Draft minutes for the:	**23rd meeting of Ecma TC39**
held in:	**Redmond, WA, USA**
on:	**27 – 28 July 2011**

1 Opening, welcome and roll call

1.1 Opening of the meeting (Mr. Neumann)

The TC39 meeting (hosted by Microsoft in Redmond, WA) was opened by **Mr. Neumann**, Chair of TC39 at approximately 10:15 AM on 27th May 2011 (TC39/2011/029 - Venue for the 23rd meeting of TC39, Redmond, WA, July 2011).

It was noted that before the TC39 meeting, on the 26th of May 2011 the TC39 ad-hoc group on internationalization (i18n Ad Hoc group) has also met in Redmond. The report of that meeting is given under 4.2 below.

1.2 Introduction of attendees

Alex Russell - Google

Istvan Sebestyen - Ecma-International

Waldemar Horwat - Google

Allen Wirfs-Brock - Mozilla

John Neumann – Ecma International

Sam Tobin-Hochstadt - Northeastern University

Douglas Crockford - Yahoo!

Brendan Eich - Mozilla

Mark Miller - Google

Luke Hoban - Microsoft

David Fugate – Microsoft

Amanda Silver - Microsoft

Peter Constable – Microsoft

Dave Herman - Mozilla

Andreas Rossberg - Google

Nebojša Ćirić - Google

1.3 Host facilities, local logistics

Mr. Luke Hoban welcomed on behalf of Microsoft the delegates and provided logistical information. It was announced that Ecma international would host a social event on July 27th evening.

2 Adoption of the agenda (2011/030 Rev 3)

Ecma/TC39/2011/030 Rev 3 contained the Agenda for the 23rd meeting of TC39, Redmond, July 2011. This was agreed with minor changes to group subjects.

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

- Ecma/TC39/2011/028 Draft minutes of the 22nd meeting of TC39, Santa Cruz, May 2011
- Ecma/TC39/2011/029 Venue for the 23rd meeting of TC39, Redmond, July 2011
- Ecma/TC39/2011/030 Agenda for the 23rd meeting of TC39, Redmond, July 2011 (Rev. 3)
- Ecma/TC39/2011/031 Draft external Liaison Report for 2010–2011 to SC 22
- Ecma/TC39/2011/032 First draft Standard ECMA-262 6th edition
- Ecma/TC39/2011/033 Ecma External Liaison Report for 2010–2011 prepared for the ISO/IEC JTC 1/SC 22 Plenary
- Ecma/TC39/2011/034 Documents mentioned in the clauses 5.1 to 5.14 of the agenda of the 23rd meeting
- Ecma/TC39/2011/035 Test262 - Status Report, July 2011

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

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

3 Approval of minutes from March 2011 (2011/028)

The minutes of the 22nd TC39 meeting in Santa Cruz in May 2011 have been unanimously approved with no changes.

4 Report from the Secretariat

4.1 Report from Geneva

Mr. Sebestyen has reported about the June 2011 GA meeting in Divonne. He said that **Mr. Breidthardt** and **Mr. Neumann** have given an excellent report about TC39 activities. **Mr. Breidthardt**, CC Chair, has congratulated to the hard TC39 work and progress. Regarding the TC39 Software Copyright problem with 3rd party software contributions and how to handle available free open source software he reported to the GA that for ES5 testing the question came up and remedy is required. The CC and the GA just had no time yet to work on these issues, but hopefully it will be picked up soon, and we can report about the results in a few months.

Mr. Sebestyen also said that on June 29, 2011 at the GA there was a celebration of the “Golden Jubilee” of Ecma. It was decided by the CC that a series of short presentations would be given including some standardization highlights of Ecma. Since ECMAScript is one of such highlights, **Mr. Neumann** gave a short presentation about the success-story ECMAScript. This involves the preparation of a few PowerPoint slides. All PowerPoint slides of the celebration have been published on the Ecma website.

On the ECMAScript Trademark matters he said that it was on its way in Switzerland, the EU and USA.

Mr. Sebestyen also said that the EU Commission has approved and published on June 1, 2011 a new Communication regarding ICT standardization that allows that non-ESO specifications (like an Ecma standard) can be formally recognized by the EU. Though the political part has already been decided, the details of the concrete implementation have to yet worked out. It is hoped that the procedure will be operational in early 2012.

