Minutes for the: *29th meeting of Ecma TC39*
in: *Redmond, WA, USA*
on: *24-26 July 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, Apple, jQuery, VuB, Yahoo, Northeastern University

1.2 Introduction of attendees

John Neumann (Ecma/Microsoft-Mozilla-Yahoo-Google)

Douglas Crockford – eBay

Allen Wirfs-Brock – Mozilla

Brendan Eich – Mozilla

Bill Ticehurst – Microsoft

Sam Tobin-Hochstadt – Northeastern University

Tom Van Cutsem – VuB

Alex Russell – Google

Rick Waldron – jQuery

Yehuda Katz – jQuery

Luke Hoban – Microsoft

Norbert Lindenberg – Mozilla

Erik Arvidsson – Google

Oliver Hunt – Apple

Dave Herman – Mozilla

Mark Miller – Google

Andreas Rossberg – Google

Jonathon Turner – Microsoft

Rafael Weinstein – Google``

1.3 Host facilities, local logistics

On behalf of Microsoft **Mr. Hoban** welcomed the delegates.

2 Adoption of the agenda (2012/046-Rev1)

Revision 1 of the draft agenda was agreed for the meeting.

3 Approval of minutes from May 2012 (2012/034)

The minutes of the May 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 Tuesday and **David Herman** volunteered to help out starting Wednesday. 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) - See Attachments

4.1 A new ES6 (5th, July 8, 2012) draft is available at 2012/042

Rev9 (July 8, 2012) of the ES6 Draft Specification is now available at http://wiki.ecmascript.org/doku.php?id=harmony:specification_drafts

Changes in this version include:

- Quasi literal added to specification
- Initial work at defining tail call semantics (still need to define tail positions in 13.7)
- Initial pass at replacing native/host object terminology with ordinary/exotic objects
- Clause 6 and others updated to clarify processing of full Unicode source code. Revised usage of “code unit” and “code point”
- Specification of Identifiers updated to use current Unicode specification devices
- \u{nnnnnn} Unicode code point escapes added
- UTF-16 encoding for non-BMP characters in string literals now fully specified
- Added functions: String.fromCodePoint, String.raw (a quasi tag function), String.prototype.codePointAt
- ECMAScript now requires use of Unicode 5.1.0, normative references updated
- A syntactic grammar notation was added for indicating when alternative lexical goals are required
- Fixed ES5 missing explicitly setting length in several array functions
- Fixed bugs: 368, 388-399, 402-405, 410-413, 415-416,418, 420-428, 430-439, 445-456,458-461 (thanks very much for all the bug reports)

Please report bugs you find at bugs.ecmascript.org.

4.2 Override Mistake (Wed)

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

4.3 Revisit for(let;;) binding alternatives (Wed)

https://bugs.ecmascript.org/show_bug.cgi?id=311

4.4 Resolve scoping rules for global lexical declarations (Wed)

https://bugs.ecmascript.org/show_bug.cgi?id=312

4.5 Review candidate spec approach for “regexp_match_web_reality”:

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

4.6 Proposal to change behaviour of DaylightSavingsTA

https://mail.mozilla.org/pipermail/es-discuss/2012-March/020832.html

4.7 Weak References (Yehuda) (Wed)

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

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

4.8 Support for full Unicode character set"

http://norbertlindenberg.com/2012/05/ecmascript-supplementary-characters/index.html

4.9 Adding forEach to new ES6 collections (separate from iteration) (DH)

https://mail.mozilla.org/pipermail/es-discuss/2012-February/020776.html

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

4.10 Object.isObject

http://wiki.ecmascript.org/doku.php?id=strawman:object_isobject&rev=1295471005

4.11 getClassNameOf

4.12 Open issues with direct proxies

http://wiki.ecmascript.org/doku.php?id=harmony:direct_proxies#open_issues

4.13 Observe strawman (discussion on July 25th for Rafael)

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

4.14 Classes

Try to reach consensus on including Maximally minimal classes in ES6

Review candidate class specification in current ES6 spec. draft clause 13.5 and associated open issues.

quasis issues and solutions based on July specification draft

https://mail.mozilla.org/pipermail/es-discuss/2012-June/023735.html and
https://mail.mozilla.org/pipermail/es-discuss/2012-June/023825.html

See

Recent destructuring issues

Objections to applying ToObject to the RHS of destructruings

Handling of missing properties: undefined or throw. Consider allowing ! or ? as a prefix to select.

Explicit undefined value trigger use of default value initializer.

"subclassing" built-ins in ES6 strawman
http://wiki.ecmascript.org/doku.php?id=strawman:subclassable-builtins

extensibly mechanism for {}.toStirng in ES6

Unresolved issues relating to iterator naming/access (Dave)

4.15 Error stack standardization (Wed)

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

4.16 Other Issues

Recent destructuring issues

Objections to applying ToObject to the RHS of destructurings

Handling of missing properties: undefined or throw. Consider allowing ! or ? as a prefix to select.

Explicit undefined value trigger use of default value initializer.

"subclassing" built-ins in ES6 strawman

http://wiki.ecmascript.org/doku.php?id=strawman:subclassable-builtins

extensibly mechanism for {}.toStirng in ES6

Unresolved issues relating to iterator naming/access (Dave)

5 Edition 5.1 Issues

6 Report from the ad hoc on Internationalization standard (Ecma/TC39/2012/0041) (Tues at 1:00 PM)

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

6.2 Multi-system prototype testing

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

7 Test 262 Progression

7.1 Addition for Internationalization

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 GA meeting

On the request of TC39 there are two subjects on the agenda of the Ad-Hoc: 1) The TC39 request for extending the TC39 Software Copyright policy and 2) The request of a TC39 member to extend the Ecma Patent Policy with a RF optional regime. On both subject updates were provided by distributing the relevant IPR Group and GA documents as TC39 documents.

It should be noted that the GA has requested both for ES6 and ECMA-402 that the current members of TC39 give voluntary RF statements to the Ecma Secretary General. The relevant GA document was also distributed to TC39 (TC39/2012/045).

