Minutes for the: *31st meeting of Ecma TC39*

in: *San Jose, CA, USA*

on: *27-29 November 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, Yahoo, Northeastern University

1.2 Introduction of attendees

Istvan Sebestyen - Ecma International

John Neumann (MS, Mozilla, Yahoo, Google)

Norbert Lindenberg - Mozilla

Nebojsa Ciric - Google

Allen Wirfs-Brock - Mozilla

Luke Hoban - Microsoft

Brian Terlson - 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

Mark Miller - Google

Brendan Eich - Mozilla

Alex Russell - Google

Waldemar Horwat - Google

1.3 Host facilities, local logistics

On behalf of eBAY **Douglas Crockford** welcomed the delegates.

1.4 List of Ecma documents considered

Ecma/TC39/2012/067	TC39 chairman's report to the CC, September 2012
Ecma/TC39/2012/068 (Rev. 1)	Minutes of the 30th meeting of TC39, Boston, September 2012

Ecma/TC39/2012/069	Draft minutes of the conference call of the IPR adhoc on 24 September 2012
Ecma/TC39/2012/070	Draft "ECMAScript Internationalization API Specification", 3 October 2012
Ecma/TC39/2012/071	Sixth draft Standard ECMA-262 6th edition, October 2012
Ecma/TC39/2012/072	Draft minutes of the conference call of the IPR adhoc on 1 October 2012
Ecma/TC39/2012/073	Final draft "ECMAScript Internationalization API Specification", October 2012
Ecma/TC39/2012/074	Liaison Statement COM16 - LS 316 on "ITU's work on script languages", September 2012
Ecma/TC39/2012/075	Notes of the ad hoc meeting on Internationalization, 5 October 2012
Ecma/TC39/2012/076	Venue for the 31st meeting of TC39, San Jose, November 2012
Ecma/TC39/2012/077	Agenda for the 31st meeting of TC39, San Jose, November 2012 (Rev. 4)
Ecma/TC39/2012/078	Draft minutes of the conference call of the IPR adhoc on 29 October 2012
Ecma/TC39/2012/079	International Registration of ECMAScript in the Republic of Korea
Ecma/TC39/2012/080	Draft minutes of the conference call of the IPR adhoc on 19 November 2012
Ecma/TC39/2012/081	Seventh draft Standard ECMA-262 6th edition, November 2012
Ecma/TC39/2012/082	ICT Proposed specifications for Identification
Ecma/TC39/2012/083	"ECMAScript Internationalization API 2.0" by Norbert Lindenberg

2 Adoption of the agenda (2012/077-Rev4)

The agenda was approved.

3 Approval of minutes from September 2012 (2012/068-Rev1)

The minutes of the September 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 – which reflect the discussions correctly and has been already shared within TC39 - 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)

A couple of additions to the agenda this week if we have time:

ES6 feedback via TypeScript: We have heard a good deal of feedback from users of TypeScript on details of ES6 proposals, particularly on classes and modules. It might be valuable to present this feedback in full separately from individual feature discussions.

Promises: There was a long thread on promises on es-discuss recently. It might be valuable to checkpoint on where this stands currently in the committee, and what work champions should pursue on an ES7 track.

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

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

7

4.2 Review new draft spec

Changes in recent spec. drafts

Global declaration instantiation added, "global object + lexical scope" model.

Program is now Script

Set/Map size is now an accessor

Set/Map clear method added

Also see spec. change log at
http://wiki.ecmascript.org/doku.php?id=harmony:specification_drafts

4.3 Syntactic support for private names:

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

What keyword should be used to declare non-private at-name bindings: symbol, unique, sym, public, etc?

4.4 Revisit function parameter scoping and default value rules

https://mail.mozilla.org/pipermail/es-discuss/2012-September/025198.html

https://mail.mozilla.org/pipermail/es-discuss/2012-October/025629.html

4.5 Extending Array Comprehensions to include multiple if clauses. Consider adding a let clause, etc.

https://mail.mozilla.org/pipermail/es-discuss/2012-September/025044.html

https://mail.mozilla.org/pipermail/es-discuss/2012-September/025105.html

4.6 Function Poison Pill methods and new function syntactic forms

https://mail.mozilla.org/pipermail/es-discuss/2012-October/026030.html

4.7 Issues with eval, new declarative forms, strict mode, etc.

4.8 The prototype/constructor object model supporting Generators/use of instanceof with generators and generator instances

4.9 String normalization and case conversion

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

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

4.10 Conventions make non-standard properties configurable

http://wiki.ecmascript.org/doku.php?id=conventions:make_non-standard_properties_configurable

4.11 Go/no-go decision: Eliminate ToUint32 wraping on array access

Sparse parameter on Array iterator constructors (eg. Array.prototype.keys/values/items that determines whether or not "holes" are included in the iteration.?

4.12 Update on modules and loaders

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

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

4.13 Useful returns from collection (cascading)

Mr. Neumann gave a brief report on the work of the IPR Adhoc Group. The group has not finished its work yet.

Up to now the Adhoc had 12 meeting, the minutes have been distributed also to Tc39 for information. Any comments and feedback should go to the IPR Ad-hoc directly or to the Ecma GA. In TC39 there are no plans to discuss these issues.

The Ad-Hoc gave the “homework” to **Mr. Neumann** and **Mr. Wirfs-Brock** to get an update of the scope of TC39 and to prepare a draft scope and Programme of Work for the TC39 RF TG.

8.1.2 Approval of the updated TC39 scope

Mr. Neumann introduced the updated scope and Programme of Work for TC39:

“TC39 Scope

Standardization of the general purpose, cross platform, vendor-neutral programing language ECMAScript. This includes the language syntax, semantics, and libraries and complementary technologies that support the language.

Programme of Work:

1. To maintain and update the standard for the ECMAScript programming language.

2. To identify, develop and maintain standards for libraries that extend the capabilities of ECMAScript.

3. To develop test suites that may be used to verify correct implementation of these standards.

4. To contribute selected standards to ISO/IEC JTC 1.

5. To evaluate and consider proposals for complementary or additional technologies."

TC39 has unanimously approved the updated scope of TC39.

No scope and Program of work has been prepared for the TC39 RF TG.

9 Date and place of the next meeting(s)

Schedule 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 **eBay** for hosting the meeting, the TC39 participants their hard work, and **Ecma International** for holding the social event dinner Wednesday evening: http://www.nycfbllc.com/#/zone8-62/. Special thanks goes to **Rick Waldron** for taking the technical notes of the meeting.

Annex 1

Technical Notes (by Rick Waldron):

November 27 2012 Meeting Notes

John Neumann (JN), Norbert Lindenberg (NL), Allen Wirfs-Brock (AWB), Waldemar Horwat (WH), Brian Terlson (BT), Luke Hoban (LH), Rick Waldron (RW), Eric Ferraiuolo (EF), Matt Sweeney (MS), Doug Crockford (DC), Nebojša Ćirić (NC), Yehuda Katz (YK), Erik Arvidsson (EA), Mark Miller (MM), D. Herman, Sam Tobin-Hochstadt

JN: (Identify missing members or those that will be here tomorrow)

…Welcome and introductions.JN welcomed members and introductions were made. DC gave us facility and logistical information. Dinner for Wednesday is set for 6 PM.

JN# Previous Meeting Minutes

JN: Review and approve?

RW: Confirm that I have had opportunity to review prior to submission.

JN: Minutes from Sept 2012: Approved.

Review and approval of agenda. Labeled as rev42 will be sent to Patrick for publication.

ES6 feedback via TypeScript

LH: Move TypeScript feedback update to later, as part of module discussion.

