Minutes for the: *35th meeting of Ecma TC39*
in: *Redmond, WA, USA*
on: *23-25 July 2013*

1 Opening, welcome and roll call

1.1 Opening of the meeting (Mr. Neumann)

Mr. Neumann welcomed the delegates at the Microsoft Campus, in Redmond, WA, USA.

Companies / organizations in attendance:

Mozilla, Google, Microsoft, eBay, jQuery, Yahoo, Adobe, Intel, Facebook (guest)

1.2 Introduction of attendees

John Neumann – Ecma International
Istvan Sebestyen – Ecma International
Mark Miller – Google
Douglas Crockford – eBay
Luke Hoban – Microsoft
Paul Leathers – Microsoft
Ian Halliday – Microsoft
Brian Terlson – Microsoft
Ron Buckton – Microsoft
Travis Leithead – Microsoft
Rick Hudson – Intel
Allen Wirfs-Brock – Mozilla
Anne v. Kesteren – Mozilla
Norbert Lindenberg – Mozilla
Erik Arvidsson – Google
Waldemar Horwat – Google
Avik Chaudhuri – Adobe
Alex Russell – Google
Eric Ferraiuolo – Yahoo!
Matt Sweeney – Yahoo!
Yehuda Katz – jQuery
Rick Waldron – jQuery
Dave Herman – Mozilla
Brendan Eich – Mozilla
Raffael Weinstein – Google
Dimitry Lomov – Google

Jeff Morrison – Facebook

Sebastian Markbage – Facebook

1.3 Host facilities, local logistics

On behalf of Microsoft **Luke Hoban** welcomed the delegates and explained the logistics.

1.4 List of Ecma documents considered

Ecma/TC39/2013/029	Minutes of the 34th meeting of TC39, London, May 2013
Ecma/TC39/2013/030	Venue for the 35th meeting of TC39, Redmond, July 2013
Ecma/TC39/2013/031	Ecma External Liaison Report for 2012–2013 for ISO/IEC JTC 1/SC 22
Ecma/TC39/2013/032 (Rev. 1)	Agenda for the 35th meeting of TC39, Redmond, July 2013
Ecma/TC39/2013/033	2nd draft Standard "The JSON Data Interchange Format", July 2013
Ecma/TC39/2013/034	3rd draft Standard "The JSON Data Interchange Format", July 2013
Ecma/TC39/2013/035	4th draft Standard "The JSON Data Interchange Format", July 2013
Ecma/TC39/2013/036	5th draft Standard "The JSON Data Interchange Format", July 2013
Ecma/TC39/2013/037	6th draft Standard "The JSON Data Interchange Format", July 2013
Ecma/TC39/2013/038	Test262 Status Deck, 25 July 2013
Ecma/TC39/2013/039	"Integrating ES and Web: Event Models" by Rafael Weinstein
Ecma/TC39/2013/040	"Value Objects" by Brendan Eich
Ecma/TC39/2013/041	7th draft Standard "The JSON Data Interchange Format", July 2013
Ecma/TC39/2013/043	Final draft Standard "The JSON Data Interchange Format", July 2013 (Rev. 2)
Ecma/TC39/2013/044	Parallel JavaScript (River Trail) Update

2 Adoption of the agenda (2013/032-Rev1)

The final meeting agenda as presented on the github server (and shown below) was approved.

Agenda for the: 35th meeting of Ecma TC39

```
in: Redmond, Washington, USA
on: 23 - 25 July 2013
TIME: 10:00 till 17:00 PDT on 23rd and 24th of July 2013
      10:00 till 16:00 PDT on 25th of July 2013
LOCATION:
    Building 27 Olympic Room
    Microsoft
    3009 157th Pl NE
    Redmond WA, 98052
```

Mr. Luke Hoban's email: lukeh@microsoft.com
Mr. Luke Hoban's local phone number: 425-706-7922

Please register before 12th of July 2013.

1. Opening, welcome and roll call
    1. Opening of the meeting (Mr. Neumann)
    2. Introduction of attendees
    3. Host facilities, local logistics
2. Adoption of the agenda (2013/???)
3. Approval of the minutes from May 2013 (2013/029)
4. ECMA-262 6th Edition
    1. Status report on latest draft (Allen)
    2. Add fill and copySlice methods to Array.prototype and Typed Arrays http://wiki.ecmascript.org/doku.php?id=strawman:array_fill_and_move (Allen)
    3. Array.prototype.values (Rick Waldron)
    4. Schedule pressure: Consider deferring to ES6 Refutable Matching. However, still make necessary changes to destructuring to future proof it.(Allen)
    5. Why new Built-in constructors should not have called as factory semantics. (Allen)
    6. Binary data proposal status (David Herman, Dmitry Lomov, Wed or Thurs)
    7. Math operations (David Herman)
    8. Stable Array.prototype.sort (Norbert)
    9. Time zones 1: Bugs or spec issues? (Norbert)
    10. Time zones 2: Time zone as property (Norbert)
5. Open issues discussion. (Allen)
    1. Symbol primitive value or object? One more time.
    2. Can computed properties name in object literals produce string prop names? http://esdiscuss.org/topic/on-ie-proto-test-cases#content-5
    3. {"__proto__":obj} and {["__proto__"]: obj}
    4. Are Typed Array instances born non-extensible?
    5. concat and typed arrays
    6. Can let/const/class/function* in non-strict code bind "eval" and "arguments"
    7. Does Object.freeze need an extensibility hook?
    8. Number.prototype.clz or Math.clz?
    9. Semantics and bounds of Number.isInteger and Number.MAX_INTEGER
    10. Number.toInteger or Number.prototype.toInteger
    11. ToUint32(length) and Array.prototype methods
    12. Should we remove [[Construct]] from the MOP and Proxy handler API?
    13. Which existing built-in properties that are read-only/non-configurable do we want to make read-only/configurable?
6. ECMA-262, Beyond the 6th Edition
    1. Object.observe/Array.observe spec changes and implementation report (Rafael - Wed or Thurs)
    2. Interfacing ECMAScript & HTML/DOM Event Loops (Rafael & Mark - Wed or Thurs)
    3. ES7 process
    4. Private symbols
    5. Parallel JavaScript (River Trail) (Rick Hudson - any day)
    6. Value objects update (Brendan)
    7. Promises (Anne van Kesteren - any day)
    8. Lacking primitives: Event Streams / IO Streams / How to do lists / How to do errors / Dates vs. timestamps (Anne van Kesteren - any day)
    9. Open Discussion: New Features without Strawmen
        1. Emitter: Standard lib module (Rick Waldron - any day)
        2. EventEmitter, EventTarget
        3. https://mail.mozilla.org/pipermail/es-discuss/2013-July/031734.html
7. ECMA-402
    1. Status report (Norbert)

8. Test 262
    1. Status report
9. Status Reports
    1. Report from Geneva
    2. Brief report from the GA meeting
    3. Implementation of the TC39 RF TG operating procedures (2013/019)
    4. W3C invitation of TC39 to the W3C technical plenary meetings TPAC 2013 (China) or TPAC 2014 (USA)
10. JSON
    1. Review and approve draft 4 JSON specification
11. Date and place of the next meeting(s)
    - September 17 – 19, 2013 (Bocoup - Boston)
    - November 19 – 21, 2013 (PayPal - San Jose)
12. Closure

3 Approval of minutes from May 2013 (2013/029)

The minutes of the May 2013 TC39 meeting have been approved as presented. Individuals that took technical notes were recognized and appreciation extended.

Rick Waldron volunteered to take technical notes of this meeting. The technical notes are included in Annex 1 of the minutes. They reflect the discussions correctly and in great details. The technical notes- have been already shared with TC39 and feedback was taken into account.

4-8 The agenda points 4-8 are discussed in depth in the Annex 1 below

9 Status Reports

9.1 Report from the Ecma Secretariat

9.1.1 Brief report from the IPR meeting

Mr. Sebestyen has reported that no reply yet from the IPR group on the software contributions by non-Ecma members, like 3rd party and Liaison organizations. He felt that the current contribution license text is already so broad that it could also be applied by 3rd parties and liaison organizations. This, however, has to be still verified by the IPR Group.

9.1.2 Brief report from the June 2013 Ecma GA. Approval of the TC39 RF TG operating procedures (2013/019)

The Ecma GA has approved the TC39 RF TG (royalty free task group) policy and the scope of the TC39 RF TG. Now, the royalty free group within TC39 has to be created. All TC39 members are requested to sign up using the new form to the TC39 RFTG. The plan is to collect the applications to the new RFTG by about mid September. The transition shall happen without interruption. It means that until sufficient members of TC39 members have not signed up to the TC39 RF TG only the existing RAND Ecma policy is applied. The new TC39 RF policy will apply only to the TC39 RFTG. It is experimental, like the TC39 software copyright policy.

9.1.3 Approval of the Non-Ecma-member contributions to the planned TC39 RF TG Work (2013/024)

This is a new "Public RF channel" that was requested by some members of the IPR group. The GA has also approved this new policy. This allows someone to write up a proposal and TC39 may incorporate it if it wants.

9.1.4 Ecma recognition program

Mr. Sebestyen explained that the GA decided that the CC should set up some common rules for the Ecma recognition program for those doing significant work in Ecma

standardization in place. There was agreement that TC leaders and Editors should be part of the recognition program, but further refinement was needed. So, the TC39 list was too long.

Nevertheless the two Editors present in the Redmond meeting could already receive the Ecma Recognition Award: **Mr. A. Wirfs-Brock** (ECMA-262) and **Mr. N. Lindenberg** (ECMA-402).

It was decided that for the 262Test **Mr. D. Fugate**, for ECMA-262 Editing work **Mr. P. Lakshman** and for the Internationalization Adhoc Group Leadership **Mr. N. Ciric** should get an Ecma Recognition award.

