a way to construct a lazy sequence either from an eager value (`array.lazy()`) or from whole cloth (`lazyRange(0, 1000)`).

DH: Second, the fact that comprehensions only do a subset of operations you want means you end up mixing code styles (comprehensions + methods), and it gets syntactically awkward.

DH: When I did the exercise with three styles, I found the generalized comprehensions nicer but no comprehensions at all nicest.

BE: The affordance of generator expressions and comprehensions is that you don't have to write a call

DH: (Gives walk through of solver.linq.js, solver.pythonic.js)

- The exercise shows a need for new methods of iterators, flatMap, filter, etc.

DH: I said last time that we need an Iterator.prototype object and we agreed to defer since it probably wouldn't break code, but we forgot that hurts polyfills that want to patch the prototype with upcoming standard methods. So we should add the empty prototype object in ES6.

WH: In expressions such as foo.lazy().map(...function1...).every(...function2...), what shuts down (i.e. calls the return method of) the foo.lazy() generator?

DH: The call to every will shut down the generator if it reaches its decision early.

DD: The minimal Iterator.prototype is empty, but available. The long term is a constructor Iterator with blessed apis.

DH: Confirm

BE: An actual Iterator is means Duck typing isn't the preferred way, just create a real Iterator

MM: Using an actual functional style, the function names you're using are

BE: The oop style prevails, desired chaining API, adapter full of goodies.

Discussion of generators and observables

WH: How would you represent a 2-d comprehension like (for (x of xs) for (y of ys) if (x % y) x+y)?

xs.flatMap(x => ys.filter(y => x % y).map(y => x+y))

WH: OK. A bit less pretty than the comprehension in this case, but acceptable.

MM: after seeing this code I will never use comprehensions

YK: *raises arms in triumphant vindication*

BE: who will own explaining to Andy Wingo and es-discuss?

DH: I will

BE: "You keep what you kill" - Richard P. Riddick

Conclusion/Resolution

- Add a prototype for iterators, but do not expose a global Iterator constructor for ES6 (leave that for ES7)

  - Between Object prototype and Generator prototype

  - Initially empty, but accessible

- Comprehensions in general deferred to ES7

4.12 Revisit spread and destructuring of string

(Erik Arvidsson , Brendan Eich)

EA: We're using ToObject in spread and all other iterable forms. Should we do the same for destructuring?

- This would allow destructuring strings and other non-objects.

```js
// Should allow:
let [first, ...rest] = "foo";
first; // "f"
rest;  // ["o", "o"]
```

STH: ToObject breaks pattern matching because you couldn't match on a number.

YK: But we agreed to a future irrefutible matching, which would be the basis of pattern matching.

DH: Array vs. Object cannot have the same semantics here in what we want from pattern matching

- if I used an array

EA: Uses iterator

DH: Not even self-evident that pattern matching syntax will work in JS

YK: (to Sam) Do you think it will it should fail for strings to destructure?

More discussion of pattern matching.

DH, BE: match must mean a different pattern language, ships have sailed for destructuring and implicit ToObject

Conclusion/Resolution

- Destructuring does ToObject

4.5 Import-into-namespace syntax (Dave)

(Dave Herman)

request slides

DH: (recapping the last meeting and the findings of the breakout group; and the fall out)

DH:

Resolution

- Changed syntax for clarity

(need slides)

Module Context 1

- Existing systems provide contextual metadata:
  - module.id
  - __filename
  - __dirname
- What is the dynamic analog of relative import?

```js
import helper from "./helper";
```

Module Context 2

- no implicit namespace pollution, plz
- JS has a dedicated contextual variable: `this`
- Solution: inital `this` binding is a context object

DD: How is this different from adding global variables, eg. `Reflect`

STH: The difference is that the value depends on where it is; unlike `Reflect`, which is the same thing.

DH: We should use `this` in top level of a module

AWB: what does that mean? `this` at the top level of a module?

DH:

Module Context 2

- Relative import:

```js
this.import("./helper").then(...);
```

- Space for host-specific contextual metadata:

```js
this.filename
```

(This is where platforms can put its host properties and objects)

- Cross-talk about `eval`

- `Reflect.global`

BT: indirect eval?

DH: Will give you the global object

DD: object to relying on `this` outside of a method

RW: Workers already to the above

MM: We can't even poison `this` for ES6

YK: if you says it's module context, you have to say how it got that value

DH: No new scoping rules. This construct just implicitly binds something.

AWB:

```js
import filename from this;
// which basically: import filename from here;
```

DD: Like this for relative

DH: Completely amenable to this

YK:

```js

import * as me from here;

me.import; // `me` is the context object

```

Conclusion/Resolution

- the api is right direction

- each module gets its own version of that object

- need some sort of access to the module contextual object

- some sort of declarative form to get at

- static contextual information about the module

"Then, during the Third Reconciliation of the Last of the Meketrex Supplicants, they chose a new form for him, that of a giant Sloar! Many Shubs and Zulls knew what it was to be roasted in the depths of a Sloar that day, I can tell you!"

