Minutes of the: *7th meeting of Ecma TC39*
held in: *Kona, HI, USA*
on: *19-20 November 2008*

Chairman: Mr. John Neumann (Microsoft/Ecma International)
Vice-Chairman: Vacancy
Secretary: Mr. John Neumann (Microsoft/Ecma International)
Attending: Mr. Douglas Crockford (Yahoo!), Mr. Brendan Eich (Mozilla), Mr. Cormac Flanagan (UCSC), Mr. Dave Herman (Northeastern University), Mr. Pratap Lakshmann (Microsoft), Mr. Scott Idaacs (Microsoft), Mr. Waldemar Horwat (Google), Mr. Mark S. Miller (Google), Mr. Mike Samuel (Google), Mr. Rob Sayre (Mozilla) and Mr. Allen Wirfs-Brock (Microsoft).
On phone: Mr. Jeff Dyer (Adobe) and Mr. Chris Pine (Opera)

1 Opening, welcome and roll call

Introduction of the attendees.

2 Adoption of the agenda (08/093)

The agenda was adopted as presented.

3 Approval of the minutes of September 24th to 26th, 2008 (08/087)

Approved as presented

4 ES3.1 draft proposal discussion

TC39/2008/095 Draft "Kona" ECMAScript 3.1 Language Specification of 07 November 2008.

Discussion of features that are retained or removed. First item on the list was Decimal, and details follow:

The attendees reviewed the current status of the proposed decimal arithmetic extensions to ES3.1. After careful considerations the attendees unanimously agreed that the decimal extensions were not yet sufficiently mature for inclusion in the frozen draft specification. There were two main area of concern that lead to this conclusion.

The first concern is the most fundamental as it is a concern regarding the actual design of the decimal feature set. The decimal proposal redefines all of the ES3 Number operators to be generic numeric operators, for example X*Y may compute either a Number result or a Decimal result depending upon the primitive types of the values of X and Y. This is an example of a generic numeric expression and in a dynamically typed languages, such as ECMAScript, with multiple numeric types it should be possible to write complete generic numeric algorithms. However, the currently Decimal design has a number of characteristics that prevent the natural coding of such algorithms in many situations. For example, while operators that are identified using special characters such as +, * , and / are generic, other numeric operators that are identified using functional notation such as min, max, and abs are explicitly defined to be non-

generic as separate versions of these functions are defined for Number (as properties of the Math object) and Decimal (as properties of the Decimal object). Thus it is impossible to write a generic numeric algorithm that uses such functions. A similar concern relates to the specific numeric constants NaN and Infinity and the isNan function. A related concern is the treatment of numeric constants, particularly small integer constants. In writing generic algorithms it is common to use small constants in expressions such as N+1 or X*2. Under the current proposal such constants are implicitly of type Number and force the type of the result to be of type number even if N or X were of type decimal. Thus, the use of such constants "poison" a generic expression forcing it to be a Number typed expression. The use of a decimal constant (1m or 2m) would produce the desired result but it is unlikely that ECMAScript programmers would routinely remember to follow that convention. Instead, it would be preferable that the language support generic interpretation of such constants where their type would be dynamically inferred based upon the other operands to the operator.

The attendees agreed that the ability to express such generic algorithms should be a characteristic of any version of ECMAScript that includes multiple numeric types. The current decimal proposal not only does not provide such support but its inclusion in its current form would make it difficult to add such support in a future revision.

The other major concern was in regard to the completeness of the current proposal. While a number of issues from the Redmond meeting have been address, there remains many places in the draft specification where the addition of the new decimal primitive types has not been fully or correctly integrate into the specification. Many of these issues have been identified in recent emails from Waldemar Horwat and others. Some of the issues are functional and not just specificational in nature. The attendee agreed that the draft specification could not be functionally frozen to include decimal until these issues were resolved.

Because of these concerns the decision was made to defer inclusion of decimal support until the Harmony revision of ECMAScript. The attendees acknowledged that very significant progress has been made in the development of the ECMAScript decimal proposal and want to thank Sam Ruby of IBM for the effort he has put into its development. The attendees encourage the continued development of the proposal by Sam and other TC39 members and are optimistic that a fully integrated and generic versions of decimal arithmetic can become an integral part of the Harmony revision.

4.1 Draft review

Review of draft occured by reviewing the comments Waldemar had sent by email to everyone. Also looked at list of concerns from Allen, and the changes are identified below:

Areas of concern include:

Object.keys(fast); *always fast so parameter not needed.*

Object.methods; discussed and resolved. All ES 3 algorithms need to be evaluated to make sure they still work in light of getters and setters, etc.

