Ecma/TC39/2014/007
Ecma/GA/2014/016

Minutes of the:	***38th meeting of Ecma TC39***
in:	***San Jose, CA, USA***
on:	***28-30 January 2014***

1 Opening, welcome and roll call

1.1 Opening of the meeting (Mr. Neumann)

Mr. Neumann has welcomed the delegates at Yahoo! in Sunnyvale, CA, USA.

Companies / organizations in attendance:

Apple, Mozilla, Google, Microsoft, Intel, IBM, eBay, jQuery, Yahoo!, Netflix, Facebook, Indiana University (guest)

1.2 Introduction of attendees

John Neumann – Ecma International

Istvan Sebestyen – Ecma International

Jafar Husain – Netflix

Filip Pizlo – Apple

Mark Miller – Google

Douglas Crockford – eBay

Allen Wirfs-Brock – Mozilla

Nicholas Matsakis – Mozilla

Erik Arvidsson – Google

Waldemar Horwat – Google

Andreas Rossberg – Google

Kevin Reid – Google

Sam Tobin-Hochstadt – (guest, part-time, phone) Indiana University

Dimitry Lomov – Google

Eric Ferraiuolo – Yahoo!

Caridy Patino – Yahoo!

Reid Burke – Yahoo

Matt Sweeney – Yahoo!

Juan Dopazo – Yahoo!

Rick Waldron – jQuery (part-time, Phone)

Yehuda Katz – jQuery

Dave Herman – Mozilla

Brendan Eich – Mozilla

Rafael Weinstein – Google

Jeff Morrison – Facebook

Sebastian Markbage – Facebook

Dmitry Soshinikov – Facebook

Luke Hoban – Microsoft

Brian Terlson – Microsoft

Ben Newman – Facebook

Rick Hudson – Intel,

Simon Kaegi – IBM

1.3 Host facilities, local logistics

On behalf of Yahoo! **Matt Sweeney** welcomed the delegates and explained the logistics.

1.4 List of Ecma documents considered

Ecma/TC39/2013/071	Minutes of the 37th meeting of TC39, San Jose, November 2013
Ecma/TC39/2013/072	TC39 RF TG form signed by jQuery Foundation
Ecma/TC39/2013/073	TC39 RF TG form signed by Ebay
Ecma/TC39/2013/074	Public consultation of 6 technical specifications by the EC
Ecma/TC39/2013/075	Venue for the 38th meeting of TC39, San Jose, January 2014
Ecma/TC39/2013/076	Last update of the TC39 RF TG form
Ecma/TC39/2014/001 (Rev. 1)	Agenda for the 38th meeting of TC39, San Jose, January 2014
Ecma/TC39/2014/002	Draft Standard ECMA-262 6th edition, Rev. 22, January 20
Ecma/TC39/2014/003	TC39 RF TG form signed by Yahoo
Ecma/TC39/2014/004	TC39 RF TG form signed by IBM
Ecma/TC39/2014/005	TC39 RF TG form signed by Apple
Ecma/TC39/2014/006	ICT Standardisation and Innovation of the EC

2 Adoption of the agenda (2014/001-Rev1)

Agenda for the: 38th meeting of Ecma TC39 (final version on Github)

```
in: Sunnyvale, California, USA
on: 28 - 30 Jan. 2014
TIME: 10:00 till 17:00 PST on 28th and 29th of Jan. 2014
      10:00 till 16:00 PST on 30th of Jan. 2014
LOCATION:
    Yahoo, Inc
    700 First Ave.
    Building E, Classrooms 9 & 10
    Sunnyvale CA 94089

CONTACT:
    Matt Sweeney <msweeney@yahoo-inc.com>
    Eric Ferraiuolo <ericferr@yahoo-inc.com>
```

Please register before 21st of January 2014.

1. Opening, welcome and roll call
   i. Opening of the meeting (Mr. Neumann)
   ii. Introduction of attendees
   iii. Host facilities, local logistics
   iv. RFTG Status (Istvan)
   v. RFTG Vote
2. Adoption of the agenda (2014/001)
3. Approval of the minutes from Nov. 2013 (2013/071)
4. ECMA-262 6th Edition
   i. Review Latest Draft (Allen)
   ii. Reconsider enumerability of concise methods. See http://esdiscuss.org/topic/enumerability (Allen)
   iii. Review toMethod and possible alternatives (Allen)
   iv. Reconsider Math.bitlen instead of Number.prototype.clz (brendan)
   v. Discuss convening a security review of module loaders & realms (MarkM, Kevin, Allen)
   vi. Compatibility semantics of functions in blocks (Luke, Brian)
   vii. Don't throw on Array.from(undefined), default to empty array (Rick, Brian) https://bugs.ecmascript.org/show_bug.cgi?id=2435
   viii. Consider not throwing a `TypeError` when `.next(value)` is called on a newborn generator that delegates to a non-newborn generator using `yield*` (Ben Newman)
5. ECMA-262 7th Edition
   i. Structured Clone (Anne van Kesteren, Dmitry Lomov)
   ii. Typed objects update (Dmitry Lomov, Niko Matsakis)
   iii. Value objects update (brendan)
   iv. Parallel JavaScript update (Rick Hudson)
   v. Advance Object.observe within the new process
   vi. Post-ES6 process mechanics
   vii. Do expression
   viii. Async/await (Luke)
6. Test 262 Status
7. ECMA-404 and IETF interactions (Allen, Istvan)