9.1.5 W3C invitation of TC39 to the W3C technical plenary meetings TPAC 2013 (China) or TPAC 2014 (USA)

Mr. Sebestyen reported that **Mr. P. Le Hageret** from the W3C has approached him with an invitation of TC39 to have again (after 2009) a joint meeting with them at the W3C TPAC 2013 meeting in China (November 2013). He told Mr. Le Hageret that in his opinion the joint meeting at this point in time at such a remote location was very unlikely. This was confirmed by the TC39 meeting. Maybe at TPAC 2014 if that meeting is in the USA in Silicon Valley that might be such an opportunity. There were several TC39 members who felt that liaison with W3C was important. It was agreed that if TC39 should have again a joint meeting with the W3C that meeting should be well prepared in advance with a clear list of topics to be discussed. Some TC39 experts plan to participate in the TPAC 2013 meeting. TC39 has appointed **Mr. A. Russell** to act as TC39 liaison representative at that meeting. **Mr. Sebestyen** was requested to communicate that appointment to **Mr. Le Hageret** (he did it…).

10 Json

Mr. Crockford has reported on the status of the JSON work both in the IETF (they want to put the informal (and expired) JSON RFC 4627 onto the Standards Track) and in Ecma TC39.

The conclusion was that the Ecma TC39 and the IETF work are orthogonal and therefore TC39 may go ahead with its own JSON Grammar standard. The goal of that standard is to document a stable grammar core which is common in all JSON applications.

The meeting looked at several drafts of the JSON standard and agreed to the final changes. It also approved to have a TC39 letter ballot on the final version of the JSON grammar standard. The plan is after the TC39 approval to have an Ecma-wide letter ballot on the JSON grammar standard. Such possible letter ballot was approved by the June Ecma GA. TC39 felt that it is rather urgent to document the JSON grammar and interchange format in an Ecma standard.

11 Date and place of the next meeting(s)

Schedule 2013 meetings

- September 17 – 19, 2013 (Bocoup - Boston)
- November 19 – 21, 2013 (PayPal - San Jose)

12 Closure

Mr. Neumann thanked **Microsoft** for hosting the meeting in Redmond, the TC39 participants their hard work, and **Ecma International** for holding the social event dinner Wednesday evening. Special thanks goes to **Rick Waldron** for taking the technical notes of the meeting.

Annex 1

Technical Notes (by Rick Waldron):

 # July 23 Meeting Notes

John Neumann (JN), Luke Hoban (LH), Rick Hudson (RH), Allen Wirfs-Brock (AWB), Yehuda Katz (YK), Anne van Kesteren (AVK), Jeff Morrrison (JM), Sebastian Markage (SM), Paul Leathers (PL), Avik Chaudhuri (AC), Ian Halliday (IH), Alex Russell (AR), Dave Herman (DH), Istvan Sebestyen (IS), Mark Miller (MM), Norbert Lindenberg (NL), Erik Arvidsson (EA), Waldemar Horwat (WH), Eric Ferraiuolo (EF), Matt Sweeney (MS), Doug Crockford (DC), Rick Waldron (RW)

JN: Brendan will be here tomorrow

Introductions.

Agenda

(https://github.com/tc39/agendas/blob/master/2013/07.md)

Discussion about getting agenda items in earlier, general agreement.

AWB: Clarifying #5 Open Issues (previously presented as a slide deck, now easier to track)

Consensus/Resolution

Will continue to use Github as the agenda tool, with the constraints:

- First Draft agenda will be published by the Ecma Secretarait 3 weeks before the meeting
-Agenda should be "locked in" 1 week prior to meeting
- Agenda URL will be posted to es-discuss immediately following the current meeting
- Allen has running "Open Items" section.

Corrections:

- What to expect in the RFTG mode
- Add JSON document to #9 JSON

Agenda Approved.

JN: Welcomes new Facebook participants who are invited guests at this meeting

4.1 ES6 Status Report

(Allen Wirfs-Brock)

AWB: Draft published, Revision 16

- Backed out the Symbols as non-wrapper types from Revision 15
- Section items renumbered for clarity
- Want to re-org/re-order the entire document
  - Primarily re-order the lexical grammar and syntax (currently out of order by 3 sections)

LH: (asked for motivation)

WH: Noticed...

- Changed for-in to disallow left side assignment expression.
- Syntax for arrow doesn't propagate the NoIn-ness of grammar rule. A NoIn arrow grammar production expands into a seqence that ends with a non-NoIn expression.

If we hadn't changed for-in to disallow left side initializers, this would break the grammar by allowing in's to leak into a NoIn expression.

However, we have changed for-in to disallow left side initializers. Given that, the regular/NoIn syntax rule bifurcation is now pointless. We have an opportunity to simplify and regularize the grammar here.

AWB: Will look into removing the NoIn productions.

LH: This was discussed recently on es-discuss

(need reference)

AWB:

- Further rationalization of alphabetizing Chapter 15

- Reminder that people should use the bug tracker for spec revision issues.
- Implementor feedback will be prioritized

LH: Re: initializer argument with an "entries" property will be used to create the entries in the Map (7.a)

"Let hasValues be the result of HasProperty(iterable, "entries")."

AWB: Explains the rationale of creating a new map from an existing map

```js
var m1 = new Map();
var m2 = new Map(m1);
```

LH/DH/YK/RW: Should be:

```js
var m1 = new Map();
var m2 = new Map(m1.entries());
```

EA: Should just use `@@iterator`, which is `entries`, but without explicitly checking for a property called `entries`.

DH: Advocate for uniform API, test for existance, assumes it's iterable, 2 element array-likes and initialize.

MM: Have we decided on the convention that an iterator of X is also an interable of X. A map.entries() gives you an iterator.

YK: map is already an iterable

DH: Should make sense to pass an iterator to Map

AWB: All the built in iterators are also iterables

DH: Agree, though this has been debated

WH: What happens...

```js
new Map([ "abc", "defg", "hi" ]);
new Map([{ 1: 10, 0: 20 }]);
```

BE: The first one makes a map mapping "a" → "b", "d" → "e", "h" → "i". The second one makes a map of 20 → 10.

AWB: The algorithm for Map should check for entries to be Object

DH:

MM: I don't think we should special case for string

AR: Agree, but not with example

MM: Making a special case for String seems like an odd decision

AR: In the case of i18n where we can't change the code point... you can imagine having a string map, but if I can just pass in a string.

... Don't object, just exploring

AWB: Objecting. What use case can you imagine where programmers intend for strngs to be array-like?

MM: None reasonable

...

MM: Question about value objects. If the value object responds to Get(0) or Get(1)

WH: with Mark, don't want special tests for different types

LH: If I do...

```js
new Map([ 1, 2, 3 ]);
```

I will get `undefined, undefined, undefined`, which is a stronger case for making the check

DH: +1

WH: Elsewhere, we've gone to detect duplicate errors

AWB: Checking for duplicates will duplicate the cost

MM: The impl of any hash table will require a test for duplicate keys

AWB: What about key, values that have been collected over time?

MM: There are use cases for duplicate key checks

LH: Historically, we make duplicate checks when it's syntactic, and this is the first time we're trying to apply duplicate checks to runtime semantics

MM: If something you didn't expect happens once, i'd much prefer an error

YK/RW: That's bad for the web

RW: Map would become a "try/catch" case

... mixed discussion about the precedent for loud or quiet handling

WH: Are there any other constructor that throw when created "incorrectly"?

RW: In non-strict mode, a program can create an object with all duplicate keys and never fail in production

...

MM:

AC: Creation can be what is the least requirement for what it takes to create a map. Taking an arbitrary structure and make a map and it's perfectly good semantics to

LH/MM: Offline conversation about what qualifies for extra defense.

DH: Select cases where there is easy to argue that there few legitimate uses, ok to have occassion sanity tests. In general, JavaScript does not go out of it's way to provide you with defensive mechanisms. It's hard to predict where people are going to get hurt, better to allow them to decide.

WH: Paying for consequences where `with` doesn't protect against collisions.

AWB: Try to apply my model when writing these algorithms, please try to read the algorithms when they are published

~~Consensus/Resolution~~

- Map contructor, accepts optional initializer
- If initializer undefined, creates an empty map
- If initializer defined, invoke the @@iterator to create the entries from.
  - For each entry, check if non-null object, throw if not (If Type(map) is not Object then, throw a TypeError exception.)
  - pull off the 0 and 1 property
  - make 0 a key and 1 value
  - No check for duplicate keys

- Remove the explicit check for an "entries" property, this is implied by the
  check for "@@iterator"

UNRESOLVED

AWB: Will put a note in the spec: "Unresolved: how to handle duplicate keys"

WH: Don't yet have consensus on how to handle duplicates, would like to discuss computed properties

4.3 Array.prototype.values

(Allen Wirfs-Brock, Rick Waldron)

AWB: Ext.js uses a `with(...) {}`

```js
function f(values) {
  with(values) {
    ...
  }
}

f([]);
```

YK: Means that we can't add common names for common objects?

RW: ...Explained that Ext.js fixed the issues, but face a commercial customer update challenge. In the meantime, it continues to break several large scale sites.

AWB: Brendan's workaround (from post on the list)

```js
values() -> @@values();
keys() -> @@keys();
entries() -> @@entries();
```

Importing from a module...

```js
values() -> values([]);
keys() -> keys([]);
entries() -> entries([]);
```

DH: Warming up to the idea of a general purpose protocol, implement your map-like protocol.

WH: But now you need an `import`

EA/AR/DH: Web breaking... but why not break

AR: Meta property, [[withinvisible]]

(Way too much support for this)

DH: This idea is fantastic

EA: Very useful to the DOM, may not need another bit on the object, maybe just a "whitelist".

MM: A very small list of names that "with" doesn't intercept

YK: Could, but semantically almost the same thing

EA: But without the extra bit on all objects

MM: Don't want to require a new bit for all objects.

DH: Need to fully explore the effects on the rest of the language..
- Blacklist for just Array or all objects?

EA: A blacklist that exists, string names, when you enter `with(){}`, the blacklist must be checked.

MM: If the base object is Array, if the name is on the whitelist

EA: Have an instanceof check? This problem happens in the DOM with Node

EA/YK/AR: We can actually use this for several use cases.

EA: The issue needs instanceof to check the prototype chain.

AWB: For objects you want to treat this way.

DH: The middle ground...

