Minutes for the: 18th meeting of Ecma TC39

held in: Mountain View, CA

on: 30 September – 1 October 2010

1 Opening, welcome and roll call

1.1 Opening of the meeting (Mr. Neumann)

The meeting (hosted by Mozilla at their premises in Mountain View, CA) was opened by **Mr. Neumann**, Chair of TC39 at approximately 10:00 AM on 30th September 2010 (2010/051: Venue for the 18th meeting of TC39, Mountain View, September 2010).

1.2 Those in Attendance included:

John Neumann	Ecma internatiponal
Allen Wirfs-Brock	Microsoft
Cormac Flanagan	UCSC
Sam Ruby	IBM
Jeff Dyer	Adobe
Nebnojsa Ciric	Google
Sam Tobin-Hochstadt	Northeastern University
Dave Herman	Mozilla
Tom Van Cutsem	University of Brussels
Mark Miller	Google
Douglas Crockford	Yahoo!
Waldemar Horwat	Google
Alex Russell	Google
Erik Arvidsson	Google
Oliver Hunt	Apple
Brendan Eich	Mozilla
Jungshik Shin	Google
Peter Constable	Microsoft (Phone)
Isabelle Valet-Harper	CC Chairman (Friday Afternoon)

1.3 Host facilities, local logistics

David Herman outlined the services for Lunch and Dinner on Thursday evening.

Mr. Horwat agreed to take technical notes for the meeting.

2 Adoption of the agenda (2010/052revised)

Adopted as modified (items 4.4 and 4.5 added)

3 Approval of Minutes from July (2010/050)

Approved as modified (spelling of Arjun Guha Brown University)

4 Report from the Secretariat

4.1 Review Ballot results of ES5 (2010/049)

It was noted by several participants that the Japanese ISO comments were excellent and very detailed.!

There was detailed discussion of the Japanese comment on 7.9.1:

```
BreakStatement:
  break [no LineTerminator here] Identifier-opt ;
```

means:

```
BreakStatement:
  break [no LineTerminator here] ;
  break [no LineTerminator here] Identifier ;
```

which then means that code like:

```
break
;
```

was rejected by the editor, but the committee felt that the problem should be corrected if there was time. The chairman verified status of the DoC document and that there is time to change the editor's response from reject to accept, and to implement change in the draft..
This is a bug in the ES5 spec. In the prehistory of ES3
We had intended the following:

```
BreakStatement:
  break ;
  break [no LineTerminator here] Identifier ;
```

but we over-abbreviated. Mr. Horwat proposed fixing the spec to the two-line form above for break and continue, and an analogous fix for return. Consensus to fix it.

Firefox conforms to the corrected grammar. Safari does something unusual: It allows a line terminator after the break but considers a semicolon after the line terminator to be an empty statement. This shows up if one writes:

```
if (...) break
; else ...
```

which is an error in Safari but not Firefox. We hope Safari will fix this tiny corner case.

If a ballot resolution meeting is needed, Mr. Horwat will represent TC39. Mr. Wirfs-Brock can't do it because he can't have dual roles in this process, also being the editor of the ISO standard.

4.2 TC39 possible relationship with Khronos (WebGL WG and Typed Arrays)

TC39 wants to work with Kronos particularly on Typed Arrays. What is the legal situation with Khronos and IPR relative to working with Ecma. Mr. Neumann thinks this need to be confirmed so we can move forward at our next meeting.

4.3 Technical Report on interoperability/conformance tests

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

We examined the question of testing and the proposed website from Mr. Wirfs-Brock and it was agreed that we want to proceed as quickly as possible to putting the website up on Ecma.org TC39 or on the external ecmascript.org site. Mr. Neumann will ask the CC to agree to hosting this on Ecmascript.org Website as soon as possible in my report to the CC. This was further discussed when Isabelle Valet-Harper joined the group for discussion. She will support TC39 request at the CC meeting and it was agreed that Mr. Neumann would call in and give the report of TC39.

Test 262:
Microsoft executed the Ecma contributor agreement.
Google hasn't yet but intends to.
Mr. Wirfs-Brock would like to publicize the site at some point as an official place for ES tests.
Several of us want to make sure that, if it's publicized, this is perceived as a work-in-progress, not as something that "certifies", "validates", ensures "conformance", etc. of implementations.