8.1.2 Draft ITU-T Recommendation "ECMAScript for IPTV services" (Ecma/TC39/2012/022) – liaison to ITU-T SG16

Three documents have been sent by ITU-T SG16 as liaison:

Ecma/TC39/2012/030 Liaison Statement to Ecma TC39 on ITU's work on Script Languages

Ecma/TC39/2012/031 ITU-T H.764 "IPTV Service Enhanced Script Language"

Ecma/TC39/2012/032 ITU-T H.762 "Lightweight interactive multimedia environment (LIME) for IPTV services

A liaison reply has been prepared and approved by TC39 (Ecma/TC39/2012/052-Rev2).

8 Date and place of the next meeting(s)

September 18 - 20, 2012 at North Eastern University (Boston)

November 27 - 29, 2012 at Bay Area (eBay?)

The intent at the next meeting is to schedule 2013 meetings. For now they will be three day meetings, subject to change as required.

9 Closure

Meeting was closed at 4 PM on Thursday.

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

Annex 1

Technical Notes:

 # July 24 2012 Meeting Notes

Present: Yehuda Katz (YK), Luke Hoban (LH), Rick Waldron (RW), Alex Russell (AR), Tom Van Cutsem (TVC), Bill Ticehurst (BT), Brendan Eich (BE), Sam Tobin-Hochstadt (STH), Norbert Lindenberg (NL), Allen Wirfs-Brock (AWB), Doug Crockford (DC), John Neumann (JN), Oliver Hunt (OH), Erik Arvidsson (EA), Dave Herman (DH)

10:00-11:00am

Discussion of proposed agenda.

Determine participants required for specific subjects.

July agenda adopted

May minutes approved

4.1 AWB Presents changes resulting in latest Draft

Draft related bug filing
Increased community participation, a good thing
Issue with numbers not matching duplicate filings, be aware

Quasi Literal added to specification
Spec issues have arisen, will review

Initial work defining tail call semantics (still need to define tail positions in 13.7)
What defines a "tail call" in ES
Existing Call forms need to be specified in how they relate to
tail positions. (call, apply, etc)

STH: Important that call and apply be treated as tail calls

YK: and accessors

STH: Agree.

…discussion of examples

AWB: Differences between accessor calls as they apply to proxy call traps, not definitively identifiable at syntax level. The function call operator and the call trap.

TVC: Proxy trap calls currently can never be in a tail position (except "apply" and "construct" traps)

STH: call should be in tail position. Clarification of known call site syntax, per spec.

Summary:

Anything that could invoke user written code in a tail position to act as a tail call.
call, apply, accessors, quasi (interpolation), proxy calls

We still need to specify the tail positions in the syntax. There's a start by DH on http://wiki.ecmascript.org/doku.php?id=harmony:proper_tail_calls which uses an attribute grammar, but the current spec draft leaves this blank.
Filed: https://bugs.ecmascript.org/show_bug.cgi?id=590

4.5 RegEx "Web Reality"
(http://wiki.ecmascript.org/doku.php?id=strawman:match_web_reality_spec)

Introduction to discussion by Luke Hoban

LH: Attempted to write a guide to make regex specification match current implementation wherein order of production matters. See *15.10.1 Patterns* in above link.

…Gives specfic examples from 15.10.1

Discussion between AWB and LH re: semantic annotations and redefinition.

YK: Do non-web implementations match current spec or web reality?

AR: Are there any non-web implementations?

YK: Rhino?

BE: matches reality because based on SpiderMonkey circa 1998

Test cases? Yes.

BT: Yes, cases exist in Chakra

LH: (Refers to examples)

NL: Do these affect unicode? We had agreement at previous meeting that web reality changes would not be applied in Unicode mode (/re/u).

LH: This is what regex is in reality… Waldemar did not want to specify because it's too hard to specify, but now the work is done

AWB: Too hard is not an excuse to not specify, good that the work is now done.

Discussion of "\u" in existing regex - \ug or \u{12} is interpreted, but differently than planned for Unicode mode

Trailing /u flag?

Makes grammar more complicated to have \u{...} only if /u flag used.

AWB: Three things to address: Web reality, Unicode support, new extensions

LH: /u the only way to opt-in to Unicode escapes with curlies, with Unicode extensions.

NL: need to reserve backslash with character for new escapes in the future, e.g. \p for Unicode character properties

OH: Fairly substantial regex in wild all created with RegExp constructor.

YK: Moving forward: Evangelize using Unicode and tacking "/u" onto all new regex?

BE, OH, AR: yes.

Decision: LH and NL to collaborate on integrated proposal

4.7 Adding forEach to Map and Set
http://wiki.ecmascript.org/doku.php?id=harmony:simple_maps_and_sets

Deferred, got to it on third day

4.9 getClassNameOf

BE: Recap, last meeting there was discussion about getting a strawman from YK

YK: I began specifying, but existing questions prevented

BE: some want to solve not only the typeof null problem, but also "array"

YK: What is the usecase for Object.isObject

DC: Polymorphic interface

AWB: "has properties"

RW: Similar to isNaN: isNull that is only for null

OH:(Reiterates that we cannot change typeof)

AWB: what is it about host (exotic) objects that need to be differentiated from native (ordinary) objects?

YK: Reclarification about things that are not objects (in the [object Object] sense) that say they are.

AWB: If we go down this path, can anyone redefine the return value

YK: My question is: either always return object Object, or let anyone change to return anything

AWB: Rephrase as "extending toString()". Removing [[Class]] from spec, but now as [[NativeBrand]].
The default: exactly as they are today. in ES6, if this property is defined, then use it, if not, use default.

Mixed discussion of real world uses of:
Object.prototype.toString.call(o)

BE: 1JS Killed typeof null

BE, OH: Like the idea of a configurable property to define explicit value of brand

YK: why is what "toString" returns so important?

AR: 2 things:
1. Fixing is not easy
2. How to correctly fix w/o making more surface area for the wrong thing

Summary

There is worry that changes to spec that affect the return of toString will have adverse impact on existing libraries and users when they encounter new runtime behaviours where the existing behaviour is expected.

Belief that we need a more flexible mechanism, whether it is AWB's configurable property that defaults when not explicitly set, or AR et al trait type test proposal.