@@withinvisible, well known symbol

```js
Array.prototype[ @@withinvisible ]
  = [
    "values",
    "keys",
    "entries"
  ]
```

AVK: Might have a more generic name, can be used with event handlers

DH: `@@unscopable`?

```js
Array.prototype[ @@unscopeable ]
  = [
    "values",
    "keys",
    "entries"
  ]
```

WH/MM/RW/YK: **actual clapping**

... Mixed discussion about performance. General agreement that penalties for using `with` is ok.

AWB: Code may override this, but at their own risk. For example

Consensus/Resolution

- @@unscopeable
- A blacklist, array of strings names that will not resolve to that object

within `with() {}`

DH: This is about the extensible web ;)

3 Approval of the minutes from May 2013 (2013/029)

JN: Need to approve the notes...

Are there are any changes to approve?

(none)

Consensus/Resolution

- Approved.

9 JSON

(Doug Crockford)

DC: Gives background re: JSON WG and presents a proposed JSON standard to be submitted to Ecma. It is an Ecma only standard, not the update of the old IETF RFC.

- Please read tonight for review tomorrow

NL: Benefit from reading the JSON mailing list threads.

YK: Will be painful.

AR: This document seems completely unobjectional

DC: IETF claims abstract formats cannot work

Mixed discussion about consequences.

(Read and be prepared for meeting tomorrow)

4.2 Add fill and copySlice methods to Array.prototype and Typed Arrays
(Allen Wirfs-Brock)

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

AWB: The Chronos group want to add methods

- fill a span of a typed array
- move copy, with care for the overlap

Array.prototype.fill (Informal Spec)

```js
Array.prototype.fill = function fill(value, start=0, end=this.length) {
  /*
    Every element of array from start up to but not including end is assigned value.
    start and end are coerced to Number and truncated to integer values.
    Negative start and end values are converted to positive indices relative to the length of the array:

      if (start < 0) start = this.length-start

    Reference to start and count below assume that conversion has already been applied

    If end <= start no elements are modified.
    If end > this.length and this.length is read-only a Range error is thrown and no elements are modified.
    If end > this.length and this.length is not read-only, this.length is set to end
    Array elements are set sequentially starting with the start index.
    If an element is encountered that cannot be assigned, a TypeError is thrown.
    Element values are assigned using [[Set]]
```

The array is returned as the value of this method
*/
}
```

Examples

```js
aFloatArray.fill(Infinity);  // Fill all elements with Infinity
aFloatArray.fill(-1, 6);     // Fill all elements starting at index 6 with -1
aFloatArray(1.5, 0, 5);      // Fill the first five elements with 1.5
aUint8Array(0xff, -2);       // Place 0xff in the last two elements
[ ].fill("abc", 0, 12)
   .fill("xyz", 12, 24);     // Create a regular array, fill its first dozen
                             // elements with "abc", and its 2nd dozen elements
```

Array.prototype.copySlice (Informal Spec)

```js
Array.prototype.copySlice = function copySlice(target = 0,start = 0, end = this.length ) {
/*
  The sequence of array elements from start index up to but not including end index are copied within
  the array to the span of elements starting at the target index.

   target, start, and end are coerced to Number and truncated to integer values.
   Negative indices are converted to positive indices relative to the length of the array.
   If end <= start no elements are modified.
   If end > this.length a Range error is thrown and no elements are modified.
   If target + (end-start) > this.length and this.length is read-only a Range error is thrown and no elements are modified.
   If target + (end-start) > this.length and this.length is not read-only, this.length is set to target+(end-start).

  The transfers takes into account the possibility that the source and target ranges overlap. Array elements are
  sequentially transferred in a manner appropriate to avoid overlap conflicts. If target <= start a left to right
```
```

transfer is performed. If target>start a right to left transfer is performed.

If a target element is encountered that cannot be assigned, a type error is thrown and no additional elements are modified.

Sparse array "holes" are transferred just like for other array functions.

The array is returned as the value of this method

*/

}

```

Examples

```js

[ 0, 1, 2, 3, 4 ].copySlice(0, 2);

// [ 2, 3, 4, 3, 4 ]

[ 0, 1, 2, 3, 4 ].copySlice(2, 0, 2);

// [ 0, 1, 0, 1, 4 ]

[ 0, 1, 2 ].copySlice(1);

// [ 0, 0, 1, 2 ]

Int8Array.from([ 0, 1, 2 ]).copySlice(1);  // RangeError

Int8Array.from([ 0, 1, 2 ]).copySlice(1, 0, 2);  // Int8Array 0,0,1

Int8Array.from([ 0, 1, 2 ]).copySlice(0, 1, 2);  // Int8Array 1,2,2

```

Moving data within an array, destructively on the calling array

AWB: Possibly rename `copySlice` => `copyWithin`

LH: Should Typed Arrays have the same surface as Array?

DH: Typed arrays better behaving and better performing since they guarantee density. (non-sparse)

- Notes concat as the only array method that expects explicit array-ness
```

RW: Do we have consensus

DH: Brendan had issue with `fill`

AWB: Brendan's issue was the similarity with `copySlice` and had suggested `fillSlice`.

DH: Not sure I understand his objection...

Consensus/Resolution

- Agreement in room
- Would like Brendan's input

4.4 Consider deferring ES6 Refutable Matching.

(Allen Wirfs-Brock)

AWB: In March, we agreed to add refutable pattern matching; began working on adding this to destructuring and realized the work involved would be too great, considering the time frame remaining for ES6.

Propose to defer refutable pattern matching.

(whiteboard)

The current spec would attempt to do a ToObject(10); and would throw:

```js
let { a, b } = 10;
```

What happens when you reference a property that doesn't exist on the object, will throw:

```js
let { a, b } = { a: 10, c: 20 };
```

To avoid throwing:

```js
let { a, b = undefined } = { a: 10, c: 20 };
```

YK: Removing the question mark breaks the consensus.

AVK: Is it hard to spec the "?" on the outside? Allowing only one level?

AWB: It wouldn't be hard, but it creates a weird language issue.

YK/AWB: It's easy to do in the grammar

LH: What was in the spec, solved 80% of the cases, we moved to a solution for 100% and this will set us back to 20%, which isn't acceptable.

AWB: What happens at the parameter list level?

YK: Ah, there is no place to put the out "?"

DH: Agrees... as long as we have a fail-soft, we're ok (YK/LH/RW agree)

YK: We could make the extra sigil mean refutable.

WH:

```js
let [a, b] = "xyz";
```

YK: Why Andreas would have argued strongly against a refutable sigil?

DH: I think this will fail without inclusion of Brendan and Andreas

AWB: Andreas is fine with dropping refutable matching

DH: Are you sure?

Current spec is fail soft

As long as Brendan and Andreas are ok with it, we can fall back to fail soft.

AC: The fail soft is consistent with JS behaviour. If you want something stricter, then the problem should be on the right side, not the left side. Otherwise you need to introduce an operator for the left.

AWB: (reads back conversation from Andreas)

DH/YK: He doesn't seem to say anything about returning to fail soft.

LH: I think we've exhausted the conversation

WH: If we don't do it now, the behavior of refutable and regular rules will be inconsistent in the future; i.e., a trivial refutable rule that doesn't actually make use of the refutable features will behave inconsistently with a textually identical nonrefutable rule.

YK: But you'll be able to opt-in to the full set of "refutables"

WH: I think it's uglifying the future.

YK/LH: It is.

DH: There is precendence in Mozilla's destructuring, that doesnt have refutable matching.

LH: If we added the bang which is the strict mode for this and adds the bang in front, opts in.

AWB: The next part...

WH: The string example:

```js
let [a, b] = "xyz";
```

Should there be implicit ToObject on the right side?

YK: We agreed `new String()` solves the problem, if that's what you actually wanted to do.

Consensus/Resolution

- No implicit `ToObject()` on the right side (eg. the string will throw)

x.x Review of Proposed Features

(Luke Hoban)

Function toString

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

MM: The one issue about Function toString, discovered since the strawman was written:

Since eval()uating a function declaration or function expression defaults to non-strict, a strict function must present the source code of its body as beginning with a “use strict” directive, even if the original function inherited its strictness from its context. This is the one case where the original local source code of the function is inadequate to satisfy this spec.

YK: Doesn't account for super, either

Discussion about identifiers captured from lexical environments.

Was the lexical environment strict?

Consensus/Resolution

Change wiki

strictness included in notion of lexical context. Thus

- always adequte for toString to preserve the original source
- result behaviour equivalence does not require injecting "use strict"

Function name property

(Allen Wirfs-Brock)

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

AWB: The spec doesn't have mechanisms for capturing the name based on the syntactic context.

LH:

```js
let f = function() {}
```

...Needs to know "f".

AWB: It's not an insignificant amount of work.

...Conversation Moves towards prioritization.

Modules

LH: Need to know that modules are going to be spec'ed asap.

DH: This is my next item to work on

AWB: Modules are the next most important and significant to address in the spec.

High priority

Standard Modules

DH: Back off on standard modules for ES6, very few things.

Standard Modules:

- Math

- Reflect

YK: All of the built-ins.

RW: If I want to avoid using a tainted built-in, `import { Array } from "builtins";`

DH: What does this directly solve?

YK/RW: When you want to get a fresh, untainted _____.

AWB: Who will write out the standard modules?

EF/YK/RW can work on this

Mixed discussion about feature dependency.

DH: Luke and I can craft a dependency graph offline.

Binary Data

On track (wiki near complete)

High priority

Regexp Updates

- http://wiki.ecmascript.org/doku.php?id=harmony:regexp_y_flag
- http://wiki.ecmascript.org/doku.php?id=harmony:regexp_match_web_reality
- http://wiki.ecmascript.org/doku.php?id=harmony:regexp_look-behind_support
- http://wiki.ecmascript.org/doku.php?id=harmony:unicode_supplementary_characters

Low priority

DH: Optimistic that we can get Modules and Binary Data to green (in the spreadsheet)

Proper Tail Calls

DH: Just need to identify the tail position and finish the spec.

AWB: It's just a time consuming project. Property access in tail position? Not tail call.

DH: Safely:

- function call
- method call

Consensus/Resolution

- Plenty of work left.

4.7 Math

(Dave Herman)

DH: Introduces need for 64bit float => 32bit float and projecting back into 64bit float. If we had a way to coerce into 32bit

- Can be simulated with TypedArray (put in a value, coerced, pull out)
- To have a toFloat32

EA: Does JSIDL Need this?

YK: Not that I know of

MM: The real number it designates is a number that is designatable as 64bit

DH: (confirms) If you have a coercion, the implementation could do a 32bit operation

WH: Note that for this to work, you must religiously coerce the result of every suboperation to float32. You can't combine operators such as adding three numbers together.

Given

x, y, z are all float32 values stored as regular ECMAScript doubles, the expressions

x+y+z

float32(x+y+z)

float32(float32(x+y)+z)

can all produce different results. Here's an example:

x = 1;

y = 1e-10;

z = -1;

Computing x+y+z using float32 arithmetic would result in 0. Computing float32(x+y+z) would not.

On the other hand, there is a very useful property that holds between float32 and float64 (but not between two numeric types in general such as float64 and float80), which is that, for the particular case of float32 and float64, DOUBLE ROUNDING is ok:

Given x, y are float32 values, the following identity holds, where +_32 is the ieee single-precision addition and +_64 is the ieee double-precision addition:

float32(x +_64 y) === x +_32 y

And similarly for -, *, /, and sqrt.

Note that this does not hold in general for arbitrary type pairs.

Here's an example of how DOUBLE ROUNDING can fail for other type pairs. Suppose that we're working in decimal (the issues are the same, I'm just using decimal for presentation reasons), and we compute a sum using arithmetic that has four decimal places and then round it to store it into a type that has two decimal places.

Let's say that the sum x+y is mathematically equal to 2.49495.

2.49495 (mathematically exact sum)

Then we get:

2.4950 (properly rounded result of invoking + on the wider 4-decimal place type)

2.50 (rounded again by coercion to narrower 2-decimal place type)

Yet, if we had invoked + on the narrower 2-decimal place type, we'd instead have gotten the result:

2.49 (mathematically exact sum rounded to narrower 2-decimal place type)

AWB: Is the proposal to expose a toFloat32?

DH: Yes and the Math object seems like the obvious place

RH: Also, toFloat16

DH: Long term, the solution will be value objects, but in the near term, this will have benefit much more quickly

WH: Found evidence that the optimizations are safe as long as the wider type is at least double the width of the narrower type plus two more bits: http://docs.oracle.com/cd/E19957-01/806-3568/ncg_goldberg.html . This is the case for the float32/float64 pair (they're 24 and 53 bits wide respectively), but not in general.

Consensus/Resolution

- Math.toFloat32()

More discussion about where (Math, Number.prototype)

4.8 Stable Array.prototype.sort

(Norbert Lindenberg)

https://mail.mozilla.org/pipermail/es-discuss/2013-June/thread.html#31276

NL: Does anyone know of performance issues that prevent going to stable sort.

DH: Tread lightly, no one wants regression

EA: Libraries implement stable sort today because they need it.

YK: D3

MM: If the answer is that performance is negligible, then we should mandate stable sort. Otherwise, we don't. We need to answer the question first.

Consensus/Resolution

- Deferred

4.9 Time zones 1: Bugs or spec issues?

(Norbert Lindenberg)

https://mail.mozilla.org/pipermail/es-discuss/2013-July/032087.html

Discussion around the semantics of Date

AVK/MM: Be able to create a date instance with a timezone, other then the current timezone

MM: ES5 implies a live communication channel into the Date instance

AWB: It's part of the algorithms

MM: We could say that we're going to stand on the ES5 spec.

Consensus/Resolution

- Deferred

4.10 Time zones 2: Time zone as property

(Norbert Lindenberg)

https://mail.mozilla.org/pipermail/es-discuss/2013-July/032080.html

NL: Dean Landolt proposed a property on Date.prototype for the timezone, that all the functions look for, born with the default timezone, but can be changed.

MM: Should be static, like Date.now()

RW: Otherwise there would be Date objects with different timezones.

Consensus/Resolution

- Deferred

Date/Timezone

Proposal 1

AVK: Having Date objects that have timezone as internal data instead of system data.

Proposal 2

NL: Pass time zone information separate from Date (as a parameter to methods)

Consensus/Resolution

- Write a strawman for ES7 (either)

 # July 24 Meeting Notes

John Neumann (JN), Luke Hoban (LH), Rick Hudson (RH), Allen Wirfs-Brock (AWB), Yehuda Katz (YK), Anne van Kesteren (AVK), Jeff Morrison (JM), Sebastian Markbage (SM), Paul Leathers (PL), Avik Chaudhuri (AC), Ian Halliday (IH), Alex Russell (AR), Dave Herman (DH), Istvan Sebestyen (IS), Mark Miller (MM), Norbert Lindenberg (NL), Erik Arvidsson (EA), Waldemar Horwat (WH), Eric Ferraiuolo (EF), Matt Sweeney (MS), Doug Crockford (DC), Rick Waldron (RW), Rafael Weinstein (RWS), Dmitry Lomov (DL), Brendan Eich (BE), Brian Terlson (BT)

4.6 Binary Data Update

(Dave Herman & Dmitry Lomov)

DH: (Introduces Binary Data) A mechanism for creating objects that guarantee a shape (Struct)

Use case that has become less important, I/O

Dmitry has given use cases where we still want control over endianness

MM: A little surprised by this direction (not objection)

If you have something like a class, do you an imagine something like an array buffer, per instance?

DH: it's still possible to overlay structs over larger sections

MM: if you want the instances to be gc'able...?

DH: they'd have to have separate backing storage, but those could be GC'd like normal

There's more work we've been doing with Rick and his group at Mozilla on parallelism patterns by describing the shape the data you're operating on. You still want the ability to do things like parallel iteration over arbitrary JS values, which means things like "object" and "any".

Those are high-fidelity drop-in replacements for JS arrays, but with no holes.

DH: once you introduce pointers, you have to bifurcate the design a bit; opaqe vs. non-opaque structs. Don't want to go into details, but will put that on the wiki.

MM: You can use structs to model more classical class object model where all the data is on the instance. (structural types)

YK: Do you imagine people using this in regular JS?

DH: Yes, but if they're writing regular JS, they'll profile and find that they want to use these sorts of structs in hot places.

```js
function ThingBackingStore() {...}

function Thing() {
  return new ThingBackingStore();
}
```

... Becomes something like...

```js
var ThingBackingStore = StructType({

- stuff: new ArrayType(object)

});

function Thing() {

- var selfie = new ThingBackingStore();
-
- selfie.stuff = ....;
- return selfie;

}
```

WH: What kind of fields can be had?

DH: Type descriptor algebra, set of combinators. (atomic types) Uint8, Uint8Clamped, Uint16, Uint32, float32, float64, ObjectRef, StringRef, "any"

- new ArrayType(T), new ArrayType(T,)

WH: no uint64?

DH: no.

BE: I'm proposing it separately

WH: what's the difference between ObjectRef and any?

DH: ObjectRef can be a object or null, while any can be any valid JS value.

DH: SymbolRef is mostly an optimization over any.

EA: What about SymbolRef?

DH: Symbols should be objects. We'll have that discussion later.

AWB: Why isn't there a Uint16Clamped?

DH: That was just needed by canvas.

MM: Only Uint8Clamped?

DH: Yes, compatibility with canvas' ImageData.data.

AR: Y U NO CLAMPED EVERYWHERE!?

```js
var Point = Struct({
  x: uint32,
  y: unit32
});
```

var p = new Point({ x: 1, y: 12 });

p.buffer;

p.byteLength;
```

WH: Can you replace the `buffer`?

DH: No, {[[writable]]: false, [[configurable]]: false, [[enumerable]]: false}

WH: Can you write to the buffer?

DH: Yes

DH: V8 team also wants reflection so that they can check the type and the field offsets.

WH: what if there's an object field in the buffer somewhere?

DH: let me outline what I'm doing, mayble that'll clear it up

if I have:

```js
var q = new Thing();
q.buffer == null; // true
q.byteLength == undefined; // true
```

One of the things we've talked about is being able to censor access to the buffer at various points.

Lets say you want some computation is working on a sub-view of a buffer, we need to be able to cordon off access between these parallel things.
```

We can imagine putting/removing things from the "opaque" state. p.buffer is originally some buffer object, but then I'd be able to set the opaque state. Having an access to the buffer will let me re-set that constraint later, ocap style.

WH: I'd like the type itself to denote if it's buffer-backed or not.

DH: once I can lock down the instances, that doesn't matter so much.

WH: it's one more thing I have to write out at creation time

WH: I'm looking at this as a backing store for classes

DH: my feeling is that it's more type and design consistent to not have this depend on having the property

AR: how do I know it's in the opaque state?

DH: I'm trying to avoid competing with the namespace for property names, so I haven't solved that yet

LH: I think that's weird. I don't think anything in the current draft indicates the per-instance-of struct type that goes along with the type.

BE: that's been diagramed.

DH: We should stratify buffer and byteLength.

WH: are there any other special names besides "buffer" and "byteLength" that you're thinking of adding?

DH: the other major ones are methods that do getting and setting for things like multi-hop getting that avoids intermediate alloc.

```js
var S1 = { foo:     , bar:     }
var S2 = { ... s1: s1, ... }
var x = new S2();
x.s1.foo
x.get("s1", "foo")
```

// lots of discussion about structs, prototype chains, and confusion about how long this has been agreed

AR: don't worry about so much about this object/buffer split; it's showing up because DH is enamoured of return overloading

DH: the main feedback I'm hearing is that they don't want default stuff in the prototype chain

// AR: what's the controversy?

DH: getOwnPropertyNames() matters and should behavle as much as possible like a normal JS object that it's emulating. If it has ergonimic issues that we discover later, so be it, but that's the general approach.

MM: if we find a problem, I'm happy to deal with it -- but I want to take it seriously and not as a detail.

AWB: I think this is extremely promising and there are lots of ways we can leverage this. But I don't see how we can get this into the ES6 spec. So many details. So many interactions.

WH: so there won't be binary data in the spec?

AWB: no, there are TypedArrays.

We might be able to do this as a self-contained thing after ES6.

Speaking editorially, we probably need to accept we won't get this done for Dec.

BE: if we're going to discuss if this is ES6 or ES7, we can have that discussion, but that's not how this started.

LH: this began as a technical update.