ES3.1 Opt in;

Future Const; Global object. Need to come to agreement on what Const is or will be in harmony to make sure we don't have any conflicts with 3.1.

reflection leakage; Needs to be studied later, but no problem seen for now.

Chapter 16; list of C16 exceptions for strict mode things. Need to revisit for Harmony

this binding for delegate functions; Chapter 10 does not apply to built in functions. Chapter 15 is not used in strict mode

Object.getprototypeof; have eliminated other constructs because we have this and if we eliminate then we need to take another look at what has been done so there is almost no support to eliminate this. It is a natural part of the reflection type.

Chapters 8, 10; Waldenar will communicate with Allen via email to resolve any further concerns or issues

Arguments array in strict mode; we have *isarray* in 3.1 but not *isarraylike*, which is open for includion in harmony

strict mode restrictions; *no strict mode restrictions in strict mode, add new clauses 12.5, delete 12.1.1 and remove restrictions on var statements in blocks*

statement grammer; reverted to ES 3 so all problems went away

var y, var x ; etc. *All disallowed. no param (x) var (x), no function (x) var (x); duplicate parameters not allowed – f(x,x)*

Issues with sort discussed.

Details of Document Review Follow:

5.2 - "step my specify"

7.1 - "format control characters may be used in identifiers, ...": No they can't, according to section 7.6.

7.3 - "except that line terminators that are preceded by an escape sequence may occur": "preceded" is not the right meaning here. "part of" ?

7.3 - The production
LineTerminator:: ... | <CR> | <CR><LF>
is ambiguous. Probably the simplest fix is to change it to:
LineTerminator:: ... | <CR> [lookahead isn't <LF>] | <CR><LF>

Even then having <CR><LF> there causes trouble for things like its use in 15.10.2.6, 15.10.2.8, and 15.10.2.12.

7.5 - Token:: ReservedWord | Identifier | IdentifierName doesn't make sense

7.8.4 - "All Unicode characters may appear literally in a string literal except for the closing quote character, backslash, carriage return, and line feed. Any character may appear in the form of an escape sequence." This is wrong about the other line terminators. Cannot have <LS> and <PS> in a string literal; the only way to put them in is using their Unicode values.

7.8.5 - Allowing /[/]/: This would be a change in this section. It's already allowed by the chapter 15 grammar. /(./ is a syntax error in Mozilla and ES3. Leave it that way.

8 - Can Property Descriptors and Property Identifiers be stored as user-visible properties of objects ? The last sentence seems to imply that they can.

8.6.1 - "change the property to being an" => "change the property to be an"
"operator in section 11.4.1, and the": remove comma.

8.6.2 - add [[ThrowablePut]] in the first paragraph after the table.
"The value of the [[Class]] property of a host object may be any value": Do you mean any string ?

8.6.2.2 - "explicit control over the handling of invalid property store": Do you really mean "store" here ?
"stores" makes more sense.

8.6.2.8 - "if O is a String object it has": add "then" before "it".

8.6.2.10 - Add comma before "the following steps".

8.6.9 - get rid of the SameValue test from 10.a.ii.1. Setting the same value twice must be an error.

8.6.10 - change [[ThrowablePut]] to [[FalliblePut]]

8.10 - The nomenclature is too inconsistent. Sometimes you refer to property descriptor properties as "writable" (as in "{value: 42, writable: false, configurable: true}", and sometimes as "[[Writable]]" (as in "Desc.[[Writable]]" in 8.10.2). Are these are two different things just as x.prototype and x.[[Prototype]] are different ?

Also, Desc.[[Writable]] doesn't make sense because there is no such internal property listed in the table of all internal properties used in this specification in 8.6.2.

Data types from 8.10 are used in earlier sections of chapter 8 before they are defined here. Can't figure out which order to read this chapter in, as text from 8.10 subtly modifies the interpretation of 8.6.2. Solving this problem by moving this content to or near 8.6.2 would help solve the others as well.

Can a Property Descriptor include both [[setter]] and [[value]] fields ? 8.10 is ambiguous on that.

"(name, descriptor), where name is a string and descriptor ": italicize "name" and "descriptor".

8.10.4 - "the following steps are taken:, the following steps are taken:". Don't repeat. Don't repeat :-)
The Note here should be a normative part of the preamble. Otherwise step 4 doesn't make sense.
Be consistent about italicization of Desc.

8.10.5 - Call the formal parameter something other than "Desc" here. It's confusing to use the same name for both objects and property descriptors.