Mr. Sebestyen also said that the EU Commission has invited Ecma to participate on July 15, 2011 in Brussels in an IPR Workshop on the experiences of "ex-ante" licensing in SDOs. He said that the experiences of SDOs and Fora & Consortia were very mixed, depending on scope, subject, culture, etc. of each body. He said that in his brief report he mentioned that in the current Ecma Patent policy the concept of "ex-ante" was not included, but Ecma has "ex-ante" experiences in the category "ex-ante RAND Zero", e.g. in TC39 practice, where its implementation depends on the goodwill of the involved, but the without the explicit help of the policy itself. He said that the CC will look into the matter if anything on that has to be done. TC39 expressed interest in the outcome of such CC discussion.

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

4.2.1 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/TC39/2011/035 Test262 - Status Report, July 2011.

The official name will be TEST262 ES5.

It was decided that a 1 page TR will be prepared for the December GA. The TR will point to the test suite itself.

The Ecma Secretariat (Patrick) will send a TR template to the Editor.

4.3 Report from the ad hoc on Internationalization standard (i18n Ad Hoc group).

4.3.1 Review of proposed draft standard

The i18n Ad Hoc group hold a face-to-face meeting on July 26th, 10-17h at the Microsoft Campus in Redmond. The agenda was to go over the first draft of the standard document.

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

4.3.2 Next steps

In the TC39 meeting the following has been decided:

TC39 will aim for frozen specification in September 2011, do a round of implementation and go for an Ecma General Assembly approval of the standard in June 2012. This delayed the original plan by about 6 months.

An immediate fast-track submission to JTC 1/SC 22 is planned.

4.4 Update of TC39 Web Page at Ecma home page

It was decided to update the TC39 webpage. **Mr. Neumann** will send the update instructions to the Ecma Secretariat.

4.5 Review of SC 22 Liaison Report

- Ecma/TC39/2011/031 Draft external Liaison Report for 2010–2011 to SC 22

TC39 has discussed the input draft and has prepared the following output that has been communicated to the Chair of JTC 1/SC 22, **Mr. Jaeschke**:

- Ecma/TC39/2011/033 Ecma External Liaison Report for 2010–2011 prepared for the ISO/IEC JTC 1/SC 22 Plenary

4.6 Review and comment on Last Call Working Draft of Web IDL (http://www.w3.org/TR/2011/WD-WebIDL-20110712/) ; deadline August 23 from Cameron McCormack <cam@mcc.id.au>

TC39 comments on WebIDL

Cameron,

Ecma TC39 met today in Redmond, WA and have agreed that we want to respond to your last call request for comments on WebIDL.

I have been asked to request an extension to the Last Call deadline to early October. Two of our members (**Brendan Eich** and **Allen Wirfs-Brock**) have agreed to thoroughly review the document and to prepare a proposed TC39 response. It is our intent to develop the final response based on their review and other comments from TC39 members during our next scheduled meeting in Mountain View (Mozilla host) and we would like to invite you to attend our meeting on September 26 and 27, 2011.

5 Latest on ES 5.1

5.1 Press Release

Mr. Sebestyen has reported that the Ecma Press Release on the ECMAScript 5.1 approval in Ecma, the ISO/IEC approval and publication, and Test 262, was released at BusinessWire immediately after the June GA meeting. The interest and response to the Press Release was very good, five times as high as to the ECMAScript 5.0 Press Release in December 2009.

5.2 Publication

The ISO/IEC publication of ECMAScript 5.1 (IS 16262) was done on May 25, 2011.

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

6.1 Handler access to proxies

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

6.2 Proxy drop receiver

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

6.3 Insights from ES.next feedback gathering presentations

o Highest perceived value features, areas of concern, significance of versioning

6.4 Update on Math, String and Number API improvements

6.5 Overview of initial working draft for 6th edition and discuss work flow for developing 6th edition draft

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

6.6 Review/resolve open issues and change requests for 6 edition

htt8s://bugs.ecmascript.org/buglist.cgi?cmdtype=runnamed&namedcmd=Pending%20TC39%20Review

6.7 Review updates to Object Initialiser super proposal

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

6.8 Syntax Discussion

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

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

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

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

6.9 Minimal classes:

https://mail.mozilla.org/pipermail/es-discuss/2011-June/015559.html

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

6.10 Generators:

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

6.11 Binary data:

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

6.12 Eval (Mark)

7 Date and place of the next meeting(s)

September 26 (Monday) – 27 (Tuesday), 2011. Location: Silicon Valley, CA, hosted by Mozilla.

November 16-17, 2011. Location: Silicon Valley, CA, hosted by Apple.

January 25-26, 2012. Location: Silicon Valley, CA, hosted by Yahoo.

8 Closure

The TC39 Meeting ended at 4:00 PM on 28 July 2011. **Mr. Neumann** has thanked the meeting participants for their good contributions, constructive discussions and the co-operative spirit of the group.

The group expressed appreciation to Microsoft and to **Mr. Luke Hoban** for hosting the meeting and to Ecma International for hosting the Mexican dinner on the 27th July in Kirkland, WA.

Item 4.2 Attachment

ECMAScript i18n Ad-hoc meeting summary (7/26/2011):

Attendees:

- Microsoft: Peter Constable, Derek Murman, Eric Albright, Luke Hoban
- Google: Nebojša Ćirić , Jungshik Shin, Mark Davis
- Yahoo : Norbert Lindenberg

------

27. јул 2011. 10.52, Nebojša Ćirić <cira@google.com> је написао/ла:

Hi all,

these are steps that need to happen if we were to meet deadline for Ecma General Assembly in December:

1. First draft of the document has to be ready for TC39 review 3 weeks before September meeting (Sept 28th).

2. Final draft if approved, will be published 2 months before the General Assembly meeting - publish date is October.

There are some unresolved issues left from our last July 26th meeting that we need to resolve in August in order to meet the deadlines. I propose a teleconference (I will book local room too for people that want to show up in person) mid August.

List of issues:

1. Collation was not reviewed.

2. Extended date/time skeletons proposed by Erik (Microsoft)

3. Skeleton for NumberFormat proposed by Nebojsa (Google)

4. Global namespace name - we used globalization, but we should try shortening it.

5. UTC timezone addition to DateTime format (and local timezone)

6. Name for options property - suggested resolvedSettings?

Things we decided on:

1. Remove LocaleInfo class.

2. Add global object that acts as a namespace for DateTimeFormat, NumberFormat and Collator classes -- and future extensions to the standard.

3. Remove regionID property since we pass currencyCode directly.

4. Each service accepts localeID property (empty, string, array of strings) and does validation/resolution based on supported locales for a service.

5. Options property holds resolved values that user asked for (numberSystem: thai if -u-nu-thai was specified).

6. For currency formatting, user has to specify ISO currency code - throw exception if it's missing.

7. Rounding mode for number formatting was defined as round half up (away from 0).

8. Error handling was defined better - exceptions for invalid parameters

--
Nebojša Ćirić

--
Nebojša Ćirić

Item 6 Attachment

From: brendan@mozilla.com
To: es-discuss@mozilla.org
Sent: 7/28/2011 12:21:34 A.M. Eastern Daylight Time
Subj: July TC39 meeting notes, day 1

== Handler access to proxies ==

Proxy handler traps need to receive the proxy as a parameter: first, or last?

Last allows trap implementors to leave |proxy| off. It's also a compatible extension to the proposal and its prototype implementations. Putting |proxy| last may also steer implementors away from touching proxy, reducing the bugs where you infinitely diverge.

First is more normal-order (proxy, name) and some find it more aesthetically pleasing.

Another alternative: the proxy could be "passed" via a data property on the handler. But the only use for the proxy reference is as a key in a weakmap, and if the handler references it, the handler could just as well be the key -- or the handler could simply hold the value associated by the weakmap.

Conclusion: no rationale for adding a |proxy| parameter to all traps.

== Proxy drop receiver ==

Sean Eagan pointed out in

https://mail.mozilla.org/pipermail/es-discuss/2011-April/013916.html

that the receiver parameter of Proxy handlers' get and set traps is useless, due to how these derived traps layer on fundamnetal traps.

Conclusion: remove |receiver| from these traps.

== Luke's report on Microsoft-internal feedback ==

Runtime extensions usable soonest, modules significant but full impact unclear, concern about versioning.

Positive: private names, classes, let, proxies, typed arrays and binary data

Concern: enhanced object literals, destructuring, quasis

Indifferent: Maps/sets, iterators, tail calls, (parameter) default values, comprehensions, spread/rest

Possible confusion about Map and Set requiring object typed keys -- they allow all key types. What may be wanted: "computed keys", i.e., objects that are equated not by identity, rather via equals/hashcode-style methods.

ES5 not viewed as super-compelling. A few "use strict"; adoption cases, unclear what it helped or hurt.

Happy to be able to feature-detect APIs, cannot feature-detect syntax (false: eval), but can't use ES5 until it is supported by most browsers in the field. Counterpoint: in that case, when it is supported enough both APIs and syntax can be used.

Subsets while prototyping the next edition, coordination among implementors, to get the good stuff out.

Concern about brevity of <script> vs. <script type="application/ecmascript;version=6">. Counterpoint: |use version 6;| in-band in the script content. Can use both, or just the latter. Agreement on usability including conciseness being important to work on for ES.next.

An offline compiler from ES.next to ES3+R would help.

Some better way to conditionally load scripts depending on UA language version support...

API and feature wishlist:

* Crypto APIs (no details re: getRandomValues vs. AES, etc.)
* Stack traces, debugging aids
* Data-binding
* Binary source format for code size,load time perf wins
* Shothand for typeof someGlobal == "undefined"
* Compiler hooks for code transformation
* Int64, Bignum
* More focus on static analysis (part of each group's toolchain)

Destructuring had love / hate reaction (object pattern without shorthand, less worthy objections).

Modules:
* Early errors make code brittle as modules evolve if you use |import foo.*|.
* Async loading implications
  * Soft failure on timeouts?
  * Scenarios where developer doesn't know whether to use static or dynamic loading?
  * Want dependent script loading to be delayed till "truly dynamically needed"
* Browser-level compoment dependency mechanism -- can support CSS and HTML too?

Classes
* Positive readability and debugability reactions
* Desugaring important for interop with existing prototypal patterns
* "class" terminology and the keyword itself raises some hackles

Object literal
* The ~!# and := cussing is not popular
* People don't want to make properties non-writable or non-configurable
* Prototype-of well-received as idea, <| syntax looks awkward
* Seems like a lot of overlap with classes, want only one

Quasi-literals
* "I don't like Perl" reaction
* String.format instead
* "Love it, I would use it all the time" reaction
* Name is confusing

Iterators and generators
* "Yes, that would be cool"
* Not something that would be heavily used unless provided by common libraries
Audience was mix of front end developers and library/middleware authors

Proxies
* Awesome... wait, do we really want unstratified ("observe")? Yeah...
* Don't immediately see applications, but could see abstracting away DOM diffs
* Could simplify sandbox, but would prefer direct browser support, at MS Web Sandbox or Caja granularity

== Luke's work on Math and String extensions ==

Math functions
* Java signum (-1, 0, or 1 return value) vs. C signbit (non-zero if negative including -0, zero otherwise)
  * What is the purpose? Translating numeric algorithms from Java, or C?
  * NaNs should be canonical so you can't distinguish negative from positive NaNs
  * Safe way to make a non-NaN float from uint32 or even uint64 data
* Math.{log1p,expm1,log2,log10,hypot}, etc.
* Math.{cosh,sinh,tanh,acosh,asinh,atanh}
* gamma, erf? Waldemar argues for including
* Open question about OS libm interoperation
* deg2rad and rad2deg? Math.M_2_PI instead? neither? not sure

String methods
* reverse (like s.split('').reverse().join('')) along with the ones adopted in May
  * Waldemar: reorders uint16 units that form combining sequences, surrogates
  * General agreement that this is novel as a hazard and reverse is too marginal a thing to add with this hazard
* want UTF-8 transcoding with byte arrays, topic for binary data

== Editor's update ==

Allen gave an update on the editorial process and his use of https://bugs.ecmascript.org/ to track ES5.1 errata and ES6 work items.

/be

____
es-discuss mailing list
es-discuss@mozilla.org
https://mail.mozilla.org/listinfo/es-discuss

From: brendan@mozilla.org
To: es-discuss@mozilla.org
Sent: 8/4/2011 5:29:55 P.M. Eastern Daylight Time
Subj: July TC39 meeting notes, day 2

A week late, I missed some of the morning due to a conflict. Thanks to Alex Russell for whiteboard photos. Others in attendance, please fill in and correct as needed. Thanks.

== Overview of initial working draft for 6th edition and discuss work flow for developing 6th edition draft ==

Allen presented the draft 6th edition and how best to develop it:

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

The phrase "extended code" as a way of specifying semantics for Harmony above and beyond "strict mode code" received some discussion (I missed most of it). In the end no one had a better term than "extended".

Allen also presented the use of cover grammars ("Supplemental Syntax") to specify destructuring assignment and possibly other syntactic extensions. The problem here is that an LR(1) grammar cannot distinguish an object or array literal from a destructuring pattern of the same form, until parsing reaches the '=' after the pattern. GLR or ordered choice top-down parsing techniques can cope, but LR(1) and therefore LL(1) cannot -- such an LR(1) grammar is ambiguous.

Note that ES1-5 cope with the existing ambiguity where x.y.z and x.y.z = w both start with a member expression by pushing the ambiguity off to the semantics, creating a spec-internal Reference type, which remembers the base object (what x.y evaluated to) and property name ('z'), and only getting the value (GetValue) if the x.y.z expression is an rvalue, otherwise using the Reference type lvalue to assign to the named property (PutValue).

Computing a "Reference tree" or "minimal AST" for whole and arbitrarily large literal patterns, to defer evaluation till an assignment operator is parsed and we know the pattern is a destructuring lvalue rather than an object/array literal rvalue (after which GetValue or PutValue would process the tree according to its rvalue or lvalue nature) is not feasible.

This is due to the generality of the PropertyAssignment and ElementList productions' AssignmentExpressions, which may embed function expressions and thus most of the grammar. We do not want to add an explicit and nearly-complete parse tree or AST to the spec.

The committee seemed to agree that the cover grammar approach seems like the best technique.

== Review/resolve open issues and change requests for 6 edition ==

https://bugs.ecmascript.org/buglist.cgi?order=Importance&list_id=384&field0-0-0=flagtypes.name&resolution=---&query_format=advanced&type0-0-0=substring&value0-0-0=TC39&product=Draft%20for%206th%20Edition

(Bug rows from the above query follow, starting with bug numbers.)

145 nor Normal All allen@wirfs-brock.com CONF --- eliminate uint32 length restriction the the length of array objects.
146 nor Normal All allen@wirfs-brock.com CONF --- Array generic array methods should not ToUint32 covert the length of non-generic arrays

We deferred these, agreeing that they seem worth trying to make as relaxations of existing index/length semantics for arrays, to align with String and avoid bogus uint32-domain work that cannot handle the "overflow" case of length == 2^32. They will change edge-case behavior. The changes may break only testsuites, but you never know.

178 nor Normal All allen@wirfs-brock.com CONF --- Must settle scoping details for block-scoped bindings

Much discussion here. The issue is whether let and const bindings hoist to block top, or start a new implicit scope (the let* or, let's call it, C++ rule). The prior work was nicely diagrammed by Waldemar in:

https://mail.mozilla.org/pipermail/es-discuss/2008-October/007807.html

Quoting from Waldemar's message (note the future-proofing for guards):

--- begin quote ---

There are four ways to do this:
A1. Lexical dead zone. References textually prior to a definition in the same block are an error.
A2. Lexical window. References textually prior to a definition in the same block go to outer scope.
B1. Temporal dead zone. References temporally prior to a definition in the same block are an error.
B2. Temporal window. References temporally prior to a definition in the same block go to outer scope.

Let's take a look at an example:

```
let x = "outer";
function g() {return "outer"}

{
  g();
  function f() { ... x ... g ... g() ... }
  f();
  var t = some_runtime_type;
  const x:t = "inner";
  function g() { ... x ... }
  g();
  f();
}
```

B2 is bad because then the x inside g would sometimes refer to "outer" and sometimes to "inner".

A1 and A2 introduce extra complexity but doesn't solve the problem. You'd need to come up with a value for x to use in the very first call to g(). Furthermore, for A2 whether the window occurred or not would also depend on whether something was a function or not; users would be surprised that x shows through the window inside f but g doesn't.

That leaves B1, which matches the semantic model (we need to avoid referencing variables before we know their types and before we know the values of constants).

--- end quote ---

In the September 2010 meeting, however, we took a wrong turn (my fault for suggesting it, but in my defense, just about everyone did prefer it -- we all dislike hoisting!) away from hoisted let and const bindings, seemingly achieving consensus for the C++ rule.

Allen, it turned out, did not agree, and he was right. Mixing non-hoisting (the

C++ rule) with hoisting (function in block must hoist, for mutual recursion "letrec" use-cases and to match how function declarations at body/program level hoist) does not work. In the example above, g's use of x either refers to an outer x for the first call to g() in the block, but not the second in the block (and various for the indirect call via f()) -- dynamic scope! -- or else the uses before |const x|'s C++-style implicit scope has opened must be errors (early or not), which is indistinguishable from hoisting.

So at last week's meeting, we finally agreed to the earlier rules: all block-scoped bindings hoist to top of block, with a temporal dead zone for use of let and const before *iniitalization*.

The initialization point is also important. Some folks wondered if we could not preserve var's relative simplicity: var x = 42; is really var x; x = 42, and then the var hoists (this makes for insanity within 'with', which recurs with 'let' in block vs. 'var' of same name in inner block -- IIRC we agreed to make such vars that hoist past same-named let bindings be early errors).

With var, the initialization is just an assignment expression. A name use before that assignment expression has been evaluated results in the default undefined value of the var, assuming it was fresh. There is no read and write barrier requirement, as there is (in general, due to closures) for the temporal dead zone semantics.

But if we try to treat let like var, then let and const diverge. We cannot treat const like var and allow any assignment as "initialization", and we must forbid assignments to const bindings -- only the mandatory initializer in the declaration can initialize. Trying to allow the "first assignment to a hoisted const" to win quickly leads to two or more values for a single const binding:

```
{
  x = 12;
  if (y) return x;
  const x = 3;
  ...
}
```

The situation with let is constrained even ignoring const. Suppose we treat let like var, but hoisted to block top instead of body/program top, with use before set reading undefined, or in an alternative model that differs from var per temporal dead zone, throwing. So:

```
{
  print(x);
  x = 12;
  let x;
}
```

would result in either print being called with undefined or an error on the use of x before it was set by the assignment expression-statement -- those are the two choices given hoisting.

But then:

```
{
  x = 12;
  print(x);
  let x;
}
```

would result in either 12 being printed or an error being thrown assigning to x before its declaration was evaluated.

Any mixture of error with non-error (printing undefined or 12) is inconsistent. One could defend throwing in the use-before-assignment case, but it's odd. And throwing in both cases is the earlier consensus semantics of temporal dead zone with a distinct state for lack of initialization (even if the initialization is implicit, e.g., in a declaration such as let x; being evaluated). Here "initialization" is distinguished from assignment expressions targeting the binding.

Trying to be like var, printing undefined or 12, is possible but future-hostile to guards and gratuitously different from const:

```
{
  x = 12;
  const G = ...;
  let x ::G = "hi";
}
```

We want to be future-proof for guards, and even more important: we want to support *refactoring from let to const*. Ergo, only temporal dead zone with its barriers is tenable.

There remains an open issue: without closures obscuring analysis, it is easy to declare use before initialization within the direct expression-statement children of a given block to be early errors, rather than runtime errors:

```
{
  x = 12;          // can be early error
  print(x);        // can be early error
  function f() {
    return x;      // may or may not be error
  }
  escape(f);       // did this call f?
  let x = 42;
  escape2(f);      // did this call f?
}
```

Some on TC39 favor normative specification of early errors for the easily-decided cases. Others want runtime-only error checking all around and point out how even the easy cases (within straight-line code in the block's direct expression-statement children) testing that reaches the block will fail fast. The question remains: what if the block is not covered by tests?