AWB: Need to explore the interaction with classes, @@create, etc. to me it's hard to separate some of the tech discussions from the schedule discussions.

DH: Objection to the exxageration of "newness" of this conversation.

BE: (Right, we've seen this)

MM: First time I've understood, so first time I'm reacting. Where the buffer is used as an instance itself

AR: No, not the buffer.

MM: What are these called?

DH: Structs, always. These examples are just to show what you can do with them

MM: the idea that these things inherit from the struct-making function's prototype...was that always there?

BE: I'd like to intervene. We're mixing things up still. Can I be blunt? The thing that set mark off was the "I don't have a strong opinion about that". If it's too late for ES6, it's ES7. We need details and there's sentiment in favor of stratification.

...Champions should have a strong opinion about these aspects

These should look like objects as much as possible

DH: Fair, but these weren't on my agenda to talk about. My dismissal was out of unpreparedness for the question.

WH: I would much prefer for this to be in ES6. TypedArrays without Struct seems half-baked

DH: YEs, agree, but the specing of modules trumps the specing of binary data.

YK: IIRC, you've said that Struct is polyfillable

WH: What does "polyfillable" mean in this case?

YK: Does not need to be in the spec for user code to use.

DH: Polyfillable in the sense of perfect semantics, but not the level of performance.

DH: yes, but this enables a class of optimiztions

LH: it's not strictly enabling the perf optimizations...right?

DH: no, you can't infer that there's an invariant that it won't change.

WH: You cannot know that a property is uint8 for example. It's hard to efficiently allocate just a byte for it if you can't tell that no one will try to write a larger value to it.

DH: I want to make another point which I had to learn along the way: I came to understand that there are 2 large, distinct classes of use-case:

1.) efficient representations of memory _within_ pure computation

2.) serialization and de-serialization

The design constraints are very different. For I/O, you want a LOT of control. And JS is expressive for those things. Doing it in a built-in library is a waste of TC39's time. Where TC39 can add something programmers can't is the memory-internal set of use-cases. Polyfilling only gets you a little bit of benefit there, but if it's built in, the optimizations can happen.

BE: the typed-array use-cases motivated the structs work for efficient overlays.

WH: but why would you polyfill something that doesn't perform or only has TypedArray?

DH: it's better to put it in TC39 even if we don't get full-structs. I'm gonna do the best I can to get it into ES6, but it's important. No reason not to have TypedArrays if we can't get it done, though. Better for us to control the sharp corners than to leave it in Khronos.

DH: I've focused on the in-memory patterns and others probably haven't thought about these as hard. But i haven't even gotten to the stuff that the V8 team brought up: for webgl, since you can have structs and they can be overlayed with buffers, what's the alignment story? TypedArrays haven't had unaligned access. The safest thing to do here is to provide a universal padding scheme that's portable.

DH: gl programmers want the ability to use structs on the JS side, but on the GL side, want the ability to demonstrate exact offsets and optimize for specific systems. They want aligned fields, but they want complete control.

WH: do they want holes? if we use C-padding, they can re-order fields explicitly to get the least total size.

DH: I'm unclear what's specified and not in C

WH: The C language spec states only that the fields must be in ascending address order. However, in order to be able to link code and have it interoperate, every ABI must spec it. As far as I know, they all specify it as "naturally aligned" using greedy allocation: for each field pick the next position that has the correct alignment.

WH: This is sufficient control to allow users to minimize padding to the minimum possible if they want to. I sometimes do this to my C++ classes. All you need to do is to reorder your fields to avoid doing things such as interleaving bytes and float64's.

DH: the proposal from dslomov's group is that the struct ctor can specify offsets:

```js
new ST([
```

```
    [0, uint32, "le"],
    ...
]);
```

MM: dave, can you state the alignment rule?

DH: every leaf type in a struct has natural alignment

WH: this is like most C systems, if you make an array of structs, the elements cannot be misaligned. struct sizes are rounded up, padding them up to their largest alignment to avoid possibly putting them in a misaligned context and breaking the alignment. A struct {x: double, y:uint8} has size 16 instead of 9 so that alignment of the double field works if you create an array of such structs. On the other hand, the struct {x1:uint32, x2:uint32, y:uint8} would have size 12 instead of 9 because it only needs 4-byte alignment.

MM: I heard there was soemthing about a reflective operation for obtaining the layout?

DH: it's work I still ahve to do.

MM: what object do you get back?

DH: TBD. Tempted to say it looks like the thing you passed in originally.

WH: I dont' see bool...deliberate?

DH: yep.

WH: let me go on record as requesting a bool field.

?: Can't you use uint8 for bool?

?: Use object type for bool.

?: What about other types such as different kinds of objects?

WH: bool is different. I am only asking for bool here. A bool stored in a single byte, so using 8 bytes for an object reference for it would be a substantial cost. A uint8 is not a suitable replacement because one of the major use cases for structs is to use them as an efficient backing store for objects, and getting 0 and 1 instead of false and true would be a really annoying impedance mismatch.

MM: how many bits per bool does an array of bool's contain?

BE: needs to be exactly 1 byte

WH: Don't want to repeat the C++ vector<bool> mistake that tried to pack it to 1 bit/bool and ended up breaking lots of things. The C++ committtee now regrets having done this.

9 JSON (Continued)

(Doug Crockford)

DC: Explaining updates made since yesterday

- - Reordered Appendix
- - Moved ECMAScript mention

The last big change is that Rick suggested removing the security section (10) and I think I agree

AWB: Those don't apply to a spec at this level

DC: Agree, I don't think it belongs

AWB: what's the start symbol of the grammar?

DC: unspecified.

RW: my suggestion was to re-order the appendix to ensure that Value comes first and that anything that is one starts the gammar

AWB: I think it should specify a start symbol.

DC: some uses of JSON won't want a bool at the top level

AWB: ..or array, or object. All this describes is the universally valid syntax, and that has to start somewhere.

DC: I don't think it does

AWB: then I don't know how to parse/validate.

YC: I think you do need a root symbol. I've had this exact issue.

AR: C++ experience backs that up. Lots of incompat.

RW: can't we just say that anything that's a value can begin the production?

MM: we have a historical conflict. As I read the ES spec vs. this, we see value vs. object/array.

DC: it should be value, then. How should we specify that?

AWB: just say that any input that can be parsed that uses that start symbol is valid JSON text

MM: we should decide if we want to be compatible with reality or the RFC. Given that we decided on value in ES, we have incompat. Shifting to value as the start symbol would be up-compat with reality and current practice.

AWB: doesn't say what a "reader" does with this grammar, so can allow/disallow however it wants. There's no universal JSON reader.

MM: JS might accept those things but not give you access to the literal content of the values

NL: this should be based on Unicode code points. We don't know how to convert between things if we don't.

DC: best practice might be to use Unicode, but this grammar doesn't need to.

NL: I differ. We care about computing in this group.

DC: if someone wants to use a 6-bit code to send JSON from a satellite, JSON is agnostic.

AWB/NL: The title says "interchange format", should probably just be a "grammar"

NL: Without reference to Unicode code points, we can't decide which characters Unicode escape sequences are equivalent to, e.g.

\uxxxx ===? ö

\uxxxx\uxxxx ===? ö

AVK/NL: current best practice is to base document formats on Unicode, e.g. HTML

WH: The description of where whitespace can go is broken. Currently it's described as only being allowed before or after one of the punctuation symbols. That means that "{3}␍" parses but "3␍" does not.

MM: Crock, what purpose does the document serve?

DC: ... i forgot what he said..It creates the stabil base standard on JSON Grammar and Interchange Format, that the different JSON applications will follow.

MM: Comparison to RFC

AVK: Refers to ECMAScript's JSON, not the RFC

AWB: Wants other standards to make references to this normative standard and not some other.

...To avoid the inclusion of anything that isn't the grammar, eg. mime type

MM: The danger of the IETF is trying to go beyond the RFC.

AWB: This document to have the grammar over an abstract alphabet, normative Unicode mapping to that abstract alphabet.

MM: Ok, a two level grammar. Still a refactoring of the RFC and that's ok.

AVK: You don't have to refactor, just define the grammar as a code point stream

note differences between code points and character encodings

MM: Why retreat from the RFC?

DC: There is contraversy in that it tolerates unmatched surrogate pairs. Want a standard that avoid that contraversy

NL: You can avoid it by talking about code points. The transfer format can outlaw unpaired surrogate, e.g. utf-8.

AVK: Right, utf-8 does not allow lone surrogates, it can only encode Unicode scalar values.

[After the fact note, we neglected escapes here...]

BE: Try to avoid duplicating efforts

DC: This is meant to be a guard against other efforts going out of bounds.

BE/AWB: Need to be addressed:

- - Goal symbol
- - "e" production
- - leading/trailing whitespace
- - character sets
-

NL/AVK: Without describing the character encoding

Discussion about character => code point

AVK: Need to define a representation you're using

AWB: Define the alphabet

AVK: Use the Unicode alphabet, without talking about utf-8, etc.

AWB: DC might be rejecting the discussion of encoding.

DH: The purpose of code point is to avoid talking about encoding.

AR: Why is this so important?

AVK: In order to describe the grammar, you need to define an abstract alphabet, Unicode is sufficient.

MM: This differs the RFC is code units instead of code points

DC: I will add a sentense in the section about JSON text, that it's a sequence of code points

AWB: Unicode code points

Include an informative note that it doesn't imply a specific character encoding

Character sets:

"json text is a sequence of Unicode code points that conforms to this grammar"

Start symbol:

value

MM: Move to declare consensus?

AWB/WH: Doug needs to finish an editorial pass

Consensus/Resolution

-
- - Pending the remaining edits, to be discussed for approval tomorrow.
-
-

Test 262 Update
(Brian Terlson)
Test262-ES6.pdf

Talk about moving test262 to github.

BT: What do we need to make that happen?

DH: Use CLA for pull requests.

DH: Lets not solve this now. Lets stick to the current process that only we can only approve PR from members of TC39/ECMA

Apporoved to move the source code to GitHub, keeping all the current process for approval of tests.

IS: We will need to have a way to download the tests from ECMA.

IS/AWB: Needs to be an update to the tech report. Describing what the different packages are good for.

