Ecma/TC39/2012/068
Ecma/GA/2012/087
(Rev. 1 – 1 October 2012)

Minutes for the:	***30th meeting of Ecma TC39***
in:	***Boston, MA, USA***
on:	***18-20 September 2012***

1 Opening, welcome and roll call

1.1 Opening of the meeting (Mr. Neumann)

Mr. Neumann has welcomed the delegates.

Companies in attendance:

Mozilla, Google, Microsoft, eBay, jQuery, VuB, Yahoo, Northeastern University, Intel

1.2 Introduction of attendees

Istvan Sebestyen - Ecma International

John Neumann (MS, Mozilla, Yahoo, Google)

Norbert Lindenberg - Mozilla

Nebojia Ciric - Google

Allen Wirfs-Brock - Mozilla

Luke Hoban - Microsoft

Paul Leathers - Microsoft

Sam Tobin-Hochstadt - Northeastern

Andreas Rossberg - Google

Erik Arvidsson - Google

Dave Herman - Mozilla

Yehuda Katz - jQuery

Rick Waldron - jQuery (took minutes)

Eric Ferraiuolo - Yahoo

Matt Sweeney - Yahoo

Douglas Crockford - EBay

Rick Hudson - Intel

Mark Miller - Google

Brendan Eich - Mozilla

Alex Russell - Google

Rafael Weinstein - Google

Waldemar Horwat - Google

Tom Van Cutsem - VUB (via phone part time)

1.3 Host facilities, local logistics

On behalf of Microsoft **Sam Tobin-Hochstadt** welcomed the delegates.

2 Adoption of the agenda (2012/061-Rev1)

The draft agenda was agreed for the meeting.

3 Approval of minutes from July 2012 (2012/056)

The minutes of the July 2012 TC39 meeting have been approved as presented. Individuals that took technical notes were recognized and appreciation extended. **Rick Waldron** volunteered to take technical notes. The technical notes are included in Annex 1 of the minutes.

4 Discussion of ES harmony (technical contributions are available and can be found on the ES wiki)

4.1 Review of list of Project Proposals for inclusion in ES 6

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

4.2 Data parallelism (River Trail) strawman changes and challenges

4.3 The Define Properties Operator:

http://wiki.ecmascript.org/doku.php?id=strawman:define_properties_operator

Object.define, Object.put, Object.merge

https://mail.mozilla.org/pipermail/es-discuss/2012-August/024402.html

https://mail.mozilla.org/pipermail/es-discuss/2012-August/024571.html

4.3.1 Discussion about scoping of the top-level

http://wiki.ecmascript.org/doku.php?id=strawman:syntactic_support_for_private_names

4.4 Array.of => Array.new

Defining Foo.new for all built-ins

https://mail.mozilla.org/pipermail/es-discuss/2012-August/024688.html

4.5 Thin arrows. Is -> still an arrow too far? See

https://github.com/JSFixed/JSFixed/issues/42

4.6 The existential operator

http://wiki.ecmascript.org/doku.php?id=strawman:existential_operator

4.7 Object.observe (Wed or Thurs)

4.8 Global scope: Recent concerns about global declarations shadowing WebIDL globals in both ES5.1 and 6:

https://mail.mozilla.org/pipermail/es-discuss/2012-August/024481.html

https://bugs.ecmascript.org/show_bug.cgi?id=78

4.8.1 Open issues and loose ends regarding super references

4.9 Proxies (Tom call in Wednesday10AM)

4.9.1 Interaction between proxies & private names:

http://wiki.ecmascript.org/doku.php?id=strawman:proxies_names

4.9.2 Revokable proxies:

http://wiki.ecmascript.org/doku.php?id=strawman:revokable_proxies

4.10 private/unique names (After 4.12)

http://wiki.ecmascript.org/doku.php?id=strawman:syntactic_support_for_private_names

4.11 Renaming Name to Symbol (After 4.12)

https://mail.mozilla.org/pipermail/es-discuss/2012-August/024278.html

4.12 Syntactic support for private names: (After 4.12)

4.13 Grammar Validation

4.14 Formal Parameter Scope

4.15 Overhead of temporal dead zones

4.16 Early errors interfere with compilation with first use

5 Edition 5.1 Issues

6 Review of final draft of ECMAScript Internationalization API Specification

• Report from the ad hoc on Internationalization standard: Specification updates, prototype development, conformance test suite

• Review of the specification

• Approval of final draft of ECMAScript Internationalization API Specification for submission to CC and GA

Materials:

- Final draft of ECMAScript Internationalization API Specification (to appear Tuesday 9/4)

- Working draft of technical report ECMA-402 Test Suite (sent out today)

- Proposal for changes to test262 web site and bug tracking:

https://mail.mozilla.org/pipermail/test262-discuss/2012-August/000106.html

And then follows the discussion about the second edition that Nebojša suggested.

6.1 TC39 review and last feedback prior to preparation of the final draft

- Final draft of ECMAScript Internationalization API Specification (likely Tuesday 9/4)

6.2 Multi-system prototype testing

- Working draft of TR ECMA-402 Test Suite (likely this week)

Google and Microsoft have been adding internationalization tests to the 262 test-suite.

6.3 Approval of final draft to CC and GA for December approval

6.4 Discussion about v2.0 and moving forward

7 Test 262 Progression

How to organize test262 site and development to support ES6 development and Ecma--402 (Internationalization API) tests.

https://mail.mozilla.org/pipermail/test262-discuss/2012-August/000106.html

7.1 Addition for Internationalization

https://mail.mozilla.org/pipermail/test262-discuss/2012-August/000106.html

7.2 Addition for ES-6

7.3 Prototype Website (http://test262.ecmascript.org and http://test.w3.org/html/tests/reporting/report.htm)

8 Status Reports

8.1 Report from Geneva

8.1.1 Brief report from the IPR meeting

9 Date and place of the next meeting(s)

- November 27 - 29, 2012 at Bay Area (PayPal)
- Schedule of the 2013 meetings:
- January 29 – 31, 2013 (Mozilla)
- March 12 – 14, 2013 (Yahoo)
- May 21 – 23, 2013 (Google)
- July 23 – 25, 2013 (Redmond)
- September 24 – 26, 2013 (Boston)
- November 19 – 21, 2013 (PayPal)

10 Closure

Mr. Neumann thanked **Northeaster University** for hosting the meeting, the TC39 participants their hard work, and **Ecma International** for holding the social event.

Annex 1

Technical Notes:

# September 18 2012 Meeting Notes

Rick Hudson (RH), John Neumann (JN), Mark Miller (MM), Norbert Lindenberg (NL), Nebojsa Ciric (NC), Allen Wirfs-Brock (AWB), Istvan Sebestyen (IS), Luke Hoban (LH), Paul Leathers (PB), Sam Tobin-Hochstadt (STH), Andreas Rossberg (ARB), Brendan Eich (BE), Erik Arvidsson (EA), Dave Herman (DH), Yehuda Katz (YK), Rick Waldron (RW), Eric Ferraiuolo (EF), Matt Sweeney (MS), Doug Crockford (DC)

Introductions. Brief Agenda tweaks

RW: Confirm that Internationalization spec is available, per last meeting

NL: Internationalization specs are available on the Wiki

AWB: Start with Internationalization

IS: We need two products: The spec and a technical report (via test suite)

JN: Would like Internationalization tests to be included with test262

AWB: Finalize by June

# Internationalization

(Presented by Norbert Lindenberg, Mozilla)

[Slides](https://members.ecma-international.org/get.php?group=TC39&file=2012_sub_tc39-2012-065.pdf)

## Review and Approve Final Draft of Internationalization API Specification

(Introduction)

NL: Final draft following major discussion in May and tweaks from July.

Primary change is to **Require Normalization By Default**

DH: Is this a performance issue? Should we leave it to the lib authors

AWB: Why do we assume lib authors will write wrappers.

YK: Because we have no evidence to the contrary

LH: That implies a judgement about design...

RW: Spec APIs are not _bad_, they often require simplification for wide acceptance and use.

Discussion returns to whether or not Normalization is _required_ by the specification or if it is implementation independant.

Agreement that generally there should be no "optional" implementation details. Conformance should be explicitly normative.

NL: Note about ES5 assumptions that a string has been normalized before parsed.

NL: Three options to support or ignore:

1. Normalization
2. Lowercase to Uppercase first
3. Numeric sorting

## Don't Use Year 0 in Date/Time Formatting

2 AD, 1 AD, 1 BC, 2 BC

Discussion and explanation

YK: Ruby has a year 0

AWB: Fundamentally Date/Time objects are ms mapped to some external designator, would there be any impact?

Decision: Date calculations will remain as is; but dates before 1AD will be adjusted in localized formatting (Date.prototype.toLocaleString, Intl.DateTimeFormat.prototype.format) to reflect no year 0.

## Conformance Tests

NL: 137 tests, reaches almost all algorithms, coverage is still thin, draft test report

LH: These have been effective in identifying bugs in Chakra prototype implementation

## Implementor's Report

(Google)

NC: Chrome/v8 has `Intl` namespace and the implementation is working towards passing the conformance tests with ~20 failures. Some tests not yet implemented. Google internal use

AWB: What is the feedback?

NC: Mostly migration based discussion for the time being. There was a previously a [Localization] API.

AWB: Issues with name conflict? Wouldn't know if it was prefixed.

NC: Will be removing the prefix

DH/YK/STH: Discussion about globals, as they apply to the future with modules. import foo from … will reduce naming conflict overall.

(Microsoft)

LH: Currently passing 100/137 of conformance tests. Dont have direct user feedback. No one is actively using the prototype.

RW: Is there any communication between implementors?

NC/LH: Only via es-discuss

(Mozilla)

NL: Prototyped in SpiderMonkey. Uses ICU for comparison, formatting and feature detection. JS/C++ for implementation, Unicode extensions not yet supported. Passing 128/137.

## Approvals

NL: Final change, move to year 0.

AWB: Need to address the "optional" spec issues.

DH: Not that leaving things unspec is evil, we should just be conservative.

RW: For the sake of clarity there should be a specific list,(via notes?) of "optional" implementation details.

DH: Agree, similar to the history of strict mode list

LH: This can be produced off-line

AWB: In the form of an Annex

DH/LH/AWB: (Discussion about implementation limitations)

Summary

Produce Annex that outlines a list of all optional implementation details. Include a rationale for each item that describes strong reasoning for optional implementations.

## Annex

Optional

All Functionality

- Supported Locales
- Default Locale
- Supported subset of unicode
- Best fit matcher for locales
- Supported Unicode Extenion values per locale
- Additional values per conformance clause

Collation

- Adherence to unicode collaiton algorithm
- Support for unicode extenions keys kf, kk, kn and parallel options caseFirst, normalization, numeric
- localized sort orders
- Default search sensitivity per locale

NumberFormat

- Support for numbering systems
- Implementation of non-decimal numbering systems
- Localized decimal & grouping separators, representation of negative numbers, percent sign
- Localization grouping
- Localized concurrency symbols and names

DateTimeFormat

- Supported data/time formats per locale beyond core set
- Best fit matcher for formats
- Supported Calendars
- Support for numbering systems
- Localized format patterns, weekday names, month names, era names, am/pm, time zone names

BE: Similar to the underspecified portions of Date

LH: Of course, we'll work together to be as consistent as possible.

## Approval of Intl

JN: If there are no objections, we will forward this document specification, ECMA-402 to the CC & General Assembly for final approval.

…No objection.

JN: With the final modifcations, this document will be submitted to the CC & GA for final approval. NL and NC to produce a list for an Annex of optional details.

JN: Any desire for ISO fast tracking?

(Discussion re: ISO benefits.)

Conclusion/Resolution

ECMA-402 Approved for submission to ECMA CC & GA

ISO fasttrack postoned (with the limited time frame of 2 months notice prior to presentation the GA, approx Oct 10, 12?)

## Intl 2nd Edition

NL: There is a need to continue work, towards a 2nd edition

JN: Agenda item for Nov.

IS: Need to determine scope and scale of needed changes.

Conclusion/Resolution

Agenda item for November 2012 to entertain a proposal.

# Parallel JavaScript

(Presented by Rick Hudson, Intel)

## River Trail (Intel)

\[Slides\](https://members.ecma-international.org/get.php?group=TC39&file=2012_sub_tc39-2012-064.ppt)

Map

- myArray.map(callback)

- myArray.map(depth, callback) // for an n-dimensional array

 - elementalFunction (element, index, source)

 - (need slides?)

DH: Not sure that the level of technical detail is yet appropriate (from the perspective of an implementor)

YK: Gratuitous API changes should be avoid

RH: Intentionally avoided using the |thisArg|, think it's complete unnecessary and exists for legacy purpose.

DH: Absolutely not the case and is very important.

RW: For example, when you have a constructor that has properties [[Put]] via map, |thisArg| allows setting the context within the callback to the constructor itself.

DH: This needs to be taken offline, away from the committee.

LH:

Examples of Map

\`\`\`js

paArray.map(function(element) {

  return element+1;

```
});

paArray.map(2, function(element) {
  return element+1;
});

paArray.map(2, function(element, [i, j], array) {
  return element+1;
});
```

LH/DH: (Discussion of explanation of River Trail semantics and the use cases)

LH: Is it the claim of this proposal to follow the ECMA 262 Semantics.

DH: Yes, up to the issue of floating point non-determinism

LH: Which means an engine cannot detect… (lost)

RH: We do rely on the programmer to know that they need to write an associative and commutative function. Tools can be provided to help.

LH: Worried about implicitly saying that a function does not match what ECMA 262 says it will mean.

DH: (Clarifies that it's _just_ JavaScript)

BE: Parallelization can be painfully slow

YK: If it's straight forward, then why not specify how Parallelization is accomplished

DH/BE: Too early to attempt to specify Parallelization detection.

Lengthy discussion of Parallelization "mode" switching semantics… Devolved. Ended abruptly when no progress was made.

Shape

- Mixing 1D and 2D operations requires an understanding of shape

- Shape determined at construction

Identity

- Accessors to non leaf elements of ParallelArrays will return a fresh ParallelArray
- References semantics for === remains consistent

Between the two:
- pa[2] === pa[2] true for only 1D ParallelArrays
- pa[2] === pa[2] always false when shape is > 1

LH: Asks for explanation…

DH: Will fill in blanks offline

AWB: Are there any other efforts that are developing competing specifications?

RH: Not exactly, but WebCL(Kronos?) is similar in the application they are trying to address.

DH: A different name?

BE: Vector?

DH: Also, the world will hate us for creating a new Array-like.

RW: Only a problem when creating constructors that produce objects with numeric indices and a length property and no Array proto API.

DH: Which it does.

RW: Then it will be a problem.

MM: The proliferation of Array-like things is unfortunate

RW: And ES6 reduces that pain by effectively eliminating arguments via rest and Array.from()
…

DH: There was also the idea of having parallel-specific methods.

Derailed to Array-like API issues on the whole… When to implement array API and when not. Why and why not…

Conclusion/Resolution

Further research and offline discussion.

# Define Properties Operator ":="

(Presented by Allen Wirfs-Brock, Mozilla)

Introduction, rationale as published:

http://wiki.ecmascript.org/doku.php?id=strawman:define_properties_operator

DH/AWB/RW: (discussion) Object.define, Object.put

RW: Have research and supporting cases from jQuery, Dojo, YUI, Node core, Underscore… Always exists an approximation of "merging" or "extending"

AWB: We should strive to fix the future and correct developers thinking about "define" and "assign"

LH: That's a dangerous scenario to put developers in, where they have to think about assignment vs definition.

RW: The newly created dom node case, for batch property assignment (originally brought forth by Doug) is the second most important use-case, but implicit define will pave innerHTML (or any dom node properties) …

DH: Shouldn't create syntax for the less common operation.

MM: Agreed.

AWB: But there is no way to:

- Batch define class-side properties
- Batch define constructor properties
- Batch define instance properties

YK: Nothing for static properties in classes yet anyway

MM: But not sure we need any syntax yet. If there was a lot of precedent for batch define, in the same way there is for assign/put, then it would make sense, but there is very little userland history for _define_

RW: Agreed, assign/put is a cow-highway to pave, but user code has barely begun to include regular use of definePropert(y|ies)

MM: (comments about private name access concerns)

AWB:

LH: Long term, we're going to have to consider features that are allowed to move private state.

DH: Essentially, you'd need inside access and list private items.

Discussion about side channel access via newly defined properties that were never expected on the object.

Discussion about the needs of Private Names, Unique Names and WeakMaps.

BE: People want Private Names as much as they want Unique Names

YK: Can we tell people to use Unique Name when they want copyable and Private Name when not.

BE: I thought of this earlier, but wasn't sure, but it could work

AWB: …

MM: Given ES6, remove the copying of private names, allow copying unique names: Can this be written as library code?

DH/YK/RW: Devs want Object.define which is Object.defineProperties

Object.assign() or Object.put() (these are the same, just different names)

Extensive discussion around whether or not Object.define()

Extensive discussion around whether or not Object.assign()

Derailed due to concise method's making non-enumerables, which means they won't copy if the rule disallows copying.

…Revisit "Concise Method Definition" (add anchor)

Object.define(target, source)

- All own properties of source
- plain object descriptor map is copied
- private names are not copied
- unique names are copied
- super mechanism (rebind super)

Object.assign(target, source)

- Only enumerable own properties of source
- Invoke [[Get]] on property list derived from source, for each property in list [[Put]] on target
- private names are not copied
- unique names are copied
- super mechanism (rebind super)… AWB To determine needs
- Returns modified "target"

DH, MM, AWB: Object.assign a well worn enough cow-path to be worth paving. Object.define isn't, and so should only be standardized after libraries have explored the space.

Conclusion/Resolution

Accept Object.assign into ES6, but postpone Object.define (or something like it) to discussion of future versions.

Reference materials and use cases: https://gist.github.com/3744794

(** The inclusion of variable length sources is imperative to match real world patterns found in the most ubiquitous JS libraries)

# Concise Method Definition, Revisited

RW: Defaulting concise methods to non-enumerable is a mistake

DH: Not sure about the decision to go non-enumerable. Users expect that things they create to be enumerable and things that the platform provides to be non-enumerable.

LH: enumerability is not a real concept with any sort of meaning.

EA: (reveals the broken-ness of the DOM)

No longer arguable.

Conclusion/Resolution

Concise method definitions create [[Enumerable]]: true

# Scoping of the Top Level

## var and the window.prototype issue

var indexedDB = window.msIndexedDB || window.mozIndexedDB || window.webkitIndexedDB || window.indexedDB;

Issue with WebIDL change.

REVISIT.

## let, const, module, class

Global scope contours

AWB: propose extra global contour, shared across all scripts, for new binding forms, to avoid colliding with Window object

others skeptical of complexity of new scoping model for globals

Conclusion/Resolution

continued on second day, resolved then

# September 19 2012 Meeting Notes

John Neumann (JN), Mark Miller (MM), Norbert Lindenberg (NL), Nebojsa Ciric (NC), Allen Wirfs-Brock (AWB), Istvan Sebestyen (IS), Luke Hoban (LH), Paul Leathers (PB), Sam Tobin-Hochstadt (STH), Andreas Rossberg (ARB), Brendan Eich (BE), Erik Arvidsson (EA), Dave Herman (DH), Yehuda Katz (YK), Rick Waldron (RW), Eric Ferraiuolo (EF), Matt Sweeney (MS), Doug Crockford (DC), Alex Russell (AR), Rafael Weinstein (RWS), Waldemar Horwat (WH), Tom Van Cutsem (TVC, phone)

Recap of Sept 18th notes.

## Proxy

(Presented by Tom Van Cutsem, Free University of Brussels)

### Enumerate Traps Return Types

TVC: change return type of enumerate() trap from array-of-string to iterator. Requires waiving the check for duplicate property names

Conclusion/Resolution

Drop the duplicate property check. enumerate() trap returns iterator.

### Revokable Proxies

TVC: The Problem

In a nutshell: with direct proxies, one can no longer write a caretaker proxy that nulls out a pointer to its target once revoked. This means that the target can no longer be garbage-collected separately from its proxy.

As pointed out by David Bruant, caretakers are often useful precisely for memory management, where one uses revocation of the caretaker to release the underlying resources, as described here.

Such caretakers were easy to define in the old Proxy design, since the link between a proxy and its target was fully under the control of the handler (i.e. fully virtual). However, with direct proxies, the proxy has a built-in, immutable reference directly to its target. This reference can’t be modified or nulled out.

Proposed Solution

Provide an extension to the Proxy API that allows scripts to null out the proxy-target reference. Of course, the API must be designed with care so that this link can’t be nulled out by any old client of the proxy. Only the creator of the proxy should have the right to revoke it.

Because revokable proxies appear to be a niche abstraction, we propose to introduce a separate Proxy constructor dedicated to creating revokable proxies. Proxies created with the regular Proxy constructor would remain unrevokable.

Proposal:

- Introduce a new constructor function RevokableProxy, next to Proxy.
- RevokableProxy(target, handler) returns an object {proxy: proxy, revoke: revoke}.
- revoke is a zero-argument function that, when called, revokes its associated proxy.
- A revoked proxy becomes unusable in the sense that any operation that would trap to the handler instead throws a TypeError.
- Revoking an already revoked proxy has no effect.
- Once a proxy is revoked, it remains forever revoked.
- A revoked proxy drops its target and handler, making both available for garbage collection.

MM: We missed this in the change to Direct Proxies, thanks to David Bruant for identifying

…Bikeshedding the spelling of "Revokable"

MM, WH: Should be "revocable"

?: "Revokable" would seem like a typo to many folks, as it's an unusual spelling of the word.

LH: This is bringing another constructor…? Do we want to duplicate all of the functionality?

RW/DH: Agree this is an issue.

MM: static on the Proxy object?

Yes.

… Discussion about the naming.

Proposal: Proxy.revocable

DH: perhaps too academic? Consider alternative names: nullable, ...?

MM: We can decide later.

Should Proxy.revokable return the tuple as an array or an object?

MM: object: non-positional + Javascript has sufficiently lightweight notation for objects.

STH: then the name "revoke" is part of the API

Question as to whether we really need two kinds of proxies.

BE: yes, non-revokable proxies have less trap overhead (no null-check)

Discussion about whether revokable proxies introduce new ways for interceptable operations to behave.

WH: Not sure about this as a feature, w/r to future hostility…

- eg. if "===" would trap to the handler how does this work with that?

MM: trapping "===" would be a significant change on its own, independent of revokable proxies

TVC: The only type test that is affected is typeof: once the proxy is revoked, it drops references to its target and its handler, so it can no longer forward the typeof test to its target

BE: It remembers "function" or "object"

TVC: right

DH:

RWS: Does this problem exist…

STH: revokable proxies don't add any new semantics b/c you can already write a direct proxy that throws the same error on every operation (it just uses more memory)

WH: Direct proxy, not allowed to muck with

MM: Are

TVC: Equivalent to a handler that implements all its traps to unconditionally throw

STH: RevocableProxy adds no new semantics to the language. except for allowing the proxy to be nullable

WH: I'm not sure about it

DH: He jsut gave proof.

MM: The semantics change to the handlers always throwing on all traps.

WH: Are there other traps affected by this?

TVC: New traps added last meeting will not be affected.

WH: So, a frozen object can "unfreeze" itself?

MM: No

WH: An object that is frozen can later refuse that its frozen

Concern about trapping isFrozen etc.: these tests are no longer stable.

MM: but are still fail-stop. The integrity guarantee (i.e. that it always returns a correct answer) is more important than the availability guarantee (i.e. that it always returns an answer)

Alternative to trapping isFrozen etc.: cache stable outcome of certain operations in the proxy, and afterwards no longer trap.

STH: After the first time isfrozen is true, mark to no longer call.

AWB: would preclude valid use cases that e.g. want to log all requested isFrozen operations.

Is there a situation where revoking allows an operation to throw where it previously couldn't with non-revocable proxies?

TVC: should not be the case

MM: Specify that…

The revoke action is observably equivalent to changing the state of the handler to always throw on all traps.

WH: An uneasy feeling about what else is hiding

BE: Mark and Tom have significant work on this issue.

MM: The ability to evict a subgraph is important

once frozen, stop trapping. once non-configurable, non-writable data, the value continues to be accessible. this loses the garbage collectability of membranes

…

DH: Question of clarification re: two constructors, are they necessary because there is no way to return two things from a constructor.

BE/MM: They create things that are just too different

BE: There is no mutable state within the proxy that is actually "mutated"?

TVC: not currently. If we want to pursue the above idea where a trap is disabled once its stable outcome is observed, then the proxy would need to be mutated.

MM: If you could falsly claim to be not-frozen after being frozen, then there would be issues with reliability, but doesn't exist.
…

MM/AR: (discussion about execution points in ES5)

AR: If you have a file name object, hand it into some API, it might throw, it might not throw… how is this different that what happened before?

WH: Again, still not sure about effects throughout proxy

MM/AR: (discussion about frozen object stability)

AR: Why do we care about the stability of frozen object w/r to revocability?

WH: (summary) Not clear what the security consequences of revoking a frozen object, either via a revocable proxy or via traps, are; this hasn't been thought about.

MM: This has been thought about and discussed off-line.

Conclusion/Resolution

- we want revocable proxies

- further discussion is needed on revoking frozen objects, either via revocable proxies or via traps

## Proxy and Private Names

TVC: We don't want Proxy to inadvertently leak private name properties.

In Redmond, we discussed that we would seperate string properties and private name properties into separate traps.

Later determined that this would become cumbersome.

Proposing to add a third argument to the Proxy constructor: a "whitelist" of private name properties that are allowed to be trapped.

WH: unique vs private names?

STH: The primary purpose of Names is to avoid name clash and non-forgeable. Uniques should be reflected, private: not.

WH: What makes them non-forgeable

STH: They are objects

DH: And are as non-forgeable as objects,

WH: Why do we need both? Unique, Private?

AWB: They are both useful

STH: Unique names give you actual unforgeability

MM: as opposed to the "unguessability" of randomly chosen strings

DH: About the whitelist, instead of _requiring_ a WeakSet, can it be anything that can be passed to a WeakSet, like an Array?

MM: Should probably use a WeakMap…

EA: An object that has a method that takes the public name string and returns the private name object, if you have access to that, you can extract the private data.

STH: Erik is right, but it a

MM: if we don't drop the .public property, don't need the whitelist but instead just the resolvePrivateName trap. If we stick with the whitelist, don't pass the .public property to the resolvePrivateName trap

TVC: Why is it a WeakSet? Because we dont want someone to provide their own collection that can extract the private data.

TVC: Why is there a resolvePrivateName trap? Say an operation is performed on a proxy involving a private name, and the proxy doesn't know this private name. Two reasonable options: 1) forward to target, i dont see results, I dont care. OR 2) throw exception. A policy decision to be made by the handler: when working on a private name that we dont know about, forward or throw?

DH: Should champions take this offline?

STH: It's pretty important and could mean a significant simplification.

WH: Would like to get rid of the public/private property flags

DH: for notational convenience, would be great if one could pass an array literal as 3rd arg

TVC: could specify that if 3rd arg is an array-like, we copy its elements into a built-in WeakSet

WH: that would be confusing: names later added to the array-like won't get added to the internal WeakSet

AWB: Not clear how using a built-in WeakSet will protect, what if it's been redefined? Are WeakSet methods non-writable?

TVC: we need to specify that the proxy calls the original/intrinsic WeakSet.prototype.get method.

STH/LH/RW: Will need to spec WeakSet

Conclusion/Resolution

Yes to third arg for whitelist, pending details to be worked out by Proxy champions.

Expect to remove the "public" part of private names.

## WeakSet

DH: Clear that we need WeakSet to match WeakMap (Set and Map)

AWB: Need to assure that WeakMap and WeakSet are not redefined?

RW: Use cases in node programs where I'm using WeakMap made it apparent that it _is_ possible to leak via redefinition.

Conclusion/Resolution

Needs to be designed, as part of the whitelist feature's needs.

SpiderMonkey tracking bug: https://bugzilla.mozilla.org/show_bug.cgi?id=792439

# Syntactic Support for Private Names

(Presented by Allen Wirfs-Brock, Mozilla)

Slides: https://members.ecma-international.org/get.php?group=TC39&file=2012_sub_tc39-2012-066.pdf

## Private Names & @Names

### Unique/Private Names

- Unique and private names (aka symbols) are ES6's solution for objects that need to expose props that have limited or controlled acccessibility.

- Currently no syntactic support for definition of use

- Imperative code patterns for using names dont mesh well with declaratinve object/class definitional forms.

eg.

```js
const secret = Name();
let o = { secret: 42 }; // does not define a "Name" prop

class MyClass extends Yours {
  secret() {
    this.mine();
    super[secret]();
  }
}

// Not allowed:
MyClass.prototype[secret] = function() {
  super[secret]();
};
//... Because super is banned outside of class
```

### Computed Property Names for Object Literals Were Abandoned…

```js
const secret = 42;
let o = { [secret]: 42 };

class MyClass extends Yours {
  [secret]() {
    this.mine();
    super[secret]();
  }
}
```

Issues:
- Allowed arbitrary expr in prop name def position
- Allowed aliasing of string valued prop keys
- Permitted same key to duplicate
- Future hostile: ties prop def to indexing (See: object model reformation)

### Proposal At-Names

[Slides](https://members.ecma-international.org/get.php?group=TC39&file=2012_sub_tc39-2012-066.pdf)

- An At-Names is an IdentifierName that is lexically prefixed with @
- At-Names are const bound to Name values by new declaration forms

```js
name @x, @y;
priv @secret;
```

- Such declarations implicitly create Name Objects
-
… Missed slides

### At-Name References

- At-Name can appear in any context where an IdentifierName would be interpreted as a literal property name.
  - As a prop name in object literals/class defs
  - After . in MemberExpressions
- Lexical scoping rules resolve such At-Name references to a Name object value.

```js
private secret;
let o = { @secret: 42 };

class MyClass extends Yours {
  @secret() {
    this.mine();
  }
}
```

### At-Names in Primary Expressions

- When used as Primary expression, an At-Name… (see slides)

```js
obj.@secret
obj[@secret]
// same thing
```

### Name Declarations with Initializers

- In a name/private declaration, each At-Name may have an initialization expression.

```js
private @secret = NameBroker.provideName(secretcode);
```

- The initializer must evaluate to a name object

- Primary use case is initializing an At-Name to a name provided via a function call or other computed value (the provided name or computed value is "secretcode").

DH: Concern about keyword naming, "private" (worried about contextual keywords)

AWB: naming can be bikeshedded, but whatever it is, there is no ambiguity because:

```js
foo @IdentifierName
```

Is always: "keyword [space] '@'"

DH: You've won me over

LH: Not sure this goes "far enough" to warrant the addition.

MM/AWB/YK: (discussion about simplification)

DH: (whiteboard)

```js
let o =  {
  private @foo: 42,
  m() {
    return this.@foo;
  }
}

o.m(); // 42

// As seen via inspection (REPL, console)
{ m() {} }
```

…Discussion about varying semantics lacking in previous proposals.

MM: Cannot avoid runtime collision?

DH: We can

MM: Should throw?

DH: Not strictly, but a possibility.

AWB: function vs. cost

ARB: strongly recommend making runtime duplicate check and throwing. (LH seems to agree? Or just acknowledge?)

## Optional Feature: Class-scoped name declarations

- Allow name/private declarations to occur as a class body element
- Any such declared At-Name are scoped to the class body.

```js
class Point {
  private @x, @y; // <=== scoped to class body
  constructor(x, y) {
    this.@x = x;
    this.@y = y;
  }
  get x() { return this.@x; }
  get y() { return this.@y; }
}
```

DH/RW: Makes sense for declaration to see it at once.

LH: Want to be cautious about what we commit to forever.

RW: Fully support this.

LH: Let's work towards understanding the entire commitment

…Discussion about path forward and whether it's too late.

DH: This does resolve an issue that stands out about max min classes: private name props cannot be added declaratively

WH: classes are not worth it without this

BE/LH/RW: Disagree. Classes already support themselves.

YK: Saying that it's too late is not a valuable argument.

DH: Just the API won't be _worse_. Allen's biggest point is that you cannot use Symbol in declarative forms with super().

DH: (rebuked)

AWB: There is a big hole in classes where super() is allowed only within class, means that name/symbol _cannot_ be used with super()

DH: imperative API is safe to start, At-Names are good, but we don't need them now.

WH: You could say the same about super() in class

DH: No. We have solid, reasonable semantics for where to allow super() and where to error.

LH: I don't think we're ready to design this syntax.

AR: Are we converging on this as an idea that needs solving? Just not now?

MM: If we decided that we had consensus today, we could improve it over the next year.

RW: (agrees)

MM: However, it _is_ late and the agreement of max min was to commit to something light weight that can be built on.

BE: I agree, but I dont think we should disallow exceptions to that cut off date.

LH: syntax was not on the table at the cut off

BE: Firm disagreement.

RW: This is actually a good example of why max min classes is a good idea. It's already an identified addition that creates a massive improvement.

WH: This actually adds the missing piece for me to support classes.

LH: I want something more minimal.

BE: Can we accept it for work, bet on it until at least the next meaning?

EA: Nice that it allows class private as well as instance private

LH: Agrees. If we're going to do this, we have to do all of it. The part that is listed as "optional" actually needs to be included.

AWB: Essential functionality is base declarations at the block level.

I have an ES.next (not 6) solution for "protected".

LH/DH: The baseline is private declaration in statement, classes, object literals. Build from there. As well as whatever the "public" version. BOTH are part of the baseline. NO PROTECTED.

AWB/WH/BE: Yes, I agree.

DH/BE: Move protected to a new strawman.

Conclusion/Resolution

Consensus on:

- Need private binding, including: statement form, import and export.
- Need private prefix form on class methods on object literal props and methods
- "public" in addition to private, but need to choose keyword
- dot notation, expression form

Deferred to separate proposal:

- "Protected"

# Renaming Name to Symbol

(Presented by Dave Herman, Mozilla)

YK: A little nervous that not identical to Lisp

DH: Gave blessing ;) the only diff is that you can't go from string -> symbol, whereas in lisp you can

YK: ok with that.

Conclusion/Resolution

- we agree that names are now called symbols (Name => Symbol)
- new alternatives for syntax to bind a "Unique Symbol" will be a keyword, one of: public, unique, or symbol ie. `public @foo;` vs `symbol @foo;` vs `unique @foo;`

## Early Errors That Possibly Should Not Be
(Presented by Luke Hoban, Microsoft)

LH: Concerns about detecting assignment to constants. Want to just parse, not build name environments, parse trees, etc.

WH: What about detecting a continue to a nonexistent label? That requires name environments.

LH: It's rare enough that it doesn't matter.

WH: Just what are we talking about? Should the assignment 3 = 1 no longer be an early error?

[Debate, without resolution]

[Discussion about whether the ES5.1 early errors should no longer be early errors.]
Consensus: No.

[Discussion about when these kinds of errors should be detected. One option mentioned was at function call time at the granularity of a function.]

WH: That's not useful. With arrow functions we'll have many lightweight functions, so the hybrid of early and late checking at function granularity does not do a good job of either early error detection or fully dynamic binding (late error detection).

DH: Not detecting typo'd free variables would be a big loss for one of the motivations for modules.

[Discussions about alternative of having early error detection of things like typo'd variables only in development tools]

WH: Concerned that these will diverge into a mess of incompatible "even-stricter" modes.

DH: macros need to expand at compile-time; we'd essentially be making static features like macros impossible

[Macros require compile time expansion]

BE: we can do the delayed errors now, and modules that use macros could be eagerly compiled; IOW we can make this decision now without necessarily closing that door

LH: need for delaying compilation of cold code is huge; large portions of web workloads consist of cold code

BE: (explains that interns and researchers at Mozilla are working on parsers that can support macros)

DH: Explains sweet.js (similar to coffeescript transcompilation)

DH/BE: Continue research of macros, offline.

MM: List all early errors and discuss the merits

AWB: Are we ok with a class of errors that are reported on each function entry? Static function analysis will simplify certain runtime semantics.

…Will allay performance issues discovered through Chakra and v8 prototyping.

Discussion of label analysis

Strict mode implementation

Conclusion/Resolution

- es-discuss will get a list of all early errors that should not be

# Performance Costs of Temporal Dead Zone

(Presented by Luke Hoban, Microsoft)

https://mail.mozilla.org/pipermail/es-discuss/2012-September/024993.html

LH: (introduces discussion)

ARB: Have you tested with let and no temporal dead zone

BE: early-boyer is likely not an accurate test for let. There are closures that capture deep chains of activations. If let is top level, why would it have changed?

LH: We suspect that a majority of the overhead was caused by the read barrier created by temporal dead zones.

…Will continue with testing and gather evidence.

MM/STH/BE: early-boyer is probably innappropriate.

LH: Believe that the motivation of temporal dead zones is not actually a common enough issue in real world code

BE: (proxy for Oliver Hunt) says the whole JSC team is against temporal dead zones on let, support it on const

LH: meta concern is that cases are adding up against let. If the perceived cost is real, measurable performance cost, there will be less buy in from developers.

WH: [as previously mentioned by Brendan], having a lot more scopes to close over (i.e. creating a new one for each loop iteration, etc.) is likely to be the main performance cost of let. What if let is slower merely due to having many more scopes?

BE: (history of let in SpiderMonkey…) No outcry that performance is an issue

MM: (countering the "not common enough" argument) Presented personal experiences where dead zones were essential in detecting problems.

Conclusion/Resolution

- No change to temporal dead zone (yet)
- Get more data, report back

# Global Scope Revisit

YK: Still not in agreement with discussion yesterday, but not blocking the

BE: Recap the problem that revealed the issue.

LH: Can we talk about let/var at the top level?

Current proposal… let would shadow var

ARB: Why?

BE: Global contour in which let binds

AWB: The rule in functions is that you cannot have a var and function of the same name.

MM: is the interaction with the global the same as a "with" scope object.

LH:

YK: class followed by class with the same name: error
var followed by let with same identifier name: error

BE: It happens all the time…

```
for (var i = 0; …) {}

for (var i = 0; …) {}
```

…Not the issue, but this:

```
var i;
for (i = 0; …) {}

var i;
for (i = 0; …) {}
```

…is a problem if `let` throws when name collision occurs.

DH: A serious issue exists… the single global object. It's beginning to feel like we're adding to the issue.

WH: The complexity is not on the number of scopes, but what they do. I'm concerned about the complexity of reflecting let, const, @names, etc. onto a global object.

WH: Also don't want random HTML attributes elsewhere on the page to knock out global let/const/etc. bindings.

AWB: No one has discussed… Any script tag, whose order can be determined… bring deferred scripts to the last script

BE:

```
A <script>
B <script> document.write("D <script>")
C <script>

( G )  <
  ^    ( D )
( A )
  ^
( B ) <-?
  ^
( C ) <-?
```

Hell.

DH: Note that multiple globals is not the problem. The problem is that the one global scope is the "window"

ARB: Implemented global lexical scope, give the impression of one, but is a scope chain.

DH: Why is it any different then having a global that isn't the window?

LH: It has to be.

BE: Points out in chart above that a function in A that refers to a `let x = 1;` in C, will create an error.

DH: Users make recursive dependencies that they don't realize.

ARB: The way I understood `let`… we resolve every variable at compile time, (I missed the next part, ARB: Can you fill this in?)

YK: (whiteboard)

```html
<script>
  function animate() {
    requestAnimationFrame();
  }
</script>
<script>
  let requestAnimationFrame;
  if ( typeof requestAnimationFrame === "undefined" ) {
    requestAnimationFrame = …polyfill.
  }
  animate();
</script>
```

BE: Recapping the Window.prototype issue

RW: There is no reason to put _everything_ on the Window.prototype. Object APIs should be own properties of the Global object ("window" in the browser case). This doesn't mean that the needs of EventTarget specification cannot be met by using the Window.prototype. The WebIDL change to move all APIs to a different semantic "space" without understanding the consequences is the worst negligence.

…More discussion…

BE: If we make let, const, class, module, function, var (all binding forms, now and in the future) are global—will this be blocked?

…Discussion…

DH: Hard to decide if a dead zone exists for `let` on the global object.

ARB:

LH: What are the mechanics of `let` accessors on the global?

…Discussion about the

Two models to consider:

Andreas:

When you have a `let` binding, it reflected observably as a getter/setter pair with some form of identification as a `let` binding.

Luke:

I can't figure out what the semantics you're describing is

DH: I think it's simple: every let binding is simply stored in a getter/setter pair

ARB: better to think of as a separate scope contour, where the getter/setter accesses it

DH: no difference

ARB: No, it is the difference between being stored inside the object vs inside the separate scope contour

DH: I still don't see the difference

WH: How are the proposed hidden attributes (such as the aforementioned identification of a binding as a 'let' binding) reflected if the global object is a Proxy?

DH: Tom has spec'ed these to "pass through"

BE: Object to the idea that there is an observable getter/setter on `let` bindings.

ARB: If you don't do it this way, you can't have a temporal dead zone in the global scope.

DH: (to BE) Can you recap your objection to getter/setter pairs on `let` bindings? Optimization doesn't seem like a long term motivation if we're moving to `module`

BE/DH/ARB…

Like modules, need to have global getter/setter pairs on let/const bindings

Unless there is a lexical scope or hidden data store…

LH: Observability?

ARB: It would be actually observable if the global object was a Proxy. Think of this in terms of a lexical contour

LH (recapping one of the alternative models)

There are two scopes:

Global Object and Global Lexical Scope… when a `let` binding occurs, there are two things created, one on the Global Object and Global Lexical Scope.

[people generally felt it was easier to comprehend the data living in the scope contour]

Conclusion/Resolution

Agreed on the AWB/MM/WH alternative model

(new binding forms)

- Allen's 1 extra global scope contourc
- Redeclaration is an error
- Shadows all properties on the Global object
- Does not create a Global property#

# September 20 2012 Meeting Notes

John Neumann (JN), Mark Miller (MM), Norbert Lindenberg (NL), Nebojsa Ciric (NC), Allen Wirfs-Brock (AWB), Luke Hoban (LH), Paul Leathers (PB), Sam Tobin-Hochstadt (STH), Andreas Rossberg (ARB), Brendan Eich (BE), Erik Arvidsson (EA), Dave Herman (DH), Yehuda Katz (YK), Rick Waldron (RW), Eric Ferraiuolo (EF), Matt Sweeney (MS), Doug Crockford (DC), Alex Russell (AR), Rafael Weinstein (RWS), Waldemar Horwat (WH), Rick Hudson (RH)

# Object.observe Update

(Presented by Rafael Weinstein, Google)

REQUEST SLIDES!

Aug 18th, Released an experimental implementation, spec complete, via special Chromium Build.

https://github.com/rafaelw/v8

Updates to strawman:

- Updating [[Prototype]] qnqueues a "prototype" changeRecord

ChangeSummary.js (experimental library)

https://github.com/rafaelw/ChangeSummary

- Prototypes "diff" view of changes
- Correct observation of "paths" (eg. o.foo.bar.baz)
- Array splice projection (minimal ops to syn)
- Basis for framework usage

YK: Question about splice projections being built in. When splice happens, many records change… is pathologically slow. Fine for v1 to leave it to library code.

RWS: Opted to leave this out…

…explains n^2 issues that arise when changes to an array occur. …explains rationale for leaving out for the foreseeable future and allow library authors to handle as they see fit for the time being.

DH: How to make a policy decision about "what" to look at in the change of an object. Agrees with this v1 decision, in favor of allowing library optimization patterns to emerge. [We can't determine the "policies" before the needs are fully understood]

AWB: Not just large scale libs, but everyday data type abstractions.

RWS: Sounds like there are specific issues?

AWB: Yes, but not to be addressed in this timeline

MM: Is the synthetic changeRecord adequate for the level of abstraction you may require

AWB: Yes, sufficient.

RW: (shared anecdotal experience writing "Fact" with Object.observe: https://github.com/rwldrn/fact)

RWS: (presenting demo)

ADD LINK

Discussion about "read" notifications and performance concerns.

YK: Willing to move Ember, despite the scale

EF: Did Angular replace all dirty checking?

YK/RWS: Not really possible to remove all dirty checking.

## Observing Computed Properties and Dependencies

RWS: Believe that it is not in scope now or ever.

## Tuning Spec, Implementation Complexity

RWS: …Is hoping to progress the strawman to harmony. A few slides that discuss remaining issues.

LH: These represent concerns and agreements of several committee members who have been involved.

### Synchronous Delivery?
NO.

### Security Mitigations
Object.getNotifier(frozenObject) returns null
(Out of 3 options: return null, throw, or do nothing notifier)

WH: What happens when the argument is a Proxy?

RWS: Returns the Proxy's notifier

STH: An invariant maintained internally, that if the proxy is frozen it ensures the target is frozen.

ARB: Not sure if this is true.

MM/STH: The proxy cannot say it is frozen if the target is not. Can the proxy say its NOT frozen if the target IS?

WK/MM/STH: **Proxy can say it's frozen if AND ONLY IF, the target is frozen.**

MM: If the notifier is derived, then the object is frozen, the notifier will continue to work as expected.

Creator of an object:

1. creates an object
2. gets the notifier
3. freezes the object
4. releases the object

This is an intended mechanism of the Proxy proposal

WH/AWB/MM: Discussion about notifications from frozen objects. The use of the notifier should ONLY be from the provider of the abstraction.

### oldValue

Most use cases would create copy of all observed data

WH: What about accessors?

RWS: accessors don't notify

WH: That's bad.

MM/EA/RWS/AWB: No, this is intended.

RWS: (Explains that getNotifier can be used to build synthetic events for accessors)

RW: Yes, accomplished this while experimenting with "Fact"

RWS: (Revisits demo to show example of what an abstraction over this looks like)

YK: (Supported use story in Ember)

…Mixed discussion about far future notifier patterns:

```js
// no-op, just for example...
function handler( changeRecord ) {
  console.log( changeRecord );
}

class Foo {
  private @x, @notifier;

  constructor(x) {
    this.@x = x;

    Object.observe(this, handler);

    this.@notifier = Object.getNotifier(this);
  }

  get x() {
    return this.@x;
  }

  set x(v) {
    this.@notifier.notify({
      object: this,
      name: "x",
      type: "updated",
      oldValue: this.@x
    });
```

    this.@x = v;
  }
}

var f = new Foo("hello");
f.x = "Hola";

// synthetic change event fired with "changeRecord"
// defined in the set x() accessor
```

Discussion regarding the feasibility of adding to ES6.

DH: Doesn't need to be in spec to prototype

BE: Worried about sheer weight of work involved in Allen's spec writing.

RH: Moving from strawman to proposal/spec is meaningful to implementor teams.

[some discussion about whether the above construction pattern has efficiency problems; needs implementation experience]

**Conclusion/Resolution**
Push for prototype implementations (Chrome already in progress). Encourage others to do the same

Officially Promoted to proposal for ES7.

# Grammar Validation
(Presented by Brendan Eich, Mozilla, Waldemar Horwat, Google, Rick Waldron, jQuery/Bocoup)

Overdue for grammar validation, would be ideal to re-validate.

WH: The following is ambiguous because yield is not a reserved word. It can't even be lexed:
boom = yield/area/
height;
Note that the current semantic restrictions on where yield expressions can go don't help because they apply after the program has been parsed; with this example you can't even get to the lexing and parsing stages.
```

Resolution on the yield issue?

Conclusion/Resolution

AWB will refactor grammar so that yield can only be used within the context of a generator function and yield will not be usable as an identifier there. This will require essentially doubling the number of grammar rules in an analogous way to how the no-in rules are handled today, but on a much larger scale.

BE: Need wider, complete validation that takes into account ASI and noIn.

BE/AWB: Who is going to own this?

Conclusion/Resolution

Waldemar is going to try to work on this

# Formal Parameter Scope

(Presented by Brendan Eich, Mozilla)

With regard to default formal parameters…

Previously, all to the LEFT are in scope,

`let *`:

```js
var b = "outer";
function f(a = x, b = a * b, c = c * g() ) {
  /*
  three scopes:
    (a),
    (a and b)
    (a, b, c)
  */
}
```

Realistically, it should desugar, scope rules included:

```js
```

```
var b = "outer";
function f(a, b, c) {
  if ( a === void 0 ) a = x;
  if ( b === void 0 ) b = a * b;
  if ( c === void 0 ) c = c * ();

  function g() {} // will hoist
}
```

AWB: functions will hoist, let and const will be in the dead zone and will fail, var will hoist and initialize undefined

BE: Can we agree to get rid of `let *`?

With `let x` this is an error:

```js
function f( a = "a" ) {
  let a;
  var a;
}
```

Not `let` bindings

```js
f(a = b, b = 3)
```

let scope would break var scope: bad

let scope with magic to break var scope: bad

ARB: (whiteboard)

```js
function f(x = (y = 42), y = 41){}
```

```
f();

42, 42
```

DH: My understanding was that temporal dead zone should error all the way to actual "blessed source code" where the binding is initialized.

Remind why only "no read before write"?

BE: 2 years ago in redmond agreement.

AWB: (explains the rationale and current semantics)

DH: is this a distinction worth making, for `let`, when we're trying to say that `let` is a better `var`

MM: `let` guards were the original reasoning.

DH: `let` guards should be different from just `let`

MM/WH: No, they shouldn't be different.

STH: Need to provide an argument beyond that

WH: guarded `let` should be not be different from a `let`. One common option for a guard must be a bottom type that lets through anything, but that's impossible with the same behavior if a guarded 'let' differs from a plain 'let'.

DH: This is far future, hypothetical

MM/DH: related to temporal dead zone discussion.

LH: The temporal dead zone opposition is perf

WH/DH: (discussion of complexity for `let` and `const`)

DH: I dont want to alienate developers for no good reason (gives supporting argument)

MM: Better to return undefined or immediately error to show where the mistake is made

DH: Want to simplify to "no read before write". The model of `let` creates a binding and = assigns a value to it. There is no way to explain that this `=` is different from that `=`. Refers to Alex's JS is top-to-bottom.

MM: But it's not because you can forward-call functions, because JavaScript allows you to interleave function and var initializations and assignments throughout your program.

DH: (missed not about read barrier)

WH:

ARB: Trying to simulate C semantics within the constraints of a dynamic system.

AWB: Yes, this was the agreement before temporal dead zone. throw if used before init.

MM, WH: Halting further discussion. We frequently eat up time on subjects like this that have already reached consensus. Let's postpone unnec revisitings.

AWB: Ok to revisit for valid reasons…

YK: worried that if no performance issues are found then it wont be revisited.

AR: Points out that there might be real issues with developer understanding. (cites some example, ask to share?)

[[[[[[[[[[[[[[[[[[[[[[

Temporarily, this happened:

Conclusion/Resolution

- `var` bindings and are in scope within the function

- cannot use `let` to shadow a parameter

- defaults can refer to any top level binding

]]]]]]]]]]]]]]]]]]]]]]

Conclusion/Resolution

Revisit when data is gathered, re: perf or unexpected behaviours

# Array.of Rename?

Recent post on es-discuss from user that doesn't like Array.of

Array.of has been implemented in all of the es6 shim libs (Paul Miller, Axel Rauschmayer, Andrea Giammarchi and others…)

## Array.of()

Makes sense, nice to say and explain.

When I reason about a program:

"Here we have an array of elements"

(elements, items, numbers, strings)

Conclusion/Resolution

No change, no revisit.

If we do `Foo.new()`, it must be _identical_ to `new Foo()`.

# Thin Arrow?

(Presented by Brendan Eich, Mozilla)

We have the fat-arrow, supported by Kevin Smith's research, it's a win. Some voices in the community don't want the unexpected behaviour of the bound lexical `this`

class, concise methods and fat-arrow are all new, powerful and composable function binding forms.

WH: Don't want two slightly different concepts with a confusingly similar syntax. It would be too difficult for casual users to remember which arrow is which [in the same way as I can never remember in C++ which variant of the ++ operator overload takes an extra dummy argument].

?: Then let's use fat-arrow with an extra 'this' parameter to stand for thin arrow.

WH: That would address the confusion, but is still unnecessary featuritis and doesn't even save much in terms of text, which was its original reason for existence. Saving a couple characters here is not worth complicating the language.

Conclusion/Resolution

Consensus holds on fat-arrow

# Existential Operator (strawman discussion)

(Presented by Brendan Eich, Mozilla)

Significant desire include a null and undefined check in syntax/operator form (a la coffeescipt)

```js
o = {}
r = o?.p.q.r
r = o?.p?.q.r
```

Mixed discussion about the needs and use cases as they apply to coffeescript code.

ARB: This is non-compositional

```
    o = {}
    r = o?.p.q.r
    r = (o?.p).q.r
    r = o?.p.q.r()
```

Results in…

```
    var o, r;
    o = {};
    r = o != null ? o.p.q.r : void 0;
    r = (o != null ? o.p : void 0).q.r;
    r = o != null ? o.p.q.r() : void 0;
```

Non-starter.

DH: Why not an operator that needs to be explicit?

```
    o?.p?.q?.r
```

LH: Why would you ever even use it on the first?

BE: Forget all of the problems with coffeescript's impl, the need exists.

YK: In the common cases, where it works, it works well. Where it doesn't, it falls apart unexpectedly.

WH: What about other contexts such as p?[x], p?.q[x], and p?(x) ? [Note that grammar problems arise for some of those.]

General agreement.

Conclusion/Resolution
Seems useful, but not now. Semantics are unclear

# Generators

## thisBinding

Generator thisBinding is the thisBinding of the original generator call.

```js
class MyArray extends Array {
  *iterator() {
    let last = this.length;
    let next = 0;
    while (next < last) yield this[next++];
  }
}
```

## Generator object API?

```js
class MyArray extends Array {
  *iterator() {
    let last = this.length;
    let next = 0;
    while (next < last) yield this[next++];
  }
}

new MyArray(4, 8, 15, 16, 23, 42).iterator();
```

…returns a generator instance that will have generator methods

MM: Take a care to expose the APIs that you expect to expose.

DH: Agrees

LH: Specify the generator?

BE: Allen is worried that normative spec will require generators when it's unnec.

DH: spec a simplified generator interface, without semantics, just to define the method interface.

MM: what is the minimal method interface?

DH: send, throw, close

Discussion around specifying a generator contract.

DH: Don't specify what can't be put on generator objects.

AWB: Need something for spec

YK: If everyone implemented iterators with generators

DH: Not super worried about this…

Conclusion/Resolution
The spec should not specify built in iterators to have 3 extra generator methods (ie send, throw, close)
(Currently in draft, needs to be refactored)

Notably: Significant dissent on throwing exceptions for control flow.

Continued discussion of note…

LH: Using a debugger, break on throw, exceptions. Want to catch at the point where they are thrown… on a StopIteration, do I see internals?

DH: Up to the debugger to determine whether or not it should expose

LH: (whiteboard, example misuse of iterator)

BE: Historically, not an issue.

YK: (Question about use of in-band return?)

LH: if this is not an issue, then why not specify

Discussion about protocol specifications, where precedent exists.

# Supplemental tests for Tests 262

SpiderMonkey and v8 are writing tests, can contribute back.

Need parity, it's hard.

# Goals

AWB: January, spec: feature complete.

LH: Multiple implementations?

Concerns about removal of new additions when there isn't enough evidence to support the removal.

Too soon to make cuts when we don't know what to cut.