Dave Herman brought up the let/var at top level equivalence implemented in SpiderMonkey, specifically in connection with <script> tags. Sketching in pseudo-HTML:

```
<script type=harmony>
  alert = 12;      // reassign built-in alert
</script>

<script type=harmony>
  let alert = 13;  // shadow built-in alert
  var quux = 14;   // this.quux = 14
  let quux = 15;   // alternative: in scope for later scripts?
</script>

<script>
  alert(quux);
</script>
```

Dave's point was not to commend the SpiderMonkey equating of let and var at top level, but to observe that if "let is the new var", then depending on how multiple successive script elements' contents are scoped, you may still need to use var in Harmony -- let won't be enough, if it binds only within the containing <script> element's scope.

Recall that Harmony removes the global (window in browsers) object from the scope chain, replacing it with a lexical environment with (generally) writable bindings. Each script starts with a fresh lexical environment, although it might be nested (see next paragraph).

For scripts that do not opt into Harmony, there's no issue. The global object is on the scope chain and it is used serially by successive script elements.

The question for Harmony scripts boils down to: should successive Harmony scripts nest lexical scopes in prior scripts' scopes, like matryoshka dolls? Or should each script opted into Harmony be its own module-like scope, in which case to propagate bindings to later scripts, one would have to

```
<script type=harmony>
  export let quux = 14; // available here and in later scripts
</script>
```

This remains an open question in TC39. Some liked the explicit 'export' requirement, the implicit module scope. Others objected that migrating code would expect the nested semantics, which was not inherently evil or unsafe.

--- end of block scope discussion ---

173 enh Normal All allen@wirfs-brock.com CONF --- FutureReservedWords should not be allowed as a function name or argument name of a strict func.

Deferred but this was considered straightforward, per the comment 0.

157 min --- All allen@wirfs-brock.com CONF --- "do{;}while(false)false" prohibited in spec but allowed in consensus reality

Approved -- this is the de-facto standard whereby do;while(0)x will have a semicolon inserted before x.

== Minimal Classes ==

Dave presented his pitch for minimal classes, posted to es-discuss previously here:

https://mail.mozilla.org/pipermail/es-discuss/2011-June/015559.html

This subset includes class C {...}, class D extends C {...}, super calls in constructors, and method syntax for defining non-enumerable function-valued data properties on the class's prototype object.

The premise is that classes have significant open issues that will take time to resolve, at high opportunity cost, without clear consensus in sight for some of the issues; whereas the minimal "profile" has consensus already and will do good in ES.next without question.

Dave quickly acknowledged Mark M.'s long-running and indefatigable effort to get classes as sugar into Harmony, and how this was not in any way lost forever via minimal classses. Some of the early work used the closure pattern, which is not the main pattern supported by the current proposal (but it is supported if you write public methods in the constructor). So, credit to Mark for his work.

Waldemar thought the approach too minimal, and suggested zero-inheritance as a different axis on which to miminize. Others disagreed with that, noting how the existing pattern (the cowpath to pave) has at least subclassing and super-call boilerplate in library code and generated JS to absorb.

General agreement to work through open issues (recorded in the wiki or not) with the http://wiki.ecmascript.org/doku.php?id=harmony:classes proposal.

Open issues and discussion/resolution summaries:

1. return allowed from constructor?

Dave H.: This is a lesser cowpath possible with functions-as-constructors-with-prototypes in JS today, we should pave it.

Mark M., others: we want minimal object layout or "shape" declarative guarantees with classes, return {unshaped: "haha"}; in the middle of a constructor for a class with public x, y, z; properties defeats this goal.

Dave: no shape guarantees.

Others: must have shape guarantees, at least "the declared properties exist, at the moment the constructor returns" (ignoring const classes, which have frozen prototypes, instances, constructors, and at least sealed instance properties).

Mark M.: argument by analogy to module objects.

Dave, Sam, Brendan: module system is second class, module object reflections do not correspond to class-as-factory instances. This is not to say "no shape guarantees", however (Brendan at least).

Alex R.: minority-use-case users can fall back to declaring constructor functions if they need to return from constructor.

Consensus is: RESOLVED, return disallowed from class constructor body.