BT: In due course backporting will be hard, but in the ES6 timeframe it should be okay.

MM: Are we anticipating backwards incompatible changes to the testing harness?

BT: There migth be some, we can probably avoid it.

MM: good, good.

BT: In ES6 a number of tests moved location. We'll create a map and move them programmatically.

AWB: I'm planning on making the reorganization within the next few weeks.

BT: We'll annotate the tests with previous and new locations.

BT: Norbert's proposal is to rename the folders to use English names and remove the multitude of subfolders.

[Discussed moved to organization of the spec.]

AWB: There could be an arbitrary number of chapters in the spec. It's somewhat convenient to be able to say "chapter 15".

BE: Core shouldn't depend on 15.

AWB: Trying to get to that situation.

MM: I don't object to a part 1 / part 2 organization, but I also don't see the point.

MM: Back to tests, I want test directories to match spec chapters.

BT: Agreed.

BT: Contributors: http://wiki.ecmascript.org/doku.php?id=test262:coreteam Need to expand.

DH: We could use github pages and make test262.ecmascript.org point to that.

DH: BT, you set up a mirror for the site. I will do the DNS switch, call me, maybe.

BT: Given the new algorithms, designating algorithm step is not always feasible. Proposal is to require identifying the section, and maybe the algorithm step.

BT: Ensuring full coverage becomes harder.

[Much discussion on algorithm step designation not minuted.]

YK: This should probably be dealt with at the champion level.

BT: Open issues: How to create cross-host-compatible collateral for realms and scripts?

MM: ??

MM: We might only be testing the slow path of JavaScript engines. Testing things in a loop will help.

[Insert poison attack.]

Consensus/Resolution

- - move the repo and test262 web presence to github.com/tc39

5.2 Can computed properties name in object literals produce string prop names? Duplicates?
(Allen Wirfs-Brock)

http://esdiscuss.org/topic/on-ie-proto-test-cases#content-5

AWB: Latest revision include computed properties

```js
```

{ [x]: 1, [y]: 2 }
```

1. Can x evaluate to a string or a symbol? The concern is that people hope to determine the shape of objects by looking at them (but engines also want to determine the shape statically)

2. Duplicates?

EA: I thought we allowed any value as a computed property

DH: Yes, and converted with ToPropertyKey

Discussion re: duplicate computed property keys

DH: Comes down to "how likely are there going to be consequences

WH: I draw a distinction between definition and assignment and I view this example as definition.

EA: If you call defineProperty twice with the same property you do not get an exception. We should have the same semantics for define property in an object literal (and class).

MM: Often enough, if you get a

MM: The importance of throwing an error at the point

YK:

MM: If these are symbols, the programer did not intend to override a previous property.

DH: JavaScript practioners consistently tell us they prefer fail-soft with opt-in. Having said that, we are deploying strict mode more widely.

BE: We don't know, people will write top-level functions.

DH: It's the direction we're heading in.
```

WH: Main motivation here is consistency. In strict mode we intentionally made having duplicate properties such as {a: 3, a: 4} be an error. That rule is simpler and easier to remember than a rule that constructs another exception such as it being an error unless one of the properties is specified using [].

[debate]

WH: This has nothing to do with allegations of trying to make the language less dynamic. I want the simplest, easiest to remember rules. Had duplicate textual properties been allowed in strict mode, I'd be campaigning to also allow them if they're specified using [].

...

Discussion re: strict mode checking vs non-strict mode checking

There was an assumption that computed properties would disallow duplicates in strict mode (throw)

MM: The failure will frustrate programmers, but the lack of a failure will frustrate programmers. You say some prefer fail soft, I know of some that want fail fast.

DH: Static checks are easy to reason about and easy to justify. Dynamic checks are harder to justify and lead to unpredictable results.

AWB: What happens when x is undefined?

BE: Let's stick to duplicates

It's unlikely to want duplicates

The most common case is that you write two lines like the above and that's what you want

We should not creep into B&D language with runtime errors

Quick poll:

- error, 7
- no error, 6
- abstain, 9

DH: I want to hear a response to Mark's point about being bitten by fail-soft

AR: In Google Wave, you'd get notices that the application "crashed"... the rest of the web doesn't behave that way. Fail soft is seen be some as "wrong" and others as "going along". I put more favor towards getting "going along".

AVK: In specifying web APIs, authors regularly requesting less errors. Back in 2000's, WebKit had asked for less errors...

YK: Identifies committee pathology

EA: Waldemar wanted symmetry with the iterable parameter to the Map constructor regarding duplicate keys.

LH: Do not agree that we need symmetry here.

BE: This is syntax, Map parameter is not symmetry.

Recapping Waldemar's position about consistency with duplicate string keys in Objects in strict mode.

Break.

Post break discussion, after the points made about @@iterator and @@create

Consensus/Resolution

- - Strings allowed
- - Strict Mode: Duplicates are an Error
- - Non-Strict Mode: No error
-
-
-
-
-

5.3 Special syntax for `__proto__` in an object literal
(Allen Wirfs-Brock)

AWB:

```js
{"__proto__": foo}
```

MM: Uncomfortable with either semantics for this issues

YK: This is new syntax that looks like old syntax

...quotation marks and no quotation marks should do the same thing.

DH: But then there is...

```js
{ ["__proto__"]: foo}
```

YK: So for dict, we want "__proto__" as a regular property?

MM: Yes

DH: Allows...?

MM: I'm ok with either decision, because equally uncomfortable

DH: What happens today?

```js
{__proto__: foo }
{"__proto__": foo}
```

Same.

MM: Then the quoted case should remain as is.

BE: The

MM: computed form:

- no compat hazard
- new special case that requires checks
  - - too much overhead
  - - syntactically unclear re: outcome.
  -
  -

Consensus/Resolution

-
- - __proto__: magic
- - "__proto__": magic
- - ["__proto__"]: no magic, just a string
- - ["__" + "proto__"]: no magic, just a string
-

5.4 Are TypedArray insances born non-extensible?

LH: Pretty sure all are extensible.

DH: I'd like them to be not extensible. There's performance benefits. No reason for expandos. Put them on another object. Gecko is not extensible.

WH: Second

DL: Current implementation allows them to be extensible

AR: does anyone think they _should_ be extensible?

crickets

tumble weed

Consensus/Resolution

- - TypedArray instances are not extensible.

5.5 concat and typed arrays
(Allen Wirfs-Brock)

AWB: Anything that's an exotic array (ie. Array.isArray(a) === true), will "spread"

...Gets weird with Typed Arrays

Proposing:

Specifically for concat, we give Typed Arrays a new special concat that auto spreads

MM: The only sensible approach has to be type compatible

DH: concat is badly designed

...If we have to do a new method, we shouldn't try to simulate bad behaviour

Conversation leans toward:

- - Two new methods that represent the two behaviours of concat, but as single operations each
- - Names need to be added to @@unscopeable

LH: Does not like a new method that does 90% of the same as another existing method

DC: can't we just lean on the new syntax? Use "..." for this:

```js
new Uint8Array([...array1, ...array2]);
```

AWB: if this isn't heavily optimized, this is going to create a large intermediate object

DH: this might create issues until people get "..." and engines want to stack-allocate args

AWB: not an arg

// agreement

Mixed discussion about general purpose apis that are lacking in the language.

BE: how should we settle those issues?

AR: with science. Look around the world and see what's most common; then put those things in the std lib

Consensus/Resolution

-
- - No concat on Typed Arrays
-
-
-

5.11 ToUint32(length) and Array.prototype methods
(Allen Wirfs-Brock)

AWB:

```js
let len = Get(o, "length");
let len = ToUint32(len);
```

If "o" is a Typed Array, this above will

Can duplicate all the algorithms to handle Type Arrays

Don't want to do that

What is the impact of changing to:

```js
let len = Get(o, "length");
let len = ToInteger(len);
```

53 bits

DH: Wrapping around?

AWB: Getting there, not done with points

For regular arrays, already there, constrained to uint 32

Only talking about Get, not Put

```js
[].forEach.call({ 1: 1, [Math.pow(2, 32) - 2 ]: "pow", length: Math.pow(2, 32) -1 }, e => ...)
```

Visits only the first (nothing after, because holes)

```js
[].forEach.call({ 1: 1, [Math.pow(2, 32)]: "pow", length: Math.pow(2, 32) 2 }, e => ...)
```

Visits the first and last (nothing in between, because holes)

```js
[].forEach.call({ 1: 1, [Math.pow(2, 32)]: "pow", length: Math.pow(2, 32) -1 }, e => ...)
```

```

Readers: That would be really slow and insane. Don't do that.

Propose ToLength()

- - return length >= 0 ? Truncate to 2^53-1 : 0;
-

WH: Note that this only applies to the length property. In other places where the Array method take positions or lengths as parameters, they already call ToInteger, not ToUint32.

MM: Are there places where the operational meaning of -1 is changing?

BE: There's hope to make this change, sure that the only things that will be broken will be tests.

WH: [[Put]] still has a restriction?

AWB: Yes

Typed Arrays are not exotic arrays

BE: Typed Arrays are going to use 64 bit unsigned for the length, gonna be nice.

## 5.14 keys(), entries() return numbers for array index properties

(Allen Wirfs-Brock)

https://bugs.ecmascript.org/show_bug.cgi?id=1560

AWB: Arv filed a bug re: Array iterators, the keys() iterator (as well as entries()). Currently it is speced to use strings for the indexes. It would be better to use numbers.

RW: Agree, the string property would be unexpected.
```

General agreement

Consensus/Resolution

-
- - keys(), entries() use numbers for array index properties

5.7 Does Object.freeze need an extensibility hook?
(Allen Wirfs-Brock)

AWB:

```js
let y = Object.freeze([1, 2, 3]);
let x = Object.freeze(new Uint8Array([1, 2, 3]));
```

The second does not really freeze the underlying buffer. So, the following does not work as the array case:

```js
x[1] = 1;
```

Discussion about the operational understanding of Object.freeze, clarifications by Mark Miller.

Lengthy discussion about Object.freeze

AWB: Proposes the freeze hook

@@freeze

Called before the freeze occurs

DH: I'm pro mops, but you have to be careful with thems.

