Minutes for the: ***28th meeting of Ecma TC39***

in: ***San Francisco, CA, USA***

on: ***21-23 May 2012***

1 Opening, welcome and roll call

1.1 Opening of the meeting (Mr. Neumann)

Mr. Neumann has welcomed the delegates.

Companies in attendance:

Mozilla, Google, Microsoft, ebay, jQuery, Apple, IBM, Lab 126, Yahoo!

1.2 Introduction of attendees

TC39 Ad Hoc Internationalization (May 21):

John Neumann – Ecma

Allen Wirfs-Brock – Mozilla

Luke Hoban – Microsoft

Bill Ticehurst – Microsoft

Michael Ow – IBM

Steven R. Loomis – IBM

Nebojsa Ciric – Google

Roozbeh Pournader – Google

Douglas Crockford – ebay

Richard Gillam – Lab126

Mark Davis – Google

Norbert Lindenberg – invited expert

Eric Arvidsson – Google

Yehuda Katz – jQuery

Rick Waldron – jQuery

Alex Russell – Google

Dave Herman – Mozilla

Matt Sweeney – Yahoo

Mark Miller – Google

TC39 meeting (May 22-23):

John Neumann – Ecma

Allen Wirfs-Brock – Mozilla

Eric Arvidsson – Google

Bill Ticehurst – Microsoft

Douglas Crockford – ebay

Matt Sweeney – Yahoo

Alex Russell – Google

Luke Hoban – Microsoft

Yehuda Katz – jQuery

Rick Waldron – jQuery

Dave Herman – Mozilla

Oliver Hunt – Apple

Brendan Eich – Mozilla

Isabelle Valet-Harper – Microsoft, Ecma CC chair (part-time, during IPR discussion on Wednesday).

Istvan Sebestyen – Ecma (part-time, via conference call)

Norbert Lindenberg – invited expert

1.3 Host facilities, local logistics

The first day is devoted to the ECMAScript Internationalization project, the 2nd and the 3rd day are devoted to the TC39 meeting.

2 Adoption of the agenda (2012/024-Rev2)

Adopted as presented with some minor changes to order of discussion.

3 Approval of minutes from March 2012 (2012/020-Rev 1)

Minutes approved with recommended change from **Allen Wirfs-Brock**. Changes follow:

It is the part that discusses maximally/Minimal classes. Last page of minutes. Replacement text for what is there now:

Maximally minimal classes:

Alex Russell and **Allen Wirfs-Brock** initiated the discussion as a status up-date to TC39. We pointed out that this proposal had recently been extensively discussed on es-discuss and that it appear to have considerable support from most of the participants in that discussion.

Luke Hoban: These aren't good enough to be a net win.

I'm not sure whether this is an exact quote. **Luke Hoban** certainly did raise the issue of whether classes, as defined by this proposal, added enough functionality to ES to justify the inherent complexity of a new feature.

Allen Wirfs-Brock and **Alex Russell** reiterated that this proposal is only trying to address the most common class definition use cases but in a way that allows for future extensions to address a broader range of use cases. There is significant value in what the proposal provides even if it doesn't do everything any might want.

Dave Herman stated he has some minor design issues he wants to further discuss, but that overall the level of functionality in this proposal was useful and would be a positive addition. He supports it.

Waldemar Horwat: These don't address the hard problems we need to solve.

Concerned about both future-hostility (making it cumbersome for future

classes to declare, say, object layout without cumbersome syntax by

taking over, say, const syntax) and putting developers into a quandry

We discussed this concern quite a bit and did not identify any specify ways in which the current proposal would block future extensions. **Waldemar Horwat** was asked to provide specific examples if he comes up with any. **Allen Wirfs-Brock** pointed out that future syntactic additions can also enforce new semantics. For example addition of a per instance state declarations and a "const" keyword to the constructor declaration could cause ad hoc this.property assignments to be statically rejected, if that was a desired semantics.

-- if they want to do anything more sophisticated, they'll need to

refactor their code base away from these classes. Unless one choice

is clearly superior, having two choices (traditional and extended

object literals) is better than having three (traditional, extended

object literals, and minimal classes). Minimal classes currently

don't carry their weight over extended object literals. Perhaps they

can evolve into something that carries its weight, but it would be

more than just bikeshedding.

The above is a statement of **Waldemar Horwat's** opinion. Other opinions expressed in the discussion aren't record in the original notes.

Alex Russell: We need to do something.

Allen Wirfs-Brock and **Alex Russell** also expressed that it is unlikely that any class proposal that significantly goes beyond will be accepted for ES6.

Debated without resolution.

In summary:

Waldemar Horwat should identify any specific ways that the syntax or semantics of this proposal would be future hostile.

Waldemar Horwat, **Luke Hoban**, and **Mark Miller** expressed varying levels of concern as to whether the user benefit of the proposal was sufficient to justify its inclusion. In order to resolve this question, both sides of the issue really need to provide better supporting evidence for the next meeting.

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

Notes on Internationalization (Attachment 1) taken by **Mr. Cira** (Monday), **Mr. Herman** (Tuesday), and **Mr Waldron** (Wednesday)) for technical discussions follow as attachments to minutes (Attachment 2 and 3).

4.1 A new ES6 (4th) draft is available at (Ecma/TC39/2012/027)

4.2 Proto A newer versions with several corrections:

http://wiki.ecmascript.org/lib/exe/fetch.php?id=strawman%3Amagic_proto_property&cache=cache&media=harmony:draft_proto_spec_rev2.pdf

4.3 Binary Data

4.4 Override Mistake

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

4.5 Revisit for(let;;) binding alternatives

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

4.6 Resolve scoping rules for global lexical declarations

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

4.7 Review proposal for __proto__ in current ES6 draft, and other approaches discussed at

https://bugs.ecmascript.org/show_bug.cgi?id=313

4.8 Consider promotion from strawman to proposal status for "do expressions":

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

4.9 Review candidate spec approach for “regexp_match_web_reality”:

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

4.10 Review updates to ‘observe’ strawman based on feedback from Rafael and Arv:

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

4.11 Proposal to change behaviour of DaylightSavingsTA:

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

4.12 Classes

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

4.13 Weak References (Yehuda)

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

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

4.14 Support for full Unicode character set":

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

4.15 Adding forEach to new ES6 collections (separate from iteration)

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

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

4.16 Object.isObject,

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

4.17 Update on prototype implementation of value objects,

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

(specifically the int64 and uint64 value objects)

5 Edition 5.1 Issues

No discussion

6 Report from the ad hoc on Internationalization standard (Ecma/TC39/2012/0026)

This agenda item was discussed on Monday, May 21, 2012. It was the only subject for discussion on that day.

Notes for technical discussion attached to minutes (Attachment 1).

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

Meeting started around 10:15 and following company members present

19 people from Mozilla, Microsoft, IBM, Google, ebay, Lab126, JQuery, Yahoo, Chair.

Norbert Lindenberg conducted the review of the proposed Internationalization standard (tentatively ECMA-402). His presentation material is in TC39/2012/033. A number of items

have been delayed until next edition since inclusion now would likely cause slippage in the December 2012 approval of the standard. So there will be a second Edition of ECMA-402 at the least. **Mr. Neumann** briefly brought up subject of TC39 requesting RF status for 262 Edition 6 and 402 Edition 1. Members felt that they had to get agreement from their corporate lawyers to make that request. **Mr. Neumann** told them the name of the lawyer in their organization that is involved in this approach.

6.2 Multi-system prototype testing

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

6.3 Next steps. Time Line to success.

Final draft to be prepared for review in August with final approval scheduled for the September meeting of TC39 and send to the Ecma GA for December 2012 approval.

7 Status Reports

7.1 Report from Geneva

7.1.1 Brief report from the CC meeting on April 24-25, 2012 (Updated Intel Patent Declaration on ECMA-262 (Ecma/TC39/2012/012))

Mr. Sebestyen reported from the last CC meeting in April 2012. He said that the TC39 reporting was done on the basis of the Chairmen's report. He said that the current status of TR/104 (262 Test) publication was not fully satisfactory yet, because the Zip file containing the collected test codes to be published had a date of September 2011 which is clearly out of date. He said that for Ecma publication and archival of the TR we need – as long as new tests are added to the test suites – on a regular basis (e.g. monthly) new updates of the Zip file. **Bill Ticehurst** (Microsoft) has volunteered to keep the Test 262 software current and send the latest versions to the Ecma Secretariat.

7.1.2 Discussion on the possible extension of the TC39 experimental software copyright policy and of the Ecma Patent policy

First, an update on the work of the IPR Adhoc was given. There are two subjects on the agenda of the Ad-Hoc: 1) The TC39 request for extending the TC39 Software Copyright policy and 2) The request of a TC39 member to extend the Ecma Patent Policy with a RF optional regime. On both subject updates were provided.

Mr. Neumann expressed concerns regarding the RF patent policy proposal how ongoing standardization projects would be treated from the RAND to the RF transition. He was concerned if the formulation of the policy takes too long, that might create problems.

Mr. Wirfs-Brock suggested that both for ES6 and ECMA-402 the current members of TC39 should give a voluntary RF statement to Ecma. That would improve the situation. **Mr. Neumann** agreed, he suggested that we should request current TC39 members to follow on a voluntary basis the example in Ecma/TC39/2012/012 (Revised Intel patent statement and licensing declaration form regarding ECMA-262). Currently the only patent statement on ECMAScript is that Intel statement. It was understood that TC39 members in this meeting cannot make such decision, but they can act as "postmen" to their own organizations, and then the organizations can decide if they want to submit such voluntary RF statements to Ecma on any of the TC39 standards (especially on ECMA-402 and ES 6). **Ms. Valet-Harper** reminded TC39 that a TC should not discuss such matters.

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

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

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

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

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

Mr. Neumann summarized the opinion of TC39 on the two drafts. They are incompatible with the current use of ECMAScript on the web. Their short-cuts are not in line with ECMAScript and are not necessary. TC39 recommends that Ecma protect its Trademark and disassociate Ecmascript from this ITU activity.

Mr. Sebestyen requested that a well prepared technical liaison should go back to the ITU-T in order that describes precisely the issues and problems TC39 has with the two ITU-T Recommendations. **Mr. Wirfs-Brock** and **Mr. Neumann** will prepare the liaison. Regarding seeking legal protection of the Ecma Trademark "ECMAScript" he certainly has to discuss this with the Ecma management, the normative references in the ITU-T documents are the ISO/IEC standard, the Ecma standard is only listed in the Bibliography. Furthermore to seek legal actions against the ITU-T can be very costly, for which we have little budget, plus it is a rather unfriendly act towards an intergovernmental standardization organization who has just send us in good faith two of their recommendation for comments. **Mr. Sebestyen** also said that actually the source of those two documents seems to be from the Association of Radio Industries and Businesses (ARIB) in Japan. Their Broadcast Markup Language (BML) is an XML-based standard developed in 1999 as a Japanese standard ARIB STD B-24 "Data Coding and Transmission Specification for Digital Broadcasting." And it has a wide spreading and implementation in industry. The other specs on LIME is more recent, only a few years old.

NOTE – The ECMAScript trademarks for Switzerland and the EU (that is what has been granted so far) are younger than the two ITU-T Specs, let alone the ARIB standard.

7.2 Report of the status for a Technical Report on interoperability/conformance tests

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

No further discussion other than as indicated in 7.1.1 above.

8 Date and place of the next meeting(s)

July 25 - 26, 2012 at Microsoft (Redmond)

September 25 - 26, 2012 at Boston Northeastern University

November 28 - 29, 2012 at Apple (Cupertino)

Meeting schedule is good for the rest of the year

NOTE – After the meeting it has been suggested that a third day be added to July and September meetings. This is under investigation as these minutes are being finalized.

9 Closure

Meeting ended around 5:15 PM.

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

Attachment 1- Monday Notes:

By Nebojsa Cira

- We need to get royalty free agreement for Ecma standard at this meeting on Wednesday
- Discussion/forms on TC39 site

- Introduction of the spec (goals, constraints)
- Locale negotiation, collation, number formatting, date formatting
- More was considered but we had to drop some
- Showing examples on how to use the functionality
- Mentioning JS buitins that use the i18n API (toLocaleString...)
- Negotiation of having Unicode extensions and options
- Options take precedence oved Unicode extensions
- resolvedOptions give you what was resolved in the end
- Implementation dependencies
    - Set of supported locales
    - Set of supported features per locale
    - Collation rules
    - Calendars and time zones
    - Various formats for numbers and dates
- Allowing for best-fit algorithms where implementations are free to provide better than default algorithms
- Extensions to the spec are allowed if prefixed by vendor tag
- Testing is hard because of variations in the data and implementations
- Discussion about default behaviours
    - Single data source?
    - More examples in the spec to make developers life easier
    - Allan - 262 doesn't specify everyting so there is a precedent
    - jQuery representative - example of navigator.language across browsers not being the same (en, en-US, en-us)
    - Ritchard - should we focus on each area and see if we can tighten up
- Locale lookup
    - Explanation about de-DE vs. de
    - Even simple lookup algo. won't give the same results
    - Determinism is an issue en->en-US or en->GB
    - Keep the best-fit algorithm as default.
- Collator
- NumberFormat
- DateTimeFormat
- Loomis: Make it clear in the spec (14.3.1,2,3) that the respecified functions behave as if the constructors were called with blank locale list and options (old behaviour is gone).
- Loomis: Add references to third-party implementations to use CLDR or UCA if don't have other options.
- Covered spec. update (collation keys removed, using internal Object type)

- Implementations
  - Add v8toLocaleString to v8?
  - Remove v8 prefix from the implementation because there is no competing proposal
  - Put it behind a flag?
  - Microsoft is continuing in the labs, updated to the current spec, couple of known limitations, but minor. They are shiming toLocaleString. TBD for when is it going to be in the browser. No vendor prefix.
- Test suite
  - Microsoft medium to long term would add tests
  - Google added 35 tests
- Community feedback
  - Passing around locale list - default is a problem - where to put it as a global - spec doesn't say that default locale can't change
  - Pattern string for date format - postponed for version 2.0
- Allen: should there be a global default at all?
- Intl taking locale list as a parameter? That list would become default. But constructors are not frequent in the code, format(), compare() are. And in toLocaleString() is a convenience method, so we don't worry much. No need to change the spec.
- ErikMS: List of default locales. Long discussion about merging lists, what if each service picks different locales? What if locales are not supported at all? 10.1. could be changed to say it could have 1+ instead of 1 iteam? Should new LocaleList() return empty list? A defaultLocale() method/static that returns system default list?
- Skeleton string for dates
  - Skeleton property maybe? (LDML) - v2.0 with agreement
  - Style property (short, long, medium)? - v2.0 with agreement
- Pattern for dates maybe don't have place in the Intl?
- Moment.js for date formatting (lang files) - http://momentjs.com/
- Date formatting - take a look at the other libraries
- Move numbering system information wrt BCP47/ (move from supplemental/)? Clarify the status within LDML spec. Leave couple rows in the table, and point to LDML spec.
- Collator variant name change? Probably good enough.
- List of default locales? Why not have LocaleList() return a list from user preferences?
- Async issue - not an issue since we have at most 9MB of data total, which is mostly trimmed down to 1MB, so it won't block long. Can be preloaded on server to avoid locale switching.
- Support for most likely subtags - v2.0, covered.
- Options in the resolved locale tag - return information being supported (and if was asked for). Drop other extensions. If user passes th only Eric says that we get back -u-nu-thai in addition (a bug).
- Bound format methods - compare is bound others are not. Make format methods bound.
- resolvedOptions - should we create a new object or allow for cached value to be returned? Should the object be mutable? Should resolvedOptions become a method to lower the confusion? Make it a function that returns a mutable object.
- Lookup algorithm spec change - NOTE -> MUST for zh-TW problem. Make it a normative must.
- Luke: editorial comment - left for later/es-discuss.
- Alex/Allen: remove LocaleList object and use plain Array.
- 2. Conformance

    - What about already defined properties? Can we add new, implementation specific values, like v8Identical for collator sensitivity?
    - We should throw if we don't recognise the value. You may recognise additional property values.
- Talk to Patrick to check if the format is conformant to the Ecma standard format.

Attachment 2- Tuesday Notes
By Dave Herman

Participants:

BT: Bill Ticehurst, Microsoft
LH: Luke Hoban, Microsoft
BE: Brendan Eich, Mozilla
DH: Dave Herman, Mozilla
AWB: Allen Wirfs-Brock, Mozilla
AR: Alex Russell, Google
EA: Erik Arvidsson, Google
MM: Mark Miller, Google
RW: Rick Waldron, jQuery
YK: Yehuda Katz, jQuery
DC: Doug Crockford, eBay
OH: Ollie Hunt, Apple

Topic: Binary data

DH: typed arrays were designed for two use cases:
- shipping data from CPU -> GPU for WebGL
- doing binary file/network I/O
DH: former requires matching the endianness expected by the GPU for interpreting shader scripts (e.g. that read bytes as int32 or float64), so they designed it to use the system's native endianness
DH: latter requires explicitly specifying endianness, which DataView API allows
DH: but casting ArrayBuffers to different types exposes system endianness, and basically the entire web is implemented on little endian systems, so the web is being written with the assumption of little endian, despite not being specified
DH: we should specify little endian. bi-endian systems (which many modern little-endian systems actually are) have HW support for little-endian; but big-endian systems can implement byte swapping by compiling shaders to do the byte swapping themselves
YK: I agree; hard for devs to reason about endianness. expecting large use cases for binary data?
DH: yes; e.g. crypto algorithms, optimizing bignum libraries; I wrote a "float explorer" that displays bit patterns of IEEE754 doubles by casting, and discovered after months that my explicit check for system little-endianness was flawed and always produced true, but I couldn't test so I only discovered the flaw by chance.
AWB: agree that most devs will not understand endianness differences
LH: do expect some big-endian UA's
DH: right, in particular game consoles; some have hardware support for bi-endian; how robust that support is is unknown. but can still implement byte-swapping by compiling shaders.
DH: right now, there's no one implementing big-endian browsers making the case that they can't implement little-endian, and meanwhile the web is being written to assume little-endian.
YK: if we standardize little-endian, does it become impossible to discover the system's endianness?
DH: yes, but we could add a feature-detection API that explicitly provides that information. you wouldn't need it though.
DH: if real perf issues arise, willing to consider little-endian by default with explicit opt-in to big-endian. still possible to write unportable code (e.g. blind copy-paste programming), but at least less likely. but let's not solve the problem till we know we have it
AWB: endianness is clear for integers, but what about floats? aren't there other variations e.g. what order the mantissa and exponent and sign go in?
DH: not sure, but we should just standardize whatever format the web is using.
AWB: yes. if you know you need big endian, you'll do the conversion
DH: mostly that's probably just for I/O, where DataView does that for you automatically
DH: standardize little endian?

EA, YK: de facto standard.
LH: I need to contact people on my end; there may become big-endian browsers
DH: broadly speaking, little-endian has won in the hardware world. regardless, this is becoming the de facto standard. HW support will likely get better and better, and leaving it unspecified is solving a problem that doesn't exist, while creating issues of its own.
BT, BE: DataView defaults to big-endian
DH: yes, people are unhappy about that inconsistency. we could change DataView to default to little-endian for consistency. but in the world, big-endian has won as the network byte order, while little-endian has won as the CPU byte order.
BE: a foolish consistency!
DH: right-- the defaults are modelled on reality, and reality is lumpy
BE: yes. <<provides some historical insight>>
YK: what observable effects would this change have on WebGL?
DH: likely none. AFAIK when you create data for WebGL, you don't use casting in ways that care about endianness, you just ship to the GPU, and that's what's sensitive to the endianness
YK: could we eliminate the observable effects by just converting all the data en masse?
DH: no, not that simple; you have to know which bytes need to be swapped
DC: I thought we were going to replace typed arrays. can we just leave it out?
BE: they'll go to W3C. this is reality; we must embrace
DH: we can embrace and then create better, more ergonomic extensions
DC: tell me more
DH: typed arrays are views over ArrayBuffers, which are buckets of bytes; we add a new view type of structs:

```
S = struct({
x: uint8,
y: uint8
});
```

S is not a struct object but a struct type. then:

```
x = new S
```

creates a new instance of this struct type

```
A = Array( S )
```

again, a type, not an instance

YK: that's a little confusing
DH: whole idea is an API for creating new types, so there's a meta-level here that is inherent complexity; can certainly bikeshed API names later
LH: idea we discussed before was making the .buffer null of a struct unless you explicitly constructed it wrapping an existing buffer
DH: yes, and once we do that, we can add pointer types:

```
S = struct({
x: uint32,
foo: object
})
```

DH: for security, *absolutely cannot* expose the buffer to casting
DH: also, enforce alignment constraints to maintain invariant that normal ArrayBufferViews never have unaligned access; will write this up and could use help checking my logic
AWB: isn't there an inconsistency about when there's padding and when there isn't?
DH: no, if you create an unaligned struct type:

S = struct({
x: uint8,
y: uint32
})

then if you construct it fresh, you can't observe the padding, and if you wrap it around a buffer, you get an error:

o = new S; // can't access buffer
O = new S(buf, 2) // error: unaligned type

DH: as always, you can do unaligned access via data view:

d = new DataView(buf, o)

v = d.get(S, 17) // unaligned read

DH: in that example, v is an object pointing to index 17 in buf, and property accesses do the proper logic to perform unaligned access
LH: since strings are pointers, would also be good to be able to have a string type
BE: Waldemar wanted that a while back
DH: I think he was talking about inline strings, which is best treated as a byte array and use Josh Bell's encoding/decoding API (in progress); but I agree we should make it possible to have all JS values; string type is a no-brainer, +1
<<everyone generally favorable>>

Topic: Classes

LH: anything we standardize finalizes basic choice of syntax; maxmin decides we're going to have a new kind of body instead of trying to make class be a constructor-like concept
AWB: we need to agree on something, so let's agree on this basic structure
MM: constructor syntax is DOA for good reason, b/c of the scope issues; in my proposal, class parameters *were* in scope but methods were instance methods; but this was a non-starter for implementation
LH: I think there are ways to mitigate: put constructor locals in scope but poison them
RW: that's too confusing
BE: howling, screaming wart
MM: what's the advantage over maxmin?
LH: syntactic weight is fairly significant
AR: there's weight, but it's a syntactic entryway to more features
LH: blocked off one key door, which is the constructor syntax which is the shortest and simplest of all possibilities
MM: one of the big advantages of JS is simple lexical scope; confusion of poisoned names is more important cognitively and the shorter syntax doesn't compensate
DH: what's the syntactic convenience?
LH: one less level of indentation
YK: at cost of clarity
RW: and you added public!
BE: which cost is worst, public keyword or not having the thing you're calling new on be hoisted to the top?
DH: public keyword is almost outright hostile to programmers; goes completely against the rest of the way JS works
YK: given that maxmin is so minimal, I would like some escape valves for making it something we can build on (extension hooks)
AWB: we need consensus on base level first, before we go into extensions to the proposal
AWB: could you live with maxmin alone?

YK: no. I will receive pressure to use class syntax for my existing class systems, but without at least one escape valve, I will have to choose between saying no or saying yes and killing features
AR: there'll be needs for things like mixins
EA: that's been debunked; RHS is an AssignmentExpressions; mixins are perfectly possible
AR: but there will be various things that can't be done
AWB: you can do this with a function that implements your hook
YK: that asks my users to call a special function
MM: there's simply no proposal for this additional feature for us to evaluate
AR: will you attempt to stone maxmin before they get out the gate before we can agree?
YK: well, no
LH: I won't try to stone classes, but I really hope if we're going to put something in the spec, a) it can actually be used in most cases where most people want to use it, and b) it be forwards-compatible with addressing remaining places
AWB: I wouldn't be behind this if I didn't think both those things are true
YK: I think those things are true; I am personally concerned that I will receive requests for things I can't implement
AWB: I think people will have issues in the short-term but they'll eventually improve
DH: rollout and adoption are important; YK, I'd like to know more specifically what you can do now that you can't do with maxmin
RW: try/catch never would've happened if people said "I can't use this now"; in 3 years, I think you'll rebuild Ember with classes
YK: here's why I know it's a real issue: people want Ember classes to work with CoffeeScript classes, and I have these exact issues
AR: no one's suggesting that it's impossible to do more than maxmin -- even in ES6 time frame -- but we need to agree on base foundation
DH: I think Luke's Ocaml syntax is too confusing, it doesn't win enough for convenience, and it's less mainstream
BE: Ocaml! I knew it!
LH: this clearly has limited amounts of support
AR: we've been through this part of the design process and sympathize, but we've already come to conclusion it doesn't work
MM: keeping scope simple is the most important thing IMO
LH: but what I miss is the instance properties, the ability to see shape of the object is very valuable, e.g. for completion lists in tools
AWB: if you have maxmin as starting point, there are ways to address it
LH: the proposed extensions I've seen are awkward
AR: this has been solved in other languages e.g. C++ initialization list
LH: I won't stand in the way
DH: pulse?
LH: I think Waldemar was totally opposed, I'm not totally opposed, just had concerns
DC: my concern: this isn't new capability; but if it doesn't address all the needs, it'll just add more confusion and won't help
AWB: there's significant debate on a lot of things that could make things easier, but there's a basic structuring that is useful: it's useful not to have to wire up the prototypes properly
DC: but a lot of people are doing this with libraries
AR: but then they're not interoperable
DC: I don't feel compelled to go forward with something we can't be sure isn't right
YK: I see this being on the way to that, but not there yet
BE: you want something, Waldemar wants something else...
YK: I think it's ok if it's a subset, but we should be clear about that, so I can hold the line against using it
DC: tactically I'm opposed to that; if we're on the way but not there, let's do nothing
AR: I was in your camp some time ago. so much complexity to build a proposal that hangs well together that you will have to throw some use cases overboard. we have a long history of not doing classes on the basis that we should wait to do it right. but we also have a long history of iterating on and improving on things that are already in the language. I have faith that we will work to iterate
DC: I'm saying we shouldn't ship until we're sure we're right

DH: design does not have empirical or rationalistic methods of validation. all you can do is discuss, prototype, build examples, and try it out, all of which are part of the Harmony process
DC: I'm fine with consensus
BE: Waldemar wants the ability to have "final" or "sealed" objects that can have errors when mis-named properties, which Mark would like as well
AR: I thought Waldemar wanted *only* that
MM: I think this is a subset of the original class proposal from last spring that Waldemar was on board with; the restrictions were only on const classes
BE: other than future-hostility concerns, which aren't falsifiable
MM: can check whether this is forward-compatible with the May proposal
AWB: that wasn't sound, it was just a whiteboard sketch
BE: Mark's saying that we can grow in the direction of the things Waldemar wants
AWB: I specifically asked Waldemar to say specifically what he thought this was interfering with, but he hasn't come back with anything; don't see why that couldn't be added with more syntax
AR: ISTM there's a fight over defaults
DH: if const is default, we will all be burned at the stake
AR: can't we get something useful if we don't solve the const problem?
BE: it's not clear to me people are asking for what Waldemar is asking for
MM: let's not spend time guessing Waldemar's position
BE: sure, let's just separate the default question from const question; can we agree const should not be default?
MM: I think that was always the agreement since last May
BE: I think when Waldemar returns, we may have to find a way to get past const, to achieve consensus
AWB: I think we should move forward, start specifying maxmin, knowing that we can remove it later
YK: I would like to make a stake in the ground
MM: until Waldemar can make his case, this isn't consensus
AR: there's some room here to suggest we can't wait forever
EA: can we start saying let's build on this instead of waiting another two months?
MM: let's not push until Waldemar has returned
AR: OK, but we do have a deadline, and there's been plenty of time to register complains
EA: I would just like to start prototyping
AWB: and I would like to start doing spec work
MM: sure, we should all push on all fronts, but we don't have consensus without Waldemar's agreement
BE: correct

Topic: __proto__

OH: it's a getter/setter in JSC
MM: one sane approach: refuses to change prototype of object born in another global; other sane approach: refuses to modify prototype of any object that inherits from that context's Object.prototype
DH: what about the getter?
MM: always works; equivalent to Object.getPrototypeOf
AWB: exposing the setter as a function means that any method call with one argument could potentially be a proto modifier
DH: can't you already do that with a function that delegates to .__proto__ ?
AWB: if you thought you deleted Object.prototype.__proto__ but someone squirreled away a copy of the setter
BE: I just don't like it
MM: not fatal for security
BE: we're inventing something new that's not been tested on some pure aesthetic of not wanting magic data properties. I just don't like it
DH: when we already do that with array.length and indexed properties anyway!
BE: this is a turd we do not want to polish
MM: I don't like the way it's specified

DH: sure. I think we agree about the behavior
BE: agreement: it's in Annex B, it'll be a pseudo-data property
MM: I don't think that's clear
EA: I would pick an accessor
BE: this isn't a clean-slate design experiment!
MM: b/c Firefox doesn't implement the accessor?
BE: b/c no one other than JSC very recently
MM: we can analyze the security properties
BE: this is not about security, it's about unintended consequences
YK: is there an advantage to making it an accessor?
MM: yes: the actual action is that it has a magic side effect
BE: that's an aesthetic preference
OH: the pre-accessor behavior of JSC was that every object instance had the magic property; from our point of view, being able to extract the accessor is in no way different from what you could already do
BE: acid test: `o.__proto__ = null` -- what happens?
OH: in old JSC, it remained magic
DH: comes down to distaste of another magic data property vs distaste of a portable, well-specified proto-updater function
YK: people will use it
BE: and maybe someone will come up with some zero day, I can't predict that
DH: it's strictly more risk for sake of a purely aesthetic reason, in an already aesthetically nasty space
BE: yes, should be a risk analysis, not an aesthetic analysis
YK: I agree
MM: I would like a more modular specification than poisoning the semantics of [[Get]] etc; but think of how expensive the magic of array length has been
BE: this is already shipping in a bunch of browsers, and in some of them it's already a magic property; the developer-facing cognitive load is a wash; developers just want it to work, they don't care whether it's an accessor
DH: I can predict the security bugs: the implementor just thinks about the normal case, but the attacker takes the accessor out, installs it on an object that inherits from a proxy to an object from another global etc. etc. and something internal breaks
MM: that's the most compelling argument I've heard. the additional testing surface area is much bigger
DH: Arv?
EA: I'm not gonna block this
OH: I think it would be nice to have a mechanism that let you specify getters/setters that weren't extractable
DH: actually proxies basically *are* such a mechanism
AWB: I don't like getters/setters that can't be reflected

Topic: Spec terminology

AWB: confusion about "host object"; I'd like to eliminate concept of "host object" entirely, introduce new terminology
- object: runtime entity w/ unique identity and exposes properties
- mundane object: object that uses default behaviors for chapter 8 internal methods
- exotic object: object that provides non-default behavior for at least one of the internal methods
DH: I would say that "plain" is better than "mundane"; also value objects, were they to succeed in Harmony, don't have identity
AWB: we can cross that bridge when we come to it
DH: anyway I like this; makes clearer where the extension points of the language are
AWB: note that proxies are exotic
MM: are array instances exotic?
AWB: yes
DH: is that a problem?

MM: no, just surprised me
AWB: I'd like functions and other "normal" things to be classified as mundane
DH: but arrays should still be exotic?
AWB: yes
DC: Ecma members can be "ordinary"
AWB: ok, "ordinary" and "exotic"
AWB: another dimension of classification?
- standard object: defined by ES specification
- built-in object: provided by ES implementation but not defined in ES specification
- platform object: provided by host environment
<<mass confusion>>
AWB: I agree these distinctions are unclear
DH: I think the first dimension was great! it moves us in the direction of not *needing* the second dimension
AWB: I agree, and I will see if it's possible to eliminate the second dimension
AWB: now, a dimension for functions:
- function: an object that exposes the [[Call]] internal method
- ECMAScript function: function whose invocation result & side effects are provided by evaluating ECMAScript code
- alien function: function whose invocation result & side effects are provided in some manner other than by evaluating ECMAScript code
- standard function: function whose invocation result and side effects are defined by the ECMAScript specification (mostly ch 15)
DH: why do we need this? job of implementation is to make it indistinguishable that a function is alien
AWB: setting up initial environment has to keep track of variable environment etc
BT: it's unobservable
DH: you can't tell!
LH: are you saying there's a flaw in the current ES5 spec?
AWB: when you activate a chapter 15 function, you don't go through any of the normal process
MM: Function.prototype.toString has requirements on functions that *must* be ECMAScript functions; so there's a distinction between "must-be" and "may-not-be"
AWB: but Chapter 15 functions need access to the current global's intrinsics (e.g. for "initial meaning of new Array" etc)
BE: you can get to that via [[Scope]]
DH: I would rather eliminate the alien function distinction, and the steps in Chapter 15 can be implemented in pure ES
AWB: I'll just have go through and see how many distinctions I can eliminate
MM: what are the "intrinsics"?
DH: things like where the spec says "as if via `new Array` for the initial value of Array"; loaders rationalize this, and each loader has its own set of intrinsics
YK: "realm" is used in basic auth
DH: "realm" is a little thesaurus-y; I like "context" but "ExecutionContext" used in ES already, so "global context"
AWB: "context" doesn't feel big enough
MM: so [[Scope]] gets you to the context?
AWB: you can share global objects between loaders!
DH: oh right, so [[Scope]] is not it -- needs its own [[Context]]
AWB: top-level environment record gets you to it
BE: don't have two independent fields that are required to agree
DH: yeah, as long as we can get to the global context via the top-lex environment record, we should just get to it via that
DH: popping the stack, I like "global context"
YK: I like "global context"
BE: "context" is almost meaningless
MM: "shire"... lulz
AWB: "island"?
DH: <<winces uncomfortably>>

DH: "home"?
AWB: not bad... already using for something else but I could rename it

Topic: do-expressions

DH: allows you to evaluate expressions with control flow or temporary variables:

```
a = do {
let tmp = f();
tmp * tmp + 1
};
```

workaround:

```
let tmp; // scope pollution!
a = (tmp = f(), tmp * tmp + 1);
```

workaround:

```
a = (function() {
var tmp = f();
return tmp * tmp + 1;
})()
```

YK: CoffeeScript gives you the latter
DH: the IIFE doesn't "say what you mean" -- boo boilerplate
YK: also, TCP hazards
MM: strongest argument in favor is for code generators
DH: well, I think the developer convenience is an even stronger argument
EA: I want this most for code generators
MM: strongest or not, it's still compelling for code generators
DH: arguments against that I've heard:
- eliminates the current structure of JS syntax where statements are always at outermost level of a function body
- exposes the completion value of statements, which is only otherwise observable via eval
YK: it's pretty clear what the result is
DC: the "do" syntax will be confusing to those familiar with do-while
YK: you can think of it as do-while-false
DH: I think it reads really naturally as English: "do this" means do it once, "do this while blah" means do it some number of times
DH: pulse of the room?
<<mostly favorable>>
LH: my standard concern: the overall cost of so much new syntax; I would certainly use this, and I like it, but all these conveniences add up in cost
DC: I don't think we ever consider the overall design
DH: sick of that accusation; we constantly consider the overall design
BE: there is an important point about the cost of syntax, and we need to take account of all of the syntax we've proposed
AWB: worth making everyone come up with a priority list?
BE: that's a little "fighty", but we will need to take stock and make some cuts at some point
DH: I imagine I'd cut do-expressions before anything else
EA: I want them more than comprehensions
DH: yeah, I've championed comprehensions for so long I hadn't really reconsidered; I love them but I could see how you might prioritize do-expressions
DH: I want to make a couple points about the big picture:
- there is a meme that TC39 doesn't understand complexity budgets and is out of control

- please do not spread this meme; it's false
- one thing the recent controversy, including JSFixed, has shown me is that TC39 needs to improve our communication
- there's a lot of misinformation, and a lot of unhappy people, when they start talking with us, end up finding out they're actually pretty happy with the direction ES6 is going in
- I will be redoing the entire wiki to focus on community first, committee second, and to make the big picture clearer
- but we all could think about ways to increase our communication with the community
AWB: in some ways the anger is actually a consequence of being open
DH: yes, but we can still improve our communication
BE: we still need to think about whether and what we should scale back
AR: people are resistent to change; once features land they'll use them
DH: tension between needing to grow a language incrementally but about the standards process taking many years between revisions; either underserve users by simply not growing enough, or freak them out by landing many things at once
YK: roll-out of individual features in browsers can help, even preffed off
DH: just want to say that we *do* consider complexity budget, we *have* cut a ton of things, and the ES6 feature set has really strong coherence
AR: go team!

Attachment 3 - Wednesday Notes

By Rick Waldron

4.10 Object.observe (Rafeal Weinstein)

Rafeal Weinstein:
- Introduction to proposal

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

MM:
- In favor of spec'ing an event loop

BE: let's subset the parts of Hixie's HTML5 spec that we need, and no more (and reference that spec so it's likelier we keep the subset relation).

AWB:
- There is too much going on at low level with this proposal

Allen's point was that all of the strawman champions and Allen, the spec editor, and experts and interested folks (Tom Van Cutsem, people on es-discuss) all need to digest the non-local effects of the local but low-level changes in the strawman.

I said this, and also offered that we would prototype in SpiderMonkey alongside V8.

RWeinstein:
- Asking for blessing to prototype in v8: approved.

I emphasized that all strawman not moved to deferred: namespace in the wiki (and there's a curation process here, which can fail sometimes) are fair game for prototyping.

Property Assignment Shorthand

Rather, "Object Literal" or "Property Definition" (in ES3, PropertyAssignment) shorthand.

eg.
function point(x, y) {
return { x, y };
}

{ x: x, y: y }

This is too unclear
w/r to destructuring: both, or nothing.
Plenty of opposition based on clarity.

Resolution: Nothing, no change.

That is, we keep the shorthand for *structuring* as well as for *destructuring* where it is clearly useful.

LH observed that the 90%+ by frequency use-case for destructuring wants the shorthand, whereas for structuring (making an object via an object literal expression) it's more like 10% or 1%. BE agrees but BE and MM at least want duality consistency: shorthand for both literals and patterns.

That's where we left ES6.

I personally think the "n00bs will see the new shorthand and artificiailly declare variables of the same name as the property just to use the shorthand" worry as an undemonstrated speculation, and too fearful. We need user-testing evidence of this!

EA raised the other interpretation of {x,y} in an expression, as a WebIDL-ish enum equivalent to {x: "x", y: "y"} -- for making string manifest constants named by the string contents -- but this is a different semantics, and rarer yet than {x,y} to make a point from x and y bindings.

AR mentioned future-hostility to set notation in JS. BE ack'ed that, figures it is un-possible given object literals being wrapped in curlies anyway.

Triangle

Is capable of giving up private names

If people have __proto__ they will not use <|.

MM:
- From a security perspective, I'd like to move __proto__ out of annex B and into normative body

BE:
- If MS puts __proto__ in IE, then it becomes defacto standard and we might as well put it in the standard.

1. __proto__
2. grawlix

Resolution: Indefinite postpone\

I.e., we defer/cancel triangle and go with normative mandatory __proto__. W00t! Kidding, mostly, but I perpetrated it, it is in all browsers but IE, and IE feels the heat on mobile (thank you Thomas Fuchs!). So __proto__ is the winner already, we're just trying to forestall the inevitable.

Moustache

o.{ p: v }

BE:
- Propose that moustache is kept, but still consider Object.extend

MM:
- Would like to consider both moustache and Object.extend

How about… both?
= assign
: define

ie.

o.{
p1: "to define",
p2 = "to assign"
};

Further choice: allow = instead of : in object literals too, not just in mustache-literals!

Positions:

Just Define
Just Assign
Both Define and Assign
Alone
Applied to object literals

Ah, you got that one. Thanks.

OH worried that the JSON theft attack (a global setter targeting well-known id in top-level JSON blob) would work, but this requires both = instead of : in the JSON blob, and the wrong MIME type for loading JSON as a <script>.

Resolution: Return to strawman for revision, needs own wiki page.

I still want mustache. My advice was to copy and lightly mutate the ObjectLiteral grammar instead of reusing it, to allow = and : as alternatives for assign and define, respectively; and for any other reason. Sometimes pattern languages diverge from expression grammars for good reason.

Computed Keys

Erik Arvidsson:
- Today, properties are static and knowable, computed keys are not knowable.

Resolution: Deferred.

But just to capture all the nuance, the idea of @privateName: value in an object literal has support and maintains the compile-time name structure that EA rightly cited.

The [anyExpr]: value idea for a property definition in an object literal

is what I believe died today. If you want to extend an object with a
computed property name, use assignment or Object.defineProperty.

Weak Refs

Yehuda Katz:
- Use case: observers

Resolution: Continued, due to lack of time.

BE: valid use-case for observers, virtualization layers, other such
bridging functions. MM/AWB need to discuss more, ideally here on es-discuss!

Value Objects

Brendan demos, explanation of the implementation and operator overloading

See https://bugzilla.mozilla.org/show_bug.cgi?id=749786 and stay tuned.