9 - Decimal support broken in most of the tables.

9.3 - ToDecimal on a Number gives the Number unchanged ?
ToNumber on a Decimal is not defined.

9.8 - ToString on a Decimal is not defined in the table.
The algorithm only works on Number values. +0, -0, etc. are Number values, not Decimal values. Also, it internally references conversions to Numbers.

9.3.1 - ToDecimal on a string results in a Number. Also, it optionally drops significant digits after the 20th.

10.2 - "functrions"

10.2.1 - "binding can not be set throw a TypeError exception": Missing comma.

10.2.1.1 - "a ECMAScrpt": a -> an and fix spelling error

"A declarative environment record binding the set of identifiers defined by the declarations contained within its scope.": Not a sentence.

10.2.1.1.x - Be consistent about spaces before the opening parenthesis of formal parameters.

10.2.1.1.6 - "The S argument is ignored because strict mode deoes not change the meaning of setting bindings in declarative environment records have .": Ah, that's what "S" is for ? You didn't explain this earlier when S was first mentioned in the other methods. Also, fix grammar errors.

10.2.1.2.x - Same comments as above.

10.2.1.2.1 - This will mean that having bindings in the prototype will prevent one from building ones in the leaf object.

Step 4. If Result(3) is false or the binding for N in Result(1) is an uninitialized immutable binding, then": What's an uninitialized immutable binding here ? Result(3) is an object, not an environment. Objects have properties, not bindings.

10.2.1.2.5 - This will always error out in DefineOwnProperty.

10.2.1.2.6: Step 3: If the binding for N in Result(1) is a mutable binding, then": What is a mutable binding ? Result(1) is an object, not an environment.
Step 4. Else this must be an attempt to change the value of an immutable binding so throw a TypeError exception.": This doesn't follow. For example, just because Result(1) has no binding doesn't mean that its prototype doesn't.

10.2.2.1 - "called with a lexical environment lex, identifier string, name, and boolan flag strict the following steps are performed": Due to several grammar errors (an extra comma and a missing one) this doesn't mean what it's supposed to.

10.2.2.x - "is call" -> "is called". Lots of other typos as well.

10.2.2.4 - There is no current lexical environment bound around the declaration of PopEnvironnmentRecord.

10.3 - "to tract the execution" (?)
What is VariableEnvironment for ? It's never used in the spec, except for a mention in 12.2 which is a bug and shouldn't be there.

10.3.2 - Can't do the arguments object this way. It's incompatible with ES3 for multiple arguments sharing the same name. You also don't want users extracting the getters and setters out of the arguments array, etc. Also, the notion of scope in which the getters and setters are eval'd is fuzzy at best and can cause problems if other definitions ever shadow the parameter names.

IsArray(arguments) should return true

10.3.3 - "Variables and functions declared in ECMAScript code evaluate in the execution context are added as bindings in the that environment record." (?)
Step 1. Let env be the running execution context's VariableEnvironment." How do those get created ? Section 10.4 should come first.

10.4 - This is still confusing. What creates execution contexts ? There is no such step in the algorithms here.

11.1.5 - This means that I can override a getter with a value property or specify two getters for the same property even in strict mode. We had agreed that strict mode disallowed such things.

11.2.1 - "where <identifier-name-string> is a string literal containing the same sequence of characters as the IdentifierName.": The meaning is ambiguous in the presence of escape codes.

11.3.1, 11.3.2 - All four of the return statements are wrong in different ways. Some return the preincremented value. Some return an lvalue instead of an rvalue.

11.4.1 - Agreed to change this to always throw if step 5 is reached in strict mode. This prevents "delete x" from deleting a global variable.

11.4.1.1 -
delete a.b
(x+y)
would cause inappropriate semicolon insertion in strict mode. Also, MemberExpression doesn't accomplish much here, since you can still write delete (4).
Fix: Remove grammar restriction altogether.

11.4.3 - host objects cannot masquerade as "function". Host objects that don't implement [[Call]] cannot answer as "function"

11.5 - What's the corresponding Decimal operation ? There are a bunch of different remainder options.

11.8.5 - Treating Unicode character codes as Decimal numbers. Which characters have Unicode numbers that are Numbers, and which ones have Unicode numbers that are Decimals ?
If you fix this and apply the same contagion rules as for +, -, *, etc., then you'll have the issue that 1e-400m > 0m but 1e-400m > 0 is false. The contagion rules need rethinking.

