| | |
|---|---|
| *Minutes for the:* | *33rd meeting of Ecma TC39* |
| *in:* | *Sunnyvale, CA, USA* |
| *on:* | *12-14 March 2013* |

## 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, Adobe, Stanford University, Intel, Apple

### 1.2 Introduction of attendees

John Neumann - Ecma

Adam Klein - Google

Edward Yang – Stanford Univ.

Dan Stefan – Stanford Univ.

Douglas Crockford - eBay

Waldemar Horwat – Goog le

Luke Hoban - Microsoft

Norbert Lindenberg - Mozilla

Allen Wirfs-Brock - Mozilla

Erik Arvidsson - Google

Rick Waldron - jQuery

Bernd Mathiske - Adobe

John Pampuch - Adobe

Avik Chaudhuri - Adobe

Alex Russell - Google

Edward O'Connor - Apple

Richard Hudson - Intel

Matt Sweeney - Yahoo

Eric Ferraiuolo - Yahoo

Yehuda Katz - jQuery

Sam Tobin-Hochstadt - Northeastern Univ

Dave Herman - Mozilla

Andreas Rossberg - Google

Brendan Eich - Mozilla

Rick Hudson – Intel

Rafael Weinstein – Google

Reid Burke – Yahoo

**Daily participation:**

# March 12 2013 Meeting Notes

John Neumann (JN), Norbert Lindenberg (NL), Allen Wirfs-Brock (AWB), Rick Waldron (RW), Waldemar Horwat (WH), Eric Ferraiuolo (EF), Erik Arvidsson (EA), Luke Hoban (LH), Matt Sweeney (MS), Doug Crockford (DC), Yehuda Katz (YK), Brendan Eich (BE), Sam Tobin-Hochstadt (STH), Alex Russell (AR), Dave Herman (DH), Adam Klein (AK), Edward Yang (EY), Dan Stefan (DS), Bernd Mathiske (BM), John Pampuch (JP), Avik Chaudhuri (AC), Edward O'Connor (EOC), Rick Hudson (RH), Andreas Rossberg (ARB), Rafeal Weinstein (RWN), Mark Miller (MM)

# March 13 2013

John Neumann (JN), Norbert Lindenberg (NL), Allen Wirfs-Brock (AWB), Rick Waldron (RW), Waldemar Horwat (WH), Eric Ferraiuolo (EF), Erik Arvidsson (EA), Luke Hoban (LH), Matt Sweeney (MS), Doug Crockford (DC), Yehuda Katz (YK), Brendan Eich (BE), Sam Tobin-Hochstadt (STH), Alex Russell (AR), Dave Herman (DH), Adam Klein (AK), Bernd Mathiske (BM), John Pampuch (JP), Avik Chaudhuri (AC), Edward O'Connor (EOC), Rick Hudson (RH), Andreas Rossberg (ARB), Rafeal Weinstein (RWN), Mark Miller (MM), Reid Burke (RB), Edward Yang

# March 14 2013

John Neumann (JN), Norbert Lindenberg (NL), Allen Wirfs-Brock (AWB), Rick Waldron (RW), Waldemar Horwat (WH), Eric Ferraiuolo (EF), Erik Arvidsson (EA), Luke Hoban (LH), Matt Sweeney (MS), Doug Crockford (DC), Yehuda Katz (YK), Brendan Eich (BE), Sam Tobin-Hochstadt (STH), Alex Russell (AR), Dave Herman (DH), Adam Klein (AK), Bernd Mathiske (BM), John Pampuch (JP), Avik Chaudhuri (AC), Edward O'Connor (EOC), Rick Hudson (RH), Andreas Rossberg (ARB), Rafeal Weinstein (RWN), Mark Miller (MM), Reid Burke (RB),

## 1.3 Host facilities, local logistics

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

## 1.4 List of Ecma documents considered

| | |
|---|---|
| Ecma/TC39/2013/002 | International registration of the "ECMAScript" trademark in Japan |
| Ecma/TC39/2013/003 | Draft TC39 RFTG Scope |
| Ecma/TC39/2013/005 | Submission of ECMA- ECMAScript to the European Commission |
| Ecma/TC39/2013/006 | Draft minutes of the conference call of the IPR adhoc on 15 January 2013 |
| Ecma/TC39/2013/007 | Current status of files for the TC39 Royalty Free patent policy |
| Ecma/TC39/2013/008 | Draft minutes of the conference call of the IPR adhoc on 4 February 2013 |
| Ecma/TC39/2013/009 | Minutes of the 32nd meeting of TC39, San Francisco, January 2013 |
| Ecma/TC39/2013/010 | Venue for the 33rd meeting of TC39, Sunnyvale, March 2013 |
| Ecma/TC39/2013/011 | US trademark registration of "ECMAScript" on 22 January 2013 |
| Ecma/TC39/2013/012 | First draft Standard ECMA-402, 2nd edition, February 2013 |
| Ecma/TC39/2013/013 (Rev. 2) | Agenda for the 33rd meeting of TC39, Sunnyvale, March 2013 |

| | |
|---|---|
| Ecma/TC39/2013/014 | Draft minutes of the conference call of the IPR adhoc on 4 March 2013 |
| Ecma/TC39/2013/015 | Ninth draft Standard ECMA-262 6th edition, March 2013 |

## 2 Adoption of the agenda (2013/013-Rev2)

The agenda was approved.

## 3 Approval of minutes from January 2013 (2013/009)

The minutes of the January 2013 TC39 meeting have been approved as presented. Individuals that took technical notes were recognized and appreciation extended. **Rick Waldron** volunteered to take technical notes. The technical notes are included in Annex 1 of the minutes. They reflect the discussions correctly and in great details. The technical notes- have been already shared with TC39 and feedback was taken into account.

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

### 4.1 Revisit the @-names discussion/resolution from the November meeting.

A significant body of work has emerged that I feel makes a strong case towards making an exception to the cut-off deadline. I'd like to urge everyone to take a moment to review this: https://github.com/Benvie/continuum/tree/gh-pages/engine/builtins

### 4.2 Module update

**From Yehuda:**

**4.2.1 - Modules Field Report (ES6 Module Transpiler Usage)**

**4.2.2 - Modules Use Cases (probably should be combined with the broader modules discussion)**

**From last meeting:**

### 4.3 Proxy issues

### 4.4 Spec update

**From Mark Miller:**

### 4.5 Private Symbols, WeakMaps, and Relationships

("Relationships" is Allen's term, and comes from an enlightening conversation Allen and I had after the last meeting.)

### 4.6 Notification proxies

### 4.7 Runtime costs of the override mistake -- a report.

**From Rafael:**

### 4.8 I plan to present a brief "implementation report" on Object.observe which will include some minor changes to the spec text during implementation

**From Doug Crockford:**

### 4.9 JSON. IETF may issue a revision of the JSON RFC. The current RFC is informational. The next one will be an internet standard. There will probably be a correction that affects ECMAScript

**From Rick Waldron:**

### 4.10 Array Extras

- findIndex: https://gist.github.com/rwldrn/5079427

- find: https://gist.github.com/rwldrn/5079436

- unique (no spec written)

From Andreas:

### 4.11 Open questions/issues with generators

(https://mail.mozilla.org/pipermail/es-discuss/2013-March/029004.html)

From Arv:

### 4.12 StopIteration, generator return and alternatives to exceptions

From Norbert:

### 4.13 Identifying ECMAScript identifiers

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

From Allen:

### 4.14 Discuss resolution for bug 1257 Make new Date(dateObj) work as expected see also 5 edition 5.1 issues

https://bugs.ecmascript.org/show_bug.cgi?id=1257

From Brendan:

### 4.15 * Discuss Googler Adam Barth's objection to template strings, in the thread ending at this post from me:

http://lists.w3.org/Archives/Public/public-script-coord/2013JanMar/0242.html

* Should we let embeddings override the default handler for template strings?

From Luke:

### 4.16 We talked in January about doing an all-up status review on ES6 feature set and timeline during the March meeting. I don't see this on the agenda yet, but I assume it's still something we want to do.

### 4.17 Update on "block scoped function semantics in non-strict"

### 4.18 Clarification of expected rules for "eval declaration instantiation"

### 4.19 Clarification of spec plan for __proto__

## 5 Edition 5.1 Issues

## 6 Second edition of ECMA-402

### 6.1 Status report

Ecma/TC39/2013/012
http://norbertlindenberg.com/ecmascript/intl.html

## 7 Test 262 Progression

### 7.1 Status report

## 8 Status Reports

### 8.1 Report from Geneva

Comments from Istvan Sebestyen, stated by John Neumann:

- The ECMAscript trademark is rather through. We have it in the most important countries, incl. US and EU.

- The EU formal recognition for ECMA-402 and TR/104 is progressing. We are now in the evaluation phase by a small group. They asked me to make a first draft for evaluation, which I did. Informally I have one "yes" so far, and no "no" yet. If the small group gives an ok, then the member countries have to vote. The process will take a few months more (but I do not know how long, because this is the first experience with the new EU policy).

- Regarding the IPR Adhoc. We are progressing. I think I can distribute an updated package to TC39 at the end of March. GA vote is planned for June 2013. Comments back to the IPR Adhoc, GA, or GA members. Just as last time.

WH: ECMA's TC39 email reflector is incompatible with gmail. If a gmail user sends an email to the reflector, non-gmail users will receive it but no other gmail user will receive it in his inbox. It will always arrive in the spam folder due to some SPF issue.

[Various people checking their spam folders now and discovering TC39 emails in there.]

## 9 Date and place of the next meeting(s)

**Schedule 2013 meetings:**

- May 21 – 23, 2013 (Google - London)
- July 23 – 25, 2013 (Microsoft - Redmond)
- September 17 – 19, 2013 (Bocoup - Boston)
- November 19 – 21, 2013 (PayPal - San Jose)

## 10 Closure

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

# Annex 1

**Technical Notes (by Rick Waldron):**

| | |
|---|---|
| **From:** | Rick Waldron [waldron.rick@gmail.com] |
| **Sent:** | Monday, March 18, 2013 4:58 PM |
| **To:** | TC39 |
| **Subject:** | March 2013 Meeting Notes |

March 12, 13, 14 (combined):

# March 12 2013 Meeting Notes

John Neumann (JN), Norbert Lindenberg (NL), Allen Wirfs-Brock (AWB), Rick Waldron (RW), Waldemar Horwat (WH), Eric Ferraiuolo (EF), Erik Arvidsson (EA), Luke Hoban (LH), Matt Sweeney (MS), Doug Crockford (DC), Yehuda Katz (YK), Brendan Eich (BE), Sam Tobin-Hochstadt (STH), Alex Russell (AR), Dave Herman (DH), Adam Klein (AK), Edward Yang (EY), Dan Stefan (DS), Bernd Mathiske (BM), John Pampuch (JP), Avik Chaudhuri (AC), Edward O'Connor (EOC), Rick Hudson (RH), Andreas Rossberg (ARB), Rafeal Weinstein (RWN), Mark Miller (MM)

## Opening

## Introduction

## Logistics

## Adoption of Agenda

Mixed discussion regarding scheduling over the course of the 3 days.

