Ecma/TC39/2013/071
Ecma/GA/2013/153

Minutes for the: 37th meeting of Ecma TC39

in: San Jose, CA, USA

on: 19-21 November 2013

1 Opening, welcome and roll call

1.1 Opening of the meeting (Mr. Neumann)

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

Companies / organizations in attendance:

Mozilla, Google, Microsoft, Intel, eBay, jQuery, Yahoo, Free University of Brussels, Facebook, Indiana University (guest)

1.2 Introduction of attendees

John Neumann – Ecma International

Istvan Sebestyen – (part-time, phone) Ecma International

Mark Miller – Google

Douglas Crockford – eBay

Tom van Cutsem (part-time, phone) – Free University of Brussels

Allen Wirfs-Brock – Mozilla

Erik Arvidsson – Google

Waldemar Horwat – Google

Andreas Rossberg – (part-time, phone) Google

Sam Tobin-Hochstadt – (guest) Indiana University

Alex Russell – Google

Eric Ferraiuolo – Yahoo!

Reid Burke – Yahoo

Matt Sweeney – Yahoo!

Rick Waldron – jQuery

Yehuda Katz - jQuery

Dave Herman – Mozilla

Brendan Eich – Mozilla

Rafael Weinstein – Google

Jeff Morrison – Facebook

Sebastian Markbage – Facebook

Dmitry Soshinikov – Facebook

Brian Terlson – Microsoft

Ben Newman – Facebook

Rick Hudson - Intel,

19 November 2013

John Neumann (JN), Allen Wirfs-Brock (AWB), Yehuda Katz (YK), Eric Ferraiuolo (EF), Erik Arvidsson (EA), Rick Hudson (RH), Matt Sweeney (MS), Rafael Weinstein (RWS), Alex Russell (AR), Rick Waldron (RW), Dmitry Soshnikov (DS), Jeff Morrison (JM), Sebastian Markbage (SM), Ben Newman (BN), Reid Burke (RB), Waldemar Horwat (WH), Doug Crockford (DC), Tom Van Cutsem (TVC), Mark Miller (MM)

20 November 2013

John Neumann (JN), Allen Wirfs-Brock (AWB), Yehuda Katz (YK), Eric Ferraiuolo (EF), Erik Arvidsson (EA), Rick Hudson (RH), Matt Sweeney (MS), Rick Waldron (RW), Dmitry Soshnikov (DS), Sebastian Markbage (SM), Ben Newman (BN), Reid Burke (RB), Waldemar Horwat (WH), Doug Crockford (DC), Tom Van Cutsem (TVC), Mark Miller (MM), Brian Terlson (BT), Andreas Rossberg (ARB), Alex Russell (AR)

21 November 2013

John Neumann (JN), Allen Wirfs-Brock (AWB), Yehuda Katz (YK), Eric Ferraiuolo (EF), Erik Arvidsson (EA), Rick Hudson (RH), Matt Sweeney (MS), Rick Waldron (RW), Dmitry Soshnikov (DS), Sebastian Markbage (SM), Ben Newman (BN), Jeff Morrison (JM), Reid Burke (RB), Waldemar Horwat (WH), Doug Crockford (DC), Tom Van Cutsem (TVC), Mark Miller (MM), Brian Terlson (BT), Andreas Rossberg (ARB), Alex Russell (AR), Mark Miller (MM)

1.3 Host facilities, local logistics

On behalf of PayPay (eBay) **Doug Crockford** welcomed the delegates and explained the logistics.

1.4 List of Ecma documents considered

Ecma/TC39/2013/050	Results of GA postal vote (The JSON Data Interchange Format)
Ecma/TC39/2013/051	TC39 RF TG form signed by Microsoft
Ecma/TC39/2013/052	Draft Standard ECMA-262 6th edition, Rev. 19, September 27
Ecma/TC39/2013/053	TC39 RF TG form signed by Vrije Universiteit Brussel
Ecma/TC39/2013/054	TC39 RF TG form signed by Facebook
Ecma/TC39/2013/055	Minutes of the 36th meeting of TC39, Boston, September 2013
Ecma/TC39/2013/056	TC39 RF TG form signed by Adobe
Ecma/TC39/2013/057	TC39 RF TG form signed by Mozilla
Ecma/TC39/2013/058	Venue for the 37th meeting of TC39, San Jose, November 2013
Ecma/TC39/2013/059	Software submitter contribution form completed by Adobe
Ecma/TC39/2013/060	TC39 RF TG form signed by Google
Ecma/TC39/2013/061	TC39 chairman's report to the CC, October 2013
Ecma/TC39/2013/062	TC39 process proposal by Mr. Weinstein (Google) and Mr. Wirfs-Brock (Mozilla)
Ecma/TC39/2013/063	Draft Standard ECMA-262 6th edition, Rev. 20, October 28
Ecma/TC39/2013/064	Draft minutes of the European Multi-Stakeholder Platform on ICT Standardization, October 2013
Ecma/TC39/2013/065	Draft Standard ECMA-262 6th edition, Rev. 21, November 8
Ecma/TC39/2013/066	Agenda for the 37th meeting of TC39, San Jose, November 2013 (Rev. 1)

Ecma/TC39/2013/067	Report to TC39 from the CC meeting held in Geneva in October 2013
Ecma/TC39/2013/068	TC39 RF TG form signed by Intel
Ecma/TC39/2013/069	ECMA-402: Draft Evaluation report

2 Adoption of the agenda (2013/066-Rev1)

Agenda for the: 37th meeting of Ecma TC39 (final version on Github)

```
in: San Jose, California, USA
on: 19 - 21 Nov. 2013
TIME: 10:00 till 17:00 PST on 19th and 20th of Sept. 2013
      10:00 till 16:00 PST on 21th of Sept. 2013
LOCATION:
    PayPal's Campus
    Building 17
    2211 North First Street
    San Jose, CA 95131,

Mr. D. Crockford's email: douglas@crockford.com
```

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 (2013/066)
3. Approval of the minutes from Sept. 2013 (2013/055)
4. ECMA-262 6th Edition proposal discussion
   i. Review Latest Draft (Allen)
   ii. Clarification of the interaction of unicode escapes and identification syntax (Waldemar)
   iii. Should 'arguments' in an arrow body be from nearest outer defining scope, or early error? (Brendan)
   iv. Finalizing the Proxy API for ES6 (tomvc) (preferably Tuesday morning)
      a. remove the `hasOwn()` trap. es-discuss thread
      b. remove the `invoke()` trap. es-discuss thread
      c. postpone standard Handler hierarchy to ES7. es-discuss thread
      d. should `Proxy` be a constructor or not? In draft spec `Proxy(t,h)` works but `new Proxy(t,h)` does not. es-discuss thread
   v. Modules (dave) (preferably not tuesday morning, as samth won't be there)

vi. the global scope contour (dave)

vii. Math.hypot() and precision (dave)

viii. reconsidering the Map custom comparator API (dave)

ix. reconsidering Object.is (dave)

x. Generator arrow function syntax (Brendan)

xi. Data on prevalence of var decls masking function decls (which is now an error) (Brian)

xii. Performance impact of Tail Calls (Brian)

5. Post ES6 Spec Process. (Rafael)

6. Internationalization Status

7. Test 262 Status

8. Report from the Ecma Secretariat (CC Report)

i. Status of the TC39 RFTG

ii. Status of external software contributions to TC39 work

iii. Status of the text copyright discussions

9. ECMA 404 W3C TAG / TPAC report (Alex)

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

i. January 28 - 30, 2014 (San Jose - Yahoo)

ii. March 25 - 27, 2014 (San Francisco - Google)

iii. May 20 - 22, 2014 (Paris France, Portland, OR, San Francisco - Mozilla)

iv. July 29 - 31, 2014 (Redmond, WA - Microsoft)

v. September 23 - 25, 2014 (Boston, MA - BoCoup)

vi. November 18 - 22, 2014 (San Jose - PayPal)

11. Closure

The agenda was approved.

For details of the technical discussions see Annex 1.

3 Status Reports

3.1 Report from Geneva (Ecma/TC39/2013/067 Report to TC39 from the CC meeting held in Geneva in October 2013)

3.1.1 Brief report about move to a TC39 RF TG

Mr. Sebestyen said that the IPR ad hoc group is still working out a solution for a software contribution from non members but the IPR ad hoc group has not finished their work. So the external contribution is not yet for software. The CC has agreed that the current text for Ecma members' software contribution might also work for 3rd parties, but it has to be checked if the patent policy can be equally applied to members and non-members. The IPR group has been contacted on that issue.