8. Report from the Ecma Secretariat (2013 Dec. GA Report)
   i. Istvan updates on GA, TC39, and IPR matters
9. Date and place of the next meeting(s)
   i. April 8-10, 2014 (San Francisco - Mozilla)
   ii. May 20 - 22, 2014 (Menlo Park - Facebook)
   iii. July 29 - 31, 2014 (Redmond, WA - Microsoft)
   iv. September 23 - 25, 2014 (Boston, MA - Bocoup)
   v. November 18 - 20, 2014 (San Jose - PayPal)
10. Closure

The agenda as posted on Github was approved.

3 Approval of the minutes from the Nov. 2013 Meeting (TC39/2013/071)

Ecma/TC39/2013/071, the minutes of the 37th meeting of TC39, San Jose, November 2013 was approved without modification.

4 Approval to start the TC39 RF (Royalty Free) Task Force

As requested by the Ecma GA in December 2013 in Las Vegas **Mr. Neumann** took the vote company by company if to start the TC39 RF TG or not. **<u>All members in attendance voted for unanimously to start already at this meeting with the TC39 RF TG</u>**. **Mr. Sebestyen** explained that practically all members present have sent their registration forms to Ecma (they were all published as TC39 and GA documents). **Mr. Neumann** announced that all TC39 members should register until the next TC39 meeting in April, otherwise they may only participate in the TC39 Plenary meetings but not in the TC39 RF TG, where the current technical work is being progressed. This policy was agreed to.

5 For details of the technical discussions see Annex 1

6 Status Reports

6.1 Report from Geneva (Ecma/GA/2013/163 – GA minutes from Las Vegas December 2013 meeting)

6.1.1 Brief report about move to a TC39 RF TG

Mr. Sebestyen said that the IPR ad hoc group has just started to work out a solution for a software contribution from non members. 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. It appears that this can be done. If so, then we can close this issue before the ES 6 tests are populated by members and non-members.

6.1.2 Ecma Text copyright suitability for TC39

The IPR committee has started discussed the matter. They agreed that a "Frequently Asked Question" could be the best solution. This would explain what from Ecma's point of view is allowed and what not with the copyright license. But this discussion is not finished yet.

6.2 Json

After successful TC voting the JSON ballot (ECMA-404) has been completed on October 6, 2013.

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 at the November meeting. The liaison letter was sent to IETF right after the TC39 meeting. No official replay has been received before the TC39 meeting. However, the IETF JSON standard is technically finished and it will be published sometimes in the next months. IETF did not listen to the request of TC39. The two specs are at the moment technically very close, but the danger is that with time the difference may widen.

We will close follow how things are developing on that subject.

7 Date and place of the next meeting(s)

Schedule 2014 meetings:

- April 8 - 10, 2014 (San Francisco - Mozilla)
- May 20 - 22, 2014 (Bay Area, - Facebook)
- July 29 - 31, 2014 (Redmond, WA - Microsoft)
- September 23 - 25, 2014 (Boston, MA - BoCoup)
- November 18 - 22, 2014 (San Jose - PayPal)

8 Closure

Mr. Neumann thanked **Yahoo!** for hosting the meeting in Sunnyvale (including the nice dinner on Tuesday), the TC39 participants their hard work, and **Ecma International** for holding the social event dinner Wednesday evening. Special thanks goes to **Eric Arvidsson** for taking the technical notes of the meeting.