Approved.

## Approval of January 2013 Meeting Notes

Approved.

## Adobe

JP: Here to help accelerate the ES6 spec, positive motivation. Excited about Modules, concurrency, debugging and profiling specifications.

BM: Background as trained language designers and implementors and here to help.

JP: Also excited about asm.js

BM: Not sure about the spec status/prospects of asm.js.

## 4.9 JSON, IETF changes

(Presented by DC Crockford)

Currently, JSON is an RFC, informational, the IETF version will be an internet standard and there is a minor correction that affects ECMAScript.

The use of "should" in 15.12.2

AR: What is the motivation of the change?

DC: The change involves the mistake of using "should" w/r to multiple same-named keys error. Multiple same-name keys are invalid and _must_ throw an error (vs. "should" throw an error)

LH: This is a breaking change

DH: The worst being the use case of multiple, same-named keys as comments

DC: That's stupid

YK: That's based on on your recommendation to use a keyed entry as a comment, so people naturally used the same key, knowing they'd be ignored.

DC: I would certainly never recommend that practice

YK: It was a side-effect

AR: Which key is used now?

AWB: The last one wins.

AR: Is that the root of the security vector?

DC: Not in ES, but in other encodings

AR: Order matters, unescaped content that follows...

DC: The current spec says "[they should not]", but will say "[they must now]"

YK: Let's define an ordering and make it cryptographically secure.

DC: (recapping to Mark Miller, who just arrived)

MM: You can't do that.
(laughs)

MM: You can't change "should" to "must"

YK: Agreed, you cannot change JSON, there are too many JSON documents in existence.

MM: Agreed.

AR: It's possible to ignore this change?

DC: Yes

DH: Then why are we creating a dead letter?

MM: ES has a grammatical specification for validating and parsing JSON. Anything that is not conformant JSON, would not parse. This change loses that property.

DC: Or we don't change the spec

MM: The way that you properly reject our favorite fixes, I think you should apply to your favorite fixes

DC: I'll consider that

AR: There is considerable opposition to this change

DC: Two choices...

1. Make it an error
2. Continue to take the last one

DC: Decoders have license to do what they want with non-conformant material. Encoders _must_ be conferment to new changes.

MM: Our current encoder conforms...

AWB: I don't think it does... reviver/replacer

MM: No, can only apply objects instead of the original objects.

AR: Did not realize the production/consumption distinction of this change.

WH: Supports this change. ECMAScript is already conformant because it never generates duplicate keys.

MM: With this change ECMAScript would have two unappealing choices:

A. No longer be a validating parser (i.e. a parser that doesn't allow any optional syntax or extensions, even though extensions are permitted by the JSON spec).

B. Do a breaking change by throwing errors when seeing duplicates when parsing.

**Conclusion/Resolution**

- Revisit this, after DC has made a final decision.
- FTR: Majority opposition, no consensus.

## 4.12 StopIteration/Generator

(Presented by Dave Herman)

https://github.com/dherman/iteration-protocols

DH: ...Confirms that there is lack of understanding for Generator "instances"

MM: Clarify terminology

DH: Generator objects are instances of Iterators, in that they have a `next` method. The current design is based on a thrown `StopIteration`

C#, Java, Python apis for comparison.

My opinion, a single method is win.

Concrete proposal:

- The benefit of a single method
- Not based on exceptions
- Compatible with Generator semantics

Iterator API object has a single method:

```js
{
  next() -> { done: false , value: any }
          | { done: true[, value: any] }
}
```

b/c generators can return an argument, if you're using a return value

```js
function* f() {
  yield 1;
  return 2;
}
```

for `for-of` this doesn't matter, but for libs like q, task.js this is useful for pause and resume with value

If we didn't care, the result value can be omitted

This API was pleasant to write and works nicely with existing idioms

MM: Requires allocation for every iteration?

DH: Yes, will still need the object allocation, but

WH: Does next return a fresh object? or can reuse the same?

DH: Can reuse

AWB: For every iterator in the spec, we need to specify a fresh or reused object?

DH: Yes.

YK: The current API, able to do yield ...returns a promise...

DH: Can still do that, this change is internal and w/r to performance, this should be highly optimizable.

AWB: Anything that uses a method based implementation, will be more optimizable through calls vs exception.

DH: I've never seen an iterator API that didn't have some performance issues

AWB: (refutes) Any method based approach can be better optimized over exception based approaches.

DH: I don't have a solid performance story, but the feedback I'm getting is that there is negative concern about the StopIteration approach, whereas this approach mitigates these concerns. Issues arise when dealing with multiple iterators

WH: If you try throwing StopIteration across iterators, it will be caught

AWB: Or it won't

EA: Surprising: If any function throws a StopIteration, it will jump out of the for-of.

AWB: I noticed this in the examples shown in the github repo

WH: Why I'm in favor... Throwing StopIteration across places where no iterators exist and if code is refactored so that an iterator is present, you'll experience unexpected behavior. (This suffers from the same capture flaws as Lisp's old dynamic scoping.)

LH: When we last talked about this, we whiteboarded the priority order. for-of is the primary case, generator authoring is the secondary case. Cases affected by this: direct API consumption as third and direct API authoring is fourth

If we're really sure the engines will do the work to optimize these things?

A. will this slow down implementation?

B. won't be willing to implement due to poor performance?

AR: No implementation wants to ship something that will potentially be slow

LH: Of course, but StopIteration has to go.

MM: One allocation per loop

WH: So is this

MM: Only if you reuse the record

LH/WH: Of course and that's what you want

MM: Then, as Allen said we need to specify this

DH: My inclination would be to use a fresh object each time

AWB: ...you know the first time, because it's the first time that next is called,

MM: My proposal is that you provide stop token as a parameter of next(stop), every time. next(stop) would return either the next value or the stop token.

DH: (clarifying) "iteration" is one time around the loop. "loop" is the entire the operation.

WH: It's possible for next(stop) to cause havoc from one iteration to another by caching one next call's stop parameter and returning it from a different next call.

[Someone had also presented a variant of the proposal where <stop> was a field on the iterator instance instead of an argument to next.] WH: This would create funky failures if, for example, you had an iterator that did a deep traversal of an object tree and said tree happened to include the iterator instance.

MM: In order to not allocate on every iteration, you have specify (???)

MM: A new stop token would be generated per loop.

WH: What's a loop? This gets confusing when you have iteration adaptors.

AWB: If the client passes in the token on next(), then it's the client's burden

MM: Anything that's unforgable, unique, or itself side affectable.

DH: Is there support for Mark's API?

RH: If you use Mark's API, overtime...

MM: My API reuses the object for the iterations of the loop, by passing it back in as an argument to next()

RH: To avoid the cost of allocation?

MM: Yes, but only as a token

EA: You can have a return value in a generator so the object passed in needs to be mutated to include the return value in case of end of iteration.

MM: That is a downside of this proposal, where StopIteration would carry the value.

DH: (examples of the two proposals)

Dave's
```js
{
  next() -> { done: false , value: any }
          | { done: true[, value: any] }
}
```
Marks's
```js
{
  next(X) -> any | X
}
```

AWB: (suggests an alternative: pass an object to next, on which next sets the result)

STH: ...is hostile to implementors and user code.

ARB: That's the C-style of doing it.

WH: Suppose the iterator takes an object and returns all the properties, but calls on itself?

DH: Mark's proposal is broken, because it doesn't work with return values of generators.

MM: Agreed.

DH: Don't think that we're approaching consensus, but don't let your idea of perfect cloud judgement. I'm asking engine implementors if this is appealing. The concern over StopIteration is real.

AWB: This is certainly better then the current plan of record

AR: Agree.

BM, JP, AC: Agree

BM: This is also future proof and works well with concurrency and the semantics are sound. It's also easy to implement and optimize.

AWB: All spec iterators/generators _must_ specify a reused iterator or fresh

MM: (further support for AWB's claim)

DH: Not sure if we're trading short term wins for long term losses. Are there long terms

ARB: There is another secondary effect that it encourages better GC

AWB: This shouldn't be a problem for a good GC

MM: I'm really only concerned about the loop aspect

AR: We have the tools to work with hot loops

WH: Alex's point about the escape valve is key

DH: Not discounting the needs of the developers/user code. The method API is appealing, vs. StopIteration

RW: Agree.

DH: (Shows example of C#)

AWB: The third use case that Luke gave, using an iterator and the fourth use case, creating an iterator.
...This API is more complex for user code
...More ways for client code to go wrong

BM: Disagree, this is a safer.

DH: Don't get out of sync. Argue that the Java and C# API are too error prone.

BM: Agree, this is actually superior to the Java and C# APIs

ARB: This is actually the path you'd want in typed languages, minimizes state space

DH: I want to see a better overall interface for iterators and generators, without jeopardizing the acceptance.

MM: In favor of this API, if the implementors are not objecting.  Although I don't like the API itself.

DH: Agree, I prefer the pure, stateless proposal

AWB: If an object is passed an argument, the object is used as the value bucket.

DH: Still mutates the object

AWB: But it mutates an object that you've explicitly provided

BM: The issue is not the allocation, but that you have to go to heap at all.

ARB: If you do this pre-allocation thing, it might be observable

BM: But that's the case either way

DH: Is the mutable version going to harm optimization?

ARB: Yes: the object may be shared, in which case mutation may become observable from the distance, and cannot be optimized away

RH: If the object being mutated escapes to an old mutation, this kills potential optimizations.

DH: Seems like consensus on the pure, stateless version of this:

```js
{
  next() -> { done: false , value: any }
         | { done: true[, value: any] }
}
```

JP: "more" vs. "done"?