BE, AWB: nominal type tests considered an anti-pattern per Smalltalk, but they happen in JS not only "because they can" -- sometimes because of built-ins you need to know

6 Internationalization Standard

Norbert Lindenberg: (Introduction and opening discussion)

Discussion, re: contributors

6.1 Last call for feedback before final draft

Function length values? Using ES5 section 15 rules would cause respecified functions like String.prototype.localeCompare to have larger length values; using ES6 rules would let them keep old values.

Leads into larger discussion about Function length property.

Decision: Apply ES6 rules to all functions in Internationalization API.

Numbering system, number formatting system. Would like to reference Unicode Technical Standard 35.

Outstanding issue:

If you have 4 different impls, 3 of them support a language that you want to support, how can you polyfill the 4th to support the language.

The constructor can re-declared?

Conclusion: There is no easy way currently, second version of Intl spec will address this.

Conformance tests being written for test262.

NL will have the final draft prepared for September meeting, but will produce drafts leading up to that meeting.

6.2 Microsoft and Google are implementing prototypes

Unicode support

AWB:

within curlies: any unicode code point value \u{nnn}
so essentially three ways within string literal:
- two old-style escapes, expressing utf16 encoding
- two new-style escapes, expressing utf16 encoding
- one new-style escape, expressing code point

BT: treating curlies as utf32 value?
AWB: curlies contain code point value, which you *could* call utf32

DH: old-style escapes always are a single utf16 code unit, so always .length 1; new-style escapes always are a single Unicode code point, so may have .length 2

NL: "<<some stupid emoji>>" = "\u{1F601}" = "\uD83D\uDE01" = "\u{D83D}\u{DE01}"

AWB: one point of controversy: what happens with utf16 escape sequences within identifiers
- no current impl recognizes suppl pair escape sequences for suppl identifier characters
- `var <<wacky identifier>> = 12` -- is that a valid identifier?
- `var \u{<<wacky identifier code point>>} = 12` -- is that a valid identifier?

NL: and, for example, what if it goes in an eval?

DH: careful! difference between:

eval("var <<emoji>> = 6")
eval("var \uD83D\uDE01 = 6")
eval("var \\uD83D\\uDE01 = 6")

AWB: disallowed:

`var \uD83D\uDE01 = 6`
`eval("var \\uD83D\\uDE01 = 6")`

allowed:

`var \u{1F601} = 6`
`eval("var \\u{1F601} = 6")`

DH: any reason to allow those?

YK: sometimes tools taking Unicode identifiers from other languages and translating to JS

DC: we have an opportunity to do this right; \u{...} is the right way to think of things

DH: we have eval in the language, so the language thinks of strings as UTF16 and should have a correspondence in the concept of programs

LH: there's just no strong argument for this inconsistency

DH: there's no real practical value for disallowing; there is potential harm for the inconsistency in causing confusion in an already-complicated space

DC: the only real value here is for attackers; no normal code uses this

BE: and maybe code generators

LH: it's just removing an inconsistency that could be a gotcha

LH: there isn't a codePointLength -- is that intentional?

AWB: since strings are immutable could be precomputed

DH: which is why you want it to be provided by the engine, so it can optimize (precompute, cache, whatever)

DH: should it be a function, to signal to programmer that it has a potential cost?

AR: but no other length is a function

DH: fair enough, just spitballing

AWB: what about code point iteration from end to beginning? and also codePointIndexOf? don't have those yet

4.1 (cont) Processing full Unicode Source Code

String Value

Conversion of the input program to code point sequence outside of standard

Trad. \uxxxx escapes represent a single char, creates a single BMP character, 16bit element

Issue: in string values, ?? (Etherpad is broken) === \u1F601 === \uD83D\uDE01 === \u{D83D}\u{DE01}. In identifiers, ?? === \u1F601 !== \uD83D\uDE01 !== \u{D83D}\u{DE01}. Inconsistency that's hard to explain to developers.

DC: This feature is more likely to be used by hackers than developers.

AWB: Two APIs

String.fromCodePoint (build string from integer values)

String.prototype.codePointAt

What's here, valid surrogate pair?

DH: Mixing the API levels is problematic, should it be scrapped?

…The problem in naming is the "At"

…If we're going to build code point abstractions, we really need a new data type.

NL: ICU has iterators for grapheme clusters, words, sentences, lines – all based on UTF-16 indices. Abstractions don't require different indices.

Need more here.

4.13 Destructuring Issues

A. Patterns discussion on es-discuss

Issue: ToObject() on the RHS?
This is currenty specified and enables things like:
let {concat, slice} = "";

This equivalence is desirable and maintain by the current spec:
let { foo } = { bar: 42 }
===
let foo = { bar: 42 }.foo;

A syntax for pattern matching against objects
match({ bar: 42 }) {
case { foo } { console.log("foo") }
default { console.log("no foo") }
}

---

let { ?foo } = {}
let ?foo = {}.foo // _wtf_

DH: Pure WAT. Let's pick the most common case and address that. You cannot presume to cover everyone's pet case

What is the right thing to do.

DH: Future pattern matching

LH: Reiteration of correct matching vs intention

More discussion, defer until AR is present

let { toString: num2str } = 42;
===
let num2str = (42).toString;

Consensus without AR is to impute undefined for missing property when destructuring, and if we add pattern matching, use different rules for patterns compared to their destructuring meaning.

BE talked to AR at dinner on day 2, thinks he heard this and may have agreed (to avoid breaking consensus). Need to confirm.

B. Defaults

Explicit undefined value triggerw use of default value initializer.

let foo = (x = 5) => x;

foo(undefined) // returns undefined by current draft
foo() // returns 5 by current draft

Issue: is this desirable? dherman and others think an explicit undefined should trigger use of default value. use case in support
function setLevel(newLevel=0) {light.intensity = newLevel}

```
function setOptions(options) {
setLevel(options.dimmerLevel); //missing prop returns undefined, should use default
setMotorSpeed(options.speed);
...
}
setOptions({speed:5});
```

Note same rules are used for both formal parameter default values and destructuring default values.

```
let foo = (…x) => x.length;

foo(undefined) // 1
foo() // 0
```

Need summary.
decision: change spec. to make undefine trigger use of default value.