Annex 1

Technical Notes (by Erik Arvidsson):

# Jan 28 Meeting Notes

John Neumann (JN), Allen Wirfs-Brock (AWB), Yehuda Katz (YK), Eric Ferraiuolo (EF), Erik Arvidsson (EA), Rick Hudson (RH), Matt Sweeney (MS), Dmitry Soshnikov (DS), Sebastian Markbage (SM), Ben Newman (BN), Jeff Morrison (JM), Reid Burke (RB), Waldemar Horwat (WH), Doug Crockford (DC), Mark Miller (MM), Brian Terlson (BT), Luke Hoban (LH), Andreas Rossberg (ARB), Istvan Sebestyen (IS), Niko Matsakis (NM), Brendan Eich (BE), Rick Waldron (RW), Sam Tobin-Hochstadt (STH), Simon Kaegi (SK)

IS: Talk about the royalty free task group

JN: Approved, 9 in favor. 0 opposed. 0 abstains.
We are now operating as a Royalty Free Task Group

JN: Netflix and Apple haven't returned the paper work.

IS: Neither from Brown University. (and a bunch of other universities that are not active any more)

## Agenda

JN: Agenda approved https://github.com/tc39/agendas/blob/master/2014/01.md

JN: Minutes from Nov meetings also approved. (https://github.com/rwaldron/tc39-notes/tree/master/es6/2013-11)

## Spec Status update

AWB: Latest draft (Rev22) is the "almost feature complete" version. (Discussion about process troubles). Spec now uses a master document and is split into multiple files.

AWB: Still some work to do on Direct Proxies.
Update spec for C-style for-let.
Loaders and Realms are integrated into the spec draft. Lots of review needed.

EA: Standard modules were deferred at last F2F

LH: We agreed on globally reachable names for all pre-modules APIs last September. New names for module loaders APIs need similar design.

AWB: Loader and Realm are speced as %Loader% and %Realm% since we haven't talked about their public names.

DH: Where does Reflect go?

AWB: Use a global.

DH: Realm can go in Reflect since it is a very reflective thing.

DH: We can probably defer System too.

YK: We need an entry point and a global named System seems to be good.

DH: Loader can go in Reflect too.

AWB: Need to update the spec draft to match the scoping rules we agreed to before.

#### Resolution

## Aside on detecting Module vs. Script (see http://esdiscuss.org/topic/detecting-js-language-mode-for-tools)

Debate about whether it's required (partial agreement that it is since they are clearly distinct)
WH: Even though they're distinct, doesn't mean that they must be ambiguous with each other at the source text level. We could easily say that a module has to start with some syntax that designates it as a module.

AWB: Spec doesn't necessarily need to address this (why?) but we should consider setting the direction here. If we're fine with the community coming up with something here we can lean on that too.
LH: This process has to happen, there will be lots of solutions, we might not pick the best one.

Potential solutions:
* YK: Bower uses metadata for packages that say they're ES6 module
* DH: Possibly do nothing, we live in this world already today
* DH: Come up with some kind of syntactical distinction
    * DH: This sucks, modules should not require additional boilerplate to be valueable
* Possibly use file extensions...

#### Resolution
DH to do the work?

## Task and Tasks Queues

Postponed until tomorrow when Raf is here.

## Process for ratifying ES6?

LH: We have now effectively met the "feature complete draft" requirement we were targetting. Are there any process requirements this group is placing on progressing the spec to ratifica

BE: Have nothing, same as ES5.
(Discussion about whether we can be strict)

LH: For example, do we have "multiple compatibile implementations" requirements? Or "test262 coverage" requirements? Our ES7 process proposal has these, but do we want/need some limitied version of this for ES6?

AWB: We just need to ship it and get to the stricter process for ES7.

MM: We can always postpone features to ES7 if they seem to be problematic.

BE: For example, there is unlikely to be an implementation of proper tail calls before we send ES6 to the GA, then to ISO. I don't think we should defer them on that basis, but it's a fact. We should not slip ES6's schedule.

LH: If by the time we ratify ES7 and there are still ES6 features that have not yet been implemented by anyone, we should re-evaluate if we can remove them.

#### Resolution
The commitee is putting no blocking requirements of implementations or tests on standardization of ES6.

## Concise methods and Enumerability

YK: People have object literals today and use these objects with for-in loops to find the properties and if they change to non enumerable consise methods things will break.

AWB: Concise methods were initially non-enumerable. This was to match the built-in classes, like Date, Array etc. One possibility is to make methods in classes non enumerable but concise methods in object literals enumerable.

RW: The same issue happens when people use object literals as the prototype.

AWB: Should Map and Set methods be enumerable then?

RW: The spec should be consistent with itself.

WH: User-defined classes should be consistent with built-in classes. Classes are new in ES6. The question is whether it's the class that turns off enumerability or the concise method that turns off enumerability.
WH: My preference is for the consise method syntax to turn off enumerability. In the rare cases that you want an enumerable method in a prototype, you can define it the longer way.

YK: Refactoring should be simple and not have side effects like these.

RW: hasOwnProperty solves the problem of filtering out class methods inherited from the prototype, so we don't need to make them non-enumerable.