Regarding Registrations to the TC39 RF TG the Ecma Secretariat has now received a a significant number of registrations. Major TC39 players have submitted their registration forms. According to the advice of the CC, who has discussed the matter in their October 13 meeting advised, that though the current modus of operation is still RAND but at the January 2014 TC39 meeting TC39 members who are participating in the meeting should vote if they want to start with the RF TG. If no vote the RAND regime should remain. He urged again that the not yet registered TC39 members if interested should officially register to the TC39 RF TG.

3.1.2 Ecma recognition program

The CC suggests that Ecma Recognition award should be awarded to anyone who has done exceptional service and outstanding contribution to Ecma. TCs may come up with nominations for GA approval.

3.1.3 Ecma Text copyright suitability for TC39

The CC has discussed the matter. It is on the opinion that the current use cases do not require changes on the copyright license. They suggested that a “Frequently Asked Question” should be prepared on what is allowed and what not with the copyright license. They also said that the IPR group should be contacted on this matter. More details in TC39/2013/067

3.2 Json

After successful TC voting the JSON ballot has been completed on October 6, 2013. The JSON standard has the number ECMA-404, and is already popular in terms of number of downloads.

It was brought to the attention of TC39 that the current IETF work on JSON may lead to incompatibilities to ECMA-404. TC39 decided to write a liaison letter to IETF on the subject. The liaison letter was approved by TC39 and sent to IETF right after the TC39 meeting.

4 Date and place of the next meeting(s)

Schedule 2014 meetings

- January 28 - 30, 2014 (San Jose - Yahoo)
- March 25 - 27, 2014 (San Francisco - Google)
- May 20 - 22, 2014 (Paris France, Portland, OR, San Francisco - Mozilla)
- July 29 - 31, 2014 (Redmond, WA - Microsoft)
- September 23 - 25, 2014 (Boston, MA - BoCoup)
- November 18 - 22, 2014 (San Jose - PayPal)

5 Closure

Mr. Neumann thanked **PayPal / eBay** for hosting the meeting in Boston, the TC39 participants their hard work, and **Ecma International** for holding the social event dinner Wednesday evening. Special thanks goes to **Rick Waldron** for taking the technical notes of the meeting.

Annex 1

Technical Notes (by Rick Waldron):

From:	Rick Waldron [waldron.rick@gmail.com]
Sent:	Wednesday, November 27, 2013 7:14 PM
To:	Istvan Sebestyen; Patrick Charollais; John Neumann
Subject:	Complete Meeting Notes for 37th meeting of TC39

# Nov 19 Meeting Notes

John Neumann (JN), Allen Wirfs-Brock (AWB), Yehuda Katz (YK), Eric Ferraiuolo (EF), Erik Arvidsson (EA), Rick Hudson (RH), Matt Sweeney (MS), Rafael Weinstein (RWS), Alex Russell (AR), Rick Waldron (RW), Dmitry Soshnikov (DS), Jeff Morrison (JM), Sebastian Markbage (SM), Ben Newman (BN), Reid Burke (RB), Waldemar Horwat (WH), Doug Crockford (DC), Tom Van Custem (TVC), Mark Miller (MM)

## Welcome

JN: (Introductions)

DC: (Logistics)

JN: ...updating status of RFTG
...Adopt agenda?

Unanimous approval

JN: Approval of Sept Minutes?

Unanimous approval

## ES6 Status

AWB: (presenting Luke's spreadsheet with remaining features that need attention) https://skydrive.live.com/view.aspx?resid=704A682DC00D8AAD!59602&app=Excel&authkey=!AAMixsO0TuyPYwc

...Function.prototype.toString still needs attention

BE: Mark is to write spec, in this case under-specify

AWB:
- Refutable pattern matching is deferred
- still need specification for enumerate / getOwnPropertyKeys/Symbols in various places
- Proxy handlers, cut
- C-style for-let

BE: We agreed on semantics

AWB: We agreed on outer capture, but won't try to update per iteration. Need per iteration binding.

YK: This is the consensus I recall

AWB:
- Modules, static semantics complete
- No loader/runtime semantics yet.
- Dave will have a complete spec this meeting

YK: (confirms that it's complete)

AWB: Yes, but not yet in the spec.
- Standard Modules deferred.

(TVC dials in)

## 4.4 Finalizing the Proxy API for ES6
(Presented by Tom Van Cutsem)

TVC: First three relate to es-discuss threads, re: simplifying Proxy. Jason Orendorff expressed concerns.
- hasOwn()

- invoke()

WH:

AWB: Looked for traps for call

BE: Total traps?

AWB: Now 14 traps.

BE: Cool. Not including hasOwn()?

AWB: Not including

TVC: The next is `invoke()` trap. Leave out for now to avoid inconsistencies with `get()`?

YK: How do you implement virtual objects? ie. an object whose `this` object is always the proxy. Can't do it reliably without invoke.

WH: Still not reliable, even with invoke.

YK: So what are the cases?

AWB: Can still implement a virtual object or full membrane, or thin wrapper.

YK: Not the use case. Want to make an object where `this` is the proxy and not the target.

TVC: Are you arguing that `this` should remain bound to the proxy object upon forwarding? If yes, this is the default.

YK: As long as maintaining equivalence between `foo.bar()` and `bar.call(foo)`

AWB: yes.

TVC: Regarding Handler API: not enough motivating use cases for proxy handlers without community use. Propose to defer.

AWB: Let's publish the library code

TVC: It's already on github, as part of my shim. I will publish it as a separate project to make it more accessible.

TVC: `Proxy` as a constructor? Currently, no `new` throws TypeError

AWB: Not really a "class"

AR: You create new ones?

AWB: No prototype

AR: Gives an instance, why not "new"

AWB: Would need an @@create

YK: Then shouldn't be capitalized

BE: We can do "proxy"

AWB: (to Tom) the concern is: if it's not "new-able", should it be little-p proxy?

TVC: Given choice between little-p proxy and requiring `new`, I prefer `new`

RW: Agree with Alex, `new Proxy()` creates new Proxies, so allow `new`

AR: Let's not duck the charge on @@create.

WH: Proxy is not a class.

YK: ...But has allocation

WH: I would hate to specify what happens when subclassing from Proxy

AWB: Must create a special constructor

TVC: @@create doesn't make sense here

BE: Is Proxy a class?

(General: no)

TVC:

DS: What is typeof and instanceof

AWB/BE: object

BE: Capital P

AWB: Ca???

DS: Whatever Proxy creates?

BE: That depends on what is created.

DS: By default?

BE: typeof depends if there is a call trap. instanceof depends on the prototype chain. All in the spec, so can create any object (apart from private state issues)

AWB: Ok, into the future... would value objects allow `new`?

BE: (int64 example)

...back to why should `new` throw on Proxy constructor.

BE: Seems counter intuitive: Proxy constructs objects. `int64` creates a value. callables construct objects

[inaudible]

BE: these are object constructor functions, which is what people want to do with "new"

AWB: this is somewhere in the middle between newing a class and a random function that returns an object

BE: in either case, it returns a new object and proxies are factories for object

AWB: yeah, I agree...spec currently calls them "proxy factory functions"

BE: pretty weird not to have "new" on this.. feels natural

YK: Intuitively, the difference between returning an object and not a value

AWB: we can do it...need to make it an exotic object with a special [[Construct]]

AR: agree. Making it exotic is good.

TVC: what's the summary?

AWB: we allow new, we do it by making it exotic.

EA: Do we REQUIRE "new"?

WH: what does Map do without `new`?

EA: Throws

BE: Why are we requiring `new` again?

RW: Needed for subclassing

AWB: my objection is that we're trying to tell a story about objects, classes, and @@create... this confuses the story

YK: there's already a weird split...saying you have to use `new` avoids the confusion

AR/RW: agree

AWB: we could go either way, and it's subtle, but most people won't see the subtlety

RW/BE: the conservative thing to do is to throw now and we can walk it back later

AWB: agree

JM: Removing the throw might change control flows?

BE: non-issue, rare.

#### Consensus/Resolution

- Drop `hasOwn()`
- Drop `invoke()`
- Postpone proxy handlers API
- REQUIRE `new Proxy` by making Proxy an exotic function that has it's own internal construct. (this is only "action" item)

## ES6 Status (cont)

AWB:
    - Symbols in place
    - Binary data deferred
    - RegExp look behind deferred

WH: No comprehensive specification of which variant of RegExp lookbehind was wanted. Followed up on es-discuss and got no good answer to questions.