2. private: section vs. private prefix keyword

Some favor sections, others do not. Waldemar cites original-JS2/ES4 private {...} braced forms to distribute privaste, static (class), etc. across a group of declarations, with the braced body being a declaration list, not an object literal. Bob Nystrom proposed C++-style section syntax here:

http://www.mail-archive.com/es-discuss@mozilla.org/msg09070.html

No resolution.

3. private variable use-syntax

The private(this).x, private(other).x syntax in the wiki'ed proposal is an intentional non-starter: too verbose, wrongly suggests that private variables are properties of some "private data record" object.

But what to use instead? @ as prefix (this-based) and infix (restricted, no LineTerminator to left, for other-based) private-keyed property refs has been mooted but is not proposed in the classes proposal, and my sense is the committee is not ready to annex @.

Meanwhile, Allen proposed (see PDF link http://wiki.ecmascript.org/lib/exe/fetch.php?id=harmony%3Aprivate_name_objects&cache=cache&media=harmony:private-name-alternatives.pdf at bottom of http://wiki.ecmascript.org/doku.php?id=harmony:private_name_objects) that we allow private name objects to be used in object literals like so:

```
  return {
    [MyPrivateKey]: ...,
    publicNameHere: ...
  };
```

and this was agreed to at last week's meeting.

Given this extension, whose [] syntax mirrors the computed property name "indexing" used with private name objects as property keys, we agreed to defer private(this)/private(other) replacement syntax from the classes proposal and revisit later, based on usability experience with private name objects including this extension to object literal syntax.

4. class-side inheritance (method, this, super)

Some (Smalltalk and therefore Ruby matter to these folks) on TC39 want, others are indifferent. No one was hostile, but we did not resolve to add class-side inheritance yet.

People agree that "static" is the wrong keyword, but may have weight for some coming from C++ and Java. General desire to use "class" but not as prefix keyword.

5. no magic syntax for constructor body

This matters more if we attempt to unify class body syntax with object literal extended syntax, or somehow make a desugaring from one to the other. The general form of this open issue is item (6), but to focus on instance properties/variables, we split this one in two:

5a. public x = x inside a constructor taking parameter x. We do not have a better alternative at this time. The C++-style public: section idea discussed on the list (proposed by Bob Nystrom, see (2) above) separates declaration from likely initialization based on constructor parameters or other constructor/instance-dependent computation.

5b. private w = ... instead a constructor. Per 3, we removed this for now, deferring to private name objects and agreeing to revisit later.

6. do not abuse lexical binding declarative forms to define properties (prototype, class)

This was contentious at first, because modules use 'export function f(...){...}' and 'export const K = ...' declarative syntax extended by prefixing with 'export', so Mark at least found the use in class syntax of declarative forms to bind prototype properties plausible:

```
  class C {
    function f() {} // C.prototype.f?
    let x;          // C.prototype.x?
    const k;        // C.prototype.k?
    class Inner{}   // C.prototype.Inner, (c = new C, c.Inner)
    m(){}           // method m, no controversy, no comma after
    get gs(){}      // getter gs, no controversy, no semicolon after
    set gs(x){...}  // setter gs, optional, no controversy
    pp = 42;        // prototype property pp, controversial syntax --
assignment?
    constructor(){} // just a method, unless the body is special syntax (1)
  }
```

After some discussion it became clear that there is no symmetry with module instances without 'const class': module exports are sealed properties, module instance objects are not extensible. Class instances by default are extensible, and class prototypes in particular are as mutable as today. In general Harmony moves binding forms away from defining properties on objects (except via reflection, as in the module instance case) and to lexical scope.

This item was not resolved, but it left all of the variations sketched above with comments ending in ? as open issues. Methods (including constructor, even if its body is a special form), getters, and setters are ok. All others are not yet resolved as consensus features of classes in ES.next.

I observed that at the rate of progress resolving open issues in the classes proposal (counting generously), we needed 2.5 more meetings to resolve the rest. But the remaining issues are actually bigger, and lack live alternative proposals that helped resolve (1) and (5b).

To make progress, we need to avoid "hovering" at a bogus "consensus" where people agree with the idea or general goal of classes, without getting concrete final agreement on all the details.

/be