BE: people don't use hasOwnProperty

YK: enumerability is broken, we should not discuss the Platonic ideal form of enumerability

BE: According to Neo-Platonic Mystics, the material world was created by the evil Demiurge, not by Sophia (wisdom). Enumerability and 'for-in' are from the Demiurge.

BE: Enumerating options:
- Concise methods and class methods are enumerable
  - Current es6 draft state
- Split the difference, concise methods enumerable, class methods non-enumerable.
  - can't refactor between es5 object literal and es6 class/concise methods depending on which is non-enumerable
- Make both non-enumerable (Waldemar supports)
  - same refactoring hazard, possibly not desirable
- Some tilde-based or better syntax to allow developers to pick, but go with option 1

YK: Hope that we can use annotations in ES.Future that can control this kind of details.

BE: Agree, so fourth option above.

YK/LH: High bar to make a change, does this meet the bar?

#### Resolution

Status quo. Keep concise methods enumerable.

## More on toMethod

AWB: Function.prototype.toMethod(superBinding, methodName = undefined)
Footgun to put this on Function.prototype.

MM: potential suprise: in the absence of toMethod, super's interpretation can't change, but now people have to account for possibility of shifting interpretation, which requires them to know about toMethod (not sure if I'm getting this right- BN)

AWB: The footgun is that we do not propagate any properties.

AWB: Suggest moving toMethod to Reflect

DH: Needs to be renamed then.

?: Suggests renaming to bindSuper.

WH: We've discussed all of this before and reached consensus. bindSuper is a bad name because it intentionally doesn't commute with bind.

WH: [reviews consensus from past meeting]

AWB: expect people to define Object.defineMethod in terms of toMethod, with copying of properties

DS: do we also need toStaticMethod?

AWB: no, you can just use toMethod with the constructor function as the superBinding / home object

BN: are static/class methods inherited?

AWB: yes

AWB: should we copy .name and/or .length to the new function object?

MM: use bind as a precedent to decide what to copy (mixed messages though: bind does copy .length, minus the number of curried arguments, and (in SpiderMonkey at least) copies the .name, though V8 gives the bound function an empty string .name)

AWB: What should happen if toMethod is called on a non ECMAScipt function object (built-in)

MM: How about just returning a clone of the function. You could even even delegate in a similar way as bind.

AWB: Only allow toMethod on unbound methods?

MM: to preserve possibility of transparent monkey-patching, either make toMethod return something that can't be toMethod-ed again, or allow built-in (non-ECMAScript Function Object) functions to be cloned, which requires an additional path in the internal CloneMethod operation (because it currently asserts that the clonee is an ESFO)

#### Conclusion/Resolution

Allen to take it back to the lab. To get it to work with bound functions and built-ins.
  * Make it match old concensus `func.bind().toMethod(...)` should throw
  * Does not work on proxies

## Clz (count leading zeros)

WH: Don't like deliberately introducing anachronisms a la 1900-origin getYear into the language. Would prefer to have it be origin-0 instead of origin-32 (i.e. return 0 instead of 32 when called on 0) to avoid hardwiring a machine word size. However, understand that we want this to compile to an efficient low-level primitive, so it would still have the toUInt32 as part of its semantics. In that case we should call it clz32 instead of clz so that later we can do clz64 (that would call a future toUInt64) or other variants.

BE: Want it to map to one single machine instruction

BE: Mislocated, should be on Math

BE: (missed the third point)

BE: CLZ - for cases when you're doing DSP-level hacking, and want to count the number of populated bits. Also important for native-to-js compilers, there is an intrinsic for this in GCC and Clang.
BE: Math.clz32 wins.

#### Conclusion/Resolution
Rename to Math.clz32 (rename and move from Number.prototype)

## Array.from

AWB: What should happen with `Array.from(undefined)`?

EA: `Array.from` is a likely replacement for `Array.prototype.slice.call` and the array generics do `ToObject`.

AWB: Array with holes would lead to to a dense array.

YK: Want to keep the holes.

BE: "Holes are from the devil"

#### Conclusion/Resolution

Keep as spec'ed.
  * `Array.from` will throw on `undefined` and `null`
  * `Array.from` will return `[]` for `3`
  * `Array.from([0, , 2])` => `[0, undefined, 2]`

## Test 262

BT: Up on GitHub. https://github.com/tc39/test262. Conversion from Mercurial done by Brandon Benvie.
BT: AttendedTest the web forward. Got a lot of PR (22).
BT: Pending contributor guidelines.
BT: Going to work on style guidelines since the PRs are inconsistent.

IS: Getting CLA working is highest prioroty.
BT: Do we need any test coverage before approving ES6?
BT: For ES7 we agreed that we need tests before final spec.
BT: Instead of using numbered section plan to use the HTML anchor names from the HTML version of the spec.

## Yield and its precendence
Code Samples: https://gist.github.com/lukehoban/8678463
LH: Presents code samples. All but "yield 1+2" seem strange. Any time you have a generator that pumps values in you want good expression semantics.
WH: Points to yield 1+2 alternatives in LH's code samples; one makes it mean (yield 1)+2, the other makes it mean yield(1+2). Strongly prefers the latter.
BE: Have to make a choice that yield should be high precedence or low precedence. yield x + y must be yield(x + y) without parens, the rest follows by the grammar.
LH: concede

:: conversation about whether implementing iterators is more common or taskjs-like scenarios are more common ::

BN: Status quo causes more errors but errors encourage parenthesization, and enables `yield 1 + 2` to yield the value 3

LH: For the async use case, important to have high precendence, but willing to concede that `yield` should remain low precenedence due to iterators use cases. Async use cases could have new "await" syntax with high precendence.

YK: task.js requires (yield a) + (yield b) paren style
BE: revenge of Lisp, you get used to it

#### Conclusion/Resolution
Status Quo

## More generators (Ben)

See issue here: https://github.com/tc39/agendas/pull/25

DH: Intention was for generators to be composable.
BN: Proposal: generator with yield* doesn't throw when receiving a value
DH: Breaks correspondance between yield* and for-of desugaring. Also... wrappers are not equivalent unless both wrapper and wrapped are newborn.
BN: Can be fixed... but is hairy. Presents updated wrapper code... Proposal is that we remove the type error for passing in a value to an unborn generator. Then, if the first yield we encounter is a yield *, pass in the value that was passed to the unborn generator.
:: Concerns with proposal (please fill?) ::
DH: Possibly other alternatives to fix this?
WH: There are two problems here:
 1. A function* can't capture the first value passed to the 'next' that invoked it. Instead, it currently requires it to be undefined.
 2. No way to pass an initial 'next' value to yield*. This means that manually doing the first yield followed by a yield* wouldn't help because the problem would then reappear on the second yield.
WH: The proposal is proposing a hidden channel (a la Perl's $_) that ties 1 to 2 here. Would prefer mechanisms to do those separately and more explicitly.
BN: yield` syntax that says delegate but don't care about value?
DH: Let's think about it more...
DH: Went through the mental excercise to do an implicit yield upon calling the generator. Leads to a lot of

other issues.

```js
function*() first {
  return yield* push(first, delegate);
}
```

#### Concensus/Resolution

BN: Have to go back and think more about this. Maybe a helper function can be created.

## ECMA-404 and IETF interactions

AWB: IETF is not comfortable referencing ECMA.
IS: IETF includes ECMA-404 as an informative reference (?)
AWB: Proposed doing a second edition if IETF wanted to work with ECMA.
:: Discussion about IETF participation, general dissatisfaction with the process and end result ::
See http://www.ietf.org/mail-archive/web/json/current/threads.html#02131 and look for "R S" (Rob Sayre) fighting the good fight

# **Jan 29 Meeting Notes**

John Neumann (JN), Allen Wirfs-Brock (AWB), Yehuda Katz (YK), Eric Ferraiuolo (EF), Erik Arvidsson (EA), Rick Hudson (RH), Matt Sweeney (MS), Dmitry Soshnikov (DS), Sebastian Markbage (SM), Ben Newman (BN), Jeff Morrison (JM), Reid Burke (RB), Waldemar Horwat (WH), Doug Crockford (DC), Mark Miller (MM), Brian Terlson (BT), Luke Hoban (LH), Andreas Rossberg (ARB), Istvan Sebestyen (IS), Niko Matsakis (NM), Brendan Eich (BE), Rick Waldron (RW), Sam Tobin-Hochstadt (STH), Rafael Weinstein (RWS), Dmitry Lomov (DL), Niko Matsakis (NM), Simon Kaegi (SK), Kevin Reid (KR)

## Function in Blocks in non strict mode.

(Ask Brian for slides)

BT: In ES6 draft in appendix B (since functions in block are not in ES5 but implemented)
BT: IE11 shipped with function in block in non strict mode.
BT: Hoists across let blocks
AWB: If you start using a let then you'll get an error. Since old code does not use `let`.
LH: If the code falls out of the known patterns you hit a rough edge and then it is better to get an error than undefined behavior.

WH: Concerned about expanding past the minimal crazy intersection semantics we agreed to in the past. The proposal expands the set of cases that produce surprise captures.
WH: [on whiteboard] This is not just about eliminating errors. It will silently change the meaning of working code:
```js
function foo() { ... 1 ... }
function bar() {
  if (...) {
    function foo() { ... 2 ... }
  }
  foo(); // Thought it would refer to the outer scope foo. Surprise!
}
```

~~~
```
~~~

~~~
```js
function foo() { ... 1 ... }
function bar() {
  if (...) {
    let foo =...;
    {
      function foo() { ... 2 ... }
    }
  }
  foo();
}
```
~~~

LH: Since 1.5% depend on the crazy behavior we have no real option.
WH: Let's limit the craziness to just the cases in the minimal intersection semantics.
BT: Allow hoisting past a let.
ARB: What was the motivation for this special case, i.e., why not drop all hoists that would normally be an error?
LH: Creates 2 bindings. The FunctionDeclaration itself mutates both the var binding and the let binding.

~~~
```js
(function() {
  {
    let foo;
    {
      function f() {}  // Why no error?
    }
    console.log(foo);  // undefined
  }
  console.log(foo); // funtion foo() {}
})();
```
~~~

DH: If you inject a let and it shadows a var it should be a static error.
LH: In strict mode it is legal so it cannot be an error in non strict.
DH: It is consistent with a var in a block.
WH, ARB: What happens if you have two functions
LH: You conceptually get two var bindings.
WH: That's not part of the intersection semantics. Shouldn't get any if you have two foo's in inner blocks.
YK: Writing new ES6 code in non strict should still work.
WH: It will silently do different and surprising things, as per examples above.
DH: If there is any intervening let binding do not hoist past the let.
LH: If introducing would introduce a static error. Do not introduce the var binding.
[Discussion on the behavior of the new var bindings, along with whether assignment to foo updates one or both.]
WH: When are the crazy introduced var bindings initialized? At entry to outer function bar (just like ES5 inner functions in the top-level of an outer function), or at the time the inner function function foo definition is executed?
Various: Set to undefined on entry to outer function. Set when inner function definition foo is executed.
Various: Executing function definition updates both the let and the var bindings. Assignment updates only the let binding.
WH: [points out (and likes) warning on cases "for which the above steps are performed" in Annex B of the current draft]
[Long discussion about warnings, SHOULD, and their audience]
LH: A SHOULD warning seems a reasonable addition to the proposal here.

#### Conclusion/resolution:
    * 2 bindings.
    * Hoist a var binding for the FunctionDeclaration, unless it would introduce a static error (ie. hoisting past a let will not cause an error but also not create the var binding).
    * Also create a let binding as per pure (strict mode) ES6 semantics. (Change from IE11 semantics). Within the scope of the let binding, assignment will only touch the let binding (ie. normal semantics).
    * The var binding is only initialized at dynamic evaluation of function declaration.
    * A SHOULD warning if there is a reference to any such var binding of a function

## Object.observe Status Report

RWS: Suggest moving OO to the second stage.
:: Approved
RWS: Wants to move OO to the third stage, which requires a spec text review.
YK: Has reviewed it already.
AWB: Might have time to review.
RWS: The plan is to ship OO in Chrome sometime around April.
AWB: Should not be a problem to advance without scrutizing the proposal. Willing to rubber stamp at this point. Since we are in a state where we are. (?)
YK: Once we are at stage 3 we are committed to not revisit.

RWS/YK to talk about task scheduling.

#### Conclusion/resolution

???

## Post ES6 process

AWB: First step is to post the process somewhere public
AWB: Second, need a place to track the progress
AWB: Thinking we should do this on GitHub. Project TC39/meta. Master table. Or maybe 2 tables?
WH: Are we abandoning the wiki?
Various: yes
WH: How do I view HTML instead of the HTML source on GitHub?
EA: We should also move the meeting notes from rwaldron/* to TC39/*
YK: We should use the GitHub API to extract the comments for keeping the paper-trail.
RWS: Agenda changes since last time
    1. Add a designated reviewer.
    2. At step 3 and 4 there is a requirement to get an approval from the spec editor.
RWS: Also managing spec in flight
YK: I'm going to work in that.

## Do Expression

(Ask for slides from ARB)

MM: An engine can statically detect an IIFE and treat it as a do expression.
DS: Important to scope var declarations to the enclosing function rather than the do block because that's what happens in a do-while loop, and it's hard to tell if you're looking at a do block or a do-while loop.