Review Proposals for inclusion in ES6

JN/AWB: Discussion re: harmony:proposals page and status of listed items.

Spread Operation accepting Iterables

RW: SpiderMonkey has implemented spread to accept iterables despite the resolution from July. Would like to revisit the resolution.

DH: The implementation was likely just a misunderstanding, will file a bug for these notes.

EA: Let's add to the agenda for further discussion.

Agenda Discussion

Missing agenda items re-added.

Begin Technical discussion as item 4 on agenda

Review of new draft spec

AWB: Summary of changes in recent spec. drafts, including:

- Global declaration instantiation added, "global object + lexical scope" model.
- Program is now Script
- Set/Map size is now an accessor
- Set/Map clear method added

Notably, size is the first accessor defined by the spec.

DH: There will be more accessors defined in the module loader spec.

EA: Ensure no prototype for methods, should behave the same as spec functions.

AWB: Explains how the spec language is laid out for accessors, wherein "get" or "set" is prepended to the section item's title.

RW: (agrees that it's clear to follow)

AWB: This is the first rev. that brings in Proxy, which leads to the restructuring of early sections and internal methods.

Section outlines…

6. Source

7. Lexical Grammar

8. Type

Abstract Engine

Language

    Lex

    Syntactic

Vocabulary

9. Helper

10. call/return/scoping

11. expressions

12. statements

13. functions

15. Libraries

Explanation of rationale leading to the restriction and reorganization of internal methods for defining spec behaviors. Leading into the organization of…

Meta Object Protocol, aka. MOP

Discussing section on invariants and how to redefine in a reasonable way. We need people who are interested in these invariants to make contributions to this section.

8.1 Language Types

8.2 Specification Types

8.3-8.5 Will be moved to after section 10, these define the concrete types

8.3 Ordinary Object - an object that uses the standard MOP semantics in 8.3. example: Objects, Function objects

8.4 Exotic Objects - any object that is not an Ordinary Object, which means anything that specifies the use of something that is not in the MOP. examples: BoundFunction, Arrays, Strings

Standard Object - any object defined by the specification.

WH: why is BoundFunction exotic?

AWB: the special semantics of [[Call]] and [[Construct]]

WH: Does this mean that all built-in functions eg. Math.sin are exotic? This labeling bothers me if the user might be able to define differences between ordinary/exotic

LH: Why aren't all objects exotic?

AWB: initially set out to replace Host Objects, initially defined as anything that has redefined its internal methods

LH: Are there any checks, "is this an exotic object"?

AWB: No, this just simplifies and cleans up the language to that describes an object that has "special" semantics

WH: Do you identify all built-ins as exotic?

AWB: Reads back from Built-In descriptions in rev12

STH: Seems like it's simply a way to ease the pains of editing the spec.

AWB: [Further explanation of

WH: ?

AWB: Every algorithm in chapter 15 is the definition of the algorithms used when [[Call]]. In a situation where we need to reify concepts where we're not sure what the execution context currently is… eg. generators

WH: How does execution context come into this discussion

AWB: The spec needs to be able to discuss these semantics (gives example)

NL/WH: How is this observable if the implementation may or may not be provided in ECMAScript code?

AWB: Only need a means by which to explain the semantics.

I need to be able to talk about the execution context of a chapter 15 function while in a chapter 15 function algorithm

DH: I'm not sure this concept of an exotic object is the best way to go about this.

AWB: At this point we're talking about chapter 15 issues, not specifically exotic objects.

DH: I think this is good to signal to implementors that something different will have to occur.

STH: This is simply a tool to aid in the editing process.

DH: Not an annex, good.

AWB: Correct, the different semantics are grouped where they actually belong.

WH: I still don't understand why there is a grouping of these types of objects.

AWB: Please review the spec… If there are any strong arguments, please share.

AWB/DH: Confirm the subject matter of sections 8.3 and 8.4

RW: (stated prior to beginning of meeting) This change makes reading, understanding and following the varying behaviors easier to follow.

WH: Reading 8.4.6 and can't follow the description…

AWB: There are issues that exist due to prior hacks (not authored by me).

STH/LH: As long as this is just a spec tool, there is nothing wrong with the chosen categorizations.

AWB: For those of you reading the spec, please re-familiarize yourself with the new terminology.

LH:

AWB: The risk for semantic change… some methods renamed, some may have different behaviors. No change to observable semantics

DH: Don't remember the proto climbing refactoring

AWB: Tries locally, then same operation on next level. Each step up is observable via proxies.

DH: not web dependent

WH: if it's not visible, why do implementations need to change?

AWB: In the presence of proxies, proto climbing is visible. The difference… What [[Get]] in the old system did was loop, instead of recursion. If a prototype was a proxy, the trap would never be invoked. In the new system, it will. The property access system needed to change to support this…

DH: We'll get feedback from proxy implementors

AWB:

MM: There are two engine based proxy implementations, SpiderMonkey and v8—let's wait to discuss the impact.

BT: So there is a low risk of semantic change in 8.x? (with no regard for proxies)

AWB: Should be no observable semantic change and this should be confirmable with tests and reviewers.

…Will add a description of what should be observable and not.

Another change happening in this edition, is specifying where each exception is thrown. Previously this was implicit, but is now **explicit**.

WH/AWB: review 10.2.1.2.6 use of ReturnIfAbrupt(value)

LH: was there concern of ambiguity before? Seemed clear…

AWB: There were places where it was unclear where the the exception should be thrown, Mark had identified the loop in ??

…Discussion about explicit exceptions

DC: Need to break.

WH: Throw on string concat over allowed length?

…Discussion about ReturnIfAbrupt…

LH: Is this a maintainability issue? If a new leaf addition is defined that uses ReturnIfAbrupt, do you have to check all call sites?

AWB: The work is already done.

… Discussion.

Break.

Meeting Notes - Publish to wiki

(side discussion, but valid to document)

Conclusion/Resolution RW to publish meeting notes to ecma wiki (in addition to publishing on es-discuss and submission to ECMA)

Internationalization Update

NL: (Summary of changes based on feedback) Spec document submitted to general assembly for approval. HTML version of the spec has been prepared, will be posted to ECMA site after GA approval. Demonstration of test402 (INTL spec testing) running on Firefox (special build, not in Mozilla repository yet).

NC: v8 Implementation has regressed, due to clean up of defineProperty use.

LH: Has an IE plugin impl., next phase is to implement directly

LH: Should have discussion re: spec intention that all checks whether this is an instance Object, is this cross-realm? or this realm?

AWB: Yes, let's discuss this. Clarify…

NL: Any questions re: Internationalization 1.0?

JN: What is the plan w/r to hosting and maintaining test402

NL: Any company that wants to invest resources into contributing tests and maintaining?

BT: Can't say specifically, but Microsoft will certainly be contributing tests

NC: Same for google

JN: Anything for submission to the GA?

NL: We'll provide a technical report of some form for submission. Will provide the number of tests, (approx) and that coverage (missing and not) is understood.

International 2.0

(get slides from NL)

(1)

Complete Spec Sep/Nov 2013

TC39 approval March 2014

GA approval June 2014

MM: Be aware of possible delays in ES6

(2)

Prioritization

(3)

High (Or part of ES6?)

- Unicode normalization
- Case Conversion
- Character properties in RegExp or as API

Discussion re: unicode, changes in ES6 to RegExp re: unicode

No changes are in draft yet.

(4)

High (cont)

- IANA timezone IDs in DateTimeFormat

- Chrome 24+ has impl

- Message Formatting, including gender and plural handling

- Not clear how template strings fit in

RW/NL: agree to loop Alex Sexton into the work on message formatting specification.