(can discuss further)

**Conclusion/Resolution**

- Rough agreement (between those present) on pure, stateless version of:
```js
{
  next() -> { done: false , value: any }
         | { done: true[, value: any] }
}
```
...To replace StopIteration

- Always has an own property called "value":

```js
var i1 = (function *f() {
  return;
})();

"value" in i1.next();
```

```
var i2 = (function *g() {
})();

"value" in i2.next();

var i3 = (function* h() {
  return (void 0);
})();

"value" in i3.next();
```

- Built-in iterators should be specified as returning a fresh value.
- See: https://gist.github.com/dherman/5145925
- Without Brendan, a champion of iterators and generators, don't have full consensus

## 4.2 Modules
(Presented by Dave Herman, Sam Tobin-Hochstadt, Yehuda Katz)

See: https://gist.github.com/wycats/51c96e3adcdb3a68cbc3

Slides (PDF, will prompt download):
http://wiki.ecmascript.org/lib/exe/fetch.php?id=meetings%3Ameeting_mar_12_2013&cache=cache&media=meetings:modules-march-2013.pdf

DH: We're committed to making this happen for ES6, it's too important to miss and I'm going to do whatever it takes. Please remember that you can always bring specific issues directly to us (DH, STH, YK).

...Not going to spend time on syntax. Focus on Module loading semantics to address any outstanding issues, spent last two months working with Sam and Yehuda and polling community leaders to get use cases to work with.

Recognize that some use cases may not be covered, but that's ok.

### Module Loading Semantics

...Currently a notion of having a string-name registry

...Move towards having a separate space for module registration

Minimalism - IN

Nested Modules - OUT

```js
module "libs/string" {
  export function capitalize(str) {
    return (make the string capitalized)
  };
}

module "app" {
  import { capitalize } from "libs/string";
}
```

The registry corresponds to a `Loader` which creates a `Realm`

MM: But you can have more then one Realm

DH: Think of `Loader` as a virtual "iframe", as a concrete way of describing it. When you create an "iframe", you get a whole new global object with it's own DOM. A `Loader` creates a "sandbox" global that can share system intrinsics.

*The `System` loader.*
```js
var capitalize = System.get('libs/string').capitalize;
var app = System.get('app').app;
```

*Custom loader:*
```js
var sandbox = new Loader({ intrinsics: System });

sandbox.set('app', System.get('app'));
```

sandbox.get('app') === System.get('app'); // true

sandbox.eval("import { capitalize } from 'app'; capitalize('hi')"); // "Hi"
```

Acts like a map, `.get()` will get the module

[Module pipeline diagram]

~~~
 module name
     |
     V
  normalize
     |
     V
   resolve
     |
     V
   fetch
     .
     .
     V
  translate
     |
     V
    link
~~~

Produces one of three values:
 * undefined: default linking behavior
 * Module: registers module instance object directly
 * {imports : [ModuleName], execute : (Module ...) -> Module, exports : [String]} : invokes execute to produce module, exports optional

...

### Use Case: Module Paths
```js

```
System.ondemand({
  "http://code.jquery.com/jquery-2.4.js": "jquery",
  "backbone.js": [ "backbone/events", "backbone/model" ]
});
```

...is sugar for...

```js
System.resolve = function(path) {
  switch (path) {
    case "jquery":
      return "http://code.jquery.com/jquery-2.4.js";
    case "backbone/events":
    case "backbone/model":
      return {
        name: "backbone.js",
        type: "script"
      };
  }
};
```

MM: This is changing the behavior only at the system `Loader`?

DH: Yes.

STH: This is the API that several people indicated was important during the last meeting.

### Use Case: ...?

### Use Case: Compile To JS

```js
 System.translate = function(src, options) {
  if (!options.path.match(/\.coffee$/)) {
    return;
  }
  return CoffeeScript.translate(source);
}
```

LH: Is this updated on the Wiki?

DH: Will update. Much of the changes are about refining API and making common things easy and most things possible

### Use Case: Custom AMD

Creating custom translations for extensions...

~~~js
import { data: foo } from "text!foo";

```js
System.normalize = function(path) {
  if (/^text!/.test(mod)) {
    return {
      normalized: mod.substring(5) + ".txt",
      metadata: { type: "text" }
    };
  }
  // fall-through for default behavior
}

System.translate = function(src, { metadata }) {
  if (metadata.type === "text") {
    let escaped = escapeText(src);
    return `export let data = "${escaped}"`;
  }
  // fall-through for default behavior
}
~~~

WH: Why would you want to do it this strange way (escape text only to then eval it) instead of just letting the text be? [It feels kind of like the folks doing eval("p." + field) instead of p[field]].

DH: (explains James Burke's summary of static asset loading)

### Use Case: Importing Legacy Libraries

(Specifically, not libraries that use CommonJS or AMD, but libraries that mutate the global object)

```js
var legacy = [ "jquery", "backbone", "underscore" ];

System.resolve = function(path, options) {
  if (legacy.indexOf(path) >= -1) {
    return {
      name: path, metadata: { type: "legacy" }
    };
  } else {
    return {
      name: path, metadata: { type: "es6" }
    };
  }
};
```

```js
function extractExports(loader, original) {
  var original =
    `var exports = {};
    (function(window) { ${original}; })(exports);
    exports;`

  return loader.eval(original);
}

System.link = function(source, options) {
  if (options.metadata.type === 'legacy') {
    return new Module(extractExports(this, source));
```

  }

  // fall-through for default
}
~~~

LH: Once we ship this, we want people to start using modules as soon as possible. How?

YK: Realistically, a "plugin" for something like require.js will have to provide an ES6 "shimming" mechanism.

LH: To paraphrase, we're providing the primitives that make the common cases easy to overcome. What about the legacy libraries that won't be brought up to date? Can we provide a simple mechanism?

DH: No, legacy libs that just expose themselves to the global object, without any sort of shimming mechanism are out of reach

LH: Thank you, that's a sufficient answer

### Use Case:

Import AMD style modules and Node style modules. Effectively, ES6 module importing from non-ES6 module.

There is no way to tell

```js
System.link = function(source, options) {
  if (options.metadata.type !== "amd") { return; }

  let loader = new Loader();
  let [ imports, factory ] = loader.eval(`
    let dependencies, factory;
    function define(dependencies, factory) {
      imports = dependencies;
      factory = factory;
    }
    ${source};
    [ imports, factory ];
~~~

```
  `);

  var exportsPosition = imports.indexOf("exports");
  imports.splice(exportsPosition, 1);

  function execute(...args) {
    let exports = {};
    args.splice(exportsPosition, 0, [exports]);
    factory(...args);
    return new Module(exports);
  }

  return { imports: imports, execute: execute };
};
```

BM: Could you postulate that exports and

DH: You could but, unrealistic

BM: Could be optimizing for module provider, but not consumer...
...

MM: What does the `Module` constructor do?
DH: Copies the own  properties of the given object.

MM: What is the job of the `System.link` hook?

STH: To go from a piece of JavaScript source code to module instance object, translate is string->string.

WH: Is it a `module` or `Module` instance?

DH: `Module` instance object

Take the source code, all the deps, finds all the exports, links them together.

The link hook can return
1. undefined for the default behavior

2. A Module instance, where everything is done and complete

3. An object, with list of deps and factory function to execute at some later time (eg. when all deps are downloaded and ready)

YK: Explains that a two phase system is required whether you're using node, AMD or anything. Now you can use ES6 provided hook.

BM: Optionally specify the list of exports?

DH: Yes.

Conversation about specific example.

MM: Clarify... noting that the positional args is similar to AMD positional args

DH: Yes.

ARB: No static checking for non-ES6 modules?

DH: Yes, it's a hole that can't be filled if we want interop from AMD->ES6 and ES6->AMD (or node)

ARB: Concern about having two imports, checked and unchecked. (implementation complexity concern)

BM: The alternative is to not support AMD and provide only one imports

STH/RW: This is an option, but a worse option.

...Discussion re: static checking for non-ES6 modules

ARB: Every single construct, import, loading etc now has two different semantics to support.

BM: Forces users into thinking about which they need... optimizing for module authors, not module users. The wrong case... otherwise enforce static checking for all module code

AR/STH: Not possible for _all_ existing code

STH: (whiteboard) Indirection via dynamic object makes static checking impossible.

For example, if you write the code:

```js
import { a } from "some.js"
... a ...
```

where `"some.js"` is an AMD library, then there's no static checking, but if you refactor `"some.js"` to be an ES6 module, you automatically get static checking. But if you don't support this use case, then there's indirection:

```js
import { exports } from "some.js"
... exports.a ...
```

And changing `"some.js"` to use ES6 never results in new static semantics.

...Mixed discussion re: dynamic checks vs static checks.

BM: Was under the impression that the dynamic checks might be too late, but it has now become clear that they happen early enough

STH: Cannot create references across `Module` instances to dynamic module code.

MM: the world of JS uses feature detection, on the AMD side... can AMD code feature test?

STH: (refers to an early slide, which shows example of importing module as single object, properties can then be tested for)

MM: (confirms that STH answers Q)

DH: Pause on the Question of dynamic import/export

(Returns to the pipeline diagram)

...The "fetch" hook is the part where you go get the bits

DH:

(Slide: 1. Load and Process Dependencies Diagram)

(Slide: 2. Link)

AR/DH: Note that browsers can provide their own plugin points for the Fetch step

MM: All of the hooks have been executed and there is no user code? If this fails, there are no side effects?

DH: Correct

ARB/STH: During

DH: Modified the registry, but there is an inflight loading happening, when the inflight is finished, it will pave the changes to registry. (last op wins)

ARB: When you evaluate a script that imports module Foo, which runs hooks that in turn import Foo into the module registry, what happens?

AWB: Why are they operating in the same Realm?

DH: It sounds like an objection to modifying the list of modules in the registry by downloading code that modifies the list of modules in the registry...

STH: Imagine we didn't have loader hooks, all you could do was eval and had two XHRs that fetched and eval'ed. We'd still have the same issues that we'd have with loader hooks, it's a problem with mutation and concurrency.

ARB: Agree that the fundamental problem will always be there, but have a problem with shared global object for all modules.

DH: If the same module is attempted to be defined in two places, that's an error and is a bug.

ARB: Only when within the same compilation stage, silent overwriting otherwise.

WH: What if module A depends on both B and C and the initialization of B fails?

DH: C remains uninitialized but present in the registry

WH: This breaks the model.  It's not C's fault that its initializer didn't run.

AWB: Mark C as never having its initializer attempt to run and run it the next time it's imported.

DH: Moving to next slide

(Slide: 3. Execute)

Produces "Result"

Note that each step in the 3 parts has an Error path:

1. load/syntax error
2. link error
3. runtime exception

...Mixed discussion re: execution despite exceptions

...Mixed discussion clarifying fetch semantics (1. Load and Process) re: dynamically building URLs to batch load? re: browser knowledge of sources?