Mr. Miller: ES5 tests are currently too wimpy. The pass rate is too high for implementations we know not to be ES5-conformant.

Fear is that journalists will look at a "98.72% conformance" grand total score and use it as a figure of merit. There will be pressure for vendors not to put up new conformance tests that they themselves fail.

Mr. Miller and Mr. Horwat: Please delete the grand total score line.

John Neumann: Concern about naming vendors in test results (as opposed to anonymous "Vendor 1", "Vendor 2", ...) on an Ecma-related site.

4.4 Internationalization standard proposal (Google)

Google and Microsoft are interested in participating. Maybe Mozilla.

Should the internationalization library standard be in lockstep with ES-Harmony, targeted for 2013? No need to wait for Harmony.
Mr. Wirfs-Brock thinks that 2012 is realistic for a separate internationalization library standard.
Mr. Horwat: We can decide later whether it's best to fold the internationalization library into Harmony or leave it separate. If it's separate, we'll need to face issues with Harmony security, multiple global objects, etc. that may need to be backported to the internationalization library.

First internationalization meeting in November. We'll share the es-discuss mailing list.

Next meeting: Nov 16 (internationalization), 17, 18

Mr. Miller and Sam Ruby: Interested in standardizing a debugging API. There is already some prior work, such as IBM's JSDT.

A number of others are interested in following the effort on es-discuss.

4.5 W3C Joint work items

Proposed agenda for joint meeting will be sent to W3C for their reaction and comment:

1. Relationship between ECMA-262, WebIDL, host object semantics, and the JavaScript binding for WebIDL. - how would we present our preferred organisation of the WebIDL document?

2. Features of WebIDL that are supported for historic compatibility but strongly discouraged from future APIs.

3. Roadmaps for TC-39 and W3C Web APIs - get everyone one the same page

4. Identifying shared goals and understanding how the groups coordinate their work towards those goals.

Another possible topic is style/design guidance for future web API to make them more natural for JavaScript programmers.

5 Progression of ES 5

5.1 Editor's Proposed Disposition of Comments Report (2010/055)

5.2 Ecma representative to Ballot Resolution Meeting

This was discussed completely under Item 4.1 of the agenda.

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

6.1 const functions extended with joining

<http://wiki.ecmascript.org/doku.php?id=strawman:const_functions>.

done

6.2 Inherited explicit soft fields

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

Not discussed and carry over to next meeting.

6.3 classes vs traits"

<http://wiki.ecmascript.org/doku.php?id=strawman:syntax_for_efficient_traits> <http://wiki.ecmascript.org/doku.php?id=strawman:traits_semantics>

Classes as sugar:

Should public fields be enumerable? Yes, so the object can be
serialized via JSON. Also, when traditionally creating objects as
records, the fields will be enumerable.

Mr. Horwat: Traits and classes-as-sugar proposals seem remarkably similar.
Mr. Horwat: How does one do class-private in the current traits
proposal? Mr. Miller: You can't.
Mr. Horwat: Concerned about easily being able to define high-integrity
abstractions using whatever we use for classes or traits. Don't want
to be able to sneak in something that looks like a ComplexNumber to
some consumer but then spontaneously changes value after being
validated.

We'll need to work out the corner cases of "this" usage in both proposals. We'll also need to address what happens when we throw out of a class block but squirreled away the value of "this" in a global variable. Probably no big deal.

Having the syntactic form "public foo = ..." to create a publicly readable, privately writable field is confusing.

Open discussion about interaction between traits and prototype chain. Traits follow prototype chain when used with Object.create; they are prebound when used with Trait.create. Debate about whether the common case is promiscuous object, nonpromiscuous objects, or both.

Dave: Somewhat prefer the Object.create approach.

Mr. Miller: Make it convenient to create defensible objects.

Making traits high-integrity is inconvenient:

```
trait class PointTrait(x, y) {
  function getX() {return x;}
  => {
    getX: getX,
    ...
    ... getX ...  // High-integrity usage
    ... this.getX ...  // User overridable usage
  }
}
```

There's no way to do the high-integrity usage on another (non-this) instance of a trait.

Schism within group about goals: "high integrity" vs. "supporting the things that people are already writing with better syntax" vs. maybe possible to get both

Mr. Miller: In the world of mashups, new compositions come up on users' computers that developers did not test ahead of time. Need to ensure that components are robust/reliable enough individually so that the compositions will just work without the need for testing.

PostMessage security alternative: Very hard to program due to asynchrony at all inter-module boundaries.

Object initializer syntax:
Angle brackets useful for disambiguating things like:

```
var arrayLike = [
  <prototype: obj>,
  "element0",
  "element1"
];
```

Per wiki, this is has [[class]] Object, not Array. Debate about whether it should be Array or whether we should allow <class: expr>.

Grammar is ambiguous because it allows the > operator in the expression inside <prototype: expr>.

Wiki proposal doesn't allow null prototypes. Agreed to allow them.

Debate about whether the closing > should be followed by a comma.

What is the "final" metaproperty? It's "nonextensible" with a confusing name. Groans -- fools people into thinking that the object can't be used as a prototype.

Objections to making "const" fields automatically nonenumerable. This tying of two independent concepts makes it hard to serialize via JSON, etc.

Private properties are controversial. Do they pertain to just the instance being constructed, or can they be applied to foreign instances?

Mr. Horwat, Mr. Miller: If sole-instance, they're better expressed via closures (with an argument about implementation speed). If they're not sole-instance then within the context of "private privateVar", the expression "a.privateVar = 3" will do something different from a["privateVar"]. Instead, it will create a hidden property on a and give it the value3.

Mr. Wirfs-Brock's "static" proposal: Algol 60's "own" variables.
At what point does a static initializer get evaluated? Function expression: at the time that the function itself is created. Function declaration: we'll have to think about it.

Mr. Horwat: Not clear there is a right answer to when a static initializer gets evaluated for a function declaration. The obvious choices are either at the beginning of the next-outer function or just before the inner function declaration would have been executed. Each choice is wrong for one of the statics s1 or s2 below:

```
function Outer(x) {
  // Initialize s1 and s2 here?
  Inner2();
  let y = ... x ...;

  // Initialize s1 here?
  function Inner1(z) {
    static s1 = y;
    ... s1 ...
  }

  // Initialize s2 here?
  function Inner2(z) {
    static s2 = x;
    ... s2 ...
  }
}
```

Of course, we should also support "static static static" to go up three function levels.

6.4 Object Initializer Extensions”

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

egal <http://wiki.ecmascript.org/doku.php?id=strawman:egal>

6.5 let: classifying some syntactic errors

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

done

6.6 binary data

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

Binary data

Given

```
const Point2D = new StructType({ x: uint32, y: uint32 });
const Color = new StructType({ r: uint8, g: uint8, b: uint8 });
const Pixel = new StructType({ point: Point2D, color: Color });
p = new Pixel({point: {x: 3, y:8}, color:{r: 100, g: 50, b: 0}});
```

p.point.y returns the primitive integer 8
p.point returns an alias (lvalue) to the Point2D substructure (block) of p. Modifying data visible via such aliasing is visible to all aliases.

Debate over whether values such as 3.8 or -1 should be assignable to a uint32 field. Current proposal says no and provices a uint32(x) coercer method: uint32(-1) = 0xFFFFFFFF.

Does calling p.point twice return block references that are === to each other? Dave says he's agnostic. Produced a bit of debate.

Special updateRef feature to avoid creating lots of block wrapper objects (the blocks themselves are aliased so are less of an issue):

```
var T = new StructType(...);
var A = new ArrayType(T, 1000000);
t = T.ref();  // allocate a homeless struct object
for (i = 0; i < 1000000; i++) {
  t.updateRef(a, i);  // no allocation of struct wrapper
}
```

Given this feature, it's no longer possible to be agnostic on ===. One can't make block references a === b if and only if a and b are aliases of the same block.

One way out would be to have T.ref() return a separate updatable block type, while p.point or a[3] would produce nonupdatable block type objects. Nonupdatable block type objects would have the above natural definition of === and not support updateRef.

The counterpoint to the above is the creation of two kinds of block types, similar to each other.

Debated desire for having blocks inherit from Array.prototype.
Debated desire for having some Array generic methods on blocks.

Endianness is invisible to users except when using blocks for file
i/o, in which case the routines doing the i/o will take a parameter
specifying the endianness of the file and convert as needed.

How does one represent variable-length strings?

Why no character types for fixed-size arrays? Punted from proposal
for time reasons for now, but are desirable.

When using this for file i/o, handling of variable-length strings will
become important. How to do this? No specific design, but some ideas
are:

Given struct {x: uint32, y: uint32, z: uint32, name: String}, an idea would be:

```
string = new Layout({
  length: uint16,
  contents: function(me) {return new ArrayType(uint8, me.length)}
});
color = new Layout({
  x: uint32,
  y: uint32,
  z: uint32,
  name: string
})
```

How to store these in memory? One can't have a pointer to a string
inside a block.

int64's: Open issue. Reference semantics are annoying, but what's a
realistic alternative?
int128's? Those come up increasingly often in SSE programming.

We briefly discussed bignums as a realistic alternative, where equality would be based on numeric value rather than object identity. But for now we'll try to keep the binary data spec as orthogonal as possible from a bignums proposal.

Mr. Horwat: Reify uint64's as 4-char big endian strings and uint128's
as 8-char big endian strings. A significant advantage is that these
would have value semantics, and ==, ===, <, <=, etc. would all work
correctly. Would need to call methods to do arithmetic or signed
inequality comparisons on them.

6.7 private names

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

Unique names
Question: In what ways does this differ from weak tables?
- A client can look up a property p using a[p] without knowing whether
p is a traditional string or a unique name.

Debate about private names leaking via proxies. It's trivial to get
at private names by passing a proxy to code that accesses them.

It's not clear what private names are trying to do well. If they want
to provide privacy, they can't be visible to proxies. If they want to
provide extension without collision (i.e. namespacing), they should be

visible to Object.keys, enumeration, etc.

Mr. Wirfs-Brock: The syntax itself is the primary usage of this proposal.. Debate over whether there should be two independent concepts of scope or one with a flag.

Mr. Eich: list of questions:
1. Private for sure?
  No: unique name weak encapsulation
  Yes: Private name strong encapsulation

Perhaps we want both "unique n;" and "private n;".

2. Visible via for-in?
  Object.defineProperty
  Assignment via private

This bit included "either/or/both" with a dashed line, unlike the solid line between the "No: ..." and "Yes: ..." alternatives to 1, above

3. Visible via Object.keys/getOwnPropertyNames?

The answer here depends not on 2 but on 1: if private names are truly private, then they don't leak via Object.keys/getOwnPropertyNames; else they do, and no worries ("weak encapsulation", par for the JS course).

4. Proxy: visible as property name?
  Yes: Leaks names. Membranes can be done using proxies.
  No: Doesn't leak names. Membranes can be done using proxies using the technique below.

Philosophical debate about weak encapsulation:
Mr. Wirfs-Brock: Weak encapsulation is good enough for most use cases
Mr. Horwat: Encapsulation that almost works is worse than either strong encapsulation or no encapsulation. The reason is that folks will code until the code appears to work and then ship with vulnerabilities. This leads to most of the web security attacks.
Mr. Wirfs-Brock: Functions and objects are roughly dual mechanisms. Why do you need both? Functions provide strong encapsulation; objects provide weak encapsulation.
Mr. Horwat: Why does executability have to imply the need for strong encapsulation and vice versa?

Mr. Miller: Desugar
```
  private n;
  x.n = ...
  x.n()
into:
  const n = SoftField();
  n.set(x, ...);
  n.get(x).call(x);
```

Analogously, desugar
```
  unique n;
  x.n = ...  // Maybe make it nonenumerable by default?
  x.n()
into:
```

```
const n = Name();
x[n] = ...
x[n]()
```

It's useful to view soft-fields as a kind of property names, but then have to address what happens on preventExtensions, freeze, etc. If you think of these private fields as properties then they should be subject to freezing and such.