LH: Don't think it's wrong that the second item (from above) doesn't freeze the data, that's not what Object.freeze does.

WH: Object.freeze is part of the API and should match what reading and writing properties does (at least for "own" properties). Having Object.freeze not freeze the data is bad design

LH: Object.freeze is bad design, Typed Arrays are bad design, we're stuck with them, so what should they do.

DH: (agrees)

MM: Object.freeze is named badly. Other than that, there's nothing bad about its design. Its purpose is to make an object's API tamper proof

LH: (agrees)

AWB: Method that creates frozen data?

```js
Object.freeze(new Date())
```

Consensus/Resolution

- - No @@freeze MOP hook.
-

5.4 Typed Array MOP behaviours (Continued)

AWB: Talking about the descriptors of the properties of TypedArrays

```js
{value: ?, writable: true, enumerable: true, configurable: false}
```

MM: Doing a defineProperty on a single one, should throw. Doing a freeze on the whole thing, is allowed.

BE: Right now we throw from freeze

MM: Making these appear like properties in the MOP, throw on individual properties changes.

```js
var b = new Uint8Array([1, 2, 3]);

Object.defineProperty(b, "1", {});
// Throws!
```

BE: This makes sense.

Consensus/Resolution

-
- - Object.defineProperty on Typed Array will throw
- - Object.freeze on Typed Array will throw
-
-

 # July 25 Meeting Notes

John Neumann (JN), Luke Hoban (LH), Rick Hudson (RH), Allen Wirfs-Brock (AWB), Yehuda Katz (YK), Anne van Kesteren (AVK), Jeff Morrison (JM), Sebastian Markbage (SM), Alex Russell (AR), Istvan Sebestyen (IS), Mark Miller (MM), Norbert Lindenberg (NL), Erik Arvidsson (EA), Waldemar Horwat (WH), Eric Ferraiuolo (EF), Matt Sweeney (MS), Doug Crockford (DC), Rick Waldron (RW), Rafael Weinstein (RWS), Dmitry Lomov (DL), Brendan Eich (BE), Ian Halliday (IH), Paul Leathers (PL),

5.6 Can let/const/class/function* in non-strict code bind "eval" and "arguments"

(Allen Wirfs-Brock)

AWB: Currently, only var and function have any rules: non-strict is not

YK: Reduce the refactoring hazards

MM: What happens in arrows?

EA: Formal params follow the strict rules (no duplicates, no param named arguments etc), but the bodies are not strict.

RW/BE: Confirm

AWB: If someone writes...

```js
class eval {}
```

And later moves this to a module...

```js
module "foo" {
  class eval {}
}
```

This will blow up

RW: But the same issue exists if:

```js
function eval() {}
```

And later moves this to a module...

```js
module "foo" {
  function eval() {}
}
```

MM, WH: We need to make sure that whatever rule we decide on, is the simplest and easiest to remember

BE: Recall the issue of micro-modes

BE: Based on the decision make Arrows non-strict, the same reasoning applies to params

EA: Strict formal parameters are an early error, strict function body have different runtime semantics so those are a refactorig hazard.

AWB: The spec draft uses StrictFormalParameter for ArrowFunction and MethodDefinition.

YK: Easy to get sanity, by opting into modules and classes

RW: The January notes include rationale regarding the boundary of module and class, but not arrow, there is no note about arrow params being implicitly strict mode

AWB: method names in sloppy mode (object literals) do not allow duplicat names.

YK: Seems OK.

... Code may exist that has methods called "eval" or duplicate params named "_"

MM:

- - eval & arguments

- - duplicate arrow & method params
- - duplicate non-data names in object literals
-

LH: Agrees that these rules should be applied where code opts-in, not by layered addition of language features

MM: Agrees with LH, in terms of the memory burden (developer end). This wont be clear to anyone but us.

-
- - If you're in non-strict, it should act non-strictly
-

BE/RW: Yes

Various: explored the consequences of allowing duplicate method parameters even in new-style parameter lists when in non-strict mode. That would be the simplest rule, but it would cause too many edge cases for duplicate parameter names in destructuring, rest parameters, etc., so we all agreed not to pursue that approach.

AWB: The rule that we agreed on, in that past is that when new syntax forms are involved.

- - Depends on form of the parameter list

MM: We need to lower the memory burden

EA: This is going to make it greater

MM: Defending exception for new forms of parameter list.

AWB: More complex set of rules if you allow multiple names in simple parameter lists.

- - Duplicate param names not allowed, except for function definitions (things declared with function) with simple parameter lists
-

MM: That's more complex

Consensus/Resolution

- General Rule

- - Non-strict code operates in consistently non-strict manner (This covers the let/const/function* cases)
- - Exception:
- - Only allow duplicate parameter names in simple parameter lists
- - Simple parameter lists are defined by those that do not include rest or defaults or destructuring.
-
-

Consensus: The name of the ClassDeclaration/ClassExpression follows the strict rules for its name. So it cannot be named "eval" or "arguments". Just like for strict function names.

5.9 Semantics and bounds of Number.isInteger and Number.MAX_INTEGER

(Allen Wirfs-Brock, originally proposed by Doug Crockford?)

AWB: What is the value of MAX_INTEGER

WH; Whatever the largest finite double

DC: But there are two

WH: But I said "double"

DC: That's ambiguous

WH: No

MM: WH is not constraining to the contiguous range.

WH: If you want 2^53, call it something else

MM: Likewise with isInteger

...Propose:

Number.MAX_SAFE_INTEGER = 2^53-1

Number.isSafeInteger => n > -(2^53)

AWB:

2^53-1, 2^53, 2^53+2

2^53+1 === 2^53

After 2^53, you can add 2

WH: Alternate proposal:

Number.MAX_CONTIGUOUS_INTEGER = 2^53

Number.isContiguousInteger = n => n >= -(2^53) && n <= (2^53);

MM: Gives history of "isSafeInteger"

Caja had a Nat test that tested that a number was a primitive integer within the range of continguously representable non-negative integers. I used Nat in a small piece of security critical code, to ensure I was doing accurate integer addition and subtraction. Because I was using this definition, Nat admitted 2^53. This introduced a security hole, which escaped notice in a highly examined piece of code which has been published several times and has been the subject of several exercises to do machine checked proofs of some security properties. Despite all this attention and examination, no one caught the vulnerability caused by admitting 2^53. By excluding 2^53, we have the nice invariant that if

- isSafeInteger(a)
- isSafeInteger(b)
- isSafeInteger(a+b)
-

are all true, then (a+b) is an accurate sum of a and b.

-

WH: OK

DC: Want to call this Integer

WH: Can't call this integer. 2^54 is an integer, just not inside of the contiguous range. Like the concept, but not ok to name it "isInteger", as 2^100 also happens to be an integer.

BE: Agrees with Mark's "Safe"

YK: Easy to explain that Integers outside of the range

AWB: Current spec checks for mathematical integer

...toInteger makes use of internal ToInteger

MM: Makes sure there is no fractional part?

WH: Yes

WH: If we have toInteger, then we need isInteger or isSafeInteger

AWB:

- isInteger
- isSafeInteger
-

MM:

- MAX_SAFE_INTEGER = (2^53)-1
-
- isInteger
- - Infinity => false
- - NaN => false
- - value !== truncated value => false

- - -0 => true
-
-
- isSafeInteger
- - -0 => true

-
- toInteger
- - Does not guarantee a safe integer
-

- ToInteger
- - Does not guarantee a safe integer
-

WH: The only place where ToInteger is divergent is +/-Infinity

WH: We already have Math.trunc, which does the same thing as ToInteger would. Don't need Number.toInteger.

5.8 Number.prototype.clz or Math.clz?

WH/AWB: Is an instance operation.

WH: If it's on Math.clz(), it will return the wrong answer if we have different value objects in the future

WH: In particular, this specifies that the value is 32 bits wide, which makes it inappropriate as something in Math. Consider what happens if we add a uint64 type. Then we'd want Uint64.clz to count starting from the 64th bit instead of from the 32nd bit. We can do that if it's Uint64.clz. We can't (without creating weirdness) if we use Math.clz for both.

AWB: Then it belongs on the instance side.

Any objections?

Consensus/Resolution

-
- - Number.prototype.clz
-
-

AWB: What about the following:

Number.isInteger
Number.isSafeInteger
Number.isFinite
Number.isNaN
Number.toInteger

Consensus/Resolution

- Remove Number.toInteger (already exists as Math.trunc)

- (Reference: https://github.com/rwldrn/tc39-notes/blob/42cf4dd15b0760d87b35714fa2e417b589d76bdc/es6/2013-01/jan-29.md#conclusionresolution-1)

-

5.13 Which existing built-in properties that are read-only/non-configurable do we want to make read-only/configurable?

(Allen Wirfs-Brock)

AWB: Previously, we've discussed setting data properties as {writable: false, configurable: true}

One of these built in properties that discussed is the length property of function

MM: Points about function properties, eg. the prototype property

EA: Classes are a constructor and the prototype, can't use function for the argument to how classes behave

MM: Don't think this is a question that should be addressed for ES6, it's too late.

AWB: Not too late, we've discussed this

AWB: The "prototype" property of the class constructor object is configurable, non-writable

AWB: {writable: false, configurable: true} allows enough control

EA: We also discussed this for methods

YK: This is part of the refactoring hazard I mentioned earlier.

MM: Don't want to consider a change of that magnitude this late in the game

AWB: All of the existing properties from ES5, should we address the whole list?

- -

When define a class:

(Foo.prototype) -C-> <-P- (Foo)

AWB: Foo.prototype.constructor property {writable: false, configurable: true}?

MM: This hazard:

```js
function Bar() {}
Bar.prototype = Object.create(Foo.prototype);
Bar.prototype.constructor = Bar;
```

Code that exists like this, once Foo gets refactored to a class, if constructor is non-writable, the above breaks.

AWB: [@@create](https://github.com/rwldrn/tc39-notes/blob/42cf4dd15b0760d87b35714fa2e417b589d76bdc/es6/2013-01/jan-29.md#48-refactored-new-operator-and-the-create-method)

```js
Array[@@create]
```

Recap:

@@create sets the prototype property of the new instance, but referencing the prototype property of the constructor itself.