C. Unresolved issues related to iterator naming/access

1. Be able to destructure things that did not opt-in
2. No implicit coercion
3. Array.from

spread works on array-like
destructuring has rest pattern

```
import iterator from "@iter"

function list(x) {
return iterator in x ?
[ y for y of x ] :
x;
}

[a, …] = list(jQuery(selector));
[a, …] = list([…]);
[a, …] = list(function *() { … });
```

f.call(f, …args)
same as
f.apply(f, args);

Summary:

(DH)
iterator is a unique name -- can't be public because iterable test not confined to for-of RHS

Destructing and spread - no iterator protocol.
(return to existing draft semantics of arraylike — [Cannot be both iterable and array-like])

Array.from:

- Will have array-like protocol, now iterable
- Will always return a copy

Array.from should… (this is a change to current specification)
1. Attempt to use the iterable protocol, if cannot…
2. Fall back to using Array-like protocol

(Filed: https://bugs.ecmascript.org/show_bug.cgi?id=588)

Continued...

 # July 25 2012 Meeting Notes

Present: Mark Miller (MM), Brendan Eich (BE), Yehuda Katz (YK), Luke Hoban (LH), Andreas Rossberg (ARB), Rick Waldron (RW), Alex Russell (AR), Tom Van-Cutsem (TVC), Bill Ticehurst (BT), Rafeal Weinstein (RWS), Sam Tobin-Hochstadt (STH), Allen Wirfs-Brock (AWB), Doug Crockford (DC), John Neumann (JN), Erik Arvidsson (EA), Dave Herman (DH), Norbert Lindenberg (NL), Oliver Hunt (OH)

Scoping Rules for Global Lexical Declaration

AWB:

1. Global scoping var vs. let and const declarations
var and function need to go on global object

2. What do we do with new binding forms?
(class, module, imports, let, const)
Q. Should these become properties of the global object?

DH: Not sure a restriction is needed, the global scope is the global object in JavaScript. With modules, globals are less of a problem.

YK: (clarification)

AWB, DH, BE: (providing background, e.g. on temporal dead zone for let/const/classs

BE: Agree there needs to be some form of additional info not in property descriptor

ARB: Need additional static scope information e.g. for modules. Need additional dynamic information for temporal deadzone.

DH: If you drop the idea that let is always let everywhere. Questions whether let should be more like var at global scope.

ARB: Does not work for modules.

AR: Reasonable to say that the global scope is never finished and that properties can continue to be defined

AWB: An example.

A const declaration; it creates a property on the global object; it's not defined yet; Before it's initialized another piece of code sets the value - what happens?

DH: (board notes)

1) 2 Contours, Nested "REPL"

- var, function go in global
- let, const, module, class… all get modeled lexically as usual in inner contour
- each script's inner contour is embedded in previous script's inner contour

2) 2 Contours, Not Nested "Uniform Let"

- var, function, go in global
- let, const, module, class… all get modeled lexically as usual in inner contour
- each script's inner contour is "private" to that script

3) 1 Contour, Global "Traditional"

- var, function, let, const, module, class… everything is a property of the global object.
- Additional scope refs in a side table of global, shared across scripts
- each script updates the side table

4) 2 Contours, Not Nested - Merged "Expando"

- var, function, go in global
- let, const, module, class… all lexical
- each script updates lexical contour of previous scripts

AWB: "Expando" was previously agreed upon, where the additional layer of lexical scope is available but shared. (Notes that Andreas did not buy into this)

DH: Agrees. Explains where "Expando" fixes the problems of "Traditional".

|---------|
| get x |
| set x |
| |---------|
	x: let

This would identify that "x" was declared with "let" and so forth.

STH:

A.
<s>
let x;
</s>
<s>
var x;
</s>

"Expando" (#4) Makes this an error

B.
<s>
let x = 1;
window.x;
</s>

C.
<s>
let x = 1;

</s>
<s>
x;
</s>

"Contour"/"Expando" both result in 1

D.
<s>
const x = 1;
</s>
<s>
if (null) {
x = 2;
}
</s>

"Contour"/"Expando" both result in no error

STH: Final debate that remains is reification on the window object. Allen is not in favor.

ARB: In favor of reification; but would like to get rid of the global object someday.

DH: Points out that non-reification will result in "WAT" from community (I agree).

Discussion about module unloading…

BE: Let's talk about unloading as a seperate, secondary conversation.

DH: Keep the global garbage dump as is - maintain consistency

AWB: No objection to a global garbage dump.

DH: If we add complexity to to the global mess, there is no win.

DC: Global is mess, we can't change it. Arguments are that consistency win, but we have an opportunity to clean this up. var can remain the same, but let is a new thing and we can afford it new behaviours.

RW: I agree, but we need to stop claiming "let is the new var" because general population will take that literally. If let has different behaviour, then "let is the new let".

DH: If you consider each script to be a "block" ie. { block }

YK/DC: Agree

DH: I have a crazy alternative… We could special case unconditional brace blocks in … scope. If you write a pair of block braces at the global scope and a let inside it, it will exist in that scope, but not global. functions, var hoisted out of the block brace.

#2 and #3, most coherent options.

DH: function prevents us from having a coherent story about implicit scope.

STH: Might want to do something other than reification

BE: Disagree with imputing curlies as way of illustrating a <script>'s top level scope.

DH: Need some way to explain where that scope lives.

OH: If you explain that let identifiers only exist in one script tag, developers will understand that.

RW: Agree.

AR: Agree, but they will say it's wrong

YK, BE: Agree with AR

AR: (Explanation of strawman developer concept of lexical ownership)

ARB: Also, want to be able to access e.g. modules from HTML event attributes

YK: The concat reality.

DH, BE: Agree on opposing concat hazards

Summary: #3 is the path. Champions spec this out and present for next in-person. (AWB, ARB, DH, RW)

Object.observe
(Presented By Rafeal Weinstein)
http://wiki.ecmascript.org/doku.php?id=strawman:observe

obj
{}[[Notifier]]
---->
{}N
[[ChangeObservers]]
[[Target]]
<----
obj

Object.observe
Object.getNotifier(obj).notify(changeRecord);

{}Function
[[PendingChangeRecords]]

When a data property is mutated on an object, change records are delivered.

[[ObserverCallbacks]] Used to order delivery

Object.deliverChangeRecords(callback);
…mitigates side-channel communication by preventing change records from escaping.

Explanation of specification history and roots in newer DOM mutation mechanism.