LH: What does the synchronous timeline of Slide 1 look like?

DH: All normalize hooks first (need names and locations), then all resolve hooks

### Use Case: Importing into Node

```js
System.resolve = function(path, options) {
  if (node.indexOf(path) > -1) {
    return { name: path, metadata: { type: 'node' } };
  } else {
    return { name: path, metadata: { type: 'es6' } };
  }
};

function extractNodeExports(loader, source) {
  var loader = new Loader();
  return loader.eval(`
    var exports = {};
    ${source};
    exports;
  `);
}
```

```
System.link = function(source, options) {
  if (options.metadata.type === 'node') {
    return new Module(extractNodeExports(this, source));
  }
}
```

### Use Case: Single Export Modules

DH: Brought this up 2 meetings ago, had a proposal that wasn't ready, it was shot down. This is something that I'm being told is very important and I agree with them. We can accommodate single exports via design protocols, but the developer community may not like it.

DH/YK: (walk through the `System.link` implementation)

DH: Can, should do better. Goal: Simple syntactic sugar. **It's important, we will address it and we will do so with syntactic sugar**. We will create a means by providing an "anonymous export". We will review the "sugar" at the next face-to-face meeting.

...Recognizes the community frustration regarding lack single/anonymous exports.

...No dissent.

LH: (Questions about how this works with the previously shown use cases)

...

YK: (Shares anecdotal experience from developing the ES6 transpiler that was adopted by Square. Positive experience.)

STH: Updated/removed junk from wiki

LH: Can imports be in scripts?

STH: Yes

DH: There was originally a use case that involved jQuery, we can't satisfy this without breaking everything (there is no way to be ES 3, 5, 6 at the same time)

But...

```js
if (...some detection...) {
  System.set("jquery", ...);
}
```

```html
<!--
once this is loaded, the jQuery module is
registered and available for all scripts
-->
<script src="jquery.js"></script>
<!--
which means all future scripts may have this:
-->
<script>
import { $ } from "jquery";
</script>
```

LH: What about concatenation cases?

DH: (whiteboards example of `System.ondemand`)

```js
System.ondemand({
  "all.js": [ "a", "b", "c" ]
});
```

AWB/STH: (whiteboard)

m.js:
```js
module "m" {
  export let a = 1;
}
```
n.js:
```js
module "n" {
  export let b = 2;
}
```

Needs:
```js
System.ondemand({
  "m.js": "m",
  "n.js": "n"
});
```

If you concatenate?

m.js + n.js = concat.js...

Needs:
```js
System.ondemand({
  "concat.js": [ "m", "n" ]
});
```

Arrays for files that contain multiple things

...

ARB: We're over-prioritizing for concatenation. The language shouldn't be hostile, but should stop at good enough. We shouldn't optimize the design of the language around a secondary concept

AWB: modules are a concrete concept in the language, we need to focus on these as a building block

LH:

STH: The claim that concatenation is going to become a not-important part of the web is ludicrous

ARB: I think that mid-term concatenation will harm performance

YK: Do you think that concatenation will go away?

ARB: In the long term, it might

YK/STH: This is what is ludicrous

...Mixed discussion re: library vs. language

AWB: There is a standard loader, defined by the language

...From Arv:
AR: Joining files to optimize download and compilation

STH: YUI optimized for reality and found that concatting is important

YK: Should Ember ship 100 files?

AR: Any modern library has a lot of files. Apps/libraries are making trade-offs to get good performance.

DC: Caching is not working. Browser will get better.

AR: SPDY will make things better

YK: Even with SPDY, there is a lot of IO

ARB: It is perfectly fine to depend on a tool for concat

EA: We are designing based on concatenation. We should take that out of the picture. We can always write compilers that does the linking.

ARB/LH: With a compiler you can do linking/bundling and existing and future tools can do this.

STH/DH: There will be holes in these.

LH: module "a" { ... } is leading developers down the wrong path

STH: Recommmend doing modules the node style, where each file is a module

YK: AMD developers use a build system that adds the name to the define(). They don't name the modules in their source. The build system names the modules.

MM: AMD experience speaks in favor of a concatenator.

STH: You will get a compile time error if you import a file that contains a module.

...

ARB: How about adding a way to just register a module as a string containing the source of its body as if it was a file.

AR: Then you have to allocate the string

...

AWB: Wants to use module "abc" { ... }. It is a good way to structure code. And you don't want to tie this to your configuration management

...

STH: The strength of the system is that it supports both

ARB: The approach Allen wants is not well supported because it lacks lexical scoping

AR: If we use a string literal we cannot check the code to do prefetching etc

ARB: It is a string so the string only needs to be lexed, then the parsing etc can be paralellized, not so with embedded module declaration

...

ARB: There is no way to not modify the global registry when defining a module.

DH: The file system (in SML) is also a shared registry. The module registry is no different

ARB: Disagree. There is no way to create a local module here

STH: JS has a lot of ways to struccture code: functions, classes etc and modules do not need to fill this role

ARB: More interested in preventing accidents due to name clashes.

...Mixed discussion of module syntax related concerns

DH: Ability to prevent people from using module syntax?

MM: Yes

**STH: For Andreas' concern, look for the names of module declaration strings, check the registry and if any already exist, error.**

...Defining a loader with right hook, prevent the mutation of the registry by anyone that does not have access to the loader

**MM: Satisfied from a security perspective.**

ARB: Would prefer for the default behavior to error, need to be explicit if you want module to override in an imperative manner.

DH: Not opposed to moving towards scoped modules in the future. Just worried about complexities.

ARB: Only concerned about import scope semantics

STH: concern is that polyfills have to use `eval` and then `System.set`

ARB: good to make it clear that polyfills are doing some special

DH: agree with ARB about polyfills

STH: This is something to be deferred without blocking progress, but ok with changing to error to achieve consensus.

YK: agree with STH about consensus, but potentially concerned.

**Conclusion/Resolution**

- Default: declarative form of a module is an error if a module of the same name already exists in the module registry.
- Using `System.set` to overwrite an existing module is not an error.
- Behavior of errors during module initialization (when some module initializers don't even get started) is still unresolved.

**# March 13 2013 Meeting Notes**

**John Neumann (JN), Norbert Lindenberg (NL), Allen Wirfs-Brock (AWB), Rick Waldron (RW), Waldemar Horwat (WH), Eric Ferraiuolo (EF), Erik Arvidsson (EA), Luke Hoban (LH), Matt Sweeney (MS), Doug Crockford (DC), Yehuda Katz (YK), Brendan Eich (BE), Sam Tobin-Hochstadt (STH), Alex Russell (AR), Dave Herman (DH), Adam Klein (AK), Bernd Mathiske (BM), John Pampuch (JP), Avik Chaudhuri (AC), Edward O'Connor (EOC), Rick Hudson (RH), Andreas Rossberg (ARB), Rafeal Weinstein (RWN), Mark Miller (MM), Reid Burke (RB), Edward Yang (EY), Dan Stefan (DS),**

## 4.14 Make new Date(dateObj)

(Presented by Allen Wirfs-Brock)

See: https://bugs.ecmascript.org/show_bug.cgi?id=1257

AWB: Propose to fix this by spec'ing the Date constructor to special case date object arguments and produce an accurate copy of the date object.

DC: Produces a clone?

AWB: Yes.

DC: a Issues with toString?

AWB: No

NL: Bug mixes ES5/ES6 spec language: DefaultValue/DateValue?

AWB: Not an issue

**Conclusion/Resolution**

- When date object is passed to the Date constructor, it makes an accurate copy of the date object.

## 4.4 Spec Update

(Presented by Allen Wirfs-Brock)

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

### RegExp global, ignoreCase, multiline, source, sticky are now prototype accessor properties rather than instance own data properties.

AWB: WH's question, "why are the flags now accessor properties?" (re: RegExp global, ignoreCase, multiline, source, sticky) is necessary to support a web compatible "compile" method. The web reality takes an existing instance and reinitializes a new instance.

MM: SES doesn't whitelist "compile", but could we specify "compile" to be SES whitelistable. Freeze, no longer modifiable by compile

Example:

```js
var re = /abc/;

re.compile("changed");
console.log(re);
// > /changed/

Object.freeze(re);

re.compile("again");
console.log(re);
// > /again/
```

// oops
```

DC: What is the history of "compile"?

AWB: Not sure, but I believe it was JScript

### ArrayBuffer and TypedArray object length no longer restricted to 2^32 limit

DH: We should talk offline about making sure this is future proof

MM: is the new length 2^53? If it goes over, there will be a length that can't be represented

...Mixed discussion about the length limitations of existing arrays.

AWB: Isn't really an issue, just allows the length of arrays to be sufficiently large. I believe this is the future proof solution

ARB: Is it inconceivable that you might want to create a TypedArray backed by a large mapped memory?

DH: Not inconceivable.
...Regardless of whether you want something 2^53 or above

YK: Agree with Allen that this is probably sufficient.

DH/AR: (brief discussion about ownership of TypedArray spec)

AWB: Can't use the same methods as Array, because they use 2^32 length.

MM: And are all generic, so really work on "array likes"

AWB: It's not clear that they _should_ work and it may not make sense to have these methods. There are possible solutions
```

WH: An example of non backwards compatible?

MM: Sparse array that's not populated up to the limit and push on it? What happens?

WH: Calls ToUint32 on it?

AWB: Yes

WH: Doesn't consider those array like?

AWB: Array spec, gets and sets, does its own Uint32
...Need to look at each of these methods and see what can be done to make these compatible.

BE/AWB: (mixed discussion about historic compatibility issues)

BE: We can try it and the first sign of trouble, back off.

MM/AWB/BE: (agreement)

WH: The main use I see of ToUint32 is to call it on the existing length. Indices passed in as arguments are generally processed via ToInteger. As such, getting rid of the ToUint32 in generic Array methods applied to non-Array objects shouldn't present much difficulty (when they're applied to Array objects the change would be invisible because Array length can't reach 2^32).

AWB: But it doesn't have to be an array
...We should move on.

## 4.5 Private Symbols, WeakMaps, and Relationships
(Presented by Mark Miller)

Slides (PDF, will prompt download):
http://wiki.ecmascript.org/lib/exe/fetch.php?id=meetings%3Ameeting_mar_12_2013&cache=cache&media=meetings:relationships.pdf

MM: Concerned about the overlap in functionality between WeakMaps and private Symbols.

...WeakMap named with the word "Map" inverts the thinking.

...Allen noted that WeakMaps are about a Relationship, between two objects.

Legend:

**base** is **key**
**field** is **map**

(Slide 2)

---
**"Map" inverts thinking**
(the use of ">>" signifies the that the item on the LEFT is usually better for most purposes then the RIGHT)

Syntax: base@field >> map.get(key)
...Describes the relationship more accurately

GC: impl(base)[field] >> impl(map)[key]
...A better implementation on the LEFT, despite most implementing the RIGHT

Lookup: Inherited props >> own props

Intuition: Relationship >> Symbol >> Map

Unique Symbols ok.

---

STH: Weak hash tables are found in programming languages and people want them in JS, but they historically fall in the RIGHT. Why can't we declare victory and go home?

MM: I feel like we can declare victory, but more work to do.

WH: Let's avoid a meta discussion.

(Slide 3)

---
**ES6 Encapsulation Mechanisms**

Closures hide lexical state (ES5)

Modules hide non-exports

Direct Proxies hide handler and target

WeakMaps hide keys, do rights-amplification

Private Symbols

___

(Slide 4)

___

**GC: base@field = value**

...Not discoverable given "base". Have to have both "base" and "field" to get value.

Abstract heap maps(base, field) => value

base _and_ field reachable -> value reachable

...Introduces an "and" into the previously only "or" GC map

Obvious representations:

1. impl(base)[field] => value

better when field lives longer

...When base dies early, it will take the association with it,

by regular GC and without needing ephemeral

2. impl(field)[base] => value

better when base lives longer

...When field dies earlier, it will take the association with it, by regular GC and without needing ephemeral

___

(Slide 5)

___
**GC by use case**

When base is known to live longer

_Just use a map_ (per Yehuda)

oo private field **is known** to live longer.

Just use representation #1

Most remaining WeakMap use cases

would do better with representation #1 (untested claim)

(From a GC perspective)

___

MM: The GC cost is significant. If you chose the wrong implementation, every object that gets trademarked is retained by the WeakMap until you invoke the ephemeron collection algorithm.

ARB: It's worse in v8 because it has an incremental GC that currently can only run that on full collections

MM: w/r to normal access, I believe it's a wash

AWB: I don't think it's a wash, case 1 requires an index, case 2 requires a hash code

(Slide 6)

___
**GC by use case**

Only need ephemeron collection when

 - you guessed wrong relative longevity

 - you care about the memory pressure

Felix's O(N) algorithm is affordable

with inverted representation

Examples: Membranes

___

MM: Membranes are an example where you still need the ephemeron collection algorithm

**b** is base object

**f** is field

**L** is left side

**R** is right side

**T** is target

**P** is proxy

(Slide 7)

___

**Transparency vs Privacy**

...Membrane:

Test:

bL@fL = vL    bR@fR === vR

...see slide

___

[Whiteboard Example, get image from Sam]

MM: Propose changing the name of WeakMap and changing the way we specify and describe them for implementors to think about.

AWB: Last meeting, we discussed cases... eg. when you invoke a method whose this object expects access to the private state, but it receives a proxy, it will break. I believe the solution might solve the problem

YK: |this| binding issues are orthogonal to this issue

BE: (Agree)

MM: (refutes)

WH: I challenge on this, there is a claim that we don't need private symbols and that they're broken, but we need to do things that private symbols are for. So why can't private symbols work the same way?

MM: This accomplishes things that we needed private symbols to accomplish, if you want to propose changing the name of w

WH: I want to see common uses of private symbols and how you would replace them.

MM: I haven't shown the broken solutions...

STH: I think that would've been much more helpful then showing what works today and that you wouldn't change.

MM: I believe this is an acid test to prove the transparency

WH: Implement private symbols in terms of whatever you're proposing.

MM: \`fT\` is a private symbol

WH: ...

MM: I'm trying to answer your question

WH: What would I need to do differently and why, using your glorified new approach.

MM: In the expansion of a class that uses a private symbol, for each declared private instance variable, there will be a private  symbol per instance variable.

WH: I think you'll need to write it down.

MM: (describes semantics, based on Slide 8)

WH: You're saying private symbols work if they adopt this semantics

WH/STH: What are the changes?

STH:

1. There is no trapping on writes and reads of proxies, passes through
2. proxy a private symbol and write sets and gets, trapped to the handler that has that private symbol.

...In Mark's defense, this is a substantial change to how we think about private symbols

AWB: It's a new type of access on an object, with a new type of state. A new, different operation with separate behaviors

YK: Will the vast majority of JavaScript programmers have to understand this? This is the most important question for me

ARB: ...?

TVC: Don't need to change proxies in anyway

YK: Not actually my concern

STH: Need to think of these as something more then properties

MM: Thinking of them as properties is what caused me to gravitate towards addressing

(Slide 9)

---
**Desugaring Private Relationships**

```js

base@field              base@field = value
field.get(base)         field.set(base, value)

```

---

(Slide 10)

---
**What about symbols and strings?**

```js

base@field             base@field = value
-field.get(base)-      -field.set(base, value)-
field[@geti](base)      field[@seti](base, value)



String.prototype[@geti] = function(base) {
  return base[this];
};

String.prototype[@seti] = function(base, value) {
  base[this] = value;
};

```

---

MM: Don't want to use string name "get" and "set" for the invocation, I want to use a symbol

WH: What is the implementation of @geti?

MM: The semantics of WeakMap's get, with exception that if the key isn't found, it tries the object the key inherits from (i.e., follows the prototype chain) until it finds a key or fails. In other words, it implements the inheritance semantics we expect of accessing properties. Given the use of representation #1, this can reuse the existing optimizations for inherited property lookup.

...?

MM: Private symbols, not discoverable

AWB: You've transformed it into a alternate MOP level operation, but you could've transformed it into a new MOP level operation... (missed this before Waldemar interrupted)

WH: This was my proposal 3 years ago...

MM: Explanation...

WH: Not allowing these things to have normal descriptor rules is a bug.

STH: Exactly, the existing proposal reuses the mental model of JavaScript programmers that we should continue to use. Same is true for encapsulation at the property level and GC mappings.

AWB: If you want true privacy

WH: ... You need

AWB: You can't proxy anything that has private state. Cant proxy any built-ins, or anything with private symbols.

BE: Asking TVC to address Waldemar's concern

WH: When the issue of membranes vs private symbols first came up, I showed an inversion approach, instead of asking the object's proxy to look up, you'd ask the private symbol's proxy. This had been intended to solve the proxy privacy leak problem (don't want object proxies to see the objects' private state), but it's essentially the same issue and solution.

TVC: Having property access by the private symbol

STH: Mark is proposing something that has similar semantics, without a new meta operation, but pays for that by making it less like true properties

WH: So which is it?

MM: If the definition of WeakMaps is something you "stick things into", then the answer is private symbols.

WH: Please show an example how we'd actually use this and how it would interact with a proxy

(break for lunch)

MM:

**private** is a pseudo-syntax mechanism

```js
class Point {
  constructor(private x, private y) {}

  toString() {
    return `<${this@x}, ${this@y}>`;
  }
  add(point) {
    return Point(this@x + point@x, this@y + point@y);
  }
}
```

Desugars to...

```js
let Point = (function() {
  const x = Rel();
  const y = Rel();

  function Point(x, y) {
    this@x = x;
    this@y = y;
  }

  Point.prototype = {
    toString() {
      return `<${this@x}, ${this@y}>`;
    }
    add(point) {
      return Point(this@x + point@x, this@y + point@y);
    }
  };
})();
```

YK: If you're calling add through a proxy?

MM: If you're calling one Point proxy from another Point proxy,
the

YK: Use case, I want the |this| binding to be the proxy

MM: Call add() method on a proxy of a Point
If the right operand is a proxy,

BM: a proxy to a point doesn't have an x

AWB: Specifically, a proxy to a point isn't registered in x

...Concern: if you had done toString simply as a get access

YK/STH: If Proxies are going to fit into the language, with any sort of concept of private state, then we need to solve the problem, and this doesn't solve the problem.

MM: Use membranes.

WH: If you come up against this problem many times and each time the answer is membranes, then we need to add membranes, not proxies.

General Agreement.

STH: Until we decide which side of the solution we want: either transparently proxying objects with private state should work, or private state can indicate that something is a genuine not-a-proxy instance of an abstraction (such as Date), then we can't possible have a solution.

YK: Hiding implementation details that manage private state that can't be proxies.

AWB: Virtual object vs Caretaker

BE: Not going to make unstratified traps

YK: only need a small amount of traps: get and set

...Can't go down this road.

(going in circles now)

BM: Notes that the next logical step is interfaces, where you define an interface and proxies implement against this.

BE: Can't apply an interface to already born objects.

BM: if you pre-plan that the object is in line with the interface then security is under control

BE: We should defer private symbols

AWB: If we do that, then there is no private branding _on_ the object

STH: Doesn't solve the problem for "what does proxy for dates do"?

More discussion about the general flaw of proxies wherein... objects that have some form of internally implemented "private" state cannot be observed and therefore not subject to trapping.

MM: Can we agree that transparency through membranes is a requirement?

AWB/STH/YK: Agree its important, but there are a lot of requirements.

MM: A champion group should go and solve these problems. Retaining WeakMaps, deferring private symbols very closely matches the proposal.

STH: ...

MM: Just want to consolidate the issues

STH: Bad idea to postpone WeakMaps, which useful and are already shipping in browsers. Bad idea to postpone anything based on a particular issue with Proxies. A separate issue with regard to the design of private symbols. None of these are solved by adding "@" to represent a WeakMap.

WH: It seems like @ for WeakMap makes sense

BE: Too much for ES6

MM: Regardless of the syntax/API, the slides show that we should encourage implementers to hang the state off the key, not the base.

AWB: The WeakMap abstraction we have is fine as is. If an implementation wants to invert its handling of keys and the map, then that's fine too.

WH: (some comment about private state?)

STH: That's not related

AWB: (re-iterates)

MM: The inverted representation is the better one.

YK: So, what is your proposal?

MM:

- **My proposal is the private symbols get postponed to ES7**
- **WeakMap gets renamed to reflect the inverted thinking**
- If we adopt the @ syntax in ES6, I want to be able to use a WeakMap on the right side

...Don't care so much about the renaming, but feel that it could add clarity to the feature

RW: (Disagree based on experience teaching the concept)

WH: no consensus on class without private symbols

MM: Rick, can you find the history of this?

RW: (recounts two meetings where Waldemar was not present, but a general consensus existing with the understanding that Waldemar would still have something to say. At Northeastern University, Allen presented @-names, in which Waldemar found to sufficiently meet his privacy requirement for classes)

...Discussion around class private state as a whole.

BE: We need to synthesize what's going into ES6. Waldemar claims this must include some way of adding class private state.

STH: I would like to stop this conversation and allow myself time to think about this.

BE: Done.

**Conclusion/Resolution**

- Sam, Mark and Allen to work on "relationships" and varied representation in ES6.

## 4.3 Proxy

BE: We've had prototype implementations for some time, we made it through the change of API. We use them extensively.

## 4.16 Current Status of ES6

(Presented by Luke Hoban)

(request some form of the spreadsheet?)

AWB: Based on this review, we should shoot for December 2014

Discussion of time tables for completed spec reviews

NL: Also need to look at progress on implementations.

NL: Concerned about impact on Internationalization API 2.0 – need to evaluate whether to base on ES5 or slip to December 2014.

**Conclusion/Resolution**

- November 2013, Final review draft
- June 2014, Editorially complete

(STH: we said June in the meeting, but July makes more sense, since we don't have to submit to the GA until September, IIUC)

- December 2014, ECMA approval

## 4.7 Runtime costs of the override mistake

(Presented by Mark Miller)

Override Mistake

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

(Request spreadsheet)

RW: Would like to see the most minimal piece of code required to illustrate the operation, measured by executing the operation in several rounds with MAX number of executions calculated to a mean value. Also, there is use of an `in` operation which will skew the results between Firefox (a bit faster) and Chrome (slow)

WH: I'd like to see comparisons of non-frozen vs. frozen to see whether the two order of magnitude Chrome slowdown is caused by freezing or inherent slowness of push and pop.

Varying, lengthy discussions around SES techniques and the implications of the override mistake.

Non-issue for code that doesn't freeze.

**Conclusion/Resolution**

- Investigate instrumentation when silently fails non-writable prototype properties.

**# March 14 2013 Meeting Notes**

**John Neumann (JN), Norbert Lindenberg (NL), Allen Wirfs-Brock (AWB), Rick Waldron (RW), Waldemar Horwat (WH), Eric Ferraiuolo (EF), Erik Arvidsson (EA), Luke Hoban (LH), Matt Sweeney (MS), Doug Crockford (DC), Yehuda Katz (YK), Brendan Eich (BE), Sam Tobin-Hochstadt (STH), Alex Russell (AR), Dave Herman (DH), Adam Klein (AK), Bernd Mathiske (BM), John Pampuch (JP), Avik Chaudhuri (AC), Edward O'Connor (EOC), Rick Hudson (RH), Andreas Rossberg (ARB), Rafeal Weinstein (RWN), Mark Miller (MM), Reid Burke (RB),**

## 4.8 Object.observe Implementation Report

(Presented by Adam Klein, on behalf of Rafael Weinstein)

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

Slides (PDF, will prompt download): http://wiki.ecmascript.org/lib/exe/fetch.php?id=meetings%3Ameeting_mar_12_2013&cache=cache&media=meetings:object.observe_implementation_report.pdf

(Slide 1)

___

- Results are encouraging
- Two Google projects are pleanning to deploy run-time support
- Use cases

?

___

(Slide 2)

___

- Spec fully implemented (behind flag)
- Mostly self hosted: changRecord allocation, enqueing and delivery is in JS
- Mutation sites of observed objects deoptimize. Observed arrays are alwas slow mode
- Biggest perf bottleneck is feting changeRecords; plan to speed up Object.freeze()

___

YK: Concern about changeRecord spam (build up of changes
delivered at the end of the turn)

AK: There are places were large numbers of changeRecords are a concern

AWB: (missed question)

AK: We could define the properties as non-configurable, this is orthogonal to the isFrozen check.

AWB: Even if there isn't a specified state for a frozen object

EA: With proxies, that becomes observable

AK: There are changes to the spec since last brought to the committee...

(Slide 3)

___

- `Object.deliverChangeRecords()` continues delivery until pending
- Added `{ type: "prototype" }` changeRecord which reports changes to `__proto__`
- Minor changes to changeRecord generation to enforce consistency invariants

___

DC: Does that cause the turn to never end?

AK: Yes, same as recursive calls

ARB: Just another footgun in that respect

(Slide 4)

___

**ChangeSummary JS Library**

- Supports dirty-checking and Object.observe (polyfill to fast/safe object observation)
- Exposes semantics for a "diffing summary (Tn -> Tn+1 changes)
- Observe
  - "Values at a path" eg. `var foo = { bar: { baz: "..." } }; observe.observePath(foo, "bar.baz");`
  - Entire Objects
  - Array splice

- ??

___

WH: What is "values at a path"

AK: If you have:

```js
var foo = { bar: { baz: "..." } };

observe.observePath(foo, "bar.baz");
```

?: Observing paths leaks objects if we only have weak maps, not weak references

WH: Please explain

?: Path observers observe mutations anywhere on an object path a.b.c.d.  Without weak refs these will leak a if d is still alive but a wouldn't otherwise be.

(Slide 5)

___
**Per Analysis (dirty checking vs Object.observe)**

- Results not surprising (this is good!)
- _nothing-changed_ case is overwhelming win
  - Discovering that nothing changed only incurs the cost of de-optimizing observed objects
- Object.oberse case never needs to keep a copy of observed data
- Object.observe appears to observed properties have changed (depending on observation "type")
- Arrays encourage "hidden" everything-changed cases eg. `unshift();`

___

AK: One of the worst cases is observing Arrays

WH: If you do that, do you get one record for the array, or one for all properties?

AK: One for every index and length

?: Which then can get (inefficiently) summarized back to one change record for the array if you use a summarizing adaptor.

EA: We knew about this from the very beginning, but decided to address it after we had implementation experience.

(Slide 6)

___
Adding Support for Array "splice mutations"

- Report on changes to Array elements (isUInt32(propertyName)) as "splice mutations"
- Degrade to "normal" property mutations if...
  - !Array.isArray(array)
  - array has non-data index properties
  - array has non-writable or non-configurable index properties
  - operation will affect indices > UInt32

___

WH: What happens when you observe a WeakMap?

AK: It's an object, but nothing interesting happens, because there are no exposed properties that will be changed. We could have synthetic events for Map, Set etc.

LH: We would have to specify these synthetic events ahead of time

AK: The requirement for Map and Set would be to come up with new types.

WH: Observers make changes between properties and getters/setters (such as what we're doing to regexp flags to support the compile method) uncomfortably likely to break user code that observes such objects.

LH: But these can already use synthetic events

WH: What are synthetic events?

LH: (explains the synthetic events api)

AR: We may want a declarative way to "squelch" certain event types.

AK: ...discussion of the cost of notifying for all changes

YK: As we add more "types" there will absolutely need to be a way filter the types you care about.

RW: It would be nice to specify the "type", similar to registering handlers for specific event types. eg. `Object.observe(o, "type:prototype", ...)` gives me only changes to the prototype. etc.

YK/AK: ...more discussion about complexity of adding the feature now or later.

AR: If you have filtering, users can choose the level of changes they are shown, without taking away any of the low level type notifications.

STH: What is the consequence of changing the system now? Can't we just do the right thing here for arrays?

LH: It introduces new complexity

AK: ...discussion of failure cases

AR: Fixing the Array thing for ES7 is a reasonable thing. But this is an example of something we will want to do in the future. This is also a chance to get the filtering feature and maintain upward compatibility.

YK: An array as an identity map

LH: An array as a tuple, its an object data type.

YK: Arrays used in Ember, specifically for numeric indices

...

YK: Do exceptions bubble up?

AK: There are many issues.

...May be better to special case splices

YK: ...discussion about spamming of changeRecords at the end of the turn and the sync behavior.

STH: If you're counting typing speed, you want all of the changeRecords

ARB: What happens when there are new observers or removed observers during a changeRecord delivery.

YK: I'm happy with the current implementation, but it would be nice to be able to squelch certain types of changeRecords

AK: Let's take this offline and look into ways of making this possible.

BE: (to ARB) are you ok with staying off the fast path?

ARB/YK/AK: ...Discussion about the possibility of a fast path for properly implemented Arrays

AWB: What happens if the object is implemented via Proxy, and doesn't have the normal representation of properties.

ARB: Treated the same as accessors

AR: To clarify, we call this the "slow path", but for apps that are going to use this, it's demonstrably preferable to existing art.

## 4.6 Symbols

(Presented by Andreas Rossberg)

ARB: Update on implementation of (unique) symbols in V8: mostly finished, and as efficient as strings as property names.

But symbols don't fully behave like objects yet, as would be required by current spec.

ARB: Propose two changes to the proposal:

1. Make symbol a primitive type
2. Change toString behavior to avoid hazard

Problem with 1: extending typeof

AWB: Symbols as objects allows for useful operations like toString

ARB: Can be solved the same way as for other primitive types: wrapper object

AWB: From a spec perspective, making them not an object is more pervasive

ARB: Very different for implementations. You can use them in context of string or object, but you can pick only one representation. Chose string-like context because that's more important to optimize. But makes special-casing all object contexts costly

DH: Would have to do this if we made this a primitive type as well.

ARB: it would just fall into similar cases as other primitive types (in v8 at least)

AWB: ...compares to primitive and object wrapper classes

DH: And we would have to provide wrapper class

AWB/DH: We should have a new typeof type

DH: Real implementations have other typeof types, eg. IE

STH/BE: ...recalling prior discussion that had no definitive conclusion re: new typeof types.

ARB: Can we really afford precluding new typeof results for all eternity?

DH: If we can get away with this, then it supports adding new types for future additions, eg. int 64

BE: This is important:

```js
typeof x == typeof y && x == y
            <=>
          x === y
```

...: The proposal from AR preserves this.

STH/DH: Giving the switch(typeof...) example

YK/RW: When would symbols be passed to this code anyway?

YK: It's a fuzzy case that is already broken, because `null` can't be checked

BE: We should just do it and that'll learn 'em. ;)

DH: A new type in typeof is good and we should just do it.

General agreement.

DH: Now that these are essentially primitives, we need ways to discern private and unique

STH/YK: Committed to a new global because we use `Symbol` to construct them.

AWB: `Symbol` is a factory that creates symbols, `new Symbol` creates instance of the wrapper class. (same as Number)

BE: Value objects allow "new"

AWB: I can define the `Symbol[@@create]` to throw

ARB: The current spec has a toString for implicit conversion, which makes it too easy to convert the symbol to a string accidentally without realizing.

STH: Have a name property, and a toString that throws

ARB: Or simply no toString. But `{}.toString.call` still works.

EA: String conversion already throws for null-prototype objects

DH: That's fair, it makes senses to have some sort of infallible   stringify.

DH: Proto-less dictionaries will become more widely used

AWB: ...recalling existing specification of ToString conversion

...Mixed discussion about ToString

**Conclusion/Resolution**

- New addition to typeof table
  - `typeof symbol === "symbol"`
- [[ToString]](symbol) => throws
  - therefore `symbol + ""` => fails w/ error
- `Object.prototype.toString.call(symbol)` => "[object Symbol]"

(ARB: or rather something else, given that we just decided symbols shouldn't be objects)

## 4.17 Legacy Compatibility for Block Level Function Declarations
(Presented by Allen Wirfs-Brock)

See:
http://wiki.ecmascript.org/lib/exe/fetch.php?id=meetings%3Ameeting_mar_12_2013&cache=cache&media=meetings:legacyblockfunctiondcl2.pdf (This is a PDF that will initiate a download!)

(Copied from PDF)

Prior to the Sixth Edition, the ECMAScript specification did not define the occurrence of a _FunctionDeclaration_ as an element of a Block statement's _StatementList_. However, support for that form of _FunctionDeclaration_ was an allowable extension and most browser-hosted ECMAScript implementations permitted them. Unfortunately, the semantics of such declarations differ among those implementations. Because of these semantic differences, existing web ECMAScript code that uses Block level function declarations is only portable among browser implementation if the usage only depends upon the semantic intersection of all of the browser implementations for such declarations. The following are the use cases that fall within that intersection semantics:

1. A function is declared and only reference within a single block:

- A function declaration with the name f is declared exactly once within the function code of an enclosing function g and that declaration is nested within a Block.

- No other declaration of f that is not a var declaration occurs within the function code of g

- All references to f occur within the _StatementList_ of the Block containing the declaration of f.

2. A function is declared and possibly used within a single block but also referenced within subsequent blocks.

- A function declaration with the name f is declared exactly once within the function code of an enclosing function g and that declaration is nested within a Block.

- No other declaration of f that is not a var declaration occurs within the function code of g

- References to f may occur within the _StatementList_ of the Block containing the declaration of f.

- References to f occur within the function code of g that lexically follows the Block containing the declaration of f.

3. A function is declared and possibly used within a single Block but also referenced by an inner function definition that is not contained within that same Block.

- A function declaration with the name f is declared exactly once within the function code of an enclosing function g and that declaration is nested within a Block.

- No other declaration of f that is not a var declaration occurs within the function code of g

- References to f occur within another function h that is nested within g and no other declaration of f shadows the references to f from within h.

- All invocations of h occur after the declaration of f has been evaluated.

Use cases 2 and 3 for a given function declaration f might occur within the same function.

The first use case is interoperable with the inclusion of Block level function declarations in the sixth edition. Any pre-existing ECMAScript code that employees that use case will operate using the Block level function declarations semantics defined by clauses 10 and 13 of this specification.

LH: Has to be a statically verifiable approach

DH: We can statically declare a conservation class of blocks

AWB: Most blocks are conditional in some form.

WH: (whiteboard)

```js
function g() {}
function f() {
  if (false) {
    function g(){}
  } else {
    g();
  }
  g();
}
```

WH: This example illustrates that #2 is not interoperable. It's not even clear to me what #2 means.

- Does the first call to g (in the else close) lexically follow the inner definition of g?
- Regardless of the answer, the second call to g is not in the intersection semantics (the inner g is never defined due to the if false) yet is treated by #2 as though it were.

AWB: A fix for this case: Add an additional clause that says the subsequent

AWB: The semantics we want, for non-legacy cases, in non-strict mode are the ES6 semantics.

DH: Not compatibility with reality.

...

Sixth edition interoperability for the second and third use cases requires the following extensions to the clauses 10 and 13 semantics. These extensions are applied to a non-strict mode functions g if the above pre-conditions of use cases 2 and/or 3 exist at the time of static semantic analysis of g. However, the last pre-condition of use case 3 is not included in this determination and the determination is only applied to function declarations that are nested within syntactic constructs that are specified in the Fifth edition of this specification.

1. Let B be environment record for the construct within g that introduces a new environment contour and which most closely encloses the declaration of f, all function code references to f, and the definitions of all nested functions that contain syntactically unshadowed references to f. This syntactic construct may be the definition of g itself, in which case B is the function environment record for g.

2. As part of the instantiation of B, its CreateMutableBinding concrete method is called with arguments "f" (the string name of the function) and false. This creates an unitialised binding for the name f. Any reference that resolves to that binding prior to step 3 below will throw a ReferenceError exception.

3. When the InitializeBinding concrete method is used to initialise the binding for the function declaration f also invoke InitializeBind on B using the same arguments.

If an ECMAScript implementation has a mechanism that produces diagnostic warning messages, a warning should be produced for each function g for which the above steps are performed.

WH: #1 is a circular definition. Some references to g will go to the outer g in the example. It is up to us to define which definition points to which.

AC: It would be beneficial to look at examples and then apply the fix rules and allow that inform the direction.

AWB: (whiteboard)

```js
// A dynamic ReferenceError.
// There is no binding to g()
function f() {
 {
   if (false) {
     function g() {}
   }
   g();
  }
}
```

AWB: The interoperable fix:

- As if there was a `let g;` at the top of the inner block...

- the g() reference points to the g() declaration in the same scope

WH: ok, put back the outer g()

AWB: (whiteboard)

```js
function g() {}
function f() {
 {
    if (false) {
      function g() {}
    }
    g();
  }
}
```

...Didn't take into account this case, will need time to consider.

AWB: Could be two inner blocks, with declaration of g()...

WH: (whiteboard)

```js
function g() {}
function f() {
  {
    g();
    if (true) {
      function g() {}
    }
  }
}
```

WH: What happens when the first g() is called?

AWB: the first g() is bound to the outer g(), then a new g() is defined.

WH: (whiteboard)
```js
function g() {}
function f() {
  {
    g();
    if (true) {
      function g() {}
    }
    g();
  }
}
```

WH: Now what happens?

AWB: Per the proposed rules, both calls to g() are bound to the inner definition via the phantom let binding [presumably with the first one encountering a dead zone?].

[MM walks into the room and expresses speechless incredulity when he learns that adding the second call to g in this example changes which g the first call to g is referring to.]

AC: There is merit to avoid being clever. Benefit to preserving any existing semantics.

BM: There should be a vision for a desired semantics goal

STH/LH: Two goals, which are competing:

1. We must remain compatibility with the web
2. Introduce block scope bindings

BM: Bad that an outer binding can change

AC: ...correlates to AS

STH: We have sensible semantics for ES6/strict mode

AWB: We have sensible semantics for both modes

AC: Confident that we might be able cover all existing use cases. But what happens when someone discovers a case that was covered?

STH: Easy to come up with sensible rules from other languages

AC: All practical purposes

[? trying to start a meta-discussion about getting sidetracked on theoretical rather than practical problems]

WH: We are discussing because we know this is a real, practical problem based on research demonstrated at the previous meeting.

STH: The only guiding principle is to avoid breaking the web.

AWB: Where we're at, is the hypothesis that we can identify and fix the cases that break the 1% of sites that have code that violates. Introduce a declaration that spans the point of declaration and the point of reference

STH: ...clarifies the subset semantics definition

AC: Two cases:

1. What to do?
2. Identify that subset.

STH: Yes

AWB: This conversation only applies to the 1% of code that exists that must be dealt with in a way that matches the 99% semantics

LH: ...recapping

AWB: JavaScript programmers will have two learn that non-strict mode always uses odd-ball semantics in addition to the strict mode semantics. If we make this work, there is one semantics to learn.

DH: Inclined to agree with Allen. A worse semantics, easy to explain vs. a better semantics that serves the 99% but loses the 1%... and we'll lose the 1% in a deep dark hole.

...Arguing for:

1. One set of rules, that is weird and less pleasant
2. Carve out heuristics for identifying ... giving the good semantics to 99%

BM: Missing analysis?

STH: Presented at last meeting

DH: Arguing for #2 (99%)

WH: Want to see the semantics of strict mode in non-strict mode as often as possible. Don't want to legitimize new quirks, even if that means that the transition rules are more complex. Hope that in a few years the strange semantics can be eliminated in favor of what strict mode is doing.

LH: This is exactly Allen's point. If Allen's point achievable? If it is, then I support it, but I don't think it's achievable.

AWB: This is a first draft.

YK: Function in block is oddball already. If we have a semantics for non-strict FIB and different from strict mode, it will make it hard to upgrade to strict mode.

Note: AC/BM please send Allen any materials that might be useful for specifying the semantics for FIB.

**Conclusion/Resolution**

- All (esp. AWB) to continue working through known issues in the 1% cases

## 8.1 Comments from Istvan, stated by John Neumann

- The ECMAscript trademark is rather through. We have it in the most important countries, incl. US and EU.

- The EU formal recognition for ECMA-402 and TR/104 is progressing. We are now in the evaluation phase by a small group. They asked me to make a first draft for evalutation, which I did. Informally I have one "yes" so far, and no "no" yet. If the small group gives an ok, then the member countries have to vote. The process will take a few months more (but I do not know how long, because this is the first experience with the new EU policy).

- Regarding the IPR Adhoc. We are progressing. I think I can distribute an updated package to TC39 at the end of March. GA vote is planned for June 2013. Comments back to the IPR Adhoc, GA, or GA members. Just as last time.

WH: ECMA's TC39 email reflector is incompatible with gmail. If a gmail user sends an email to the reflector, non-gmail users will receive it but no other gmail user will receive it in his inbox. It will always arrive in the spam folder due to some SPF issue.

[Various people checking their spam folders now and discovering TC39 emails in there.]

## 4.10 Array Extras

(Presented by Rick Waldron)

(notes from EA)

RW: Wrote spec (using spec prose) Array.prototype.find and Array.prototype.findIndex. Found in most PL and JS libraries

MM: What about start indedx? Does it go before the thisArg or not?

RW: None of the existing languages supports passing a start index.

RW: Not sure if this is for ES6 or Harmony

AWB: Feature creep

DH: Common, makes sense.

**Conclusion/Resolution**

- Exception being made. Approved for ES6.

## 4.15 Template Strings

(Presented by Adam Barth [present], Mike Samuel [phone] guests from Google)

BE: What can we do to template strings to make them secure by default and not an XSS hazard.

- tagless?
- a default tag?

AWB: They're useful for making strings.

BE: (reiterating) The obvious problem: using backticks with no prefix/tag, you  create potential for xss.

DH: Ok, so we've outlined the security argument, but important to note that tagless are useful and the security fix is still not enough to "fix" the bug picture security problems.

DC: No placeholders in tagless?

BE: Possible

YK: ...recalling discussion with Mark where the rabbit hole became extremely deep when trying to avoid accidentally coercing to a string.

MM: String.prototype.trim apply to delayed template string, which turns it into a string.

...Delayed template string inherits from template string

YK: Implicit coercions.

MM: This is one option

BE: Make the default case somehow not present, or neutered

AWB: Remember for "+" and explicit coercion cases, we're falling into the DefaultValue mechanism, which has a unique Symbol hook (in ES6), which means that what is sent to the object initially does not have a toString. The DefaultValue for the delayed template string is...

...DefaultValue applied in concatenation could throw,

MM: ...specifically the deferred template string. Specifically for the tagless template string case.

(RW: some terminology issues addressed)

WH: Understand the utility of tagged template strings, with possibly a default tag that does string concatenation, but what's the rationale for deferred template strings?

[some discussion, with no answer to WH's question]

MM: If the behavior is for the DefaultValue to throw, reject the implicit coercion. However, if you're writing a REPL and want to  stringify, you can explicitly call toString

STH: This is insane

DH: This is destroying the usefulness of tagless template strings.

YK: It's not clear how taking a nice feature and making it harder to use will fix a security issue, the problem will still exist with string concatenation.

DH/RW: Agree

Mixed discussion re: security issues in the DOM.

AR: Security issues revolve around reasoning about data and behavior which you'd like to think is benign, but combining them create abilities that they shouldn't have.

DH: By calling out template strings, we're effectively blaming "strings" for these security issues.

AR: innerHTML

STH: That's the problem

AR: This isn't about hard-and-fast, right-and-wrong. Security is about doing the right thing more often. Security bugs are just a subclass of bugs that bite very hard. The biggest thing any system can do to improve the security situation is to set the economics of doing the right vs. wrong thing in such a way that it's "cheaper" to do the right thing. So we can't pretend that this is about some perfect answer; it's about the economics of designing a solution that leads to doing the right thing more often, and that's a question about probabilities and psychology. We want to build a honeypot for doing it the right way.

MM: Key, either get rid of tagless template string or make them unpleasant; have we created a situation where developers would or would not have used the feature...

AR: What is the reality that adding a no-op tag? What does it actually provide? Stopping developers from using tagless by giving them a tag that does interpolation.

DH: Strings are the most used data structure in all programming. Yes, they are used heavily in secure code construction.

...Then there is the pitchforks and torches cost of making this harder.

MM: Also, the cost of companies that will be screaming about being bit by a security issue

RW: But these are no different then the existing security issues today.

AR: That's not a fair argument to discussion.

MM: The screaming over vulnerabilities is probably more costly then developer hardship.

...If you use a tagless template string in particular context, you can have a an autoescaped context.

BE: So, you could change innerHTML setter to check if the RHS was a tagless template string

MS: (phone) If you don't try applying the string concatenation operators to the tagless template string, you get secure auto escaping.

innerHTML

MM: Let's call that "Mike's Original Proposal"

STH: (listing)

1. Current status with default handler that interpolates
2. No tagless template strings
3. tagless template strings with no interpolation
4. tagless template strings produce delayed template string objects, with toString
5. tagless template strings produce delayed template string w/ special handling that coerces
6. tagless template strings produce delayed template string w/ special handling that throws
7. Allow overloading of the default handler for the tagless template strings

MM: (whiteboard)

```js
class DelayedQ {
  constructor(callSiteId, ...args) {
    this.force = function(handler) {
      return handler(callSiteId, ...args);
    };
    this.toString = function() {
```

      return stringer(callSiteId, ...args);
    }
  }
}

```js
html`<script>
  "${...}"
</script>`

```

WH:
```js
html`<script>
  "${foo()}"
</script>`

function foo() {
  return `...`;
}
```

WH: Using a tagless template string to construct an english sentence and I stick it into html, what do I get out of that?

AWB: You'll probably get garbage, because the innerHTML parser rules will be applied.

WH: How does foo in the example know how its deferred template will be interpreted? It becomes a really sneaky part of the API, where foo can return different things depending on the context it's called in -- can be string concatenation, various kinds of escaping, etc. This is too brittle and dangerous.

LH: innerHTML should've been forced to apply cleansing on all strings.

BM: How about immediately constructing and scrubbing?

STH: Same issue with E4H

DC: Don't forget about the off-browser cases.

YK: So the socket API should check strings that are created in Node?

AB: Yes

YK: Ok, I said this to be ludicrous.

AB: If you eagerly coerce to a string, you lose the structure of the object: which part of was a template and which part was data.

WH: We have no way to tell which part is which. [In the example above, if foo returns a deferred quasi, it's like it returns a bag of stuff that it wants joined together but it doesn't say what the stuff is (individual characters? names? rows in a table? entities? script statements? HTML elements?) or how it should be joined. It's not clear to anyone reading foo's code how what the semantics of the bag of stuff is. This is too brittle.]

STH: The point Yehuda is making, is that people make code and ship it out over the wire.

AB: ...describes the way Ruby server programs create outbound blobs that from templates and data that are flatten just before sending.

YK: I wrote that.

...Discussion about JavaScript developers, use of strings vs. DOM authors creating APIs that can handle template strings

AR: This is a new string of a different type

YK: We'll need to blacklist all strings if this is the case.

MM: ...introduces Mike Samuel's work

BM: Summarizing...

1. Adam: We control all user code
2. Yehuda: We control no user code

AK: Specifically, Yehuda refers to all existing code that follows a form where:

`"string..." + " is concatenated";`

and used to assign to innerHTML (or similar)

MS: Able to successfully migrate a codebase from ad hoc work-arounds. This used closure templates. Statically compiled version of templates.

STH: Is the argument leaning towards tagless or tagged, or is that not relevant?

MS: The system worked with nested script, css and html. In cases where developers needed, they circumvented the system.

YK: ...recalling hardships of Rails upgrading for a similar auto-escape system.

...

YK: Happy with current system + html default handler provided by the browser(*)

DH: Template strings are useful for other reasons: multiline strings, etc.

MM: Yehuda's suggestion (*) is good

STH: Established that we wanted a delay handler, which produced a delayed template string per Mark's suggestion

MM:

1. Immediate String interpolation
2. Delayed Template String

(Summarized from 1 & 4 from the list above)

LH: This delayed template string concept is now a second string type in the language and I'd be afraid to use them because they I can't rely on them.

MM: Is anyone opposed to #1? (Adam)

RW: Will any of the implementors here remove the template string  feature or tagless template string handler.

No?

LH: I was not comfortable with "quasis" when they required a tag, and even less if they produce a delayed thing.

...This forks the string "type"

WH: That's exactly what I was trying to say earlier. Deferred quasis are too brittle.

MM: Can process based on context

WH: This is just another bug. Makes the problem far worse if you have a function that returns a template string.

DC: The reason we went with template string over String.prototype.format was because this was safer.

MM: (to Mike) Do you find #1 & #4 acceptable?

MS: Yes.

AB: (whiteboard)

```js
var firstName = ...
var lastName = ...
elem.innerHTML = `<h1>Hi ${firstName} ${lastName}</h1>`;
elem.innerHTML = "<h1>Hi " + firstName + " " + lastName + "</h1>";
```

LH & AB: ...discussing the value of #1 & #4

STH: Reviving #2 & #3

0. Drop template strings
1. Current status with default handler that interpolates
2. No tagless template strings
3. tagless template strings with no interpolation
4. tagless template strings produce delayed template string objects, with toString

AR: Leaning on #2 or #3

YK: Nobody wants #2 or #3

WH: I want #2

DH: If we go with #2 or #3, we should just take it out of ES6

Discussion about a provided tag, called "s"

RW: Which means we now have a binding "s" and minifiers and existing code conflicts.

MM: Argument for #3...

DH: This is a tremendous failure of our duty.

MM: My preferred option is that we agree on something
...#2 does not subtract value

BE: Opportunity cost

MS: (Arguments for the current system.)

WH: Some browsers used to have a behavior that allowed regexp to be called without exec, and called directly—should we be able to take that out?

STH: Should a programming language not be allowed to have a feature that all other languages have?

AB: I'm not a programming language designer, I can't answer that

STH: That's a cop-out, because that's what you're asking us to do.

DH: (disdain)

WH: Originally, I wanted #1, now I want #2 because its too easy for people to forget to add "html``"

BM: #0 and #1 are the best cases, #0 adds no additional security risks. #1 is the pure option. #2 changes because you add to the boilerplate. The rest are a new thing.
...I'm for #1 because either you do it or you don't.

STH: Answer this...

Is this something that all languages should do? Or just JavaScript because it's bound to the web?

AR: Yes, we have to pay the cost for the good of the web and yes other languages should've done this (ie, not had the equivalent of untagged template strings).

...wants #3

RB: re: #3, important to have to think about what kind of handler need you need to use.

YK: That might work for your case, but the wider world will just use the interpolation tag.

...

DH: We're flailing. We should just go #2 and just move on, because I'm failing to make my case.

WH: I like #1 and #2, which are similar, but prefer #2 purely for a couple of practical considerations:

- Forgetting the html in html`...` causes a syntax error instead of introducing a security hole.
- s`...` is preferable to `...` because it's practical to search for s` in general-purpose editors, while it's not practical to search for only the occurrences of ` that do string concatenation.

AC: If using this feature and just want to use string interpolation, why should I have to think about "raw" when "raw" doesn't really even reflect what I want.

(Allen leaving. 4, 1, 3, 2)

AR/AC: ...continued discussion

Beginning to devolve.

YK/STH: arguing that it's not the language's job to hand hold

AR/AK/AB: ...disagree

WH: (whiteboard)

```js
AT&amp;amp;amp;T
```

(illustrating the dangers of blindly coding with html`` concatenations everywhere)

(Running out of time)

MM: What is the intersection?

DH: We're headed the wrong way, forcing into a design approach that we don't do.

RW: Should we table?

DH: Yes, that's where I'm heading.

AC: We may never get consensus

DH: We won't give up, we've gained consensus under worse duress.

**Conclusion/Resolution**

- Tabled.