(5)

Wait

- DateTimeFormat improvements

- Need feedback on 1.0

- Pattern strings, highlevel specifiers

- Info for date pickers

- Date intervals, relative dates, durations?

- Expose ToLocalTime?

(6)

Wait

- Resource Bundles

- Needs investigation

- Maybe module system can be used?

WH: How are resource bundles different from JSON?

NL: That's generally how they are stored, the challenge is loading the right bundle for the application's current locale

DH: can this be stored compressed in a binary format?

AWB/LH/YK: (discussion and agreement on HTML/CSS involvement in resource bundling is too browser-centric)

WH: Rather than providing a method that loads a bundle from some web page (which will likely clash with how a particular web server is structured), provide utilities such as decoding a bundle from a string and selecting the right bundle (e.g. the current language's) from a list of bundle names.

MM: Abstract the problem to a data loading issue

EF: Whose job is it to define the information in a resource bundle?

EF/NL: generally the library or application using the data has to define its structure and provide bundles for the locales it wants to support; there is an issue when third parties want to add more locales

(7)

Medium

- Text segmentation: word and line breaks
- Editors, offline indexing…
- Chrome already has impl
- Browsers names for languages
- Display names for languages, countries, scripts
- Number parsing - no currencies/percent/dates

MM: Presumably there are other standards bodies that we can use the data from.

WH: This gets into the political issue of what countries are called based on local law. Is Taiwan a country?

(8)

Medium/Low

- Calendar Support
- Info for date picker
- Conversion between calendars
- Calculations within calendar (add 3 days)

(9)

No

- Title case, too many house rules

- Language Detection, too specialized

- Encoding detection and conversion, value decreasing.

(10)

- Script reording in Collator

- Pseudo-numbering systems in NumberFormat and DateTimeFormat

(11)

Approval?

JN: Who is working on this?

NL: First meeting had representatives from Mozilla, Google, Microsoft, Amazon.

JN: Will provide minutes of meetings to ECMA?

NC: Yes.

JN: Despite potential operational changes from ECMA, let's move forward with this project. Continue to report as adhoc group via this group.

Conclusion/Resolution

TC39 Approves to move forward with 2.0. NL will submit slides to ECMA for minutes record.

String Normalization

http://wiki.ecmascript.org/doku.php?id=strawman:unicode_normalization#add_normalize_method

NL: AWB has removed a number of references to normalization from the current spec that did not reflect reality

String.prototype.normalize(form)

MM: Any sequence of utf-16 has a valid, specific normalization?

NL: Will have to check if the normalization spec has anything about unpaired surrogates.

MM: Either we make the function total, in the sense that it always returns a string, or total in the sense that we define where it throws exception.

NL/LH: (agree w/ always return a string)

NL: Are we in agreement to spec this?

MM: Requirements: (moved to resolution)

Conclusion/Resolution

Yes, requirements:

- total,

- deterministic

- idempotent normalization (normalizing the result of normalization again will return the first result)

WH: Note that Unicode got this wrong a while back (their normalization algorithm wasn't idempotent, and it didn't even form proper equivalence relations). They fixed it since then and now explicitly state that it's idempotent.

String Case Conversion

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

LH: Why isn't this in the Internationalization standard?

AWB: Is there a reason this isn't in the Intl 1.0?

NL: Case conversion wasn't considered in original scope for Intl 1.0; we then forgot to add it when respecifying String.prototype.localeCompare and friends.

RW: There is a Case Conversion item in Intl 2.0, is this the same?

NL: Correct, but these functions are in Language spec. Should this be added to ES6?

AWB: Don't think that we should start moving Intl into ES, or at least not until ES7

NL/LH: Not being in ES6 doesn't prevent implementation or spec authoring.

LH: Doesn't need to be in the wrong spec just to move forward.

Conclusion/Resolution

Goes in Internationalization but doesn't prevent specification or implementation.

Eliminate ToUInt32() warping on array access

AWB: Sparse parameter on Array iterator constructors (eg. Array.prototype.keys/values/items that determines whether or not "holes" are included in the iteration.?

Can we eliminate functions returning Reference values from the specification

AWB: Arrays use ToUint32, which does modulo arithmetic.

WH: Yes, but it doesn't actually warp indices. Ones larger than 2^32-2 are not array indices; they're not treated modulo 2^32.

WH: Also note that strings such as "0.0" and "007" are intentionally not array indices either, even though they're within the array range. The array index code checks that the value round-trips to a number and back to the same string; this is what keeps indices over 2^32-2 from warping.

MM: What are the practical benefits of this?

AWB: 2 Things, we could do this at 2^53 and truncate there instead of warping

MM: Is there a history of implementors that have worked out these issues?

AWB: Every new array(like) operation needs to have this behavior

DH: Concern that this will break the web

AWB: IE had a problem for a long time that went nearly unnoticed.

MM: strategic to postpone this cleanup until we have integers. Don't see a reason to make this change

DH: Concerned about code that would even have arrays this large

AWB: The point is to avoid craziness after 2^53

MM: What is the handling of the 2^53 edge condition that this change will benefit?

AWB: I have to review the spec and can follow up tomorrow.

Look at 15.4 of ES5.1, when you go over the edge, results in expand properties being set on the array. Additionally, the 2^32 edge condition:

```
var a = [];
a[Math.pow(2, 32) - 1] = true;
```

Conclusion/Resolution

Tabled until AWB has further impact research.

The prototype/constructor object model supporting Generators/use of instanceof with generators and generator instances

AWB, presents UML diagram to illustrate…

The Generator Constructor doesn't need a global name. Assume it's accessible at System.Global.

Each generator function has a unique prototype object with own properties for next, send, throw, close, etc. These prototype types may share references to the built-in mplementations of those methods.

```
g1 instanceof Function; // true
g2 instanceof Function; // true
```

But no way to test if either g1 or g2 is a generator function rather than an ordinary function.

…

Possible Solutions:

1. Make Generator a subclass of Function, allows instanceof checking (on some kind of special System object)

2. Make a global built-in Generator constructor

3. @@hasInstance(): void

Anomalies…

1. instanceof will be true for non Generator functions

2. g1.constructor === "Generator"?

DH: If we go against the behavior of the language we'll end up with…

MM: Agree up until the reflective Generator

DH: Function creates function, Generator creates generator

YK: Anything other than that is pure WAT.

AWB: Need to get our terminology straight.

MM: Going forward, we've created this class system… Having Generators be a subclass…

lost track, sorry. Hope Mark can fill this point in later.

YK/MM/LH: like…

class Generator extends Function.prototype {}

class g* extends Generator {}

DH: The way to check "is this a generator?"

Object.getPrototypeOf(f) === "Generator";

MM: Of all the things we're talking about, creating Generators reflectively is the least concern. Something like…

Function.makeGenerator(…); returns generator function

… If that was important to provide, but likely not.

MM/LH: Generator is a zero-arg, no-op.

LH: Reiterates that class g* extends Generator {} clarifies thinking about the diagram

https://dl.dropbox.com/u/3531958/tc39/generator-diagram-1.jpg

DH/LH: (Discussion of this inside Generator)

MM/AWB: (Converge on diagram of inheritance relationship)

DH: Which parts do we surface as public API?

MM: Abstaining.

DH: Important to retain Python naming to avoid WAT. Ok with not surfacing anything to public API

WH: Would like to at least expose "Generator". [Note: the object I was referring to got renamed to "GeneratorFunction" later in the discussion and further down in the notes here.]

DH/WH: (Disagreement on exposure of public API)

AWB: Need a value for .constructor

DH: Ok, .constructor dictates the requirement.

MM: Does this mean that if you call the Generator constructor with a yield?

AWB/RW: Error

??

DC: Are we adding Generator because it qualifies as important enough to stand on its own?

DH: Reflective evaluation is powerful enough to stand on its own. A huge gulf between `with`

MM: There is no immediate benefit…

WH: It's easier to include then to exclude it, for spec benefit.

MM: Agreed, only benefit is specification symmetry.

WH: Function does reflection, so it makes sense.

MM: Consensus on exposing GeneratorFunction via some imported module?

All: Yes.

MS: Can I determine if an object is a generator function?

DH:

f instanceof GeneratorFunction;

f.__proto__ === GeneratorFunction.prototype;

f.__proto__ === (function *() {}).__proto__;

Conclusion/Resolution

- Per diagram (https://dl.dropbox.com/u/3531958/tc39/generator-diagram-1.jpg), but without exposing "Generator"

- "GeneratorFunction" exposed via a module

November 28 2012 Meeting Notes

John Neumann (JN), Norbert Lindenberg (NL), Allen Wirfs-Brock (AWB), Waldemar Horwat (WH), Brian Terlson (BT), Luke Hoban (LH), Rick Waldron (RW), Eric Ferraiuolo (EF), Doug Crockford (DC), Yehuda Katz (YK), Erik Arvidsson (EA), Mark Miller (MM), Dave Herman (DH), Sam Tobin-Hochstadt (STH), Istvan Sebestyen (IS), Andreas Rossberg (ARB), Brendan Eich (BE), Alex Russel (AR)

Syntactic Support for Private Names

BE/LH: Concern that the syntax required too much declaration

LH: Can delve deeper when presenting TypeScript findings. We don't have any experience with the impact of this syntax.

AWB: This is where we left it at the last meeting and I haven't had an opportunity to respond to feedback.

Mixed discussion regarding syntactic pains and impact of @-names

LH: Before it can go forward, someone will need to go back and address the existing issues.

AWB: The concern is the double declaration and we discussed adding a private prefix for method declarations.

YK: Alternatively, module bound private names, where declaration scopes to the module.

Discussion of Kevin Smith's modular @-names proposal…

AWB: The same logic applies to global vs. lexical namespace, and modules.

WH: If you have an @-name somewhere in a module, is it scoped to this module

YK: If you use an @-name it implies binding, and you need to explicitly export

AWB: References an implicit declaration if doesn't already have one?

YK: Yes. (draws example on whiteboard)

```
module "view" {
    export class View {
        constructor(id) {
            this.@id = id;
        }
    }
}
```

DH: Points of clarification…

1. If we're talking about @-names explicitly scope to a module, no declaration nec. Does that make them private or unique?

- You would have to declare specifically

Reviewing Kevin Smith's gist (https://gist.github.com/3868131) on projector…

ARB: How does this avoid the use of the same name twice?

AWB: You're expected to know your module.

BE: (draw comparison to Go, Dart, CoffeeScript)

DH: The goal is simply to avoid repetitive declaration lists

LH: The notion that declaration of private names as a runtime construct is great, but the syntactic representation needs to be intuitive to "this is a private thing". So far, this feels at odds with those intuitions

DH: Disagrees, this is a static concept and declarations within are intuitively static.

WH/AR/STH: There is no precedence in the language to scope limited binding forms.

DH: Painful if you're required to list out everything

WH/YK: Sharing private names across classes is a problem that needs to be solved.

DH: Implicit scoping is asking for trouble (gives examples)

AWB: Common case where a symbol only needs to scoped to a class, for that case, we have a proposal on the table that covers everything except for fields without a lot of redundant declaration. Beyond the scope of a single class, it seems an explicit declaration at an appropriate level is desirable.

DH: Tied to classes implicitly? But allowed to explicitly bound to other scopes

AWB/DH: Clarification on private for classes.

ARB: Would need to hoist the thing outside the class?

AWB: Only if you're contributing to the thing outside of the class.

BE: If you want an outer scope, put a block around it.

…Kevin's proposal seems to have no support here.

LH/AWB: back to the @-names, we left it at "it's too chatty"

LH: I want syntax for privacy, something less

Discussion about import @iterator in classes…

LH: The immediate problem w/ YK's example on board is that it's unclear that @id…

Developers don't want to think about binding names as objects

AWB: (modifies whiteboard)

```
module "view" {
    private @id; // allows declaring a private name called "id"
    export class View {
        constructor(id) {
            this.@id = id;
```

```
            }
        }
}
```

Move private @id…

```
module "view" {
        export class View {
                private @id;
                constructor(id) {
                        this.@id = id;
                }
        }
}
```

BE: This all may be developer and future hostile.

WH: (agrees)

BE: Developers want declarative form to define an instance field with a private name in one step. We separate those two.

WH: I want to preserve the option of saying one thing

BE: That's what Luke wants.

LH: Most developers don't want to think about declaring their names before use.

AWB: Then we can't address private within a class without addressing field declarations in a class.

(moves `private @id` out of example)

WH: We can't allow this to now be declarable in multiple contexts.

LH: The current behavior of @-names is not what developers expect it to be.

DH: I have contradicting experience. (ie. Racket define-local-member-name)

LH: If you had to do this for every property that you're ever going to use…?

DH: (Agrees with Yehuda's complaints)

BE: Then we need field syntax first.

LH/YK/WH: (nods of agreement)

DH: Let's punt on this for ES6. Too late

EA: How to do @iterator?

DH: We can make that work, but this is too large and too late. There are too many questionable issues w/r to declaration for the sake of scoping a name, without creating a field.

MM: Let's not discount ES7 development.

AWB: I disagree and don't think that we should defer on addressing this.

BE: If we wait and defacto standards emerge, then we're too slow.

MM: Intend to advocate:

- postpone explicit field declarations to ES7 and things that might conflict until.

BE: Agree

LH: The concern is @iterator?

standard private names and public names

WH: What is the point of contention for the existing field declaration proposal?

BE/AR: (explanation of constructor declaration and hoisting issue)

BE: (whiteboard)

```
// Mark's proposal from a year ago
// http://wiki.ecmascript.org/doku.php?id=harmony:classes
constructor(id) {
        private id = id;
}
```

MM: (reiterates rationale)

LH: (whiteboard)

```
// TypeScript...
private id;
constructor(id) {
  this.id = id;
}
```

AWB: What happens when there is foo.id is in the constructor?

DH: So private is to statically reject programs that appear to poke at things that are assumed private?

LH: Confirm

MM: So they foo.id will refer to the same id field?

LH: Yes.

…Unusable for ES6

DH: Proposes… Exactly the semantics as shown, but syntactically only allows field declaration position in classes and import/export.

WH: Important that you will want to _declaratively_ (not imperatively) list fields. Guaranteed to be there in instances of a class. Those who want to lock down the class further might also want extra class attributes that disallow other expando properties, etc. (not the default case, but something you might want to do).

LH: This is now a different discussion. If we introduce a form (re: whiteboard example)…

DH: A future compatible subset of what we discussed before.

Discussion about the baseline problem: Needing two lines to declare a private field.

YK: Not sure why the example that Luke approves of is different from the given syntax.

Discussion of computed object/class-literal properties, rejected due to

- Runtime duplicate checking
- Static object literal optimization

LH: Computed properties might be worth revisiting

EA: But you can't predict what the property name will be…

AWB: And you still need to go through the declaration steps…

BE/DH: Revisiting previous consensus on unique name for iterator

DH: No revisit on consideration for string name for iterator

LH: Revisit on square bracket computed properties.

AWB: Square brackets are future hostile… Explanation of the [] Reformation http://wiki.ecmascript.org/doku.php?id=strawman:object_model_reformation

BE/DH: (volley re: import iterator)

BE: If there is a standard library prelude in ES6 for @iter, that buys time to fully specify for ES7

AWB: Let me summarize… Take the @name proposal without the declaration.

DH: Understand, but the thing we do now needs a coherent story

AWB: Yes, we provide pre-declared @names and that's the end.

LH: Normal lexical bindings?

AWB: No, @name bindings

DH: max/min: only in property name position

WH: Can you stick an @name on any arbitrary object?

AWB: Just getting rid of declaration

BE: max/min

AWB: Existing @names in spec

- @hasInstance
- @iterator

DH: the whole benefit of unique names is no name clash

BE: This is why we decided that iterator should be public, because there is no way to avoid existing properties. We want new things to have no clash.

ARB: Want to use it for properties to avoid cross cut

BE: Use for stratified traps.

AWB: Any symbol is fine, doesn't need to private

DH: Why would you decide to expose something as visible…

AWB: There are cases where certain properties might want to be extended or customized.

Discussion of reorganization of Meta Object operations in order to simplify Proxy specification.

WH: The class proposals only permitted private properties on instances of the class. It was never the intent to allow you to create a private instance property @foo of instances of class C and then attach it to random objects unrelated to C.

AWB: At the root, symbols - ie. unique names are a powerful tool

MM: Always in consensus that symbols where a means of assigning and looking up a property by a unique, unforgeable name. It was never specifically tied to classes.

WH: Disagree. That's only one of the privacy proposals. Something got lost in translation in the attempts to merge the two privacy proposals. The class one would let you look up a class-private @foo on any object, but only the class could define an @foo property.

DH/AWB: (Discussion of pre-defined fields on class instances.)

AR: Similar to my constructor pre-amble…

LH: (Summarizing) No clash between the use of symbols at a lower level.

YK: Imagine a map literal…

AWB: Hypothetical Map that used [] for key, might use symbols for keys in that map, now there is an ambiguity at access time.

LH: Care about not having private names becoming lexical scopes

AWB, WH: (in response to question about why @'s are necessary) In the past we've attempted to work through several proposals that don't and it never works

ARB: Concern with meaning of @-syntax being dependent on context: sometimes denotes symbol itself, sometimes value it indexes. Might potentially be ambiguous in some circumstances, e.g. modules

MM: State a proposal:

- We don't have in ES6: a private or special declaration form.

- We allow @identifier, that is a symbol

let @foo = Unique();

- After a dot, in property name position, @foo does not refer to a new literal @foo, it refers to the value of the lexically enclosing @foo.

We address Andreas's issue with modules separately.

DH: b/c tied to variable declaration forms, no way to know upfront what one of these names is.

AWB: Clarify…

DH: (Re-explains)

AWB: (refutes)

DH: What I was hoping for was declarative syntax closing in… but realize it's totally generative.

ARB/LH: This is a hack

DH: The syntax looks static but is not at all.

MM: Is the hack a subset of all the non-hack things we want?

DH: This shows the same problems as we discussed earlier.

AWB: This what symbols are…

MM: The "@" is what makes it clear that this is not static

DH: You could say the same thing about brackets.

MM: Always knew @ was dynamic. Opaque, unforgeable and generative.

ARB: (to DH), Once this is defined in a dynamic context it becomes dynamic.

WH: (illustration of perceived hoisting and dynamic rebinding issues)

```
class C {
    private @name;
    f() {
```

```
            for(…) {
                x.@name = 5;
            }
            var @name….
        }
}
```

DH/MM: (discussion of future additions)

BE: We already have with nested function declarations name binding and generativity. Why is it ok for function, but not symbol?

DH: Punning the syntax to make it look static, but it's not.

BE: agreed, that was the fork we took to a bad path, punning "after-the-dot identifier"

DH: People rightly complained early on about static understanding/knowable aspects of syntax. (eg. do I have to look up in scope to know what prop means in { prop: val })

AWB: Most developers will align [] with dynamic property access, vs. .@foo aligns with "static" property name access.

LH: Clarification that we're not talking about the object literal case, but in fact the non-breakable, historic language syntax of property access with []

BE: (Hypothetical future with object [] reformation)

Discussion about implications of hypothetical future with object dereference reformation with ES6 objects.

MM: (to

LH: Moving back to the conservative position to build up from

AWB: Still have symbols

LH: Yes

YK: This is the max/min problem, writ large.

MM: Reminder of workload, wherein ES7 should look like just another phase of development and it's ok to defer to ES7.

AWB: Return to where we were before @-names were introduced

MM: yes

YK: Returning to [iterator]?

Yes.

LH: Still support Mark's proposal (see above)

DH: Only strict mode you get the duplicate error?

MM: true

DH: Could do semantics of strict mode and allow the collision

MM: I don't think this introduces a strict mode runtime tax.

Conclusion/Resolution

STH will provide a summary.

- Symbols, unique and private are runtime concepts

- Only additional syntactic support for them in ES6 is the square brackets in literal forms.

- Strict object literals throw on collision.

(Today, duplicate checks happen at compile time, this will no longer be the case when [prop]: val is used in an objlit)

const s1 new PrivateSymbol();

const s2 = s1;

```
var x = {
    [s1]: 33,
    [s2]: 44
};
```

In this context, within the square brackets: AssignmentExpression

(Re: Symbol constructor binding: http://wiki.ecmascript.org/doku.php?id=harmony:modules_standard)

Experience With TypeScript

Presented by Luke Hoban (request slides)

Findings…

Classes: Statics

- Statics are used frequently
- Imperative update is awkward when using an otherwise declarative construct

Classes: Privates

- Frequent asks for Privacy
- TypeScript added compile-time-only privacy
- Not quite the same as current private names syntax proposal
    - w/o further sugar private names syntax proposal will feel awkward in practical class

Classes: Automatic base constructor calls

- Missing super calls

ArrowFunctions

- Want thin arrow

Classes: Decorators

- w/ classes available, teams want to use them
- Biggest block is when existing class library supported some extra "magic" associated w/ class/method declarations
- No solution yet, not sure what this looks like.

MM: (re annotations) Note that "@" is no longer reserved for ES6…

DH: Point out that we are future proof here.

MM: Let's postpone discussion of the feedback

Modules

- ES6 modules
- compiled to JS which uses AMD/CommonJS
- …

Modules: Namespaces

- Two common patterns for large code structure
    1. On demand loaded modules
    2. Namespace objects to reduce global pollution
- External Modules address #1
- TypeScript allows internal module re-declaration to grow the object
    - Effectively, a declarative form for object extension with build in closure scope and syntax that matches large scale structuring use cases well.

LH: (the transition from AMD/CommonJS of today to modules a la ES6 is not going to be an easy transition)

Loading order…

LH: When you have circular references, current modules make it appear easy to ignore these issue.

Modules: "modules.exports =" use case

- Not addressed in TypeScript
- Critical for interop with existing CommonJS/AMD code
- Supportive of "export =" syntax proposal

```
// something.js
export = function() {
    return "something";
};
```

```
// other.js
import something = module("something");
var s = something();
```

DH: (whiteboard)

```
export = function() {};
```

```
import "foo" as foo;
foo();
```

Async

- Top requested addition for TypeScript is C# "await"-style async
- Generators + task.js help, but likely not enough
    - Wrapping is still very unnatural in any real examples
- But light sugar over generators + task.js would serve
- Feeds into promises discussion - have to standardize the task objects.

(shows example for task.js and identifies "spawn" which returns a promise object)

Mixed discussion about the history of async discussion through generators, promises, Q.async

Modules Update

Presented by Dave Herman

(whiteboard)

```
// Modules looked like…
module X {
    export module Y {
    }
```

```
}

// Moved to…
// "foo" doesn't bind anything into this scope,
// just adds to the module registry
module "foo" {
        module "bar" {
                // no more exporting…
        }
}

module "foo/bar" {}
```

"bar" is not exposed as a property of "foo"

```
import "foo/bar" as m;
```

AWB: (clarification of his understanding of the original way that nested modules work)

STH: Yes, that was the way, but there was a realization that much of the earlier approach was flawed and these updates lead to revisions.

One important use case for modules is to configure module references, so that libraries can import jQuery (for example), and get the appropriate version of jQuery specified by the page. Further, it's desirable to be able to use different code for the same library name in different context. Originally, the modules proposal managed this via lexical scope, as follows:

```
module M1 {
    module jquery = "jquery.js";
    module something = "something_that_uses_jquery.js"
}

module M2 {
    module jquery = "zepto.js";
    module something_else = "something_else_that_uses_jquery.js"
}
```

However, this has two major problems:

- Inheriting scope across references to external files is potentially confusing, and disliked by a number of people
- Once we decided to share instances of the same module, the "parent scope" of a module is no longer well-defined

Therefore, we abandoned the idea of inheriting scope across external references. However, this had two consequences that we did not immediately appreciate. First, we no longer had a method for managing this configuration between module names and source code. Second, scoped module names no longer had nearly as much use as originally.

Thus, Dave and I revisited the design, abandoning the use of lexical scope for managing module names, and introducing module names that could be configured on a per-Loader basis.

DH: The registry with the string names is now where the sharing mechanism occurs.

Loader…

System.baseURL = "…";

System.resolve = function(…) {

…

};

Mixed discussion…

- "global namespace" as in "per realm"

-

ARB: This seems to create a parallel global object for modules. Giving up lexical scoping for one global namespace.

DH: There is no way to get rid of the global object

STH: Yes we tried.

DH: (explanation of registry table)

- Per loader

MM: Separate name registries?

DH: Either are fine

…Provide a minimal set of APIs to allow devs to build there own.
…Sane default behavior
…Default resolution:
- baseURL + "Crypto/sha1" + ".js" when no config has been done, this is the base default behavior.

```
// foo.js
export = 42;

// bar.js
export function bar() {
}

// foobar.js
module "foo" {
    export = 42;
}
module "bar" {
    export function bar() {
    }
}
----------
<script>
System.baseURL = "assets/";
</script>
<script async>
import "foo" as foo;
import "bar" as bar;
</script>
```

WH: What happens if…

import "foobar" as fb;

(given the above "files")

STH: Answer: fb is an empty object. You also get modules named "foobar/foo" and "foobar/bar" defined.

[WH's question was related to a claim in the discussion that there is no need to have module .js files be distinguishable at the textual level from top-level script .js files]

Mixed discussion w/r loading protocols… and resource loading (files from server, etc) seems to be out of scope?

DH: How is there anything special about JavaScript as the one asset to know about in browsers?

<link rel="prefetch"…>

BE: Before imports, prefetch dependencies… but an out of line module import is not a hint, it's a requirement.

DH: Help the browser know in advance about its…

AR: a "prefetch" attribute for scripts? Requests script but doesn't execute.

RW: Until import?

AR: Yes

EF: Don't want to prefetch lazily loaded code later in the program. Don't want to load packages with same dependencies twice.

Bundle A, Bundle B

Each share common dependencies. Leads to unbounded number of combinations of pre-build bundles.

A loader should have a way which it can be told, upfront, about the dependency graph. Allowing the system to know all dependencies in advance, so that it doesn't have to compute transitive dependencies for all, every time—to make smart choices about IO.

November 29 2012 Meeting Notes

John Neumann (JN), Allen Wirfs-Brock (AWB), Waldemar Horwat (WH), Brian Terlson (BT), Luke Hoban (LH), Rick Waldron (RW), Eric Ferraiuolo (EF), Doug Crockford (DC), Yehuda Katz (YK), Erik Arvidsson (EA), Mark Miller (MM), Dave Herman (DH), Sam Tobin-Hochstadt (STH), Istvan Sebestyen (IS), Andreas Rossberg (ARB), Brendan Eich (BE), Alex Russel (AR), Matt Sweeney (MS)

Approval of ECMA/TC39 Scope Declaration

(Presented by John Neumann)

JN: (presents scope document for approval)

Conclusion/Resolution

Approved.

Scoping for default arguments revisited

(Presented by Allen Wirfs-Brock)

See Slides

AWB: (Review legacy requirements)

Two Params, Same Name allowed (non-strict)

```
function f(x,x) { console.log(x); }
f(1,2); // logs: 2
```

Parameter and a Var with Same name

```
function g(x) {
    var x;
    console.log(x);
}
g(1); // logs: 1
```

Function Declarations Override Parameter Bindings

```
function h(x) {
    function x() {return 2;}
```

```
    console.log(x());
}
h(function() { return 1; }); // logs: 2
```

Proposal Part 1

- Simple ES<=5.1 parameter lists introduce "var bindings" in the top level scope of a function
- All ES<=5.1 rules apply
    - Duplicated parameter names
    - Parameter names may be same as var or function Declaration
    - (missed) see slides…

ARB: These are simply the requirements.

Proposal Part 2

- If a parameter list uses ANY parameter syntax introduced in ES6, new rules apply:
    - Destructuring Parameters
    - Default value initializers
    - Rest Parameters
- New Rules:
        1. Parameter lists introduce "let bindings" in the top level scope of the function
    A. No duplicate param names
    B. Parameter names may not be the same as any other function top-level
        2. TDZ rules apply to parameter default value initializers
    A. Hoisted top-level function/var declaration are initialized after parameter initialization
        3. "strict" arguments object (copy of actual args, no parameter joining)

DH/YK/LH: This is problematic for extant offending code, that is updated to use ES6 syntax. One syntax change shouldn't have adverse effects on other, not directly related, syntax.

RW: If offending code exists, it would be smart to fix the issues, new syntax does new things.

YK: Sympathetic, but disagrees

New Rules Examples:

function f(x, x, …rest) {}

Syntax Error: duplicate parameter name (Rule 1.A)

function f(x, {a:x, b:y}) {}

Syntax Error: duplicate parameter name (Rule 1.A)

function f([x]) { let x; }

Syntax Error: redeclaration of parameter x (Rule 1.B)

function f([x]) { var x; }

Syntax Error: redeclaration of parameter x (Rule 1.B)

function f([x]) { {var x;} }

Syntax Error: redeclaration of parameter x using hoisted var (Rule 1.B)

function f([x]) { {let x;} }

Valid, redeclaration is in inner block

function f([x]) { function x(){} }

Syntax Error: redeclaration of parameter x (Rule 1.B)

function f([x]) { class x {} }

Syntax Error: redeclaration of parameter x (Rule 1.B)

WH/AWB/ARB: discussion about parenthesis on parameters

ARB: Points out that this is why 1JS becomes a problem where we introduce micro-modes to make things work

AWB: We need to have these to make these things work correctly

MM: Also nervous about these micro-modes, but want to see the rest of the proposal

New Rules Examples, Cont…

function f(x, y=x) {}

Valid, x has been initialized when y's default value expression is evaluated.

function f(x=y, y) {}

Runtime: ReferenceError exception y not initialized (Rule 2)

const a = "A";

function f(x=a) {}

Valid

function f(x=a) { var a;}

Runtime: ReferenceError a not yet initialized (Rule 2.A)

function f(x=a) { const a = "A";}

Runtime: ReferenceError a not yet initialized (Rule 2.A)

function a() {return "A";}

function f(x=a()) {}

Valid, a is initialized in surrounding scope, at time of parameter default value initialization.

function f(x=a()) { function a() { return "A"; } }

Runtime: ReferenceError a not yet initialized (Rule 2.A)

MM, WH: I want to preserve:

1. A Function, as soon as it's in scope is already initialized

2. A Var variable, as soon as it's in scope is already initialized

ARB: Agree, but these things simply should not be in scope.

DH: We're making mistakes with let all around. We should wait to continue this discussion until this afternoon when all are present.

MM: Luke, where does the let research stand?

LH: Incomplete.

AWB: If there are disagreements with these rules, then someone needs to write rules that cover all these cases.

ARB: I proposed a set of rules and posted to es-discuss. Basic idea: a function with default arguments behaves as if it was a wrapper function supplying the default arguments. So initializers cannot see any definitions from the body.

Simplest example:

```
-------------------------------------
| var foo = function() { return 1; }
|
| function bar( a = foo() ) {
-------------------------------------
|  var foo = null; <-- not in scope until after param
|                                                    default evaluation
| }
```

- https://mail.mozilla.org/pipermail/es-discuss/2012-October/025657.html
- https://mail.mozilla.org/pipermail/es-discuss/2012-October/025669.html
- https://mail.mozilla.org/pipermail/es-discuss/2012-September/024995.html

YK: What are observable issues with the proposal?

STH: Problems where scopes aren't seen

var declarations are not seen in parameter defaults

DH/YK/STH: Multiple nested scope might work

DH: If in strict mode, get errors. Not in strict mode, no errors. Resolves the refractor surprise issue.

YK/DC/RW/AR: (Agree with Andreas' proposal)

STH: AWB proposal has Reference Errors, ARB proposal simply says "not in scope"

(whiteboard)

(6)

```
function f(x, y = 7) {
    ...
}
```

<===>

(5)

```
function f(x, y) {
    let x1 = x;
    let y1 = y ?? 7;

    (5)
    return (function(x, y) {
        ...
    }).call(this, x1, y1);
}
```

WH: (whiteboard)

```
function f(x, x=7, z=x) {
    let x1 = x;
    let x2 = x ?? 7;
    let z1 = z ?? which x?;
}
```

DH/YK: Discussion re: 1JS issues w/r to strict and non-strict

AWB: Recap… There are clearly issues that exist. I invite anyone here to formally specify these.

DH: Andreas and I can work together, with all of these scenarios in mind (request for complete list above)

RW: Available in these minutes and here https://gist.github.com/4171244

Discussion about how other languages enforce default parameter values and existing precedents.

AWB: How does a user, that isn't familiar with JS semantics, come to understand that declarations in body… ?

MM:

WH: There was previously concerns with multiple scopes, which is clear why you've gone with a single scope

YK: Noted, non issue b/c {} doesn't create a scope bucket in JS today.

DH: (whiteboard)

```
function f(x) {
    console.log(x);
    let x = 12;
}
```

f(6); ?

DH: Imagine if this wasn't a redeclaration error, what would occur?

ARB: There are two errors here… there would still be a TDZ error.

WH: (in response to rant about scoping of let itself) This is not on the agenda

AWB/STH: This is absolutely the agenda.

Devolved.

Break

Conclusion/Resolution

Agree that Andreas will draft a proposal for next meeting.

Cascading this returns

(presented by Rick Waldron)

RW: returning "this" from

Map.prototype.set, Set.prototype.add, WeakMap.prototype.set

DC/AR/YK/EF/EA: Supporting agreement

MM/AWB/RW/BE/EF: (Discussion to determine a criteria for making this API specification distinction)

Conclusion/Resolution

Consensus… with the criteria that these methods are not simply a set of uncoordinated side effects that happen to have a receiver in common, but a set of coordinated side effects on a specific receiver and providing access to the target object post-mutation.

Issues With Eval

(presented by Allen Wirfs-Brock)

AWB: Existing issues with eval w/r to new declarative forms, strict mode, etc In particular, what grammar is allowed in eval.

ARB: E.g. allowing module declarations in direct eval would introduce local modules. (Agreement that we don't want that)

BE: But would be fine in indirect eval.

DH: System.eval is a much nicer way to do indirect eval

ARB: lets promote that as the "correct" global scope eval instead of diverging direct and indirect eval more

MM: Now we have 3 evals

DH: No way around 3 (1: direct, 2: indirect, 3: explicitly tied to a loader). But we can promote loader eval as the better one: direct but without ugly (0,eval)(src) syntax.

AWB: what about deletable bindings?

… All eval bindings in ES5 are deletable.

DH/BE: in strict eval, you can't delete locals…

AWB: (whiteboard)

eval("var x; delete x;"); // ?

MM: if this is strict, then no.

BE: let's talk about non-strict.

AWB: Change to "let":

eval("let x; delete x;");

ARB/MM: Illegal.

Consensus/Resolution

New declaration forms, even from non-strict mode eval cannot be deleted.

Eliminate functions returning Reference values from the specification.

AWB: Only want to remove the language from the spec, not reason to be a feature of the spec.

BE: Exists from the ES1 days, for VBScript-style DOM APIs in IE

LH: No objection.

Conclusion/Resolution

Consensus.

Revisit Nov. 27 Resolution on iterables in spread.

BE/RW: Recounting history, re: Array.from & spread delegation

BE: Changed mind about the forEach inconsistency

AWB/YK: Have to maintain consistency to enumerable methods

BE: Let's not remain slaves to legacy, Array.from, for-of and spread use only iterable.

RW: What about pre ES6 environment?

BE: Can fall back to array-like if needs.

BE/MM/RW: Both iterable and array-like fallback

Agreed.

Discussion about ES5 code running in ES6 environments

RW: Can't decorate, what then?

YK/BE: The polyfill has to work harder by wrapping the arraylike

BE: (whiteboard)

Array.from: iterable -> array

Array.from(iter(oldObj));

RW: No.

Return to two step on Array.from and iterator protocol on for-of, spread.

Conclusion/Resolution

Add iterator protocol to arguments object (should exist on all things.

Array.from:

1. Iterator protocol
2. Array-Like

for-of & spread:

1. Iterator protocol

Collection APIs review…

AWB/BE: Mixed discussion re generalized iterator API

BE: issues with values()

AWB: Don't care what it's called

DH: Relevant for Maps and Sets, as they've used the names keys(), values()

BE: (whiteboard)

1. for (v of a) // a has @iterator or throw
2. for (v of values(a)) // this already has a meaning
3. for (v of myValues(a)) // more possible …

…Change to a property called "elements" and it settles most of the argument

If we have method based dispatch for _some_ things, we should have it for everything.

Here is the reference AWB was searching for at the meeting regarding why only having function forms of keys/values/items is a problem: https://mail.mozilla.org/pipermail/es-discuss/2011-November/018332.html

MM: can we agree on… 3 methods:

1. iterable over values
2. iterable over keys
3. iterable over [key, value]

Naming to come…soon?

Do not have these on Object.prototype

…MM: Map is a proper stratified properties collection. Objects are not. Arrays, when used as collections, the indices are the "keys"

DH:

Collections
.keys
.values
.items
-> Array
-> Map
-> Set

Dict
keys(o)
values(o)
items(o)

keys - iterable of keys
values - iterable of values
items - iterable of pairs

DH: Possible? Dict constructor, where…

Dict() is shorthand for Object.create(null);

Implements…

keys(o)

values(o)

items(o)

And still allows .property and ["property"]

BE: Ensure that I don't have to copy into a new Dict just to get keys, values, items

DH: Won't have to.

ARB: better name: 'entries', not 'items'. (agreed)

Conclusion/Resolution

Dave Herman to craft an addition to module standard library for Dict with api shown above that creates proto-less objects as Dicts and has api that can be used on all collections.

Allen Wirfs-Brock to update existing spec language to reflect:

.keys()

.values()

.entries()

-> Array

-> Map

-> Set

The syntax of let

AWB: Making let a reserved word breaks the web.

- Removing let from non-strict?

- Crafting contextual syntactic

MM: Propose that let is not a binding form in non-strict code. When we first experimented with let, we knew the consequences, but now we know the outcome.

The remaining problematic code:

var let;

let[x] = 5;

WH: Two ways to look at it from a syntactic point of view. It's either a problem with let syntax, or it's a problem with destructuring syntax. Restricting just one of the two to strict mode would eliminate the clash with ES5; doesn't matter which one.

DH: We _could_ limit destructuring to strict mode to solve this issue and this would likely encourage more migration to strict mode.

(several nods of agreement)

BE: This leads into the reality of strict mode and the changes to runtime semantics (gives ex. of concat issues)

LH: Opposed to not restricting any new syntax to strict mode. If we just disallow this very specific example:

var let;

let[x] = 5;

LH: I think we can get away with it.

DH: Could refine that to only apply if 'let' not in scope as a variable.

ARB: No.

DH: Doesn't change the parse.

AR: But how the AST is constructed afterward.

BE: Back to Dave's earlier proposal, I don't think this will get us buy in, strict mode has a bad rep.

LH: Just one more hack is not always bad, but in fact how progress of anything that wants to avoid breaking back compat.

STH: Can we do the Apple experiment with Luke's proposal?

Two proposals on table:

1. All things that meet the grammar for let are let declarations and we don't reserve let. (LH)

2. Like 1, but `let` is not reserved if there is a lexical (not top-level) variable named `let` in scope (DH)

Leaning towards Luke's proposal, implementors not offended.

MM: (whiteboards proposal)

EA: No, because it kills let destructuring.

BE: We're not sure how big the problem really is… We could make the change and approach addressing the breakage via evangelism.

RW: My thinking is that we have ideal resources to find the uses of "let" in existing code and evangelize before ES6 publication.

WH: That would be ideal

MM for DC: Should we defer let to ES7?

DH/BE/YK/AWB/WH/ARB: Disagree

AWB: What if we just have const?

Nope.

DH: Our sense of aesthetic shouldn't get in the way of progress.

STH: This isn't that gross.

BE: I move that Luke's proposal be drafted for ES6

LH: A search of indexed web reveals 3 uses of var let.

?: Proposal to try parsing as a let statement first and fail over to an expression statement that just happens to use let as an identifier if that fails.

WH: No, negative parsing rules like that are known to cause byzantine problems. Better to disambiguate on the first two tokens alone just like we disambiguate on the first { token alone to distinguish a block from an expression statement that happens to start with an object literal. The first two tokens would be let followed by either an identifier, [, or {.

Conclusion/Resolution

In non-strict code: let, with single token lookahead (where the single token is either an Identifier, "[", or "{"), at the start of a statement is a let declaration. (Accepted breaking change)

Extend new let grammar restriction?

let (x) = …

let ?maybe = …

let !must = …

Continued discussion… Do we want to preemptively disallow these: (, ?, !

DH: Want to allow parentheses in or around patterns for analogy with (x) = 7.

WH: That's half of the analogy, and the parentheses in that case are actually around a subexpression. Note that var (x) = 7 is not currently allowed, and I don't see any reason to permit it. Propose to continue existing behavior:

(x) = 3; // allowed

({ y }) = 3; // allowed

var (x) = 3; // disallowed

var ({y}) = 3; // disallowed

DH: Why are you opposed to parentheses? Not useful now, but would like to use those in the future.

WH: I'm not opposed to parentheses in the pattern language in general, but there is no point in putting them in until we have some good use for them. Prefer to omit now simply to future-proof our design options.

DH: OK

Conclusion/Resolution

Agreed to the semantics presented by the allowed/disallowed example above.

Extending Array Comprehension

(Presented by Brendan Eich on behalf of Jason Orendorff)

Begins here: https://mail.mozilla.org/pipermail/es-discuss/2012-September/025044.html

BE: Originally seen in ES4 but never made it to ES6 (brief history of comprehension and rationale)

(whiteboard)

[x for x of a]

[[x,y] for x of a for y of b]

[[x,y] for x of a for y of b if x % y]

Proposal: restricted language, paren free heads, arbitrary sequences of let and if.

Allows?

[w if w]

[z let w = z * z if z > 4]

Mixed discussion about necessity

Discussion about also providing while clauses. Rejected for a (flawed) technical reason (incorrect claim was that they couldn't mutate a variable), but not much interest in including while clauses anyway.

DH: Allow if anywhere makes it more expressive and allows for earlier outs. The let is necessary for nested loops (storing outer values for use in the inner loop)

WH: To clarify proposal: No semicolons? No commas?

BE: Correct.

MM: All agreed, whatever is allowed here is also allowed between parens for generator comprehensions.

BE: No cost, no loss, use case gains.

Conclusion/Resolution

Consensus on Jason's proposal: for, if, let, const can be interleaved. Applies to both Array Comprehensions and Generator Comprehensions

yield, the identifier?

AWB: yield * 5?

BE: yield is reserved inside of generators

Function Poison Pill Methods and new Function Syntactic Forms

Reference: https://mail.mozilla.org/pipermail/es-discuss/2012-October/026030.html

AWB: Should all new function forms, in non-strict mode, all have poison-pill properties for arguments.caller, arguments.callee, Function.caller, Function.callee.

WH: These are new forms, unlikely to have the bizarre engine semantics that poisoning was designed to eradicate. Why are we bothering with poisoning them at all?

BE: notes that es-discuss preference was to uniformly poison

MM: No security problem to have the function behave the same with respect to either strict or non-strict

Conclusion/Resolution

New forms are like old forms per non-strict and strict (reduce surprise factor)

Conventions make non-standard properties configurable

http://wiki.ecmascript.org/doku.php?id=conventions:make_non-standard_properties_configurable

MM: The other non-standard bits that implementations add, should be configurable so SES can repair or remove it.

Proposal: All non-standard properties that are put on standard built-in objects by implementations must be configurable: true and actually deletable.

WH: The proposal should apply only at the surface of built-in objects. It should be perfectly fine for an implementation to create an object X with nonconfigurable properties and set b.p = X (where b is a standard built-in object) as long as b.p is configurable.

MM: Agreed.

WH: Note that this object tree scanning approach doesn't protect against language extensions. For example, consider an ES5 sanitizer applied to an ES6 script. The sanitizer wants to restrict the script to only the built-ins it whitelists. The sanitizer walks through the built-in objects, deleting ones it hasn't whitelisted but then is blissfully unaware that the script can get access to non-whitelisted generator classes by defining and running a generator function via language syntax instead of following object links.

MM: Yes, that is a hole. In fact, we've been blocking implementations that accept E4X syntax for that very reason. However, we don't currently try to parse a script to see if it uses future syntactic constructs we don't know about.

BE: Backs Mark's rationale

Conclusion/Resolution

Luke and Mark will chat offline.