AWB: Is this sufficient for implementing DOM mutation event mechanisms?

RWS: Yes, those could be built on top of Object.observe

AWB/AR: Good, that should be a goal as well.

TVC: [If you're in the finalization phase and another observation is triggered, what happens]?

MM: FIFO event queue, deliveries run to completion

DH: Consider a two level, nested event queue

RWS: Very close to internal event queue, but is not. A single observer is being delivered changes, but not necessarily in the order that they occurred.

YK/RWS: Agree on delivery of script data mutation first, in any context.

RWS: Explanation of how mutation is handled and data binding as a whole.

DH: Concerned that it's too complicated and may conflict with expectation of run-to-completion.

RWS: Agree, but feel as though it is unavoidably complex, but this is for library authors to build better data binding abstractions.

YK: Can confirm that this proposal addresses web reality pain points.

DH: Not sure there is a good policy for knowing when to process what and when on a queue.

(stepped out, missed too much, need fill in)

TVC and RWS discussion of how Proxy can benefit from Object.observe
Unless/until we have an actual use case for virtualizing the observation system, don't let proxies virtualize observation: proxies have their own internal notifier like normal objects. Object.observe(proxy, callback) registers callback on the proxy. Proxy handler needs to actively observe target and re-notify its own observers for observation to work transparently across a proxy.

AWB: Concerns about whether the overall complexity is something that belongs in a general purpose language spec

LH: The complexities are such that they meet half way between policy that allows for too much, and for not enough.

DH: Agrees, the conversation has been helpful and agree that the complexity is on the right track for the right reason. Need to ensure that the right middle ground is met. Maybe current state is too high level, but closer than original too low level state.

BE: agree with DH, want to avoid premature/overlarge spec, do want implementation and user-testing. Let other impls know when spec is ready for trial impl.

Summary of next steps:

DH: Coordinate with YK colleague, to do real world work. Update TVCs prototype? Implementation prototypes.

(How to leverage developers to work on mini projects with prototype implementations)

RW: Would like to get access to a build that I can bring back to devs at Bocoup, where we can put dev resources towards developing projects with Object.observe; for example converting existing Backbone applications, etc.

RWS: Agree and will arrange.

Weak References

DH: The GC issue.

MM: A security concern, how to determine what is adequately privileged. WeakMap does not have this issue, WeakRef does

YK: WeakMap doesn't meet the use case

DH: WeakMap meets its own use case really well. WeakRef portability issue: non-determinism. If the web relies on un specified behaviour, you get defacto "worst case scenario".
Safer: only null between turns, as the web does today? If we go with traditional WeakRef, it's conceivable that the non-determinism is not an issue. Again, safe if in event turns.

Discussion about determinism/non-determinism.

Discussion about finalization, and whether it is a necessary part of the proposal. MM considers it important, AWB, ARB think it's too much of a hazard. Agreement at least that weak refs are useful without.

Only considering post-mortem finalization (finalizer does not have access to the collected object; it's already been collected), so no "zombie revival" issues.

BE: programmers will expect some sort of promptness to finalization, whereas it's not possible to provide any such guarantees; not testable

YK: frameworks will have to periodically eagerly collect empty WeakRefs myself, which they can live with, but it's definitely less convenient; anyway, setTimeout FTW

Script Concat Issue

DH: remember that the purpose of ES6 modules is to do sync-style loading without runtime blocking on I/O; this means that if you want to do configuration before loading, you have to *run* one script before *compiling* another:

```
<script defer>
System.set("@widget", patch(System.get("@widget")));
</script>
<script defer>
import widget from "@widget";
</script>
```

not the same as…

```
<script defer>
System.set("@widget", patch(System.get("@widget")));
```

import widget from "@widget";
</script>

Not possible for people to concat scripts for deployment and have the configuration happen before the loading

Submitting for discussion: the shebang as "a concat seperator" that…

- Fixes the concat ASI hazard
- Allows for artificial parsing boundary
- Note that this will change semantics of var hoisting

EA: concatenation of modules will require non-trivial compilation anyway; there will be ways to do this kind of thing with translation, without needing built-in support

DH: and loaders also make it possible to deploy multi-file formats

Discussion about the reality of concatenation hazards of modules

Defer, but still open for future discussion.

Fix "override mistake"

The can put check

```
var p = Object.create(null, {x: {writable:false, value:42}, y: {{get: function(){return 42}}})

var o = Object.create(p);

o.x = 99;
o.y = 100;
```

Property in a prototype object that is read-only cannot be shadowed.

Just the same as get-only accessor.

Causes SES/Caja grief on impls that follow spec. Must replace proto-props in prototype object to be frozen with accessors where the set function manually shadows.

Summary: There is no change for now, needs to be looked at when subclassing is addressed.

 # July 26 2012 Meeting Notes

Present: Mark Miller (MM), Brendan Eich (BE), Yehuda Katz (YK), Luke Hoban (LH), Rick Waldron (RW), Alex Russell (AR), Tom Van-Cutsem (TVC), Bill Ticehurst (BT), Sam Tobin-Hochstadt (STH), Allen Wirfs-Brock (AWB), Doug Crockford (DC), John Neumann (JN), Erik Arvidsson (EA), Dave Herman (DH), Norbert Lindenberg (NL), Oliver Hunt (OH)

Maxmin class semantics

YK: namespacing pattern: class that goes inside existing object; like Ember.View

DH: `Ember.View = class ...`

AWB: or `Ember = { View: class ... }`

AWB: early error list
- naming class eval/arguments
- duplicate class element names
- extends expression contains a `yield`
- method name constructor used on get, set, or generator

MM: `yield` should not be an error!

DH: definitely not! burden of proof is on the rejector; there's no reason to reject here

YK: why can't we do a getter?

DH: there's no way to declaratively figure out what the actual function for the class is, because the getter *returns* the function

AWB: class declarations create const bindings

AR: can you justify?

AWB: why would you want to overwrite it?

RW: what about builtins needing to be patched?

DH: those are independently specified to be writable; the relevant question is whether user programs will want to patch up local class bindings

AWB: whether this is a good idea probably depends on whether you're a library writer or application writer; if you aren't exporting class definitions

AR: you could still say `const x = class`

YK: that distinction isn't useful; every app has stuff like libraries

AR: restriction needs justification

DC: my preference is only for the expression form so there's no confusion

RW: surveyed ~200 developers, majority did not want const bindings by default

MM: I like crock's suggestion, just don't do the declarative one

EA: what?

LH: that's just putting cost on everyone else rather than us

MM: no, I'm talking about saving the cognitive cost to user

YK: if we went with const by default, I'd agree we shouldn't do declarative

AR: goal is most value for shortest syntax, without footguns; the analogy with const seems tenuous

AWB: this is subtle, and most people won't even notice

DH: I don't buy that there are significant errors being caught, there's no benefit to engines, there's not enough benefit to users, and it's clear there are costs. so I don't see any reason to do const binding by default

<<general agreement>>

MM: I'm opposed to declarative form. but if it is going to be declarative, should pick a declarative form and say it's the same as that, and let is the only clear candidate

DH: I'm not convinced function is impossible

MM: the expression extends is the killer. makes it impossible

LH: I'm convinced it can't hoist

DH: why not a more restricted syntax for declarative form in order to get hoisting?

```
{
class Sup extends Object { ... }
class Sub extends Sup { ... }
}
```

LH: surprising that you can't compute the parent

DH: there are surprises in each alternative we've talked about here; but I claim it's surprising to lose hoisting

OH: relevant analogy here is the fact that other languages with declarative classes don't care about order

LH: CoffeeScript does; it translates to `var x = ...`

AR: pulse?

DH: I think we all acknowledge this is tricky; I feel strongest that leaving out the declarative is failing in our duty

MM: if we leave out the declarative, then people will simply learn that the language is let c = class

BE: why are we debating this?

STH: Mark and Doug are arguing it

BE: over-minimizing and failing at usability

YK: `let x = class extends Bar { }` is just crazy

DH: that's laughable as the common case

AWB: this came from the hoisting debate

BE: I thought we agreed to dead zone. if we get stuck on this we'll never finish classes

LH: agreed; we need a separate proposal for hoisting

DH: happy to revisit later if I can come up with better alternatives

MM: we have adequate consensus that declarative desugars to let

AWB: classes are strict?

STH: I thought class did *not* imply strict mode

AR: does *anyone* want that?

<<no>>

AWB: default constructor has empty body? we'll get back to this

AWB: local class name scoping? similar to named function expression, but const bound?

DH: const bound?

AWB: just like NFE

DH: I actually didn't know NFE's had a const binding!

AWB: is this a bug? should we reconsider?

MM: avoids refactoring hazard

MM: my first choice would be to fix function: within function body its name is const; second choice is for class to be consistent

BE: not sure why we're talking about this, can't be changed

MM: in that case the class expression form should follow NFE

<<general agreement>>

DC: I disagree with the scoping decision about class declarations

DH: confused what we're talking about

STH: in body of class declaration, should there be a fresh scope contour

OH: it's not uncommon to overwrite the class

MM: example:

```
class Foo {
self() { return Foo }
}
```

...
new Foo().self() === Foo // can fail

this is very confusing for this to fail

DH: why would you ever want the extra scope contour?

STH: Rick gave a good example:

```
class C {
m(x) { return x instanceof C }
}
var y = new C;
C = 17
y.m(y)
```

DH: not compelling; you mutated C! if you need the earlier value, you should save it; the confusion would only arise if you expected C to be a static class like in Java, but that's not how JavaScript bindings work

RW: the common pattern being the defensive-constructor pattern:

```
function C() {
if (!(this instanceof C)) {
return new C();
}
...
}
```

DH: now I'm that much more confident that there should not be another scope contour; I don't see any compelling argument

AWB: let me throw up another justification: class declarations often appear at global scope, not uncommon for somebody to write class body where there are references to the class; at global scope, anybody could have assigned to that value

DH: I don't want to poison non-global cases just to protect against one hazard of global code, when global code is hazardous anyway

AWB: I would put protecting global code at a higher priority than a subtlety of inner bindings, but I'll go with the flow if I can't convince you

DC: I don't want to hold this up

MM: are you willing to go with the function parallel?

DC: yes; I don't prefer it but I won't hold this up

AWB: missing extends, what's the default? intrinsics

<<agreement>>

AWB: extends null: prototype is null, Foo.[[Prototype]] extends intrinsic Function.prototype

<<agreement>>

AWB: extends a constructor:

class Foo extends Object { }
Foo.[[Prototype]]: (Object)
Foo.prototype.[[Prototype]]: (Object).prototype

IOW, class-side inheritance

MM: I disagree, the history of JS does not have it

BE: I disagree with that claim, history shows some examples on both sides

EA: people do refer to `this` in static functions; they have the freedom to use the class name or `this`, and they do both

LH: CoffeeScript does class-side inheritance, but they don't do it like this -- they copy

BE: but they will avoid the copy once you implement dunder-proto

MM: you can't depend on it

BE: this gives programmers more flexibility to do it however they want

MM: but then people can't use a this-sensitive function!

BE: not true, the contract of a JS function includes its this-sensitivity

Arv, AR: <<nod visibly>>

LH: at end of day, plenty of static functions in JS that are this-sensitive

YK: that's the style of program that I write

EA: some style guides say don't do it

LH: backbone does this

MM: so Foo will inherit Object.create, Object.getOwnPropertyDescriptor, etc?

DH: that does mean we'll be more and more hampered from adding methods to Object

EA: but now we have modules

BE: true, that's the right answer

MM: polluting of statics with everything in Object is fatal; those are just not relevant to most of the class abstractions people write; when I write

class Point { }

I don't want Point.getOwnPropertyDescriptor

AWB: you only opt into that with class Point extends Object; with class Point { } you don't get any of that stuff

DH: <<feels giddy and sees the clouds part and sun shining through, with angels singing from on high>>

YK: also, there are override hazards of pollution: if someone freezes Object, then you wouldn't be able to override sweet class method names like keys(), so the ability to avoid that pollution is important

MM: valid point. thing is, we don't have static form b/c you can supposedly use imperative assignment, but that won't work for frozen classes

BE: that's just an argument for statics in the future

AWB: minimality ftw

AWB: class Foo extends Object.prototype?

LH: this surprised me when I saw it in the spec

AWB: older version of class proposal had a "prototype" contextual keyword for this case

DH: what happens if you're defining a meta-class? you can't tell whether it's a prototype or a constructor

BE: that's a smell

AWB: constructor trumps object

BE: YAGNI, cut it

AWB: so what do we do if it's not a constructor?

DH: throw

BE: that's more future-proof

<<general agreement>>

AWB: extends neither an object nor null: type error

DH: actual type errors in JS, yay!

RW: curious: what if Foo extends { ... }

DH: non-constructable, so error; but could use a function literal

AWB: extends value is constructor but its prototype value is neither an object nor null: type error (existing semantics of new: silently uses Object.prototype)

<<agreement>>

AWB: Foo.prototype is an immutable binding? builtin constructors are immutable, user function(){} mutable

<<some surprise>>

MM: make .constructor mutable but .prototype immutable

YK: why? (I want mutable)

MM: nice for classes for instanceof to be a reliable test

YK: why?

AWB: classes are higher integrity; association between constructor and prototype actually means something now

BE: I'm moved by higher-integrity, self-hosting with minimal work

STH: not compelling to make self-hosting *easy*, just possible; defineProperty is just fine for that

DH: most everyone seems to agree that .prototype is immutable, .constructor is mutable. Arv and AR, thoughts?

EA: that's fine

AR: yup, that's fine

AWB: method attributes: sealed? (writeable: true, configurable: false, enumerable: false)
- configurable: false -- you've established a specific shape

YK: you don't want to switch from a data property to an accessor?

AWB: non-configurable but writable is reasonable

MM: this depends crucially on our stance on override mistake; this prevents me from making an accessor

AR: I don't see why we're considering making this anything other than writeable: true, configurable: true

BE: Allen feels having the shape be fixed is useful

<<discussion>>

BE: so consensus is writable: true, configurable: true

<<agreement>>

AWB: methods are not constructable?

DH: what?

MM: biggest benefit: this further aligns classes with builtins

MM: three reasons for this:
1. precedent in builtins
2. using a method as a constructor is generally nonsense
3. to freeze a class, I have to freeze the .prototype of the methods on the prototype!!

LH: compelling for me: never seen a class-like abstraction on a prototype of a class-like abstraction

MM: I have, but you still can; just do it in a way that's obvious, don't do it with method syntax

BE: hard cases make bad law! (agreeing with MM -- use a longhand)

YK: so you can say classes really only existed as builtins, now they're expressible

AWB: get/sec accessors *are* constructors? that's just the way they are in ES5

BE: is there precedent in builtins?

AWB: nothing explicit

YK: I'd prefer consistency between these last two cases

AWB: accessor properties on prototype are enumerable

BE: what about DOM/WebIDL? accessors on prototype?

LH: they're enumerable, yes

AWB: suggestion: concise methods should be the same for both classes and object literals
- strictness
- enumerability
- constructability
- attributes

AWB: breaking change from ES5: get/set functions non-constructable

AWB: class accessor properties:
- enumerable: false, configurable: false

AR: no

EA: no

YK: when you use an accessor you're trying to act like a data property

BE: so compelling argument is: accessors are enumerable, configurable, and writable

AWB: Luke suggests that default constructor should do a super-constructor call with same arguments `constructor(...args) {super(...args)}`

BE: default constructor in CoffeeScript, Ruby

AWB: perhaps needs to test for Object constructor and not call it

DH: no observable difference!

MM: if there's no observable difference, go with simplest spec

AWB: other places where we do implicit super call? I say no

DH: I say no.

LH: I agree, I think there's no clear way for us to do it, but I also think there will be many, many bugs

BE: irreducible complexity here, caveat refactorer

getPrototypeOf trap

TVC: (introduction)

__proto__ writable destroys invariant that [[Prototype]] link is stable

Frozen objects should continue to have stable prototype chain

getPrototypeOf trap result should be consistent wth target object's proto

MM: if the proto can be changed, the proxy should…?

TVC: spec interceptable [[Prototype]]

[[Prototype]] is currently an internal prop

Would need to become internal accessor prop or split into [[GetProto]] / [[SetProto]]

[[GetProto]] / [[SetProto]] would trigger traps for proxies

AWB/BE: This is good

YK: Do we want an analogous setPrototypeOf trap?

TVC: Yes

AWB: If you have capability to set prototype ?

TVC: proxy.__proto__ should just trigger the proxy's get trap

var p = Proxy(target, handler)

p.__proto__ // => handler.get(target, "__proto__", p)
p.__proto__ = x // => handler.set(target, "__proto__", x, p)

…

Trapping instanceof

Function [[HasInstance]]

x instanceof Global answering true if x and Global live in separate frames/windows

var fp = Proxy(targetFunction, handler);

x instanceof fp // handler.hasInstance(targetFunction, x)

MM: Explains concerns originally raised on es-discuss list by David Bruant, but shows the cap-leak is tolerable
…

DH: if hasInstance private name on instanceof RHS...

MM: What Object.prototype does private name inherit from?

AWB: Probably null

BE: the E4X any (*) name had null proto in SpiderMonkey, was true singleton in VM

AWB: functions have home context, but no reason for objects to

DH: this is a new idea of value that is not really any object

OH: if it has no properties and no prototype

BE: cannot be forged.

Discussion about unforgeability.

DH: Trapping instanceof use case

Trapping Object.isExtensible

Currently Object.isExtensible doesnt trap same for isSealed isFrozen

var p = Proxy(target, handler)

Object.isExtensible(p) => Object.isExtensible

Direct Proxies: "internal" properties

Issue raised by Jason Orendorff; auto unwrapping is dangerous if built-in methods return non-primitive values

Case:

```
var arr = [o1, o2, o3];
var it = arr.iterator();

var membraneP = wrap(it);

it.next.call(membraneP)
```

Solution (?)

Instead of auto-unwrapping, delegate to a nativeCall trap (which auto-unwraps by default)

[[PrimitiveValue]]

BE: nativeCall trap is back door between built-in this-type-specific method impls and proxies. Not good for standardization. Better to make such built-ins generic via Name object internal property identifiers, a la AWB's subclassing built-ins strawman

Discussion moved to Subclassing…

MM: re: what you want syntax wise

AWB: one way to address, not use instance that is automattically created, create new array and patch the proto

… BE: (back to nativeCall trap)

AWB: Let's continue the issue of subclassability on es-discuss

TVC: defaultValue slide

See slide?

BE/AWB: defer this to reflect spec handling, non-observable way.

Proxies and private names

TVC: getName(target, name.public) instead of get(target, name.public) -- this way get trap that doesn't expect name objects won't break on unexpected inputs

DH: has, delete, ...? bigger surface area

TVC: you'd still have to branch in the code, so this is cleaner for user

YK: debugging tool will want to be able to see these things

OH: a built-in debugger will have hooks into the VM

YK: many debuggers use reflection

BE: so it's just a matter of having a bunch of XXXName traps. in for a penny, in for a pound

STH: this is simple and straightforward, we know how to do it

BE: when in doubt use brute force (K. Thompson)

STH: when brute force doesn't work, you're not using enough of it

TVC: if getName returns undefined, forwards to target; so default behavior is transparent proxying

TVC: otherwise, getName takes public name and returns [privateName, value] to show that you know the private name and produce the value

STH: what about set?

TVC: returns name and success value

DH: what about unique names?

TVC: same mechanism

DH: so name.public === name?

MM: I like that

MM: are unique names in?

DH: I think so

BE: are they actually distinguishable?

MM: have to be if name.public === name or name.public !== name distinction

DH: (named) boolean flag to Name constructor

DH: do we have some way of reflecting unique names?

TVC: Object.getNames() ?

DH: ugh...

AWB: maybe a flag to Object.getOwnPropertyNames({ unique: true })

BE (editing notes): flags to methods are an API design anti-pattern

TVC: VirtualHandler fundamental traps throw, should they forward instead?

<<agreement>>

TVC: and rename to Handler?

<<agreement>>

MM: next issue: freeze, seal, defineOwnProperties each modify configuration of bunches of separate properties, and can fail partway through; we tried & failed in ES5 to make it atomic

MM: current unspecified order means could break

MM: tom did something in his code that's beautiful: order-independent. just keep going, remember you failed, do as many as you can, and then throw at the end

STH: if target is proxy, weird unpredictably stuff can happen

DH: no worse than anything that does for-in loops, right?

TVC: well, it's getOwnPropertyNames

MM: that's specified to for-in order, right?

DH: but what does for-in order say about non-enumerable properties? <<evil grin>>

MM: <<cracks up>>

AWB: sounds like an ES5 bug!

VirtualHandler

VirtualHandler
Rename VirtualHandler to just Handler?

Tom Van-Cutsem's Proxy presentation slides:

http://soft.vub.ac.be/~tvcutsem/invokedynamic/presentations/TC39-Proxies-July2012.pdf

Template strings

AWB: first order of business, to ban the term "quasis"

<<applause>>

AWB: proposing "string templates"

DH: a lot of people say "string interpolation" in other languages

AWB: must use ${identifier}, don't allow $identifier

EA: uncomfortable with that

BE: troublesome to identify right end of identifier

EA: withdraw my objection

AWB: untagged quasi is PrimaryExpression, tagged quasi is CallExpression

AWB: at runtime, tag must evaluate to a function

DH: well, you just do a call and *that* does the check

AWB: lexing treated similarly to regexp; add a new context called "lexical goal" so lexer can tell what a curly means (like a flex(1) mode)

AWB: default escaping should be equivalent to normal strings

BE: we should canonicalize line separators to \n

AWB: for both cooked and raw?

BE: raw should be raw!

AWB: raw tag is a property of the String constructor:

String.raw`In Javascript '\n' is a line-feed.`

DH: that's pretty badass

BE: too long a name; wanna import a small name from a module

AWB: well, importing takes more characters than renaming with a var declaration

BE: let's put off the bikeshed in the interest of time

AWB: simplify call site object (first arg to prefix-tag function): it's just an array of the cooked elements since that's the common case, with a .raw expando holding array of the raw elements, both arrays frozen

BE: is there a grawlix problem with ` syntax?

DH: I've tried polling and opinions are utterly mutually incompatible

BE: what about mandated prefix but with existing e.g. ' or " quotes

LH: that's just wrong, the most common case will be unprefixed

MM: proposal for object literals inside ${...} context, based on object literal shorthand {foo} meaning not {foo:foo} but rather {get foo() foo, set foo(bar) {foo=bar}} to sync variable foo with property (!)

STH: that is going to be utterly unexpected

MM: ok, not gonna argue for it

Map and Set methods: conclusion

AWB: what's left on the agenda?

RW: Erik is gonna take another whack at the error stack proposal

BE: forEach on maps and sets -- how about common signature, set passes e as index:

```
array a.forEach((e, i, a) => ~~~)
map m.forEach((v, k, m) => ~~~)
set s.forEach((e, e, s) => ~~~)
```

FILED: https://bugs.ecmascript.org/show_bug.cgi?id=591
FILED: https://bugs.ecmascript.org/show_bug.cgi?id=592

Scoping for C-style loops

NL: the wiki page for `` makes it sounds like they solve problems for internationalization/localization, and they don't

DH: I'd love help with a documentation hack day for the wiki

LH: another agenda item we skipped: for (let ; ;) binding semantics

DH: I thought we came to agreement on that at the Yahoo! meeting?

AWB: we had a long discussion and consensus was to make for (let ; ;) bind on each iteration

AWB: subsequent to that, considerable discussion on es-discuss about that, issues associated with closure capture occurring in the initialization expressions; couple different semantics to work around that, with more complex copying at each iteration; another approach is a new kind of Reference value, got really complex

AWB: working on the specs, I took easy way out for now; defined it a la C# (per-loop lexical binding); just for now b/c it's simple, understandable, and there's still controversy

AWB: another option is not to have a let for of C-style loops

STH, DH, OH: no!!!

DH: this needs another trip around the block but no time today

MM: my opinion is it doesn't matter what happens with closure capture in the head, b/c it's an esoteric case that will be extremely rare

BE: I think the January semantics is still probably the right answer:

```
var g;
for (let f = () => f; ; ) {
g = f;
break;
}
g(); // returns () => f
```

OH: it logically makes sense

_____

es-discuss mailing list
es-discuss@mozilla.org
https://mail.mozilla.org/listinfo/es-discuss