Mr. Eich: If you don't think of these soft fields as private fields *on the object* (so subject to preventExtensions etc.), then only inherited soft fields (not private name objects) can handle this novel use-case.

A proxy having the right to get at an object's private field names is equivalent to a proxy having the right to obtain all weak maps for which the object is the key. The security implications are the same. If a proxy can do a faithful membrane without one of these rights, it can do a faithful membrane without the other of these rights. If a proxy has no rights to get at an object's private field names, the membrane will still work as follows:

proxy[name] does not trap
object[proxy] calls a trap on the proxy

Mr. Eich: Good point -- need a new trap here? Or is this get with an object-type property name?

When a name object itself (not merely an object merely containing privately named fields) is passed through a membrane, the membrane wraps the name in a proxy p and then uses object[p] traps to maintain the membrane. Private names would be used completely on either the wet or dry side of the membrane, so they'd stay private. Public names that cross the membrane get proxied by the membrane.

Tom VC: Cormac's paper on virtual values (linked from the harmony:proxies wiki page, see <http://slang.soe.ucsc.edu/cormac/proxy.pdf>) already provides a solution for this case:

obj[proxy] traps the handler's "geti(obj)" trap

obj[proxy] = value traps the handler's "seti(obj, value)" trap

We could argue about the precise names of the traps, but Mr. Neumann likes this design.

6.8 catch guards

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

Catch guard proposal, resurrected from 1998. Well-liked all around. Proposed to move into Harmony if there are no objections by the next meeting.

Mr. Miller speculating: Could the if syntax be extended to pattern matching? Not easily. Pattern matching doesn't have a concept of failing; also, semicolon insertion would change the meaning of existing programs.

More debate about block ===. Reached consensus that two (nonupdatable) block references a and b should be === if and only if they are the same type and alias to the same data. Updatable block references will have separate rules.

Another alternative for mutual recursion:
let [f, g] = [function() {... g ...}, function() {... f ...}];

This assumes that f and g are visible in the initializer, which would bring the barrier issues back. Read barrier would be needed. Debate about whether write barrier would be needed, even if we have expressions for typed/guarded let.

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

6.9 Simple modules: external modules and scope

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

Done /needs closer look

6.10 Proxy.[[Construct]] reform

Discussion on the use of put vs. defineOwnProperty in the ES5 spec as they relate to proxies and library classes. Efficiency concerns with creating descriptors for defineOwnProperty. Most array classes turn out to use put for objects which can be proxied; the places where they use defineOwnProperties tend not to be proxyable because they're done on locally created Array objects.

Can a proxied [[construct]] return a primitive? Yes.

Mr. Eich: How do we do feature detection and top-level patching in ES-harmony?

```
if (!window.fooQuery)
  var fooQuery = ....
```

let clarifications:
let x and var x at same scope: error
let x at top scope of a function or catch block with parameter x: error
let x at script top leve: ok

```
module M {
  export let x;
}
```

var binding that would hoist across a let binding: error

let has a read barrier before the let statement gets executed:

```
{
  let x = 1;
  if (x) {
    alert(x); // Read barrier error here because inner x hasn't been
initialized yet
    let x = 2;
  }
}

{
  x = 42; // Write barrier error because x hasn't been initialized yet
  print(x);
  let x = 99;
}
```

Questions about whether

```
  let x;
```

should be treated as:

```
  let x = undefined;
```

so it has a barrier too. Could take either position on this one.

Mr. Eich: What if let and const don't hoist at all (as in C++ scoping)?
This was Mr. Horwat's position before the grand ES4 scoping compromise
a few years ago.

Mutually recursive example (wouldn't work with const without some extension):

```
let f, g;
f = function() {... g ...};
g = function() {... f ...};
```

6.11 Pragmas

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

Discussion of pragma syntax:

```
use bignums;
use harmony;
use modules {A, B, C};
```

7 Date and place of the next meeting(s)

November 16 – 18, 2010 Location Apple Computer-Bay Area (note that November 16 will be the kick off meeting of the new TC39 ad hoc group on Internationalization).

8 Closure

Meeting closed at 1630 after Mr. Neumann expressed appreciation to Mozilla for meeting facilities and lunches, and to Ecma (dinner).