MM: With regard to function.name and function.length and making them "tamper resistant", but mucking around with the built-in prototype chain has unknown implications and it could be addressed in ES7.

This change allows the actual Array.prototype to be changed.

WH: When does @@create get called?

AWB: when `new` is used.

Consensus/Resolution

- {writable: false, configurable: true}?

- - length property of functions: yes
- - prototype property of functions: no
- - new properties, ie. @@create: yes
-

-
-

TC39 + W3C

Discussion joint meeting with W3C at TPAC, Nov 11-15, in Shenzhen, China.

We decided not to do it this year, but may be next year, at the 2014 TPAC - provided it is on the US West Coast.

However, we need a clear set of concrete common topics to discuss.

We agreed that Alex Russell will serve as "TC39 Liaison" at the Shenten meeting. Istvan Sebestyen will inform the W3C officially. May be there will also be other TC39 experts in Shenzen.

5.1 Symbol primitive value or object? One more time.

(Allen Wirfs-Brock)

EA: There is discontent that there isn't private state. Symbols don't cover this. Unique Strings solve the uniqueness case

Proposal:

Postpone Symbols to ES7

BE: The reason we seperated private and unique was exposure in Reflection modules

YK: You don't need unique symbols when you can just expose private symbols.

MM: The @@iterator symbol must be transitively immutable

In the relationships case, the WeakMap

BE: There are classes that outlive any instances

Why can't we just have (private) Symbols

MM: Two subsystems that aren't supposed to be able to communicate with each other should be able to share anything that is transitively immutable.

BE: Can we unwind the split between private and unique?

YK: (fill this in)

AWB: We deferred private symbols

Private state should not be mixed up with Private Symbols

Symbols are guaranteed uniqueness, wrong way to go for private state.

BE: We aren't going to resolve this now, need to take it to es-discuss

AWB: For the spec, how do I spec Symbols?

Strings won't guarantee

MM/BE: (quick discussion about uuid strings)

WH: What you're saying is that we need a gensym?

AWB: Essentially, what we need is a gensym

BE: Andreas implemented Symbol

AWB: Dug in against wrapper objects for Symbols

1. (did someone catch this one)?

2. Unique objects, unforgeable, can't set or access properties. Are actually objects.

BE: ARB says that v8 internal architecture makes it hard to add new

Consensus/Resolution

- - Leave the spec as it is now

- - Postpone until next f2f
-
-

5.12 Should we remove [[Construct]] from the MOP and Proxy handler API?
(Allen Wirfs-Brock)

AWB: recapping @@create changes...

```js
new C(...args);
```

Essentially breaks down to:

```js
[[Construct]] =>

let obj = C[@@create]();

return C.[[Call]](obj, ...args);
```

YK: This means that [[Construct]] will always call [[Call]].

AWB: The way the built-ins work, override @@create, eg. Date, creates a private data slot for the time

```js
function String(value) {
```

- if (!(this has an unitialized instance)) {
- return "" + value;
- }
-

```
    this.value = "" + value
}
```

String[@@create] => { value: uninitialized instance }

```

WH: Disapproves having String when called as a function do different things based on this. This breaks the invariant that String(x) always returns a primitive string.

WH, MM: Also concerned about adding a new uninitialized String instance type as a specing helper but which becomes reified and user-visible. Someone could call String's @@create directly, obtain one of these values, and cause mayhem. Too much surface area of potential problems here, and this is unnecessary complexity.

YK: Objects to removal [[Construct]]

AWB: A Proxy trap?

BE/YK: Keep

#### Consensus/Resolution

- - [[Construct]] remains.
-
-
-

## Anti-Pattern to call a constructor without new
(Allen Wirfs-Brock)

AWB: In ES6, with class, it will be an anti-pattern... Don't call without "new"

BE: This is style/convention

Promote the use of `new` with classes

MM: Might want a constructor to refuse to initialize an instance of that class if the call object is not the

EA: Three browsers have implemented Map, Set, WeakMap, WeakSet and all are allowed to be called without `new`, which breaks subclassing
```

General agreement that this is bad.

AWB/MM: Function.prototype.@@construct

MM: If it implies runtime overhead that is not easily optimized, that would be a perfectly valid argument against. Does it?

In general, wherever we can replace a [[Foo]] internal property with an @@foo unique symbol named property, without penalty, we should. Especially if proxies would otherwise need a special trap for [[Foo]].

YK: Need to be careful when we change the MOP since other specs refers to the mop methods.

Consensus/Resolution

- - Giving up on the convenience of calling constructors without new, with any expectation
- - Throw when Map, Set, WeakMap, WeakSet are called without `new`

JSON

Any objections to sending the JSON draft 7 version to the general assembly

DC: Made changes. Specifiy code point. Removed summary of grammar. It was redundant. As well as the whitespace issue.

JN: We will have a final draft standard distributed on July 29 by the Ecma Secretariat. TC39 should vote (1 vote by each TC39 member company) on the final draft. Deadline for voting: 1 week. Send members' vote to JN and the Ecma Secretariat. If you don't reply to this thread then it is an implicit approval. If the TC39 vote is successful (it has to be more than 50% of YES), then the Ecma Secretariat will submit JSON for the Ecma GA letter vote (we have the permission to do this by the June Ecma GA). That takes 6 weeks.

6.2 Interfacing ECMAScript & HTML/DOM Event Loops

(Rafael Weinstein)

RWS: (A single slide) How does ES inegrate with the rest of the specified environment with regard to scheduling tasks.

- Enqueue A Task

- - The environment _must_ run the task at some point in the future
- - The task _must_ be run **after** **all** previous enqueued tasks
- - The task _must_ be run on an empty stack.
-

- Enqueue A Microtask

- - The environment _must_ run the microtask at some point in the future
- - The microtask _must_ be run **before** **all** previously enqueued tasks
- - The microtask _must_ be run **after** **all** previously enqueued microtasks
- - The microtask _must_ be run on an empty stack
-
-

WH: Note that this defines a total order.

MM: We need to decide how tasks or microtasks that originate from EcmaScript behave

MM: No nested event loop?

General agreement that the ES spec not support nested event loops. If the browser specs require them, i.e., for running JS code while a modal dialog is blocked, then the browser specs would need to state that this is an intended violation of the ES event loop model.

YK: Timing is another issue

MM: Promise scheduling, fifo

Discussion re: the host vs.

w3c bug...

Consensus/Resolution

- - Needs more offline discussion
-
-

Value Objects Update
(Brendan Eich)

ValueObjects.pdf

BE:

Use Cases:

- - Symbol
- - int64, uint64 (53 bits not enough)
- - Int32x4, Int32x8 (SIMD)
- - float32
- - Float32x4, Float32x8 (SIMD)
- - gignum
- - decimal
- - rational
- - complex
-

Overloadable Operators

- - | ^ &
- - ==
- - < <=

- - << >> >>>
- - + -
- - * / %
- - ~ boolean-test unary- unary+
-
-

Preserving Boolean Algebra

- - != and ! are not overloadable to preserve identities including
- - X ? A : B <=> !X ? B : A

... Too fast, request slides.

http://www.slideshare.net/BrendanEich/value-objects

"complex and rational cannot be composed to make ratplex"

AVK: Multiple globals will cause issues.

BE: That is not an issue with this proposal. It is an issue with multiple globals.

... we need literal syntax for readability.

... no solution for user defined literal suffixes.

BE: Some have requested mutable value objects in order to represent small tuples and be able to do updates on them in a loop.

WH: This no more requires value objects to be mutable than incrementing a loop counter requires integers to be mutable. It's the variable that holds the integer 3 that's mutable and an be changed to refer to a different integer; you can't change the integer 3 itself to be 5. If the value is a small tuple and the source and destination are the same, it's easy enough for a compiler to transform a functional-style tuple update into imperative code if it likes.

WH, MM: Don't want mutable number literals/objects. No `new Float32x4(a, b, c, d)`. This would break === (which would then need to do identity matching instead of same-value matching).

```js
```

typeof x == typeof y && x == y

<=>

x === y

0m === 0

0L == 0

0m == 0L

```

BE: typeof become advisory

AWB: You can register typeof result once during registration. That way we can enforce that it does not changes.

#### Consensus/Resolution

- - NaN requires separately overloadable <= and < [Slide 5]
- - Intersection means function identity matters, so multimethods can break cross-realm [Slide 9]
- - Mark objects that I or i as bignum suffix conflicts with complex [Slide 11].
- - Always throw on new -- value objects are never mutable and should not appear to be so, even if aggregate [Slide 12]
- - Need to work through any side channel hazard of the typeof registry [Slide 13] and the multimethod dispatch "registry"
-

## 6.5 Parallel JavaScript (River Trail)

(Rick Hudson)

...need slides

RH: We have to go parallel to keep up with other langauages

YK: Don't want to fallback into sequential
```

Various: Debate about what happens when the parallel computations have side effects that introduce dependencies between them. Options are either devolving into sequential computation or throwing an exception.

RH: The code behaves the same way as sequential code but goes faster if there are no side effects.

WH: What happens if there are no side effects but some of the computations throw exceptions? Which exception do you get?

RH: Any of them. There are also other implementation options here.

WH: If it's any of them, then this is not like sequential code.

WH: What exactly is a side effect? How would a programmer know that some ECMAScript construct has an internal side effect?

WH: In particular, suppose that I want to use a parallel loop to fill a big matrix with random numbers. Is calling a random number generator considered to be a side effect or not? If the answer is yes (it is a side effect), then how would one fill a big matrix with random numbers in parallel, as that is something that one would reasonably want to be able to do?

Consensus/Resolution

-
- - Throw instead of falling back to sequential.
- - Focus on concurrency/scheduling in ES7. Make sure it fits with other concurrency constructs (promises/event queues)
- - Discussion/Acceptance in ES7 process.
-

RWS Proposal For Specification Process (for ES7 process)

Consensus/Resolution

-
- - Go forth
-

7 Internationalization

NL: Implementations of ECMAScript Internationalization API:

- Microsoft has shipped it in Internet Explorer 11 beta
- Opera has shipped it in Opera 15 (based on Chromium)

Future Meetings

Sept 17-19, Boston

Nov 19-21, San Jose