```js
```

```
do {
  ...
}

while(true);
```

WH, DH: Expression statements cannot start with `do` to avoid syntactic conflicts.
BE: Just like `function` for FunctionExression which cannot start ExpressionStatement.
DH: Just use a parentheses if you really need to start ExpressionStatement with `do`.
AWB: What step is this in the new process? Step 0?
ARB: Can write spec
DH: Primary reviewer.

#### Conclusion/resolution

   * do-expressions as generailsation of block statements
   * Progresses do-expressions to the next state in the new process.

## Security Review for Loaders/Realms

MM: We have considered a security review for Loaders & Realms
YK: Talked about this earlier, seemed positive
STH: Concerned about definition/scope
KR Gave vague definition
MM: We'd like to do the same things we did in ES5
KR Considering porting SES to ES6.
DH/STH: this is an excellent goal
MM: What's the implementation status of Loaders/Realms
DH: Unpolyfillable, YK + DH tried some things w/ iframes, didn't work
DH: That's part of why we think it's a fundamental new primitive

#### Conclusion/Resolution

- * MM et al to port SES to ES6, contact module champions w/ results

## Typed Objects

Dmitry Lomov + **Niko** Matsakis

NM: Not objects
WH: Likes that === compares identities, but would prefer that == compare values.
NM: THat would be dangerous since structs are mutable.
WH: In that case, how do you compare values? That's a really common thing that users will want to be able to do.
ARB: Accessing a sub struct of a struct does not allocate any object.
NM: You can stack allocate

```
Line = new StructType({from: Point, ...})
line = Line(...)
while ( ... ) {
  foo(line.from)
}
```

DH: If `line.from` heap allocates it would be suprising.
WH: How does stack allocation work if the reference to the allocated object escapes the scope?

WH: Important to be able to mark StructTypes to always generate opaque fat pointers even if they don't have any 'any' field.
WH: This helps with optimization. Knowing that all instances of a StructType are opaque lets NaN-boxing implementations optimize out protection against NaN-injection on reads.

## Value Objects

http://www.slideshare.net/BrendanEich/value-objects2

# Jan 30 Meeting Notes

John Neumann (JN), Allen Wirfs-Brock (AWB), Yehuda Katz (YK), Eric Ferraiuolo (EF), Erik Arvidsson (EA), Rick Hudson (RH), Matt Sweeney (MS), Dmitry Soshnikov (DS), Sebastian Markbage (SM), Ben Newman (BN), Jeff Morrison (JM), Reid Burke (RB), Waldemar Horwat (WH), Doug Crockford (DC), Mark Miller (MM), Brian Terlson (BT), Luke Hoban (LH), Andreas Rossberg (ARB), Istvan Sebestyen (IS), Niko Matsakis (NM), Brendan Eich (BE), Rick Waldron (RW), Sam Tobin-Hochstadt (STH), Rafael Weinstein (RWS), Dmitry Lomov (DL), Niko Matsakis (NM), Simon Kaegi (SK)

## Parallel JavaScript

RH: API is stable
RH: Parallel array data type is no longer there. Instead as methods on arrays and typed objects.
NM: To be clear you can have an array with structs in it.
RH: The sweet spot is games and image processing.
RH: No current show stoppers on implementation. Some things to work out related to GC.
The strategy going forward. Be complimentary to the typed object spec. Will track it and spec parts of it Typed Object spec. "Close follower". Will move in tandem.

LH: To move to phase 1 we need to see some examples.
YK: Agree, we need to see where we're at.
AWB: Agree, we should have presentations to move from phase 0 to phase 1.
RH: I have presented twice already...
YK: Moving to phase 1 should require a presentation.
YK: Concerned that we are exploding the API with mapPar, filterPar, fooPar etc.
YK: Prefer static functions
EA: Or a standard module `import {map} from '@parallel'`
YK: `this` in functions?

```js
arr.map(obj.method, obj)
```

DH: The signature should just match the existing functions.
RH: Not less surprising. Just as surprising.

