Minutes for the: ***21st meeting of Ecma TC39***

held in: ***San Bruno, CA, USA***

on: ***21 – 24 March 2011***

1 Opening, welcome and roll call

1.1 Opening of the meeting (Mr. Neumann)

The TC39 meeting (hosted by Google in San Bruno at the YouTube Offices) was opened by **Mr. Neumann**, Chair of TC39 at approximately 10:30 AM on 21st March 2011 (2011/012): Venue for the 21st meeting of TC39, San Bruno, March 2011).

It was noted that before the TC39 meeting, on the 20th of March 2011 the TC39 ad-hoc group on internationalization has met (Ecma/TC39/2011/011 - Venue for the 3rd meeting of TC39 ad hoc group on Internationalization, San Bruno, March 2011). A report of that meeting will be given under 4.2.

1.2 Introduction of attendees

Erik Arvidsson - Google

Mike Samuel - Google

Alex Russel - Google

Nebojsa Ciric - Phone - Google

Andreas Gal - Mozilla

Istvan Sebestyen - Phone- Ecma-International

Tom Van Cutsem - Phone - Mozilla

Waldemar Horwat - Google

Allen Wirfs-Brock - Mozilla

John Neumann – Ecma International

Sam Tobin-Hochsted - Northeastern University

Dave Herman - Mozilla

Douglas Crockford - Yahoo!

Cormac Flanagan - UCSC

Brendan Eich - Mozilla

Mark Miller - Google

Luke Hoban - Microsoft

David Fugate - Microsoft

1.3 Host facilities, local logistics

Mr. Horwat and **Mr. Arvidsson** welcomed on behalf of Google the delegates and provided logistical information. It was announced that Ecma international would host a social event on March 24th evening.

2 Adoption of the agenda (2011/013)

Ecma/TC39/2011/013 contained the Agenda for the 21st meeting of TC39, San Bruno, March 2011. This was agreed with minor changes to group subjects.

3 Approval of Minutes from January (2011/008)

The minutes of the 20th TC39 meeting in January 2011 have been approved with no changes.

4 Report from the Secretariat

Regarding the question on the trademark status of ECMAScript at the January 2011 meeting **Mr. Sebestyen** has contacted the Ecma Trademark lawyer. It turned out that due to a misunderstanding there has been a delay in the application process, but now everything is on track again. The Swiss trademark application (that had to be filed first) will end in August 2011. All other applications (EU, USA) will be done later, but they will carry the priority date of the Swiss filing. It is still open if we will get the trademark assigned, as ECMAScript has been around and got well-known without trademark for about a decade.

Regarding TC39 documentation he said that in Ecma/TC39/2011/015 for Ecma TC39 archival purposes a snapshot from the ES wiki has been taken as before the meeting. Those documents are mentioned in the clauses 6.1 to 6.9 of the agenda of the 21st meeting. It was agreed that if those documents are changed during the meeting, the changes will be published as TC39 documents as well.

Then the question was asked about the publication status of ISO 16262 3rd Edition. The status of work there is on the technical level, and the Ecma Secretariat tried to find out exactly where it was, and what was the expected date of ISO publication. It turned out that there was some delay on the ISO side, the notification of completion of the fast-track was then issued by the SC 22 Secretariat the day after our urging. See also Ecma/TC39/2011/020 “Results of Final Review of SC 22 N 4595, ISO/IEC DIS 16262”.

Regarding the TC39 Software Copyright policy it was reported that for ES5 testing the question came up, if software from non Ecma members can be accepted. **Mr. Wirfs-Brock** noted that this is already coming up in the context of the Test262 project, where some third parties are coming up with tests and we would like those tests to be included.

There was some discussion if the CC and the GA should be approached for a way for non-members to sign software contribution agreements. The conclusion of the discussion 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.

So for the time being Ecma should not directly accept third-party contributions. However, an Ecma member could serve as an intermediary. So, Ecma would see only Ecma members software contributions. It was agreed that **Mr. Neumann** will report about this question in his next CC report.

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).

4.1.2 Non-member software contributions

This issue arises with the progression of ongoing work on Test262, where some contributions (tests) have been contributed from a non-member and not-channelled through a member. While this is not a pressing issue, and TC39 will continue to find solutions, we may require specific changes by the CC and GA to the targeted IPR and licensing policies set up for TC39 use sometime in the future.

4.2 Report from the ad hoc on Internationalization standard.

Ecma/TC39/2011/014 contains the Agenda for the 3rd meeting of TC39 ad hoc on Internationalization, 21 March 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

5.1 Press Release

It will cover ISO/IEC, Ecma, and Test 262, and should be ready for approval at the May meeting. **Mr. Neumann** has the action to get the first draft out to TC39 for review and discussion. The intent is to get TC39 approval at the May meeting so it is ready to go at the June GA meeting.

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

The relevant Ecma TC39 contributions are the following:

Ecma/TC39/2011/015 Documents mentioned in the clauses 6.1 to 6.9 of the agenda of the 21st meeting

Ecma/TC39/2011/017 Document "Classes and Traits - Looses consensuses and choices"

Ecma/TC39/2011/018 Document "Harmony Proxies: strawmen"

Ecma/TC39/2011/019 Document "Declarative object and class abstractions"

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

6.1 Done Classes and Traits
http://wiki.ecmascript.org/doku.php?id=strawman:classes_with_trait_composition

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

6.2 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 Quasis Done and not moved to Harmony proposal but new cycle needed

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

6.4 Function.prototype.toString
http://wiki.ecmascript.org/doku.php?id=strawman:function_to_string

6.5 Done Infix operators: has(no), div(no), mod(no), ??(ok), min, max

6.6 Done Number constants: MAX_INTEGER, EPSILON

6.7 Done Number.prototype. compare

6.8 Update on modules

Done and moved to Harmony proposal

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

6.9 Done and wiki moved to harmony proposal, with concepts in this paper also moved - - Update on binary data

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

6.10 Done but further work for next meeting - Records and tuples

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

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

6.11 - Array comprehensions

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

6.12 Done and promoted to Harmony - Generators and generator expressions

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

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

6.13 Done - Pattern matching

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

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

6.14 Done - Completion value reform

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

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

6,15 Done Extensions to Proxies:

6.15.1 Done - Proxy derived traps:
http://wiki.ecmascript.org/doku.php?id=strawman:proxy_derived_traps

6.15.2 Done - Handler access to proxy:
http://wiki.ecmascript.org/doku.php?id=strawman:handler_access_to_proxy

6.15.3 Done - Proxy set trap: http://wiki.ecmascript.org/doku.php?id=strawman:proxy_set_trap

6.15.4 Done - Derived traps of default forwarding handler:
http://wiki.ecmascript.org/doku.php?id=strawman:derived_traps_forwarding_handler

6.16 - Function proxy prototype:
http://wiki.ecmascript.org/doku.php?id=strawman:function_proxy_prototype
http://wiki.ecmascript.org/doku.php?id=strawman:completion_let

6.17 - Function.create

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

6.18 Done - Paren-free grammar changes

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

6.19 - Multiple global objects

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

6.20 Done Revisions to object abstraction proposals: (updates pending)

Done Object initializer

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

DonePrivate names http://wiki.ecmascript.org/doku.php?id=strawman:private_names

6.21 ECMASript object model and internal metaobject protocol: (content pending)

ES5 internal methods and aligning them with Proxy handlers

ES5 internal nominal typing and generalizing usage of [[Class]]

6.22 Done Enhanced Math Object

Spreadsheet comparing ECMAScript Math functions to various C/C++ math libraries

7 Date and place of the next meeting(s)

May 24 – 26, 2011. Location Santa Cruz, CA, hosted by UCSC

8 Closure

The TC39 Meeting ended at 4:40 on 26 March 2011. The group expressed appreciation to Google for hosting the meeting and to Ecma International for hosting the dinner the night before.

Item 4.2 Attachment

Attendees of the Internationalization Ad hoc meeting on March 21, 2011:

Nebojsa Ciric - Google

Erik Arvidsson - Google

Axel Hecht - Mozilla

Jungshik Shin - Google

Addison Philips - Phone - Amazon - W3C

Shawn Steele - Phone - Microsoft

Peter Constable - Phone - Microsoft

Mark Davis - Phone - Google

Started implementation of collator in Chrome and hit a problem:

```
var coll = locale.collator();
array.sort(coll.compare);
```

Compare method gets bound to the undefined or global object at the call site. Erik mentioned that this problem will be solved in Harmony by passing additional "this" parameter to for-each and likes. We would like to propose extendiCiricng this syntax to the sort method too.

We discussed each part of the API in order to get detailed parameters of each constructor and method.

General:

- Add options property to each class that would give you actual value for the user parameters. For example, if user asked for islamic calendar, and we only have islamic-civil, we set calendar property to islamic-civil. Allows developer to iterate until satisfied with the result.
- Use Unicode identifier vs. BCP47 in the API

Collator:

- numeric - specifies numeric sort (9 comes before 12)
- ignoreVariants - ignore all of case, width and kana
- ignoreWidth, ignoreCase and ignoreKana - subvariants we may implement to fine tune the behavior
- ignoreAccents - ignore accents
- ignoreSymbols - ignore punctuation and symbols
- variant - phonebook, ... - string

NumberFormat:

- Allow patterns to specify grouping, currency symbol position and sign location
- Start with ICU patterns and see if they work for everybody
- Don't support overrides for grouping separator and decimal point for now
- Specify both currencySymbol and currencyCode as override

DateTimeFormatSymbols:

- Added Era and day period methods (AM/PM)
- Moved all methods to DateTimeFormat class
- Remove DTFSymbols class

DateTimeFormat:

- Specify calendar names better (move work to Unicode/LDML and point to their document).
- Allow short/long dateType to get value from the system or cloud.
- .options[skeleton] should contain best match for the given skeleton

Item 6 Attachment

Notes kindly provided by **Waldemar Horwat.**

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.

NOTES FROM THURSDAY

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.