—Vinz Clortho[src]

 # July 31 2014 Meeting Notes

Brian Terlson (BT), Dmitry Lomov (DL), Waldemar Horwat (WH), Allen Wirfs-Brock (AWB), John Neumann (JN), Rick Waldron (RW), Eric Ferraiuolo (EF), Jafar Husain (JH), Jeff Morrison (JM), Mark Honenberg (MH), Caridy Patino (CP), Sebastian Markbage (SM), Istvan Sebestyen (IS), Erik Arvidsson (EA), Brendan Eich (BE), Mark Miller (MM), Sam Tobin-Hochstadt (STH), Domenic Denicola (DD), Peter Jensen (PJ), John McCutchan (JMC), Paul Leathers (PL), Eric Toth (ET), Abhijith Chatra (AC), Jaswanth Sreeram (JS), Yehuda Katz (YK), Dave Herman (DH), Brendan Eich (BE), Ben Newman (BN)

Notes from secretariat

IS: ES6 delay accepted, but please don't delay this again.

- TC52 working in a similar way and process to TC39's ES7 approach: Frequent releases of incremental versions of standards. They also use the same kind of RF policy.

- TC52 is looking at how TC39 is proceeding

- TC52 are more polite

IETF and Internet Architecture Board liaison.

- JSON work and looking for liason. We published Ecma-404 and they are publishing their standard and have asked for review/comment. Need to nominate someone as liaison.

ITU liaison.

- Using in JSON for communication standard

Meteor group has joined Ecma

JN: Recommend putting out a call for liaisons, including the information you have. List roles and expectations, we'll put it on the next meeting agenda to establish appointment.

AWB: There has been a notification for these roles.

JN: Is anyone here prepared to volunteer now? Or at the next meeting. Need someone to at least collect the communications out of those organizations.

Conclusion/Resolution

- John Neumann to stand in as liaison

9.1-8 Date and place of the next meeting(s)

JN: Need to fill in the venues.

DH: January 27-29, 2015 at Mozilla (Downtown SF, CA)

EF: March 24-26 2015 at Yahoo (Sunnyvale, CA)

JM: May 27-29 2015 at Facebook (Menlo Park, CA)

ET: November 17-19 at PayPal (San Jose, CA)

RW/YK: Will decide on Sept. 2015

Conclusion/Resolution

- John Neumann will update the agenda and schedule.

4.4 Follow up: Instantiation Reform (@@create)

JM: Found cases where we set up state before calling super. I'm convinced that there are sufficient workarounds (via calling <<SuperClass>>.call() in the legacy style).

https://gist.github.com/jeffmo/bf30e7154ab3c894b740 -- "#_Before.js" is an example of a pattern that exists now, "#_After.js" is an example of how one might fix this pattern

JM: (gives examples that amount to two step initialization)

WH: C++ doesn't allow this kind of bottom-up construction (can't initialize a subclass instance before the superclass instance is initialized), and use cases like this arise once in a while. The usual workaround is to pass through Options objects.

JM: (example of type ahead classes in FB code base -- see 2_Before.js and 3_Before.js in the above gist)

AWB: (draws model of two phase allocation)

YK: The problem is allocatuion vs. initialization, in your model mutates before calling super.

- Need to make sure the allocation has happened before you get into the constructor.

- It looks like the only way to fix this is to have "two constructors", which we can't do

SB: We don't want to support this pattern, but there is nothing to stop user code from doing this.

AWB: "Fragile Base Class Problem"

- Start at derived class

- super'ed up to base class

- Base class invokes method that's defined on the subclass

- The problem is that the object isn't set up yet.

This is a bug.

AWB/YK: (Further discussion of how to avoid this pattern)

JM: Special case refactoring in subclasses isn't trivial.

- Both direction have down side:

  - TDZ approach negates certain cases

  - Non-TDZ approach allows for decoupling of allocation/instantiation

YK: Lifting the TDZ doesn't solve the problem. It happens to work in this case because the base class doesn't allocate.

AWB: There is way to do this in the new design, use the construct method

SM: Foresee a tooling solution (e.g. linting for properly placed calls to super())

AWB: Will always come to a place where a problem can't be solved with your existing inheritance model and you'll simply need to refactor. It's not that inheritance has failed, just that the class heirarchy needs to be refactored.

JM/SM: Refactoring is the correct approach, but it can be idealistic in some scenarios. Imagine a TypeaheadBase class that has been subclassed 100s of times. It's not until the 101th time that you realize you need to refactor the base class (and, thus, all pre-existing subclasses)

Discussion about subclasses that require two phase construction (with instance side initialization methods)

Mixed discussion about allocation phases.