11.9.3 - The contagion here is from Number to Decimal. This is inconsistent with +, -, *, etc., where the contagion is from Decimal to Number. It should be the same for all arithmetic operations.

11.9.6 - Don't need to call ToDecimal on values that are already Decimals.

11.13.1.1 - The strict mode restrictions are ambiguous. What happens in this case, where g does not exist ?

g = (function(){throw "foo"})();

What about this ?

g = eval("var g = 5; 2");

Agreed that 12.1.1 is gone. Agreed that there are no strict mode restrictions on var placement.
Agreed to allow redundant var x declarations.

Agreed that in strict mode we disallow name conflicts within the same (hoisted) scope of:
parameter vs. parameter
parameter vs. var
parameter vs. function
function vs. var
function vs. function

Implementations are free to report these as errors early, at the same time as syntax errors. This will require adding cases to chapter 16.

12.2 - This breaks ES3 and existing practice. Consider with(o) {var x = 3} if o.x exists and has the value 7. This code currently sets o.x to 3; the proposed change would leave it at 7.

13 - "code code"

14 - The syntax of the use strict directive is incompatible with the lexer grammar. There is no such separate token. What happens if someone escapes a character within the use strict directive token ? The spaces before "use" and at the end are mandatory ? Is it mandatory that the semicolon follow without an intervening space ? How does the semicolon interact with semicolon insertion ?

Strict directives are ambiguous with statements.

There should be no "opt" after UseStrictDirective's definition.

15 - Debate about whether to change the spec to require implementations to ignore extra arguments passed to built-in methods (5th paragraph of chapter 15). This would interfere with arity checking in future variants of strict-arity mode because currently programs that pass extra arguments are non-portable, while they would become portable if the spec mandates that extra arguments are ignored. Also, there are some methods on which we have a placeholder for locale objects. Mozilla makes use of an extra argument in its string replace

function. Agreed to revert to ES3 text. But remove the statement that an implementation is permitted (but not required) to throw an exception.

Some of the "Strict Mode Restrictions" paragraphs are normative (15.1.2.1.1). Some are informative (11.13.1.1). We need to clearly distinguish the two. Perhaps we can move all the strict mode restrictions to an informative annex. Or, make them “Notes” instead of giving them their own section number – “Notes” are meant to be informative.

15.2.3.14 - remove the "fast" parameter. If an implementation defines an order for "for-in", this should follow the same order.

15.4.4.11 - sort is broken by getters that return , setters, read-only properties, non-configurable properties if there are holes, non-extensible objects, etc.
need to explicitly specify the “this” value passed in. the default this value should be undefined.
All the “Call” should change to “Call the [[Call]] …”
Also this algo seems to be broken in the presence of getters/setters, read-only properties, and non-extensible objects.
All legacy algorithms from ES3 need to be reviewed - many of the algo might be written with the naïve assumption that they are operating on data properties that are mutable.

Decimal is out because the spec isn't ready and there are some problems that are not small spec errors -- generic behavior of functions, etc.

Wednesday high level items
Discussion about reflection:
- Compatibility with const/let.
- Extracting getters and setters exposes too much information -- we'll need to either spec which ones are == to each other or live with undefined behavior.
- Name conflicts with Prototype and other libraries.

List of strict mode restrictions from ES4 discussions of a few months ago:
- No null-to-global-object this propagation (if non-strict mode doesn't already do this)
- Throw on writes to read-only properties
- Throw on deletes of dontdelete properties
- delete o.x when x is not in o but in the proto should throw
- Reference before definition causes static errors (in what contexts ?)
- Arity checking (conflict with 3.1 ?)
- Global variable auto-creation
- Duplicate formal parameters, parameters with same name as var or local functions, etc.
- Duplicate names in object initializers
- FunctionObject.arguments (not in ES3 but used in practice)
- Use of arguments object (maybe ?) (conflict with 3.1 ?)
- Useless expressions (maybe ?)
- Prohibit with and eval (if non-strict mode doesn't already do this)

Debate on setting properties and SameValue check: Is NaN a single value or possibly many, distinguishable via implementation-defined means. Choices:
- Require that NaNs be indistinguishable even if we adopt IEEE 754-2008.
- Allow read-only NaNs to be "replaced" with other NaNs, with the result being that the original NaN stays.
- Never do SameValue tests. Replacing a read-only value is always an error even if it's being replaced by the same value.
Resolution: Third choice. Get rid of SameValue checks.

Discussion on what it means to be an Array, a RegExp, a Function, etc. Host objects can have any Class value. Some places in the spec distinguish on "x is an Array object", others distinguish on [[Class]]. