Minutes for the: **_22nd meeting of Ecma TC39_**

held in: **_Santa Cruz, CA, USA_**

_on:_ **_24 – 26 May 2011_**

1 Opening, welcome and roll call

1.1 Opening of the meeting (Mr. Neumann)

The TC39 meeting (hosted by UC Santa Cruz in Santa Cruz at the University) was opened by **Mr. Neumann**, Chair of TC39 at approximately 10:30 AM on 24th May 2011 (TC39/2011/022 - Venue for the 22nd meeting of TC39, Santa Cruz, May 2011).

It was noted that before the TC39 meeting, on the 23rd of May 2011 the TC39 ad-hoc group on internationalization has met. The report of that meeting is given under 4.2 below.

1.2 Introduction of attendees

Mike Samuel - Google

Alex Russell - Google

Istvan Sebestyen - Ecma-International

Waldemar Horwat - Google

Allen Wirfs-Brock - Mozilla

John Neumann – Ecma International

Sam Tobin-Hochstadt - Northeastern University

Dave Herman - Mozilla

Douglas Crockford - Yahoo! - Phone

Cormac Flanagan - UCSC

Brendan Eich - Mozilla

Mark Miller - Google

Luke Hoban - Microsoft

David Fugate – Microsoft

Avner Aharon – Microsoft

Bill Frants – Periwinkle (invited)

1.3 Host facilities, local logistics

Mr. Flanagan welcomed on behalf of UCSC the delegates and provided logistical information. It was announced that Ecma international would host a social event on May 25th evening.

2 Adoption of the agenda (2011/024 Rev 2)

Ecma/TC39/2011/024 Rev 2 contained the Agenda for the 22nd meeting of TC39, Santa Cruz, May 2011. This was agreed with minor changes to group subjects.

The relevant Ecma TC39 contributions for the meeting are the following:

- Ecma/TC39/2011/020 Results of Final Review of SC 22 N 4595, ISO/IEC DIS 16262
- Ecma/TC39/2011/021 Minutes of the 21st meeting of TC39, San Bruno, March 2011
- Ecma/TC39/2011/022 Venue for the 22nd meeting of TC39, Santa Cruz, May 2011
- Ecma/TC39/2011/023 TC39 chairman's report to the CC, April 2011 (Rev. 1)
- Ecma/TC39/2011/024 Agenda for the 22nd meeting of TC39, Santa Cruz, May 2011 (Rev. 2)
- Ecma/TC39/2011/025 Draft press release: ISO/IEC and Ecma International ratify “ES5.1” (Rev. 1)
- Ecma/TC39/2011/026 Documents mentioned in the clauses 6.1 to 6.32 of the agenda of the 22nd meeting
- Ecma/TC39/2011/027 Status Report "test262", May 2011

Other documents are mentioned via their URL to the ES Wiki.

The more detailed technical notes by **Mr. Horwat** are attached to this report.

3 Approval of minutes from March 2011 (2011/021)

The minutes of the 21st TC39 meeting in March 2011 have been unanimously approved with no changes.

4 Report from the Secretariat

Regarding TC39 documentation **Mr. Sebestyen** said that in Ecma/TC39/2011/026 for Ecma TC39 archival purposes a snapshot from the ES wiki has been taken (status: as before the meeting). Those documents are mentioned in the technical clause 6 of the agenda of the 22st meeting. He explained that long-term archival by an SDO is a typical requirement for an open standardization process, that a decent SDO has to follow. However, unfortunately the ES wiki is an unofficial tool operated outside of Ecma and also does not have a long term guarantee to stay alive for decades – however practical it is for the time being. He explained that the “snapshot” currently is taken by “printing” the content of the technical contributions as PDF files and merge them in a single file. For the Ecma Secretariat this is a rather time consuming exercise, and he was curious if a simpler and better solution existed. TC39 said “Yes”, and they have promised to come back later with concrete practical proposals to solve the problem.

Then the question was asked about the publication status of ISO 16262 3rd Edition. Everything is finished, we are just waiting for the publication. According to the information from ISO and IEC the final version of ISO/IEC 16262:2011 will be published on 15th of June 2011.

Regarding the TC39 Software Copyright policy it was reported in the March 2011 TC39 that for ES5 testing the question came up, if software from non Ecma members can be accepted.

As agreed in the March meeting the CC has been notified about the issue. The conclusion of the discussion in March was that the CC should be made aware that such request has been made, but we should first get more experience with the experimental policy before we ask for additions / changes. This view was also shared by the CC. So for the time being Ecma should not directly accept third-party contributions. However, an Ecma member could serve as an intermediary.

Mr. Sebestyen also said that on June 29, 2011 at the GA there will be a celebration of the "Golden Jubilee" of Ecma. It was decided by the CC that a series of short presentations should be given including some standardization highlights of Ecma. Since ECMAScript is one of such highlights, **Mr. Neumann** is requested to give a 5-10 minutes short presentation about the success-story ECMAScript. This involves the preparation of a few PowerPoint slides. After the 50 years celebration all PowerPoint slides will be bound together and published on the Ecma website.

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

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

The slides of the report given to TC39 is to be found in Ecma/TC39/2011/016 - Status report "Test262" of March 2011.

The performed tests show where the most current problems in implementations are (Strict mode, Annex C, Chapter 10).

4.2 Report from the ad hoc on Internationalization standard.

Ecma/TC39/2011/0xx contains the Agenda for the 4th meeting of TC39 ad hoc on Internationalization, 23 May 2011.

A summary report on the ad hoc group's work on Internationalization was given. For more details see attachment to this report.

The plan is to have a final draft for the 2011 September TC39 meeting for approval at the 2011 December GA.

4.3 W3C Joint work items

None.

5 Progression of ES 5.1

5.1 Press Release

In TC39/2011/025 Rev. 1 the second draft of an Ecma Press Release has been circulated. It covers the upcoming ECMAScript 5.1 approval in Ecma, the ISO/IEC approval, and Test 262, and it should be ready for publication immediately after the June GA meeting. **Mr. Neumann** has requested TC39 members to review the draft, and to suggest some quotes. For deadline for comments the end of May has been agreed.

5.2 Publication

On the Ecma side everything has been done. We are waiting for the ISO/IEC publication, which is scheduled for June 15, 2011.

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

6.1 Syntax (Tuesday)

6.1.1 Promotion decisions regarding RegExp

"Match Web Reality" (including RegExp.prototype.compile)
http://wiki.ecmascript.org/doku.php?id=strawman:match_web_reality
Other RegExp extensions and API improvements. (who are the champions??)

6.1.2 Pragmas -- don't need to settle on a concrete design, just agreement that we should future-proof a pragma syntax

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

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

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

6.2 objects and classes (Wed. AM)

6.2.1 "Subtyping" and setting the prototype of various literal object forms

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

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

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

6.2.2 Object Literal Extensions

review for possible promotion
http://wiki.ecmascript.org/doku.php?id=strawman:basic_object_literal_extensions

6.2.3 Private names

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

6.2.4 classes

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

6.2.5 Scoped Object Extensions –

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

6.2.6 "Trademarks/guards" (the language feature, not the legal concept).

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

6.2.7 quasis

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

6.3 API's (Wed PM)

6.3.1 soft fields

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

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

6.3.2 better random

http://wiki.ecmascript.org/doku.php?id=strawman:random-er

6.3.3 simple maps and sets

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

6.3.4 Full Unicode Support:

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

6.3.5 Promotion to proposal status for various enhanced math functions and operators. See the "Number and Math Enhancements", Mr. Neumann will be preparing a list of specific proposed functions based upon the spreadsheet

Also see http://wiki.ecmascript.org/doku.php?id=strawman:modulo_operator

6.3.6 String extras

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

6.4 functions (Thurs AM)

6.4.1 Function.prototype.toString

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

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

6.4.2 Arrow Function vs Block Lambda

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

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

6.4.3 Array comprehensions

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

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

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

6.5 control flow (Thurs PM)

6.5.1 concurrency

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

6.5.2 Deferred Functions - Deferred functions enable writing asynchronous code in a linear style where you would otherwise use callbacks and CPS transformations

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

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

6.5.3 * Enumeration -- again, doesn't need much conversation, just agreement that we want to nail down the semantics

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

6.6 modules (Thurs PM)

6.6.1 simple module functions (was "generative module expressibility")

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

6.6.2 Multiple globals -- shouldn't need much conversation, just agreement that this reality should be reflected in the spec.

6.6.3 Better databinding via object change notifications

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

7 Date and place of the next meeting(s)

July 27 (Wednesday) – 28 (Thursday), 2011. Location: Redmond, WA, hosted by Microsoft.

Tentatively, the September meeting (likely on 28-29, September 2011) is planned to be in Boston, hosted by the Northeastern University,

8 Closure

The TC39 Meeting ended at 4:17 PM on 26 May 2011. **Mr. Neumann** has thanked the meeting participants for their hard work and co-operative spirit.

The group expressed appreciation to the University of Santa Cruz and to **Mr. Flanagan** for hosting the meeting and to Ecma International for hosting the dinner the night before.

Item 4.2 Attachment

ECMAScript i18n Ad-hoc meeting summary (5/23/2011):

Attendees:

Jungshik Shin (Google),

Mark Davis (Google),

Richard Gillam (lab126 for Addisson)

Nebojsa Ciric (Google).

Shawn, please take a look and see if there are any pain points with changes we talked about (skeleton for number format, removal of calendar parameter in date time format, and collation in general).

- Language matching
- Date time formatting
- Number formatting
- Collation
- Parameter handling

Language matching

- Match base language (de-DE) with supported locales.
- Keep the original extension for matched locale.
- If one of the requested locales is exact match to supported locales then exact match should be picked.
- Order in the list breaks ties.

Example:

request [A, B], supported {A, B}, pick A, order breaks tie.

request [A, B], supported {A', B}, pick B. A' is a near perfect match.

request [A, B], supported {A', B'}, implementation dependent.

Date time formatting

- timeType - long and full values are optional.
- dateType - medium and full values are optional.
- Added link to LDML for skeleton fields description (to strawman).
- Narrow width is optional for date time symbols ('abbreviated' and 'wide' are required).
- Calendar option removed from settings. We'll use locale -u-ca-calendar_name to specify calendar.

Number formatting

- Do we need integer style? Probably not since decimal style can do the same if we pass integer as a value.

- Keep pattern parameter for cases where users want strict format.

- Add skeleton to help with automatic placement of: sign, currency, percent symbol and grouping digits.

- Added link to LDML spec for number format (to strawman).

Collation

- Drop type parameter (search/sort), and let user set sensitivity level for search.

- Added detailed section with parameters to strawman (numeric, ignorePunctuation, sensitivity(base, case, accent, all, default...).

- User can specify some of the parameters in the languageID i.e. -u-kn-true.

Parameter handling

- Open issue: If parameter is not a valid type (Date, number or string) what do we do?

- Throwing an exception is ugly since each format/constructor would have to be enclosed in try/catch.

- Returning a default (current date, 0 or "") may lead to problems when developer misses the error he made.

- Return undefined?

- If language id is ill formed identifier:

 - Initial well formed subtags are preserved ("und" if empty) and others are dropped.

- If language id is invalid (but well formed):

 - xy-666 - keep it and do best you can - implementation dependent.

Summary done by: Nebojša Ćirić

Item 6 Attachment

Notes kindly provided by **Waldemar Horwat.**

Tuesday notes:

To: es-discuss@mozilla.org

Sent: 5/26/2011 7:22:23 P.M. Eastern Daylight Time

Subj: May 24-26 rough meeting notes

A whole bunch of agenda reordering

Test262 report slideshow

Debate about who should host the test262 website

Deferred discussion

[For the purposes of these notes, Harmony and ES.next are synonyms. We were using "advanced to Harmony" to mean "advanced to ES.next".]

"RegExps match reality":

Waldemar: Omitting these error cases was deliberate in ES3. They shouldn't be in the base language any more than getYear or HTML string functions should be. If they go anywhere, it should be in Annex B.

Debate about whether these are extensions or conflicts with the spec. MarkM: conflicts should be codified in the spec; extensions should not be normative.

Debate about whether these are syntactic or semantic extensions and whether they fall into the chapter 16 exemption. Waldemar: even if they don't currently fall under the chapter 16 extension, we could make them fall under the chapter 16 extension.

Brendan, Allen: In the next spec we'll have an optional normative section of web reality extensions. Put these in that section, along with things like getYear and compile. Consesnsus reached on this approach for Harmony.

Lookbehind support is promoted to required normative.

Multiline regexps are out of Harmony. There is no currently written-up proposal.

Pragmas:

Would rather have these than string literals. Don't require "use" to be a reserved word. Controversy about whether unrecognized pragmas should be ignored (Brendan: both ways bite back), but (I think) that's an unsolvable problem.

Consensus on moving these to Harmony.

Versioning:

```
<script type="application/ecmascript;version=6">  (RFC 4329)
use version 6;
module LOL {
  ...
}
</script>
```

There are good reasons to have the metadata both externally (in the script tag) and internally (in the script). External versioning allows implementations to avoid fetching a script at all if they won't understand it. Internal versioning helps in the case where the external version is detached.

Brendan's idea:

```
<script-if type=...>
  ...
<script-elif type=...>
  ...
<script-else>
  ...
</script>
```

Consensus on moving some form of versioning into Harmony. The strawman is a bit light at this time, so no specifics yet.

MemberExpression <| ProtoLiteral syntax:

Why can't ProtoLiteral be a variable? Could extend the syntax to allow expressions there, with <| doing a shallow copy. Copy internal properties as well? Not clear what that would mean for some objects such as dates.

Shallow copy is problematic if right side is a proxy. Would need a clone-with-prototype handler.

Waldemar: <| seems too ad-hoc. Would want a more general mechanism that also allows for creating an object that's born sealed.

Brendan: Object literal extension defaults seem ad-hoc. Constants (and maybe methods) shouldn't be configurable.

MarkM: "Mystery Meat" problem here. Not comfortable with Perlish "punctuation soup".

Allen: Words are too verbose, which was the feedback from past meetings.

Waldemar: Improve the usability. Would prefer to set configurability en masse on the properties of an object rather than having to mark each one. := should go back to being called "const" and should come with the right defaults so that no other modifiers are needed in the common case.

Discussion about dynamic vs. static super lookup. When a method is extracted, "super" used in a . (or []) expression stays bound while "this" is dynamic. Note that a bare "super" not in a . or [] expression means "this".

MarkM: Gave example of why super can't have simple dynamic semantics such as "start from the 2nd prototype". This causes infinite loops in class hierarchies. More complicated semantics where both a "this" and a "super" are passed in every call might be possible.

Brendan: super without class is too weak and causes problems in nested functions.

MarkM: super must be in a . or [] expression.

What should super.foo = 42 mean (where super is an lvalue)? What if foo is not an accessor?

Private names:

Note that you now can't use the public name gotten from for-in or getOnwPropertyNames to index into the object.

Should the name objects be reflectable via getOnwPropertyNames at all?

Waldemar: Objects to leaking the presence of private names. Wants no reflection on private names, with proxies doing membranes via the getI/setI approach.

Discussion about interaction of shallow cloning with private names.

Advanced to Harmony, with the big open issue of reflecting on private names.

Classes:

"constructor" is a contextually reserved word.

Waldemar: Classes as an abstraction should have a clear way of (preferably enforceably) documenting the shape of instances: what properties they have, what they are (getters/setters/guard), etc. This proposal is too weak here.

Waldemar: A const class must define its instance properties; otherwise the constructor won't be able to create them.

MarkM: Instances aren't sealed until the constructor finishes.

Waldemar: If it's the local constructor, that won't work. If it's the entire chain of constructors, then that can be indefinitely later, and the derived classes can muck with the unfrozen/unsealed contents of the instance. For example, if the attributes like const don't take effect until all constructors finish then a derived class can alter the value of this class's const instance fields.

MarkM: Wants clarity about scope and time of evaluation.

Brendan:

```
class C {
  get x() {return 42}
  x = [];
  static x = [];
}
```

Issue about puttig a mutable x = [] on the prototype and the instances colliding because they all try to insert elements into the same array rather than each instance having its own array.

Waldemar: Don't like the subversion of const away from its semantics in the rest of the language, where it means write-once-by-definition with a read barrier before the definition. Here a const field can be mutated many times even after the constructor finishes. That won't work for guards.

Brendan: Proposed correction:

```
constructor(x, y) = {
  x: x|| default_x,
  y: ...
}
```

MarkM: Had earlier declarative instance construction proposal, but Allen convinced him that the imperative style is the most familiar to ECMAScript programmers.

Allen: No I didn't. Declarative construction is important.

Waldemar: There's someplace you need to specify instance attribute properties, and some of them can only be specified at the time the property is created.

Also, need to be able to interleave declarative initialization of instance properties with computation of temporaries so declarative initializations can share work.

Discussion of older, closures-based version of the class proposal, which have a simple declarative syntax:
http://wiki.ecmascript.org/doku.php?id=strawman:classes_with_trait_composition&rev=1299750065

This one is simple but defines properties only on class instances, not on the class constructor or prototype.

Mixed proposal:
http://wiki.ecmascript.org/doku.php?id=strawman:classes_with_trait_composition&rev=1305428366

This one is more complicated because it allows definition of properties both on class instances and on the class constructor and prototype.

MarkM wrote a table comparing the three proposals:

	Closures	Mixed	Separate
Class properties	none	static in class	static in class
Prototype properties	none	public in class	decl in class
Instance private	lexical capture (in all)		
Class private inst	private decl	private in ctor	expando only
Public inst properties	public decl	public in ctor	expando only
Return override	none	none	optional

Constructor code	yes	yes	yes
Class code	no	yes	no

expandos are dynamically created properties: this.foo=42

MarkM's preference is Closures (best), Mixed (middle), Separate (worst).

Group trying to come up with a variant of the Mixed proposal.

Brendan: What's the order of initialization in a class hierarchy? Declarative instance initialization should be able to rely on super instance initialization having run once they call the superconstructor.

Scoped Object Extensions:

Waldemar: These use the same mechanisms as early ES4 namespaces.

What happens when extensions conflict with each other? Can't be a compile-time error. Run-time error.

Extension "objects" (perhaps better called Extension records) are themselves not extendable.

Waldemar: What happens when you write to an extension property? Delete one? Proxy one?

Peter: Make them immutable.

Alex: Won't work. Such things in the prototype will inhibit shadowing.

Peter: Make them writable but nonconfigurable.

Waldemar: What happens when you have an extension A.count, and then code does A.count++? In the current semantics this creates an invisible non-extension property on A and doesn't work.

Peter: You shouldn't be writing to extensions.

Waldemar: We've already ruled out the immutable model. Can't change the get rule without also changing the put and related rules.

DaveH: Let's look for alternatives. Could this be layered on top of private names or such?

Waldemar: No.

Debated extensions vs. private.

Lexical restriction of extensions is a blessing as well as a curse. If a module extends Tree with a where method, that module can't call a Tree utility in another module that relies on "where".

Waldemar: Membranes don't work. The proxy would need to get a hold of the lexical scope.

getOwnProperties won't see extension properties because it's a function.

Freezing won't work -- won't see extension properties because it's a function.

DaveH: Agree that this solves an important problem, but the issues of the implications on the object model are severe.

Sam: We'll need to reify GetOwnProperty for proxies to take lexical scopes as arguments.

Brendan: Not just proxies. Proxies and put are sub-problems of the larger problem of making a significant change to the metaobject protocol.

Luke: There are other things already promoted to Harmony that are as underspecified and cross-cutting as this.

Brendan: The issue is that this proposal is so new.

Waldemar: The issue is that this probosal is not new; over the years we've ran against deep problems when working in this area in the context of namespaces.

Debate about the Harmony process.

Brendan's summary of consensus: We'll try to get into ES.next but not promoted to Harmony yet. Need to address the problems with semantics and get some code experience.

Trademarks/guards:

[didn't take detailed notes because I was driving discussion]

Very long debate about merits and process.

Not promoted to a proposal for Harmony. Dave: Insufficient time to experiment with this before 2013.

Random: First paragraph accepted, second not.

Quasis:

Backslash sequences are not interpreted when passing to the quasitag function. The default is join, so:

`foo=${void 0}\n` evaluates to "foo=\\n".

`\u1234` evaluates to "\\u1234".

`$\u1234` evaluates to "\u1234".

Allen: Worried about yet another escaping syntax.

Waldemar: Also unhappy about confusion of escaping syntax overlap. Example: `foo = \$`. RexExps containing $ characters would be misinterpreted.

Waldemar: Alternate proposal: Use only the regular \ syntax for escaping. An unescaped $ followed by an identifier would turn into a substitution. All other $'s would be passed through literally. Also, it should be up to the quasi parser whether it gets the escaped or raw text (with the default being escaped). The quasi parser function could specify its desire via a property on the function.

Devate about what could go into substitution bodies.

Waldemar: Grammar is not lexable via the same kind of lexer technology used for the rest of the language; lexer is not a state machine any more. OK with allowing only sequences of dot-separated identifiers.

String formatting: Not on agenda, so not in Harmony.

Brendan: We haven't got it together yet. It doesn't address injection attacks.

Classes and privacy revisited:

- Return allowed?

- Verbose repetition of public/private prefixes

- One or two namespaces for instance variables

- Require a prefix like "public" to define prototype data properties

- Private proto-properties
- private(this) vs. alternatives
- Attribute controls
- Half-override of get/set pair

Return allowed? No consensus. Return is useful for migrating current code and memoizing but conflicts with inheritance.

Verbose repetition of public/private prefixes: Agreed to solve this by allowing comma-separated declarations: public x=3, y=5;

One or two namespaces: Consensus on two.

Require a prefix like "public" to define prototype data properties: Consensus on yes.

Private proto-properties: Consensus on yes.

Long discussion about private instance variable access syntax. private(expr).x is ok as a long form for an arbitrary expression, but we need a shorthand for private(this).x. No consensus on which shorthand to use.

Waldemar: Use just x as the shorthand (every private instance variable x would also join the lexical scope as sugar for private(this).x)?

Attribute controls: Allen has a proposal.

Half-override of get/set pair: Allen has a proposal.

Categorization:

No review, no advance:

Unicode

Concurrency

Enumeration

Simple module functions

No review, advance:

Multiple Globals

Maps and Sets

Math enhancements

Function.toString

Name property of functions

Review, advance:

String Extras

Array Comprehensions

Completion Reform

Classes

Extended Object Literals (the goals were advanced, the syntax used by the strawman was not)

Quasis

Review, no advance:

Soft Fields

Object Change Notifications

Deferred Functions

Guard Syntax

Paren-Free

Arrow Function/Block

Completion reform:

This is a breaking change. There's no way to specify a language version for eval code, short of a pragma.

Extended Object Literals: Waldemar and others objected to most aspects of the current syntax; it produces a punctuation soup and possibly conflicts with guards.

Object Change Notifications:

Luke: We tried and failed to make proxies work for this use case (as well as for copy-on-write).

Waldemar: Why can't this be done using proxies? I understand why proxies can't support the API proposed for object change notifications, but why can't they solve the larger user need here?

Luke: Too inefficient and would require full membranes to support object identity.

Allen: This allows people to break into abstractions by putting observers on objects they don't own.

DaveH: Proxies are deliberately less powerful than this in that they don't allow you to attach behaviors to arbitrary objects that you don't own.

MarkM: Notification happening synchronously is a security problem. Observers can run at times when code doesn't expect malicious changes.

Waldemar: The same argument applies to getters, setters, and proxies. You need to lock things down to guard against those anyway, and the same lockdown would presumably apply to observers.

Cormac: Just hand around *only* the proxied versions of observable objects. No need for membranes.

Sam: This would let you observe private names.

Luke: No it wouldn't. It would only show alterations to properties whose names you can see via getOwnPropertyNames.

MarkM: This could be done by turning the current true/false extensibility property into a tri-state true/false/observe property. Existing properties can be replaced with getter/setter pairs. The extensibility hook would be notified when new properties are created.

Waldemar: This won't notify you of property deletes. A delete will delete a configurable getter/setter pair without notifying any observers.

MarkM: Is observing deletion needed?

Luke: Yes, particularly for arrays.

Discussion about deferred vs. immediate notifications. Immediate is preferred.

Brendan: This might be a better "watch".

Cormac: There is a lot of overlap between this and proxies. Is there a way to do some synthesis?

MarkM: This has a lot of open research issues.

Cormac: Proxies may have flaws that this would fix.

MarkM: We should work on this in parallel, so that this informs proxies.

Not advanced to Harmony.

Quasis:

`$\u0061` is a reference to the variable <a>, not the string constant "a".

Waldemar: Why do we need variables with unguessable names?

Resolved: Don't name these variables at all. They're still created at the top-level scope but don't have lexical scope names.

Waldemar: What happens when you have an invalid escape sequence so you can't generate a decoded string to pass to the function (even though the function is only interested in the raw string)?

Resolved: Modified the proposal to remove the interleaved arguments and instead put both the raw and the decoded strings on the frozen identity object. Decoded strings are missing if they don't decode.

Advanced to Harmony.

Deferred functions:

Discussion (and lots of confusion) about the semantics of the proposal. The Q class is not part of the proposal. Continuations are meant to be one-shot, possibly with reusing the same continuation object across calls.

MarkM: There is also an error-handler in addition to the then-handler.

MarkM: Issue with chaining values.

Peter: Every time a function that contains an await statement is called, it returns an new Deferred object. The semantics of Deferred are built into the language.

```
class Deferred {
  then(callback) {this.callbacks.push(callback)}
  constructor() {
    this.callbacks = [];
  }
  callback(result) {
    for (var v: this.callbacks) {
      v(result);
    }
    this.callbacks = [];
  }
}
```

Cormac: "then" never returns anything interesting.

Waldemar: How do you await multiple things concurrently? A sequence of await statements will wait to launch the second until the first one is done.

Peter: Call these multiple things without using await and then use a combinator to wait for all of the results.

MarkM: This requires separate turns.

Brendan: This is a syntax full of library choices that could be done in other ways that we should not be specifying yet. (Examples: Form of Deferred objects, turn issue.)

Deferred implemented using Generators? Need a top-level dispatch function. Also generators don't let you return a value.

DaveH and Brendan: Deferred is coexpressive with generators. However, there are policies underneath Deferred (scheduling etc.) that are premature to standardize. Solve this problem using generators as they are.

Luke: Concerned that things we're shooting down have more value than things we've adopted.

DaveH: Need to keep the pipeline going.

Waldemar: How do you simulate return values using generators?

DaveH: Use the last yield to pass the return value. The caller will then close the generator.

Not advanced to Harmony.

Guard Syntax:

Technical aspects ok, but some are worried about the effects of this change on the ECMAScript style. Cormac's student Tim will work with DaveH to prototype this this summer.

Not advanced to Harmony, pending more user experience.

Paren-free:

Compatibility issue with:

```
  if (cond) a = b;
```

catch, for, for-in heads must not be parenthesized.

MarkM: Cost of incompatibility is too high, particularly for things like for(;;) heads. Would prefer to have full upwards compatibility, including old semantics for for(a in b).

Several others: Don't want significant parentheses.

MarkM: We must not make it impossible to write code that runs on both old and new syntax.

Not advanced to Harmony.

Arrow Function/Block:

```
function f() {
  a.forEach({| | return 3});
}
```

The return will return out of f. Note also that the implementation of forEach could have a try-finally statement that catches and revokes the return. This kind of cross-function return catching is new.

The cage around block lambda is too tight.

Luke: Concern about call syntax strangeness. This kind of syntax only works if it's designed holistically.

Debate about completion value leaks.

Waldemar: Use of | conflicts with common use of trademarks.

Alex: Objects to new "little pieces of cleverness". Too many things to teach to people.

Not advanced to Harmony.

Next meeting two days Jul 27-28.

---

es-discuss mailing list

es-discuss@mozilla.org

https://mail.mozilla.org/listinfo/es-discuss

……… old text

Wednesday notes:

Ask the GA for a way for non-members to sign software contribution agreements?
Waldemar: Thinks this would be a hard sell in the GA. They'll be annoyed at increasing provisions for non-members to participate.

Istvan: This should not be "too innovative", at least until we get a lot more experience. Ecma should not directly accept third-party contributions. An Ecma member should serve as an intermediary.

Allen: This is already coming up in the context of Test262. Third parties are coming up with tests and we would like those tests.

John: Are the third parties willing to assign the code to one of the
Ecma members?

Binary data:
Typed arrays != ArrayTypes. DaveH is proposing ArrayTypes but not ArrayTypes.
In practice there is little difference between them. ArrayType(int16)
is drop-in replacement for Int16Array.
DaveH is proposing ArrayBuffers (not currently on wiki).
Every instance of an ArrayType or StructType (instance that contains
actual data, not the type object itself) is also an ArrayBufferView.

Discussion of whether the raw buffer should be exposed for ArrayTypes created for in-program (non-i/o) use only. With this view, any client can extract the buffer and must get a specified view (big endian by default).
Allen: Ability to extract the buffer from "new new ArrayType(int32, 100)" forces implementations to store it as big-endian even if they would rather not. This impedes performance.

Slide with:
MyType MyType(ArrayBuffer buff, uint32 idx = 0, uint32 length = buff.byteLength - idx, boolean networkByteOrder = true);
Waldemar's misinterpretation of the bool: networkByteOrder = true means big endian. networkByteOrder = false means localByteOrder, which can be big or little endian, depending on local hardware. To avoid such interpretations, if this bool is meant to distinguish between big and little endian, call it bigEndian.

Brendan: What if we don't allow aliasing of multiple-sized interpretations for the same data? i.e., once something is an int32, can't access it as int16 or bytes?
Waldemar: This would break the common case of creating structures that contain crc's or digital signatures of parts of their content.
MarkM: Any reason to allow a fuzzy (rather than explicitly specified) byte order for such cases?
Waldemar: No.

Waldemar: What's the practical user-visible difference between ArrayBuffers and Blobs?

Alignment: Atomic types within a struct must be naturally aligned. Struct lengths must be naturally aligned to the largest data member.

What about unaligned use cases? These are fairly common in file formats. Most processors have some way to access unaligned types which is substantially faster than extracting single bytes in ECMAScript and shifting, so we'd want to allow it in some form.

DaveH: Solution: Packed types vs. unpacked types.

Luke: Fall back to data views if structs don't solve a problem well.

Waldemar: Most files contain lots of string data in various formats (UTF-8, UTF-16, ASCII, etc.). The complete lack of string support is the major obstacle to using this framework to parse files. User programs will have to parse UTF-8 and construct strings byte-by-byte (which might be O(n^2) on some implementations if accumulating a string by concatenation.)

Consensus on moving what's currently on the wiki page into harmony, with ArrayBuffers and strings as important goals for additions.

Doug: Concerned that we started with just moving data to GPUs but are now expanding scope into the much larger issues of file i/o.
Allen: Why not make a separate task group to standardize just this?

Module loaders:
Waldemar: evalLoad: Is it always asynchronous? Yes, but should discuss this on a different thread.
Example:
l.load("my-module-url", myCallback); // The module at my-module-url references global g
l.defineGlobal("g", 42);
Note that mere compilation of my-module must not begin until after the main thread completes; otherwise it wouldn't see the global "g".

l.defineGlobal("f", function() {...});
has the same behavior as:
var f = «function»;
except that «function» is created in the global context of the caller of defineGlobal, not l's global context. «function» is then bound as l's global variable "f".

Waldemar, Brendan: resolver.resolve/load/eval don't work well with redirects. You want to use the target of the redirect as the cache key, but by then you're in load and it's too late to do the resolve cache lookup. Origin policies also use the target of the redirect.

Custom loaders: when base.Array is not the usual Array object, what happens when evaluating an expression like []?
DaveH: Core of the object is the standard Array, but the mutant Array constructor gets to run too.
Waldemar: What exactly does that mean?
Allen: Emphasis on constructors is misplaced. Other contexts are also important.
Waldemar: Agree. Suppose you swap Array with RegExp. RegExp methods refer to hidden magic properties that are not on standard objects. They work and stay hidden because construction is coordinated behind the scenes with the other methods.
MarkM: Make base optional in loader.create.

Move Modules to proposal status?
Doug: Modules are one of the most important features, but too early to move it to proposal.
Debate over what "proposal" means.
Doug withdraws objection.

Quasis Q&A:
Q. Do we have plans to standardize functions like safehtml?
A. No, we will not standardize the functions.
Q. How does safehtml decide when quoting a message like "**bold** word" into HTML whether to keep safe tags such as or whether to escape them into ? There are plenty of use cases for both situations and neither dominates.
A. Plain strings are assumed to be raw text. To avoid escaping things like HTML you need to pass in a parameter of a special type other than string.
Debate about whether quasis are a lexical or syntactic grammar production. They're lexical and deliberately don't include an expression subgrammar for SubstitutionBody. The text is lexed via SubstitutionBody and later parsed.
Concern about grammar complexity of substitutions, particularly with nested `'s.

Brendan: Why currying and thunking and assignable substitutions and all this complexity?
None of the use cases (other than Decomposition Patterns, which also use setters) require thunks.
Brendan: By default simply interpolate the components if no function is specified.
Lively debate on the default. Doug: This will bring back security holes. DaveH: Simple concatenation useful for logging.
DaveH: It would be good to have an HTML sanitizer in ECMAScript, just like how we have encodeURI, regardless of who standardizes it.
Waldemar: Often want to refer to properties of a late-bound object, not the local variables. See internationalization example below.

Concern about lack of ability to indirect, as needed by, for example, internationalization. How do you obtain messages from a library and pass them indirectly into code? Each message captures variables at the point where it syntactically lies in the program, but here we want to late-bind the parameters.

To do this, you'd need to write tables of:
English:

```
function hellomsg(o) {
  return msg`Hello, ${o.world}`;
}
function basemsg(o) {
  return msg'All your ${o.base} are belong to us";
}
```

French:

```
function hellomsg(o) {
  return msg`Bonjour, ${o.world}`;
}
```

etc.

Status? Brendan: "Sounds like this needs another run through the body shop."

Infix operators:
?? Replace undefined with a different value
Do we want to also change the default parameter syntax to use ??= instead of = ?
Metadiscussion about complexity budget.

has
mod
div
min
max

Syntax is compatible with ES5 (as long as there is a [no linebreak here] before the keyword) but precludes some considered extensions such as attributes decorating definitions.
Debate about reverse direction of "has" vs. "in".
Brendan: Not clear if "has" is worth it.
"mod" produces the same sign as the divisor (as opposed to %, which produces the same sign as the dividend).
How is "div" different from a division followed by truncation towards zero? They're almost the same, but the intermediate division can

round up to the next integer if it's within 1/2 ulp of an integer,
while div wouldn't.
Brendan: Sympathetic to "mod" and "div" because the versions that
people should use are obscure or wordy.
Why not ** for power?
"min" and "max" feel weird as infix operators.

Number.prototype.compare(v, t): |n - v| < t?
Waldemar: This is the wrong comparison in many cases. Often you want
to do a relative one. There is no single such comparison that would
work in a majority of cases.

Number.EPSILON: nextdouble(1.0) - 1.0
Number.MAX_INTEGER: 2^53 exactly. It's the largest double d such that
d - (d - 1) = 1.
Complaints about the name MAX_INTEGER. It's not the maximum
representable integer; in fact, every finite double greater than
MAX_INTEGER is also an integer.

Various math functions on spreadsheet:
Some are commonly used (hyperbolics, exp2, erf, erfc, gamma).
Some are useful to avoid cancellation (e^x - 1 near x = 0).
Some are useful fp format manipulations (nextafter, split into
sign/mantissa/exponent and back).
Most of the rest are esoteric.

Object initializers:
Is the thing created by [<proto: myProto>, "a", 2, false] an Array (by
isArray) or not? Yes.
Syntax collision with E4X.
Brendan: Don't like comma after >. Too easy to miscount number of elements.
Waldemar: Like comma after >. Without comma, the above example would
look like a greater-than expression myProto > "a".
Method declarations within object literals:
Are those allowed in arrays? No. You can't make an array literal
with its own toString method.

Object literal property modifiers:
p: value
p const: value
var p: value
var p const: value
sealed p: value
sealed p const: value
sealed var p: value
sealed var p const: value
method p() {}
sealed method p() {}
set p(v) {}
var get p() {}
sealed put p(v) {}
sealed var get p() {}

Waldemar: Why is "var" a prefix but "const" a suffix? Too confusing.
The property name p should be in a consistent place (preferably last
because that's existing usage for getters).

Brendan: "var" and "const" in the same property declaration? "var" does not intuitively mean "dontEnum".
Brendan: configurable=true, writable=false combination not necessary?
Waldemar: Since any keyword is usable as a property name here, what does this do?
var const:value
Is it defining property "var" with attribute const, or property "const" with attribute var?
What if someone makes a typo and forgets the property name? Many of the above forms will declare a property named "var", "const", etc.

Private names in initializers: Controversy about 'private' capture semantics, orthogonal to object initializer syntax.

```
class c {
  method m() {return this.a+this.x} [HERE]
  new (a) {
    a: a,
    {this.x = somebody(this)}
  }
};
```
Controversy about what punctuation goes into the place marked [HERE]. If classes are based on object initializer syntax, there has to be a comma there. Brendan: users won't expect that.

```
class D {
  <superclass: C, frozen>,
  new (a) {
    <closed>,
    a: a,
    {this.x = somebody(this)}
  }
};
```
Note that the closed property is applied after the initializer block is run.

Debate about class private vs. instance private vs. ergonomics of protection by scoping.
Allen: Private must satisfy:
- Can't reflect on it
- Proxies can't trap them
- Don't need to stick lots of closures into each instance of a class just to close over private members.
Dave: Some of these can be relaxed.

Discussion about how the private mechanisms here subsume some of the internal browser ones used to implement the DOM.

Initializers can dynamically created @-variables simply by assigning to them.

Waldemar: This is a problem:

```
class D {
  private iv1;
  ...
  new(a) {
    method m(v) {@iv1 = v;}
  }
}
my_d = new D;

class Evil {
  new(a) {{my_d.m.call(this, 33);}}
}
```

Evil gets to create @iv1 properties on its own objects without D's knowledge.

Allen: To fix this problem, disallow dynamic creation of @-properties
from constructor initializers.

Discussion about preventing methods from working on fake instances.

Dave:

```
var kye = Name.create();
function Thing(amIBlue) {
  this[key] = amIBlue;
}
Thing.prototype = {
  meth: function() {
    return this[key] ? "blue thing" : "red thing"
  }
}

var evil = Proxy.create({... get: #(key) {...} ...});
thing.meth.call(evil);
```

Dave: Every time a private name gets passed to a proxy, the name gets
wrapped. (to what?)

```
private dance;
Thing.prototype.dance = fun() {...}

{
  private á;

  function Complex(r, i, á) {
    this.real = r;
    this.imag = i;
    this.á = á;
  }

  function setá(á) {
    this.á = á;
  }

  Complex.prototype.equals = fun(other) {
    return this.real == other.real && this.imag == other.imag &&
thia.á == other.á;
  }
}
```

Now you can again create fake Complex objects by creating an object
with real and imag properties and introduce an á property with setá.
This can fool the Complex equals method.

Debate about branding.

Brendan: Should instance variables be non-properties (i.e. not inherited from protype even if they don't use private names)? Long debate of cognitive load costs and other attributes pro and con.

Date example: Users inherit from Date, but that breaks because Date objects contain a hidden time field, which inherited ones don't. If that field were a private name, this would work.
But how would it work? A possible scenario would be that all BrendanDate objects (derived from Date) share the same hidden time field, in which case changing one would affect the others.
To fix this, BrendanDate objects should call own-property methods, but then there's no inheritance. How would initialization of the hidden field work?

Allen/DaveH: Private names proposal on wiki will be modified to support strong encapsulation. Split proposal into runtime semantics and syntax/scoping halves.

Open-ended discussion of many aspects of this proposal. Not sure how to summarize it. It's also unfortunate that Mark isn't able to present his proposal due to an emergency.

Thursday notes:

DaveH's presentation on using generators to write asynchronous code.

How do you compose generators? yield*

Waldemar: Given yield*, writing base-case trivial generators that don't yield becomes useful but its syntax is a problem. Generators should not be distinguished from functions by the presence of a yield statement.
Cormack: Why can't yield* work on both generators and functions, doing a type test?
Dave, Waldemar: Flattening using type testing like this leads to trouble. What if the return type of the generator/function is a generator?
Deferred issue about Next API throwing vs. multiple methods.

Moved generators to harmony

Proxy handler parameter:
Why not use "this"? Crossing levels; interferes with inheritance of handler object.

Brendan: delete doesn't need a receiver parameter because it only works on own properties. It will still get a proxy parameter.
Bug on Tom's slide: set needs a receiver parameter. get and set need a proxy parameter.

Completion values source statements are currently statically uncomputable:
eval("1; if (...) 2;") can produce a completion value of either 1 or 2
eval("1; while (...) 2;") ditto
Dave: Propose making the above examples return either 2 or undefined.
That fixes behavior for #-functions:
#(...) {1; if (...) 2;}

#(...) {1; while (...) 2;}

eval("3;;;") would still produce a completion value of 3, not undefined.

Refutable matching and switch extensions:
Multiple objections to syntax chosen for pattern-matching switch:
colon vs. no colon after default clause, need for blocks, etc.
Refutable matching doesn't retrofit into imperative switch syntax
well.
Waldemar: Refutable matching is half-baked at this point, with too
many syntactic and semantic problems. Not clear it's worth its added
complexity.

The refutable matching wiki has the following consequences on
irrefutable matching:
Pattern [x,y]:
Matched to [3,4], produces x=3, y=4.
Matched to [3,4,5], produces x=3, y=4.
Matched to [3], produces x=undefined, y=undefined. (wiki spec bug.)
Pattern [..., x, y]:
Matched to [3,4], produces x=3, y=4.
Matched to [3], looks up negative array indices. (wiki spec bug.)

Pattern [x,y] behaves like [x,y,...] for refutable matching. (wiki spec bug.)
Can't match on zero-length arrays. (wiki spec bug?)

Lucas: Feature value should overcome complexity costs.
Waldemar: if guards introduce unknown syntactic and semantic complexity
Waldemar: where can you use refutable matching outside of switch/match
statements and perhaps catch guards? switch/match statements are too
heavyweight and differ too much from irrefutable matching assignment;
catching doesn't really need destructuring but benefits from
conditions. The typical usage (as in Perl) is to use them in if
statements: if (pattern =~ expr) {we have matched!}

catch({z,w} if z < w): OK, but then you can't get to the entire
exception object from the catch clause.
catch(z if z instanceof T): Useful

Waldemar: Refutable matching should integrate trademarking to be compelling.
Concern about backing ourselves into a corner by implementing
irrefutable pattern matching in catch guards that will later preclude
refutable matching. Brendan's example: catch({x, y}) would succeed
on {x:3} now but fail later if we change to refutable pattern
matching.

#-functions:
Alex, Erik: No longer support #-functions with a different semantics
from functions.
Brendan's proposal:
#(a,b) {a + b} // Inherits "this" only lexically
#(this | a,b) {...} // behaves like regular functions with respect to "this"
Alex: Make function objects soft-bind "this"
Dave, Waldemar, Brendan: Object to soft-binding "this".
Allen: We should respect the intent for "this" binding, as expressed
by whoever is defining the function (as opposed to whoever is calling
it).

Alex: Debating soft-bind use case of users wanting to override "this"
in a function bound with Function.bind.
Waldemar: We're discussing raw closures here, not functions produced
by Function.bind.
Lively debate.

Discussion on what #(x | ...) would mean. Brendan: it would bind "x"
instead of "this" with the caller-supplied this-value. "this" would
then be unconditionally inherited from the enclosing scope.
#(self = this | a, b): Default value for "self" is "this".
Waldemar: Don't like defaulting here. How would you call such a
function indirectly via Function.apply without supplying a value for
"self"?
Cormack: Use | syntax for regular functions?
Luke: User confusion with two slightly different ways of doing the
same thing (function vs. #). Users will then want to use #-functions
for constructors, etc.
Alex: How do you express functions in terms of #-functions?
If we do #-functions, does that eliminate the "const foo(x,y) {...}"
shorthand for const function definition syntax? Brendan: Yes.
Alex: Willing to consider #-functions with a different semantics from functions.
Waldemar: Don't want implicit "return" of completion value in
#-functions due to leakage and inversion of behavior vs. functions.

Records:
To sort field names or not to sort:
Brendan, DaveH: Nor sorting may be cheaper
Waldemar: Don't like hidden state. A canonical order such as sorting
(as currently on wiki) is much simpler conceptually. When specifying
behavior of various methods that create records, no need to be careful
about property order.
Philosophical discussion: Can one make a language simpler by adding
features? Yes, over the long term, if new simple features obviate old
complex features and people forget the old way of doing things.
DaveH: Fear of Structured Clone between web-workers.
How does toString work on records?
Duplicate property names in #{p: v, p: v}? Error.
Duplicate property names between explicitly specified properties and
properties introduced by ... in #{p:v, ... rest}? ... wins?
=== is elementwise.

Tuples:
Brendan: Does anyone here really hate records and tuples?
Lucas: Likes dicts (mutable, but otherwise identical to records) in
preference to records.
Allen: Not much benefit to dicts over some way to provide null for the
prototype.
Waldemar: No, a null prototype is not sufficient. Even if you don't
have a prototype, in lots of contexts the language will still look for
a toString property in your object and call it if it finds it. Thus
you're not safe using a null-prototype object to store arbitrarily
named properties. Dicts or tuples avoid that problem because they're
a distinct primitive type.
Debate about weak vs. strong keys.
Dave: If you have a size-querying method, you can't have weak keys
without becoming nondeterministic.
____