MM: Do we have consensus on the instantiation reform we agreed to yesterday?

yes.

[JM agrees on the grounds that there are at least legacy style workarounds for that 101th subclass and the rest of the patterns he found a la `<<SuperClass>>.call()`]

YK: will not be ok with this solution if it switches on the instantiation of the constructor

...Need help covering this...

WH: I insisted on having a syntactic switch for function-vs-generator rather than switching on the presence of a yield. The reason was that functions and generators are radically different and it makes sense for a trivial generator to have no yields. In this case I'd mildly prefer to have a syntactic switch as well, but it's not as crucial because I haven't seen any good examples of where apparently problem-free code would unexpectedly go wrong. If you don't call the superclass constructor, you'll miss out on superclass initialization, which would be a problem even if the presence of a super call didn't statically switch modes, so the mode switch hasn't created a problem where there wasn't one. Inserting or deleting an "if (false) super()" does change things, but I don't see why one would be likely to do that. [I suppose that you could stylistically mark inheriting constructors whose super() calls are deeply buried with an "if (false) super()" at the top :).]

MM: I agree there is a smell with super; would I adopt syntactic marking? I want to see them first.

WH: On the fence, suppose we do this, what if you call super in a method not marked? What does that do?

DH: It's ok for class body to have more syntactic distinctions than outside functions acting as constructor

AWB: You could imagine that I have a splat that indicates "I do not want allocation for this class"

Conclusion/Resolution

- Agreement to MM proposal: Allen to be the champion and work out the details remaining

ES7 Items?

DH: We should focus on ES6 items, we have limited time in face to face.

DD: I don't think we should de-prioritize ES7, given the train model.

AWB: We have no choice but to prioritize ES6

5.7 Array.prototype.contains

(Domenic Denicola)

Follow up from RW's Stage 0 in

DD: Presents: https://github.com/domenic/Array.prototype.contains/

DH/RW: Parity with String

RW: Most arguments were noise and Domenic's proposal addresses them all.

AWB: One of the previous objections: string.contains is looking for a substring, not an element.

EA: Same thing with indexOf

JM: Can you give an example where this is a problem?

MM: Consistently inconsistent parity with indexOf

Conclusion/Resolution

- Advance to Stage 1

DD: Would like to do Stage 2 and 3 asynchronously due to the simplicitly of this proposal.

AWB: I'd like the process to be respected.

MM: If Stage 2, 3, and 4 are complete by next meeting, then we can advance to Stage 4.

Discussion re: the ES7 process.

AWB: Concerns about lack of review that results in more work later in the game.

MM: We don't have a mechanism to come to consensus outside of these meetings. These meetings _are_ the time that we're able to work on these issues. This is my allocated time

5.1 Math.TAU

(Brendan Eich, Rick Waldron)

https://gist.github.com/rwaldron/233fd8f5aa440c94e6e9

BE: Math.TAU = 2PI

WH: OK, but only if it's called Math.τ :-)

MM: Opposed: one letter shorter, not well known, not well taught. PI is known, taught and ubiquitous.

Conclusion/Resolution

- Rejected.

Exponentiations operator

(Rick Waldron)

https://gist.github.com/rwaldron/ebe0f4d2d267370be882

RW: All other languages have it. Why can't we?

RW: Needs hihger precedence than multiplication

MM: Right associative?

RW: Yes, same as all other languages.

BE: **?

MM: Want to make sure that it does the same as the built in %MathPow% and not not any overload.

RW: Confirm

DH: Wants to point out that adding syntax does have a cost. But thinks it is fine and there is a lot of precedent.

Conclusion/Resolution

- Approved for Stage 0, in 6 minutes.

Precision of Math trig functions

(Dave Herman)

DH: V8 made the result less exact in the name of performance. Are we going to have a "race to the bottom"?

DH: People are now starting to implement these algorithms in js since they cannot depend on the built ins.

DL: Java lies. Implementations do not follow the Java rules.

WH: Looked at the posted data and found it pretty compelling that the status quo free-for-all is not good. Some functions that should be monotonous aren't. Some results are significantly off.

WH: fdlibm should be the lower bar for precision. It almost always gets things exact or within 1 ulp.

DH: Talked to an expert (Dan Gohman) and he offered to come to this meeting; I told him he could wait and see how the conversation goes and maybe come in the future.

WH, DL: We need to invite experts to get the right answer to this.

MM: When doing the sputnik tests we just looked at the results and based the precision on what browsers did when the tests were written.

AWB: We need a champion.

DL: V8 is planning to fix this and make the results more accurate.

Conclusion/Resolution

- Need to bring in experts.