AWB: No one has stepped up, deferred to ES7
- Completion reform has bugs, but that's just work for me
- Global scope, Dave has some possible changes he wants to discuss at this meeting
- Eval scope, now simplified because can't eval a module
- function scope, needs spec some design issues still open
- Promises, Domenic has a complete delta, just needs to be added
- Internal Method Invariants. If we're going to keep this section, someone needs to provide spec ready text.

MM: If Tom will collaborate with me on this, I will commit to producing this.

AWB: I can live without this section

MM: ES5 had that section, I will talk to Tom about creating this for ES6

...

AWB: For the end of this year, we need a feature complete spec. Essentially a "candidate spec". This means all the features there are sufficiently specified to allow an implementor to implement. I think we have a shot at it. There is work to do. Questions remain in modules, but Dave can update us. Dave and Jason (Orendorff) have been working like crazy to finish modules, including a reference implementation.

MM: The spec we want to be feature complete includes modules?

AWB: Yes.

MM: The loader stuff as well?

AWB: Yes.

... Over the next 6-9 months, we need implementors to work and provide feedback. We should push forward with what he have now (this includes modules)

## 4.1 Review Latest Specification Draft

(Allen Wirfs-Brock)

(request slides)

Discussion re: arguments.caller, arguments.callee in non-strict and strict mode.

Discussion re: default param aliasing

BE: (tries aliasing on SM)

YK: Issues with arguments.caller, arguments.callee in framework uses.

BE: Do we want three types of arguments?

YK: It's not really three types

...

BE: Fair to say there are three observably different kinds of arguments
-

YK:

MM: Adopt some semantics where aliasing

BE: Deep aliasing of destructured parameters is bad

AR: Walk back poisoning entirely?

MM: No.

MM: Any simple parameter is aliased and any new style parameter is not
... on defaults I could go either way.

AWB: Back to scheduling

BE: If there is destructuring, no deep aliasing.

WH: Easier to explain if the rules are compositional.

#### Consensus/Resolution

- Parameter issues
- default params, alias is independent
- destructuring has no name for top level object, no aliasing
- rest has no aliasing
- non-strict mode arguments have unpoisoned caller and callee

... Continue discussion re: do Arrow Functions have an arguments object?

## 4.3 Should 'arguments' in an arrow body be from nearest outer defining scope, or early error? (Brendan Eich)

AWB: Arrows don't have an arguments object, but `arguments` is a keyword inside arrow functions.

WH: `let arguments = ...`?

not allowed, recapping ES5 strict rules that were applied to new forms in ES6.

questions about conditionally reserved words

AWB: what if the outer function _does_ say `let arguments = ...;`, what is it in the arrow function?

MM: (whiteboard)

```js
function f() {
var arguments = 1;

[3, 2, 1].forEach(v => v + arguments);
}
```

YK: `this`, `super`, `arguments` should refer to its outer.

WH: 11.6.2.2 and 13.2.1.1 are inconsistent. The former omits keywords such as "arguments", while claiming to be based on the latter.

#### Consensus/Resolution
- `arguments` is lexically captured by arrow functions
- 11.6.2.2 and 13.2.1.1 are inconsistent (file a bug)
- yield cannot mean "yield to the outer"

AWB/BE: yield just follows the rules for yield

MM: (whiteboard)

```js
function foo() {
  var yield = 8;
  return function * bar() {
    yield baz();

    function baz() {
      return yield;
    }
  }
}

var f = foo();

console.log(f().next().value);

// 8
```

AWB: ...Computed property keys: No dynamic checking for duplicate computed property name in object literals or classes

```js
{
  [expr1]: 1,
  [expr2]: 2
}
// Does not check if expr1 === expr2
```

MM: Cannot statically repeat properties in object literals

AWB: This isn't a static object literal

MM: ...

YK: The most common usage will by for symbols, eg. @@iterator

AWB: We don't have a mechanism for runtime checks like this and shouldn't be adding such checks

MM: The create and overwrite paths are very different (define v. assign)

WH: Pick one or the other. If we're going to do static checking, do it consistently. Either always check or never check

AWB: The complications I ran into were in gets and sets, it's not just "does this property exist"...

BE: We do want computed property gets and sets
...We should do checks.

AWB: It's not that non-strict doesn't check...

MM: given that we do dynamic checks in strict mode, you prefer NOT doing them in sloppy mode?

AWB: I don't like the dynamic checks either way

MM: that wasn't the question

AWB: the precedent in ES5 for object literals is that strict mode has "more" static checks.

AWB: "from a language design perspective", we should have them the same in both cases

BE: I'm w/ WH and ARB, we want the check in strict mode

AWB: so you don't want the check in sloppy mode?

BE: I don't think there is only one consistent position, and I'm ok with no dynamic check in non-strict mode

... discussion of the current static checks in non-strict mode ...

#### Consensus/Resolution

- No change, this is the semantics:

```
var name = "x";
var o = {x:42, x:45}, // static strict error
    o2 = {x:42, get x()}, // static error
    o3 = {x:42, [name]:45 }, // dynamic strict error
    o4 = {x:42, get [name]() {}}; // dynamic error
```

AWB: the consensus is that for dynamically computed property names we will dynamically check the things we would have statically checked.

WH: Pop quiz. What does the following parse as?

```js
baz = ...
function foo() {
  var yield = 8;
  return function * bar() {
    yield
        baz
    yield
        * baz
    yield (baz) => yield * baz
  }
}
```

[Almost nobody guessed all of these right.]

Current answer, with all optional semicolons inserted:

```js
baz = ...
function foo() {
  var yield = 8;
  return function * bar() {
    yield;  // The yield expression rule has a [no LineTerminator here]
        baz;
    yield // The yield * expression rule doesn't. No optional semicolon here!
        * baz;
    yield (baz) => yield * baz; // The * here is a product of two variables
  }
}
```

#### Consensus/Resolution

- Update `yield * [Lexical goal InputElementRegexp]` => `yield [no LineTerminator here] * [Lexical goal InputElementRegexp]`

## Class/optional yield arg ambiguity

```js
function *g() {
  class foo extends yield {} // is that an object or the class body?
  {}
}
```

Proposed solutions:

- Disallow trailing yield in extends clause
- Requires an extra parameter to Expression and AssignmentExpression
- extends LeftHandSideExpression
- would be only place an expression isnt explicitly Expression or AssignmentExpression

BE: (whiteboard)

```js
// Don't want to require these parens:
class C extends (A + B) {

}
```

BE: Should have no trailing yield, as a static error

... lost track here...

WH: (whiteboard) Counterexample to claim about never requiring parentheses in expressions that would be unambiguous without them:

```js
a = b * (yield c)
```

WH: Think of "extends" as having the same precedence as assignment operators. That's why class C extends (A + B) would require parentheses.

#### Consensus/Resolution

- extends LeftHandSideExpression
- would be only place an expression isnt explicitly Expression or AssignmentExpression

## Cross-Realm Symbol Registration

(need slide)

AWB:

```js
// key is a string
Symbol.for(key) => aSymbol // creates a new Symbol if key is not registered

Symbol.keyFor(aSymbol) => key
```

- Where for all strings S: `Symbol.keyFor(Symbol.for(S)) === S`
- the use case for `Symbol.keyFor` is serialization

Revisiting discussion from last meeting, re: Symbol.

Discussion regarding the value, or lack of, registered Symbols over Strings.

DH: Is it necessary to be `Symbol.keyFor()? What about `Symbol.prototype.key`

DS: Is there direct correspondance between `Symbol` and `toString`?

DH: If the symbol is registered, `toString` does the same as `Symbol.keyFor`

AWB: is this good?

DH: What does `Symbol.keyFor` return if the symbol is unregistered?

undefined

#### Consensus/Resolution

- Agree on proposed API.
- If the symbol is unregistered: `Symbol.keyFor(unregistered symbol)` returns `undefined`
- `Symbol.keyFor(not a symbol)` throws

AWB: An issue about Symbols not being usable as WeakMap keys...
...which is ok...

This lead to discussion about (re)naming of WeakMap. It's possible that WeakMap may be renamed SideTable

## Introduce a Prototype object that contains sloppy arguments object @@iterator function?

Discussion re:
- `arguments` prototype
- `arguments.prototype.constructor`

AWB/MM: instanceof is generally misused

AWB: Current spec: all built-in iterators. Self hostable

MM: Issues are with consistency.

BE: (whiteboard)

```js
Array.prototype[Symbol.iterator] !== (function() {
  return arguments[Symbol.iterator];
})();
```

MM: ...future JS programmers learning classes without understanding prototypes. Let's not unnecessarily introduce new abstractions that can't be understood within these semantics.

(meta discussion)

AWB: (describing a spec that used class inheritance for all new inherited objects, eg. `Array.Iterator`)

BE: Return to the question... should `arguments` be iterable, and how?

AWB/MM: (discussion re:

#### Consensus/Resolution

...?

## Conventions for ignore over-ride of @@iterator

MM: Why is this a spec issue?

AWB: @@iterator is bad example, @@toStringTag is better

BE: Use `undefined`, not "falsey"

AWB: re: `new Map(?, "is")`

BE/RW: `new Map(undefined, "is")` defaults to empty list

AWB: Happy with undefined.

#### Consensus/Resolution

- Just use `undefined`

## (function Foo() {}).bind(x).name?

See: http://wiki.ecmascript.org/doku.php?id=harmony:function_name_property

AWB: What about anonymous function expressions?

RW: No name is made

AWB: What about bound anonymous function expressions?

RW: Same, no name.

DS: Could bound functions delegate their name? Do we want to do that?

BE: Might be interesting to find out, can chat with Brandon about this.

#### Consensus/Resolution

- "bind" wins over "bound"
- Brandon needs to review the spec.

## 9 ECMA 404 W3C TAG / TPAC report
(Alex Russell)

AR: (recapping JSON specification leading to 404)
... There are people at the IETF that want more restrictions, eg. a number types, character encoding.

WH/MM: what does that mean?

AWB: Let's avoid doing what they're doing.

AR: They want to restrict all the things, that we're simply not willing to do.
... The draft revision includes non-normative "advice". They've also diverged on what is "valid JSON".

RW: Specifically, they've changed JSON so that JSONValue is not the top level grammar production, using JSONText instead. This means only "{}" and "[]", which is incompatible with Ecma-404.

AR: (proposal to work together)

STH: 3 levels.

1. Bytes on the wire. For example reject UTF-32
2. Sequence of unicode code points that make up a valid JSON sequence. This is covered by ECMA 404.
3. Semantics of this structure. For example, reject numbers with a thousand digits.

#### Consensus/Resolution

- Alex will draft statement to send to IETF last call, due Thursday

## 4.9 Reconsidering Object.is
(Dave Herman)

DH: Object.is might be a mistake

AWB: Could be trying to fill one of two purposes:
- the finest distinction possible
- what you really wish triple equal was

For general use, you want -0 and +0 to be equated, NaN to equal NaN

MM, WH: Multiple NaNs are not visible in JavaScript

AWB, DH: Different NaNs are distinguishable if aliased via TypedArrays

AWB: Hence Object.is does not discriminate as finely as possible

WH: That would be a mistake. ES1-5 clearly state that the NaN values are indistinguishable, and some implementations rely on that for NaN-boxing. Object.is should consider all NaNs to be the same.

[discussion]

AWB: NaNs are technically still not distinguishable in ECMAScript because an implementation has the freedom to write any quiet NaN bit pattern into a TypedArray, not necessarily the one that generated the NaN; in fact, writing the same NaN twice might generate different bit patterns.

Discussion about equating NaN values (https://github.com/rwaldron/tc39-notes/blob/master/es6/2013-01/jan-29.md#conclusionresolution)

MM: The spec allows 0/0 to be written to a TypedArray twice with different actual bit pattern.

More discussion re: IEEE NaN

#### Consensus/Resolution

- Status Quo

# Nov 20 Meeting Notes

John Neumann (JN), Allen Wirfs-Brock (AWB), Yehuda Katz (YK), Eric Ferraiuolo (EF), Erik Arvidsson (EA), Rick Hudson (RH), Matt Sweeney (MS), Rick Waldron (RW), Dmitry Soshnikov (DS), Sebastian Markbage (SM), Ben Newman (BN), Reid Burke (RB), Waldemar Horwat (WH), Doug Crockford (DC), Tom Van Custem (TVC), Mark Miller (MM), Brian Terlson (BT), Andreas Rossberg (ARB), Alex Russell (AR)

## Report from the Ecma Secretariat (CC Report)

(Istvan Sebestyen)

### Status of the TC39 RFTG

## 4.2 Clarification of the interaction of unicode escapes and identification syntax

(Waldemar Horwat)

WH: In ES3 we added the ability to use unicode escape sequences in Identifiers, ie.

```js
var f\u1234o = 17;
```

The restriction was that the unicode escape sequence still had to be a valid identifier. ES3 and ES5 never allowed unicode escapes to substitute non-user-data characters of other tokens such as reserved words or punctuation.

the contention is that ES6 has an incompatible lexical grammar change that lets you write things like:

```js
\u0069f(x===15)

// if(x===15)



There also was a bit confusion about whether escape sequences can occur in regexp flags, even though the grammar never allowed them there either:

/abc/\u...

// unicode for the flags

```

AWB: Things that came up in ES5:

- can you declare a variable that has the same unicode escape sequence as a keyword?

```js
var f\u0

```

- introduced in ES5:

```js
// allow
foo.for()
```

ie. Identifier vs IdentifierName

WH: Cannot use escapes to create identifiers that would be invalid. Also opposed to allowing escapes inside keywords; there should be just one spelling of the keyword ```if```, and it should not include ```\u0069f```.

So what should we do about \u0069f(x===15) ? It depends on how we interpret the ES3/ES5 rule that states that escapes cannot be used to create identifiernames that don't conform to the identifiername grammar.

Option A: Treat the if there as an identifier because there are some contexts in which "if" can be used as an identifier (notably after a dot), making this into a function call.

Option B: The if there cannot be an identifier in this context, so it's a syntax error because we're trying to spell a reserved word with an escape.

AWB: Agree there is ambiguity

MM: https://code.google.com/p/google-caja/wiki/SecurityAdvisory20131121

"Handling of unicode escapes in identifiers can lead to security issues"

Records the vulnerability that has now been fixed in Caja at the price of additional pre-processing. This vulnerability which was caused by ambiguity in interpretations of the ES5 spec by different browser makers.

STH: If there are systems that need to search code for specific forms

MM: It would be harmful to code that looked at keywords, then this could circumvent those assumptions.

AWB/WH: (recapping acceptable use of reserved words as identifiernames)

BE: We can fix it, but it's just not how ES6 spec works

WH: If it is a ReservedWord, it may not be spelled with an escape.

MM: This solves Sam's static code case

BE: No escape processing upstream?

WH: (agreeing)

AWB: We can specify in the grammar that where we write "if" it means that exact character sequence

...Anywhere we express literal keywords, we mean those character sequences.

#### Consensus/Resolution

- ReservedWords, including contextual, can only be spelled with ascii characters, ie. the literal character sequence.
- No escapes allowed in such ReservedWords

## Performance impact of Tail Calls

(Brian Terlson)

BT: Wondering if any implementors have begun work on these? Are there considerations for existing code that will become tail call?

YK/AWB: Any examples?

BT: Stack frame manipulation

BE: It's not a zero work to new work, it's an old work to different work.

STH: There's a lot of work on this subject, presumably tail calls should be able to run as fast as it does currently. No advice that's implementation independent.

ARB: Standard techniques should be applicable. Foresee a lot of work.

YK: The only real value for practioners is for compile-to-js cases.

DC: This is actually the most exciting feature for me, because it allows

BE: Will have someone work on this for SpiderMonkey

RW: Agree that implementors will feel the pressure once practioners experience the benefits that Doug describes.

DH: Allows for real cps transformations that won't blow the stack and don't require awful setTimeout hacks. FP idioms being available to JS.

#### Consensus/Resolution

- Share implementation experience

## super and object literals
(Allen Wirfs-Brock)

(needs slides)

AWB: Issue: how do you mixin some methods that reference super?

```js
Object.mixin(obj, ???);
```

In the process of mixing, Object.mixin will rebind super references to the target. The big problem: `super` is currently explicitly illegal within an object literal:

```js
Object.mixin(obj, {
  toString() {
    return `mixed(super.toString())`;
  }
});
```

BE: are we asking to allow super anywhere?

MM: We're not adding a restriction?

AWB: No, removing.

MM: Strictly a simplification.

Discussion re: Object.mixin

WH: Curious about the design of exposing super to user code, but only via the Object.mixin API. If we're going to be storing and retrieving super from a hidden slot, this seems a very roundabout API that's going to bite us.

AWB: Allow super in concise methods

EA: All object literals?

RW: No, because the property value could be defined elsewhere. Ensure invalid in function and it's ok

EA/AWB/RW: Allow super in concise methods within object literals.

Clarification of Object.mixin capabilities.

MM: (has issue with the naming)

AWB: Let's defer discussion of naming.

YK: We should allow super in function expressions within object literals

MM: Refactoring hazard

DH: There is always a refactoring hazard when scope is involved (super)

RW: On board with Erik and Yehuda, super should be allowed in both concise methods and function expression literals that are the value of properties defined in an object literal.

DH: `Object.mixin` creates a new function when rebound?

AWB: Yes.

MM: (whiteboard)

```js

{ foo() {}, ... }

// vs

{ foo: function() {}, ... }

// vs

{ foo: (function() { return function () {}; })(), ... }

```

DS: Concern about having a reference to a function object that doesn't equal the rebound method

```js
function f() { super.foo(); }

Object.mixin(o, {
  f: f
});

o.f !== f;
```

BE: No way to define a property on a concise method declaratively.

WH: RebindSuper doesn't copy expandos (referring to Allen's claim that it does) http://people.mozilla.org/~jorendorff/es6-draft.html#sec-rebindsuper

(The actual copying of expandos takes place in MixinProperties http://people.mozilla.org/~jorendorff/es6-draft.html#sec-mixinproperties

DH: Issue: bind does a similar operation, but doesn't copy expandos.

... Any other deep traversals? If you have a non-callable object, it only does a shallow copy?

... The existance of super creates an inconsistency.

AWB: Alternatives are always clone or never.

DS: All methods should be copied to avoid the distinction

YK: Don't copy exandos?

EA: Happy to go back to Object.defineMethod

YK: Still need to decide if it copies expandos

DH: That's the smallest operation that you can build on

WH: Object.mixin breaks membranes, no way to intercept the super rebinding when the method is a proxy.

AWB: There are many operations like this

EA: No different from Function.prototype.bind

MM: What happens when the method is a Proxy?

AWB: A proxy for a method is not a function.

MM: A Proxy whose target is a function?

AWB: It's not an ordinary ECMAScript function

MM: Anything we do, we should ask "What does it do across membranes?" There are two criteria that often come into conflict:

- Security
- Transparency

Discussion about Security vs. Transparency

EA: What happens when do bind on a function proxy?

MM: fail?

DH: This is shocking.

MM: bind is a perfect example, there is no conflict between security and transparency. You'd like bind to work on proxy functions

EA: (whiteboard)

```js
Function.prototype.bindSuper
```

MM: They're saying, do the [[get]] on the proxy, you don't get bindSuper back, you get a proxy for bindSuper ... membrane safe.

YK: Change bind?

DH: Can't change bind, varargs

Mixed discussion re: home binding.

AWB: Expose the home binding via trap?

DH: trap makes sense to me

MM: From the method you have access to the home binding?

AWB: yes

MM: Don't like that

AWB: Another way

WH: The method calls "super" and expects to reach it's super

AWB: There could be a super call trap

YK: Any objects to bindSuper?

DH: No idea what this means.

BN: What is the material difference between defineMethod and bindSuper?

bindSuper: like bind, but only changes super. Could be defined in terms of Object.defineMethod:

```js
// illustrative only
Function.prototype.bindSuper = function(homeObj) {
  return Object.defineMethod(homeObj, this);
};
```

changing this and super is a two step change:

```js
function f() { super.foo(this); }

var o = {
  foo(target) {

  }
}
(fill in later
```

DH: bindSuper is the max/min of define

- takes one target argument
- copies code and changes super references to target

on a bound function?

BE: On a function with this and super, changing both will create two new functions

AWB: This is a "clone function"

DH: Meaning, only clone the this-binding, not the expandos

...

AWB: you'll need to bindSuper, then bind

AWB: If you want it to work in either direction

WH: binding super after binding this will cause problems. That would be an anti-feature that breaks abstraction. A lot of times, code will return a bound function specifically to prevent you from changing this. Changing super in such a function would break the abstraction.

?: Want bind and bindSuper to commute

WH: Don't want them to commute. They're fundamentally different. bind can only be done once and freezes the this binding. bindSuper can be done repeatedly and doesn't freeze the super binding.

?: You can already rebind this in a bound function

MM: No. If you bind it again, it doesn't mutate the bound this value; the second one is ignored.

DH: (whiteboard)

- mixin -> defer, focus on primitive
- defineMethod -> not proxyable
- bindSuper -> good
- proxying -> good
- composition with bind

bind().bindSuper() -> ERROR.

bindSuper().bind() -> OK.

bindSuper can be called on the result of bindSuper (which is why YK/MM dislike the use of "bind")

Alternative names:

resuper, bindSuper, supersede, withSuper, super?

withSuper, bindSuper?

bindSuper(obj[, ...])

BN: what does super.valueOf() return?

DH: should be this, similar to what super evaluates to in Smalltalk (according to AWB)

- static error vs dynamic error? DYNAMIC.
- where is super given a binding (other then class)?
  - class methods
  - method shorthand
  - in obj literal wherever name inferrable

Discussion re: naming. The shed is pink? It's more of a mauve, I think. You would.

#### Consensus/Resolution

- Remove Object.mixin
- "toMethod()" wins -- debate about argument order
- debate about what [[MName]] is and what it's derived from
  - super delegation uses [[Name]]
  - there's a prototype property for name as well
  - and functions with names have an own property that's .name, while function.prototype has a .name that's a null string
  - .name is configurable, non-writeable, but not necessarialy an own property -- depends on how the function was defined
  - clarification that ".name" has no effect on [[Name]]
  - clarification that ".name" has no semantic effect on other methods that might consume a name
  - copied: length
  - result name: whatever we decided [[mname]] was
  - bound functions cannot be converted to methods
  - bind().toMethod() -> throws

Function.prototype.toMethod(home[, mname])

Dave, please review the details above.

## Reconsidering the Map custom comparator API

(Dave Herman)

DH: Something incredibly gross about having an API that allows exactly one string, but I know we need to solve the bigger problem which is being able to provide performant custom comparators.

Can we just get rid of this argument?

WH: [Recaps consensus decision from prior meeting and the reasoning route by which we arrived at it.]

MM: (gives memoization example)

DH: This can be addressed in ES7

Discussion re: -0/+0 difference.

It was pointed out the only difference between the default comparator and the is comparation is the handling of -0/+0 and that a subclass of Map that ditingishes between +0 and -1 using Object.is can easily be written in ES code.

#### Consensus/Resolution

- Remove second param to Map and Set constructor
- Defer to ES7

## Math.hypot() and precision

(Dave Herman)

http://people.mozilla.org/~jorendorff/es6-draft.html#sec-math.hypot

DH: Oliver Hunt brought this up, do we want to maximize precision (by sorting) and take the performance hit? Or do them in the provided order.

Prefer the latter. Oliver prefererred sorting for precision but taking the performance hit.

BE/DH: He's not here.

Referring to IEEE 754

Luke provided the original spec text, but it's changed since then.

BE: need to look at SpiderMonkey implementation and possibly provide new spec text.

WH: Sorting doesn't matter much in this case; it's a second-order effect. Cancellation is impossible because all squares being added are nonnegative.

WH: What does greatly matter is not overflowing for values > sqrt(largest finite double). What does hypot(1e200, 1e210) do?

BE (runs it on bleeding edge Firefox): About 1e210

WH: Good. We do want to avoid the intermediate overflow that would turn this into +?.

[more discussion]

?: This isn't just about hypot. How should we specify precision in general for things such as transcendental function.

WH: It's a moving target. Do not want to encode precision requirements in the standard on anything other than basic arithmetic or number?string conversion in the spec because those are complicated and how to specify them varies depending on the function. Best thing to do is link to some existing writeup describing best practices.

BE: I'll beat the drum to get a spec. Dave's right that it's bad language.

#### Consensus/Resolution

- Brendan to propose replacement for last two steps.

## 4.10 Generator arrow function syntax

(Brendan Eich)

BE: This isn't a big deal and should be easy to bring into ES6. Experience so far has been that people love arrow functions and generators and want a generator arrow

(whiteboard)

```js
// current
x => x * x;
(...) => { statements }
(...) => ( expr )

// proposed generator arrows...

// Irregular
() =*>

// Hostile to ! (async function)
() => * { ...yield... }

// Not good
() => * (yield a, yield b)

// Ok if 1 token
x *=> x * x;

// Bad (ASI)
*() => ...

// Hostile to !
(x) =* {...}

```

WH: Don't like *=> because it swaps the order from function*.

WH: The ! problem in =>* can be solved by using % or ? instead of !. Would prefer those characters anyway.

BN: Another (strawman) possibility is the presense of yield.

BE/WH: No

DH: Recalls implied generator (yield presense) footgun

DH: There is not a 1-to-1 correspondance to where you'd use function or function *. Arrow is not a replacement for all functions that want lexical this.

#### Consensus/Resolution

- No addition, revisit for ES7

## for-let

(Brian Terlson)

BT: We've shipped for-let without fresh bindings per iteration (according to the current spec) but we're ok with updating.

MM: Consensus?

RW: recalling the consensus from yahoo 2012

DH: Need consensus on the semantics of capturing in the expression positions

DH: if there's something that "closes over" that variable, what's that referring to? I remember that thead, but I don't reacall the otucome

AWB: no definitive outcome... no satisfactory solutions

DH: we have this job on this committee... ;-)

BT: that we shipped in IE has no weighting on this?

AWB: nope. Should have looked at the spec which has notes to this effect

(discussion about binding per iteration)

AWB: C# addresses this by saying "this is insane, so for C-style of or, we have per-iteration bindings, not per-loop bindings"

MM: so let in the head of the loop creates only one location?

(yes)

EA: if we don't resolve this today, we sould fallback to what IE 11 does.

DH: sure, but we have to go through this thread

AWB: The first time you initialize, create an extra scope contour, the zeroth iteration. This is where the capture occurs and the subsequent iterations propagate to that scope.

AWB: if you order these things right, the 3rd part happens at the end, but before your propagate

MM: you mutate and then the value gets copied... seems fine

```js
var a = [];
for(let i = 0, f = () => i * i, a.push(f); i < N; i++) {
  a.push(f);
}
for (let f of a) {
   console.log(f());
}
```

```js
for(let i = 0, f = () => i++; i < 1; f()) {
}
```

This is an infinite loop. Reasoning:

1) The outer scope receives its initial value for i and f. Critically, f's i now binds to this outer binding.
2) The outer scope forwards these values into the first iteration of the loop
3) In the beginning of the 1st loop iteration, the test is executed. At this point, i is still zero.
4) After, still on the first iteration, the test for i < 1 fails because i is zero.

5) Since we never modify the loop variable, this must be an infinite loop.

BE: FWIW, Dart has the same semantics.

```dart
void main() {
  foo(fun) {
    print('pre');
    print(fun());
  }
  for (var i = 0, inc = () => i++, j = foo(inc); i < 5; inc = () => i++) {
    print(i);
    inc();
  }
}
```

outputs

```
pre
0
1
2
3
4
```

#### Consensus/Resolution

- Brand new outer scope created around the entire loop that has variables that are declared in the loop head, and it gets the initial values
- There is a new scope for each iteration that receives values from the previous iteration

## 5 Post ES6 Spec Process.

(Rafael Weinstein)

Train model.

#### Consensus/Resolution

- Sounds reasonable, we're going to try it.

## Ordering of scheduling of microtasks

BE: FIFO

AWB: In the ES6 we need to say something.

?: Examples of why browsers want to use priority queues to schedule tasks

[Debate about whether in ES6 we need to mention the priority queues]

?: DOM and other tasks are beyond the scope of the standard. Just say that ES6 tasks are in FIFO.

WH: Would prefer to mention a richer priority structure in the spec; otherwise other groups (W3C) will want to fit their tasks into our FIFO, which is not desirable. At the very least we must say that other tasks with visible effects may get arbitrarily interleaved between the ES6 tasks we talk about in the spec, so don't assume that nothing can come between adjacent ES6 tasks in the FIFO.

MM: Rafael and I went throught the existing DOM behavior...

YK: Disagrees with Rafael. Bucketing. Series of buckets. The first bucket is the cheapest operations and the last bucket is the most expensive bucket. If a bucket adds something to an earlier bucket then you go back to to earliest bucket that has items in it. Each bucket is a FIFO queue.

WH: Can you reorder the operations so that the DOM operation happens next to each other.

YK: I think a priority queue is isomorphic to buckets.

AWB: In ES6 we only have one class of priority which is the priority of Promises. We do not need to spec that there might be different priorities.

#### Consensus/Resolution

- ES6 spec needs to spec that Promises are serviced in a FIFO queue
- Other non ES6 tasks might be interleaved arbitrarily
- Interleaving of the Promise queue by other non ES6 operations

# Nov 21 Meeting Notes

John Neumann (JN), Allen Wirfs-Brock (AWB), Yehuda Katz (YK), Eric Ferraiuolo (EF), Erik Arvidsson (EA), Rick Hudson (RH), Matt Sweeney (MS), Rick Waldron (RW), Dmitry Soshnikov (DS), Sebastian Markbage (SM), Ben Newman (BN), Jeff Morrison (JM), Reid Burke (RB), Waldemar Horwat (WH), Doug Crockford (DC), Tom Van Custem (TVC), Mark Miller (MM), Brian Terlson (BT), Andreas Rossberg (ARB), Alex Russell (AR), Mark Miller (MM)

## Follow Up on IETF JSON WG Communication document

(review)

#### Conclusion/Resolution

- Unanimous consent

## 4.5 Modules

(Dave Herman, Sam Tobin-Hochstadt)

(request slides)

(fighting Google Hangouts for fun and profit)

DH: Spec draft: Done.

Initial implementation for Firefox, POC for spec: https://github.com/jorendorff/js-loaders

End-to-End Draft Done:

- Linking semantics

- Loading semantics
- Execution semantics
- Loader API
- Dynamic: CommonJS, AMD, Node compatibility layer
- Bonus: literate implementation starting to pass simple tests

DH: there's a compat layer for dynamic import systems. Hope is that in a couple of months it can ship in nightly FF.

EA: is there a license on this?

DH: no, but we'll add one.

AWB: if you're contributing to ECMA, it has to be under the ECMA software contribution rules (ie, BSD license)

AR: can you add a license today?

ARB: how are these files related to each other?

DH: things are extracted from the source, and the short version is that the actual wording is now done. Then next steps are to improve the formatting and work with AWB to reconcile editorial issues.

AWB: yes, this is what we've done with Proxies/Promises/etc. Something we know how todo. if there are normative changes, they'll get resolved.

ES6 Draft

- Updated syntax for new parameterized grammar
- Static semantics done and implemented

DH: stuff that has been through the editorial process are the syntax, the static grammar, and static semantic rules. Don't have a doc dump this morning but can generate one today and send it around.

DH: didn't have a chance this morning to do it yet.

ARB: was only looking at the docx fragments so far...

DH: wanted to talk a bit about the last bits of the semantic cleanups. The distinction between scripts and modules and the browser integration: `<script async>` was a way to allow you to use the module system at the top level. This creates 3 global non-terminals. I say you dont' need `<script async>`.

Clean Ups

- Scripts and Modules

- Three goal non-terminals (Modules, Script, ScriptAsync) is one too many
- Elimintaing imports from scripts simplifies the Loader API—loading is purely about modules
- `<script async>` was a non-start anyway

DH: in additon, the idea of using `<script async>` was misguided.

AR: no, we can fix `<script async>`

WH: we discussed this at previous meetings, what happened to the need for script async?

DH: `<module>` instead. A nicer path forward for the web. Automatic async semantics. More concise than `<script async>`.

- Not part of ECMAScript
- As concise as `<script>`
- More concise than `<script async>`
- allows inline source
- implicitly strict (as all modules are)
- Named `<module>` provide source up front
- Anonymous `<module>` async but force exec

AR: this has real security issues.

YK: this can have other forms -- `<script type="module">`...etc...

DH: the goal here is to come up with the cleanest design we can come up with on the JS side and drawing a sharper distinction between scripts and modules. You get to start in a clean scope. Top-level decls are scoped to the module.

DH: you also get implicit strict mode. And inline source.

AR: this is never going to work in the field. This will violate expectations.

YK: can you show me what's secure today that does blacklisting?

AR: no, but that's not the argument.

<module> is a better 1JS

- Better global scoping: starts in a nested scope
- To Create persistent globals, must opt in, by mutating window/this
- Conservation of concepts: no special semantics required for reforming global scope, just a module
- A carrot to lead away from blocking scripts
- still accessible to scripts via `System.import`

WH: Universal parsing is a problem. HTML parsers know that escaping rules are different within a script tag but not within some other random tags such as module.

EA: you can see a transition path that starts with `<script type="module">` and move to `<module>` later

(some agreement)

DH: if `<module>` turns out not to work, we have other options. One of these is likely to work

AR: agree.

Alternatives:

- script module
- script type="module"

DH: benefits include: top-level decls are local. You can persist by opting in (window.foo = ...), but it's not a foot gun via var. No special semantics for the global scope; it's jsut a module. Still accessible via System.import.

```html
<module>
var foo = 1;
</module>

<script>
foo; // undefined.
</script>
```

DH: Recapping the concatenation story.

(didn't we all do this before?)

YK: `<module>` and `<script module>` are morally equivalent

STH: to address issues with `<script async>`, we need a separate entry point. Any of the others listed here address the use cases brought up that `<script async>` addresses.

WH:now you have 3 things

(emphatic disagreement)

DH: there are 2 terminals in the global of JS in this world, not 3. `<script>` and `<script async>` have the same terminal production

DH: we've had a hard time working through the global scoping semantics. We don't need special semantics for a reformed global scope. Deflty avoids implicating global semantics. Just write modules.

ARB: you still need the lexical contour, no?

DH: yes, but lets talk about that at the end.

DH/YK: there's an unnamed module loading timing that's before `<script defer>`

EA: you can't wait till domcontentloaded to run stuff. Need stuff running sooner.

ARB: Multiple module elements?

AR: Document order

EA: I jsut don't want us to cripple the system for this use-case, e.g. the HTML5 doc.

AR: you need both.

YK: a sync attr seems fine.

DH: we can't operate as if the web didn't exist, but we can't define HTML elements, so we need a story but not the answer.

BE: partial progress if possible

DH: lets talk about the loader API. Will send out slides soon;

Loader API
- https://gist.github.com/dherman/7568080

```js

var l = new Loader({
  normalize: n,  // Promise<stringable>
  locate l,      // Promise<address>, formerly "resolve"
  fetch f,       // Promise<source>
  translate t,   // Promise<source>
  instantiate f, // Promise<factory?>, formerly "link"
});
```

address -> where to find this

source -> the source text

WH: what's fetch for?

DH: gets the bits that are stored. The translate hook provides ways of transforming bits into other bits.

YK: `instantiate` is the bridge for legacy module forms: amd, node, etc.

WH: What exactly is promised in each step?
(see gist)

Core API

```js
// Loader API
// Promise<void>
l.load(name[, opts]);

// Promise<void>
l.define(name, src[, opts]);

// Promise<Module>
l.module(src[, opts]);
```

WH: If load is called twice, does it reuse the same module?

DH: Yes

WH: Concerned about options. How close do the options have to be to reuse instead of loading twice?

DH: the main option here is that you can skip the locate step. Load lets you start at multiple places.

DH: define() lets you do things after the fetch step, while module() lets you do the anonymous module thing.

EA: I'm worried that define() is a new eval().

DH: yes, that's what it's about. Downloading and evaulating code. Some people say "eval is evil" and I say "there would be no JS on the web without it". This is a more controlled and sandboxed way of doing it.

WH: define returns a promise of void. How do you safely use the defined module?

EA: I'm worried about CSP.

DH: l.import() as a convenience api. Kicks off a load and forces an execution of the module. Resolves to the module object. It's a nice way of doing it all in one shot.

DH: the registry API

```
// Registry API, methods on a Loader instance
l.get(name)    // Module?
l.set(name, m) // void
l.has(name)    // boolean
l.keys()       // Iter<string>
l.values()     // Iter<Module>
l.entries()    // Iter<[string, Module]>
```

MM: delete?

BE/RW: size property?

AR: if you can change it, you can delete...why not have it?

DH: you might astonish a running system

STH: it only removes it from the mapping. Agree with the misgivvings, but we should have it.

YK: I can imagine delete for security purposes

STH: clear() would be insane to use, but....

MM: if we have an existing contract, we should have it, else we should define some supertype

DH: it's a no-op in JS to do that. People are warming up on delete()

WH: what do these things actually do? eg. `l.define(name, ...);` then `l.get(name)`?

YK: there's a turn between when things are done and when they're ocmmitte. You see the "old" view.

WH: What happens when you call define twice with the same name?

DH: They race. Name stays what it was until one of them gets fulfilled; it's nondeterministic which one.

WH: How do you find out if there's a pending define on a name?

DH: You can't.

WH: That makes it impossible to write safe modular code unless you're the very first thing to run. Otherwise anything you try to define could be racing with something already started.

DH: `set` is synchronous "add this eagerly". Get is sync get.

DH: this is an inherently racy API because module loading is racy.

WH: why not placeholders indicating that a load is in progress?

DH: there's an implicit one in the system, and we try to have sanity checks, but ....

STH: now it's observable that things are loading in some order

WH: That turns a race condition into a reliable fail.
If someone tries to load a module and then I load a module of the same name, I want that to fail.

STH: (explains pollyfill)

WH: True, but don't see how that's relevant. I have no problem with module replacing. I want it done in a safe way, not in a racy way.

DH: We have handling

EF: Do yo have slides for all the exceptions?

DH: Do not, Jason has it documented, but could not be here.

STH: There are thousands of lines that explain all of this very precisely.

WH: I asked this earlier, what happens when I call define with the same name twice and was told it's non-deterministic.

DH: I stand by the statement that this is non-deterministic, there are too many cases.

(explains several common and uncommon cases)

e.g., jquery dep fails, but something else depends on it. zero refcount. Common deps may cause successful subsets to succeed or fail together.

WH: agree that you'll get non-determinism. But should we hide the started/unstarted state from the user?

(discussion of observability)

BE: Let's take

DS: The registry is global or tied to a particular loader

DH: A built in loader called "System"

Decoupling Realms from Loaders

(smells)

- Loaders are really about modules, but eval/evalAsync represnt scripts
- Mixing concerns: scripts vs modules
- Mixing concerns: module loading vs sandboxing
- Capability hazard: `new Loader({ intrinsics: otherLoader })`

Realms: Globals, Intrinsics, Eval Hooks

(facts)

- Global object & Function/eval are "intertwingled"

- Intrinsics and standard constructors, prototypes are "intertwingled"
- Realm and intrinsics are "intertwingled"
- Everything is deeply "intertwingled"

YK: (explains import from registry)

... Can create a different Loader for maintaining state in a specific module while loading another module of the same name.

Realm API

A realm object abstracts the notion of a distinct global environment.

~~~
```js
let loader = new Loader({
realm: r, // a Realm object
});
```
~~~

https://gist.github.com/dherman/7568885

~~~
``js
let r = new Realm({ ... });

r.global.myGlobal = 17;

r.eval(...);
```
~~~

AWB: This "Realm" as a global object is questionable.

DH: Talking about "Realm", "Loader" and "Module", likely should be in a "module" module.

Seperable From Loader API

- Important, but it's a separate concern
- Loader defaults to its own realm
- Realms; sandboxing, sync script executuon
- Loaders: async module execution

DH: a "Realm" is a "virtual iframe" from the same domain. It's a global with the DOM stripped out of it.

WH: Think there should be commonalities between API surface of Realm and Worker

MM: When you create a new Realm, what do you provide in order to give it initial state?
... Let's just keep this in mind.

Realm API

```js
let realm = new Realm({
eval: ...,
Function: ...
}, function(builtins) {

  // global object starts empty
  // populate with standard builtins

  mixin(this.global, builtins);
});
```

DH: First object contains things to explicitly add to the Realm's global.

EA: What happens when you omit it.

DH: you get an object that inherits from `Object.prototype`

WH: Is the global object always accessible at `this.global`?

DH: Yes

EA: This is the wrong default, we need something else.

MM: The only properties on global object that are non-configurable are `NaN`, `Infinity`, and `undefined`

DH: Better to start out empty and fill it yourself then to create something that almost looks like global

EA: The most common case is a global environment with all the builtins, should be the default

RW: agree

MM: So, if the callback is omitted, you get a realm that is the default environment with all the builtins.

EA/RW/DH: yes

STH: If the callback is provide, the realm's global is a null prototype object

(summarize change)

- no callback, default environment with all the builtins
- w/ callback, object with null [[Prototype]]

WH: What does the init provide?

DH: Allows you to whitelist what goes into your realm

AR: what about the second arg? can that be folded into the first?

DH: yes, perhaps "init: function(...) { }"

Indirect Eval

```js
let caja = new Realm({
```

```
indirectEval: function(args) {
    return anything
}
});

caja.eval(`
  (0, eval)("1 + 2")
`);
```

STH: I was really skeptical of this, but I was persuaded.

WH: Why not use something simpler such as define realm.evalToken as the function to compare against what the eval identifier evaluates to in order to check for direct vs. indirect eval?

(Offline conversation with MM/DH/YK/BT: indirectEval hook returning the result of eval wont' work without a way to refer to eval. Fix is to make indrectEval a translate hook similar to directEval.)

Direct Eval

```js

let caja = new Realm({
  eval: {
    direct: {
      translate: (s) => { ... }
    }
  }
});

caja.eval(`{ let tmp = f();
             eval("tmp") }`);
```

Direct Eval

```js
```

~~~
let caja = new Realm({
  eval: {
    direct: {
      fallback: (f, ...rest) => { ... }
    }
  }
});

caja.eval(`{ let tmp = f();
            eval("tmp") }`);
```
~~~

WH: Why so many levels of destructuring in the first parameter to the Realm constructor? Do we need then all?

Function

~~~
```js
let caja = new Realm({
  Function: (args, body) => { ... }
});

caja.eval(`
  Function("x", "return x + 1")
`);
```
~~~

ARB: Why not spec it by defined concatenation plus eval?

STH: Explains the issues the exist with toString.

WH: If the engine validates the arguments before calling Function then that limits the ability to provide new syntax for Function (for transpilers).

DH:

- Can't create new arguments syntax

- Simpler apis lose strong guarantees from validation?
- (need the third reason)

Function*

```js
let caja = new Realm({
  Function: (args, body, prefix) => { ... }
});

caja.eval(`
  (function*().constructor("x", "return  x + 1")
`);
```

`prefix` is the string that is either "function" or "function *"

DS: concern for changes in semantics of

discussion re: eval of source code

DH: Returning source code is the more conservative

MM: script injection problems because people want to create source from concatenation. If we translate an array of source pieces, then we're not doing concatenation in the spec.

MM: The string that's the prefix piece determines the constructor. "function" => Function, "function*" => Generator

DH: two place you can reach the function constructor

- `Function` global
- `Function.prototype.constructor`

If you mutate `Function.prototype.constructor`

1. Create a new Realm
2. Save the original `Function.prototype.constructor` to the side
3. Create a new `Function` in that global
4. Mutate its `Function.prototype.constructor` to whatever you want

MM: This strategy has been field tested in SES

STH: Function.prototype.__proto__?

BT: Object.prototype

Realm API

```js
let r = new Realm({
  eval: {
    indirect,
    direct: {
      translate, fallback
    }
  },
  Function
});
```

DH: A compiler for ES7 -> ES6. The compiler has some static source its carrying around and comes to:

```js
|-
|-
|-
eval(x)
```

And contains source:

```js
{t1, t2, t3}
```

Translate to an Identifier "eval", no way around this

```js
...
translate: function(s, src) {
  return compile(s, src);
}

fallback: function(f, ...rest) {
  return f(...rest);
}
```

WH: Fallback needs a this binding to handle cases with 'with' such as this one:
a.eval = <something user-defined>
with (a) {
  eval(b)
}

[agreed]

Cross-Domain Modules

- browser loader should allow cross-domain loads
- ServiceWorker requires cross-domain sources (XHR) and sinks (<img>, <script>, etc)
- Browser's `System.define` should be a sink
- No Problem: type of src parameter unconstrained in ES Semantics—polymorphism to the rescue
- IOW each loader determines for itself what types are allowed for source

Streaming Modules

- Might want to support streaming I/O in the future

- Polymorphism allows admitting stream objects for `.define` and `.module` in future

YK: We've built a layer under the new constructs and CSP can work at that level: `loader.eval`, etc. A small core that we can build on top of.

#### Conclusion/Resolution

Realm constructor semantics changes:
- changing "eval" to:
  - directEval
  - indirectEval
- Remove "Function" hook
- no callback, default environment with all the builtins
- w/ callback, object with null [[Prototype]]
- fold callback into first argument as option, eg. `init: function(...) { }`
- "fallback" hook needs an additional first argument: `this` binding

Dave and Allen, follow up on:

AWB: Not all intrinsics are named?

DH: Yes. Don't need to be hookable, create a new Realm and get the intrinsics.

AWB: Do you only have to define the constructor of the intrinsic? Wire up first?

DH: Don't think those things need ot be customizable. We can extend the optional named parameters of the Realm constructor in a compatible way.

... Can fully customize the relationship between a Loader and Realm.

EA: Does modules have a hard dependency on Realms?

DH: no

AR: Seems like it's adding?

DH: Nothing new, we're just defining what was there.

AWB: Prioritize?

DH: Modules, Module loader first. Realms later.

STH: (explains that Module Loader can't be deferred, needed by browsers)

AWB: Realm seems very essential...

DH: Not necessary, don't put off the Loader API

ARB: What is the confidence level of this spec?

AR/DH: Very confident, but assume there are small bugs to address.

AWB: Very excited, feel like we've finally gotten it.

MM: Really like this, but would like to attempt reimplement SES on top of this and Realms API

ARB: (likes Realms as well)

STH: re: confidence in design... Dave, Yehuda, Jason and I have made revisions for a few cases:
- making sure everything held together at the lowest level
- have addressed all of the use cases that were necessary

DH: No more use cases, no more churn. There will be bugs, but we will discover these during implementation and address accordingly.

ARB: Happy with the design now, this is great. I agree that there wont be major changes anymore. Realistically no way to know if the spec is spec'ing what we think it is, until we get to it. What damage would it cause if we defer this?

AR: Alot

DH: Massive

BE: You'll lose momentum and gain a perceived failure hazard.

DH: JS Community will walk away.

ARB: is this all true?

YK/AR/RW: Yes.

Discussion about time line of ES6 implementation

(break)

## 4.6 The global scope contour
(Dave Herman)

Relevant:
- https://github.com/rwaldron/tc39-notes/blob/master/es6/2012-07/july-25.md#scoping-rules-for-global-lexical-declaration
- https://github.com/rwaldron/tc39-notes/blob/master/es6/2012-09/sept-19.md#global-scope-revisit

DH: I want to make the case that we don't think we need a second global scope contour.

AWB: We require a different set of rules for how to interact with new binding forms

DH:
- A clean story for what the development model is for how you share code
- Starting your script in a module, you're already in local scope
- no need to be scoping bindings in a global scope
- if you want to scope things, use a module
- if you want mutation, use a script

AWB: Can let, const, class be script scoped?

DH: Could disallow let, const, class in Script

EA/RW/WH: no.

ARB: How does that not violate 1JS

Propose:

  let, const, class scoped to Script, as if it had an implicit scope

EA: This would prevent people from transitioning from function to class.

RW: Developer console hazard.

ARB: class has TDZ

ARB: HTML event attributes might need to see the declarations

RW: Use developer tools as an example. The transition has hazards where user code thinks it has created a global binding, but actually hasn't.

```html
<script>
let foo = 1;
</script>
```

`foo` appears undefined in the console.

YK/JM: discussing concerns about fragmentation of the mental model of global binding in scripts

BE: Again, we have a consensus on this subject.

MM: In the body of a `with() {}`, what does let, const, class do?

AWB: It's a block. They're scoped to the block.

BE: What is the new input that we're reopening this for?

DH: The new input is that the module tag or type=module is a better model

RW: Concerns that there will be transitional inconsistency across browsers.

EA: The status quo would allow you to see the binding in a later script

Proposed scoping won't allow this.

EA/RW: let, const, class should be "var like" at the global level in Script.

DH: There is a potential confusion. The local scope contour is less confusing than the extensible contour.

BE: It's bad if people type:

```html
<script>
class C {}
</script>
```

And C disappears after the `</script>`

EA/WH/RW: Agree.

DH:
- let is the same as var globally
- const is the same as Object.defineProperty(...)
- class is the same as var globally

EA/RW: ^^^^^ Agreed.

BE: Speculation about module uptake doesn't trump the expectation of cross script global bindings.

#### Conclusion/Resolution

Status Quo. Yehuda commits to work through existing issues for alternate paths.

Next meeting agenda:
 https://github.com/tc39/agendas/blob/master/2014/01.md