```
parallelModule.map(arr, func, ...)
// [T] -> .... -> [T]  // same return type
```

NM: What would you call `from`?

WH: It's too confusing to require completely different static function style for invoking parallel maps as compared to non-parallel maps. mapPar etc. method style is better than the proposed alternatives.
DH: It is always nicer to write a method call than a function call.
DH: Don't want a "bring me a rock" exercise.
EA: File issues on GitHub on the drafts (that are also hosted on GitHub).
MM: Worked well for Promises.
YK/MN to talk through the concern about a "ton of methods".

#### Consensus/resolution:
  * Move Parallel JavaScript to phase 1
  * Talk offline about design issues further

## Structured Clone

DL: Is implemented in all browsers. Part of HTML spec. Hixie speced it. Hixie is happy with TC39 moving this to ES.
YK: Like to object to this motion. It is currently a giant set of scenario hacks.
DL: We want to add language objects that we want to transfer.
AWB: Cloning framework in ES.
DH: Is it possible to reform or do we have to start from scratch? Seems hard to reform. Too many issues.
MM: Fears that if we do not take it over and introduce something new. The old will continue to exist. We need a path to replace the existing system, including what PostMessage does.
DH: We need a roadmap. How do we handle transferables?
DH: Hixie (or Anne) added Map and Set to structured clone to HTML spec.
DL: We cannot add extensibility mechanisms if we do not own the spec.
YK: We should own the spec. Opposed to DOM specific extensibility methods. General extensibility mechanism are important.
BE: How would symbols work?
AWB: We have a symbol registry. As long as both side cooperate. Serialize to a registration key. The two sides need to agree.
MM: It is unobservable that the symbol is not the same across the two workers.
WH: What if you have the same symbol registered under two different strings?
MM: That can't be allowed.
BE: in the cross-machine limit, structured clone is serlialization/deserialization; start there, allow optimizations like Sun XDR, Van Jacobsen TCP/IP, Google protocolbuffers
[discussion on optimization]
WH: What do we mean by optimization?
DH: [explains difference between opaque and structured clone, including implications on optimization]
WH: Are we going to settle on one of the two or do both?
DH: Both
[more discussion on optimization]
BE: [explains history of prior work]
BE: Can't tell Hixie and Anne to stop adding to structured clone.
Anne: Want to give them assurance that we will take over the effort.
WH: What's the consensus?
YK: We'll take it on. Move to stage 0.x

## defineGetter, defineSetter etc in Annex B?

BT: IE ships this.
MM: It would just be speced using defineProperty etc
AWB: Firefox does some strange things.
BE: please enumerate "strange things", file bugs
YK: It starts at level 1. Or 3? there are already implementations.
RW: What sites?

BT: Will attempt to furnish a list of sites...

#### Consensus/Resolution:
  * Makes sense to put in ES7 annex B
  * Brian to write an initial speec draft

## Process document

Process doc is now public
https://docs.google.com/a/chromium.org/document/d/1QbEE0BsO4lvl7NFTn5WXWeiEIBfaVUF7Dk0hpPpPDzU

## Scheduling for next meeting

April 8-10 at Mozilla, San Francisco
May 20-22 at Facebook, Menlo Park
July 29-32 at Microsoft, Redmond
Sept 23-25 at Bocoup, Boston
Nov 18-20 at PayPal, San Jose

## Async/await

LH: https://github.com/lukehoban/ecmascript-asyncawait
LH, MM: await syntax is important because the precedence of await needs to be different than yield.
LH: async functions could be combined with function*; in such a thing we'd need both yield and await
MM, WH: What would the behavior of such a thing be?
LH: That's a seperate proposal - something we can discuss later.
BE: Syntax conflict with => functions (elided parameters). what does:
  async() // newline, no semi here
  => {...} ...
mean? We can make it be an async arrow if we want, but second line looks like 0-param arrow function expression....
WH: async (a, b, c) => await d looks too much like a function call of a function named 'async'. Need to parse a long way before figuring out it's an async lambda. This wouldn't fall under the existing cover grammar.
LH: I'll look into that.
DH: Initially concerned about hard-coding the scheduler.
LH: Identical to Q.async. There is only one way to do this.

MM: September Promises consensus is superior to the Promises spec we have now.
[Debate about whether we'll end up with two Promises APIs]
WH: Do the two APIs use the same scheduler (that would be hardcoded)?
DH: No.

LH: Can we move async/await to stage 1?
General agreement
AWB: This means that we agree that this is something in a future version of ES

Conclusion/resolution:
  * Moved async/await to stage 1.
  * Next step is to write real spec language

## Promises discussion

MM: Advocacy for .then()/.cast()
STH: Advocacy for .chain()
LH: Proposal of Promise.chain() compromise
YK: I would probably be ok with this
MM: I would probably be ok with this
... extensive discussion ...
MM: A resolve of a resolve does not create multiple levels of wrapping. Without chain this is not observable.
BE: .chain() requires over-wrapping/-unwrapping, without chain this is unobservable and therefore optimizable -- says to reject chain (surprised by own position changing)
YK: This persuades me that we shouldn't have .chain()
STH: I strongly disagree, but I'm not going to hold up ES6 over this

Conslusion/resolution:
* Promise.cast is renamed to Promise.resolve (remove old Promise.resolve)
* Keep then, reject chain (NOT DEFER, reject!)
* Renaming .cast thus removes over-wrapping (always-wrap) deoptimization in old Promise.resolve

Some further discussion on keeping the spec inheritance/AOP-friendly by not using .then internally some times, short-cutting through internal methods or internal helpers other times
YK, MM, AWB have details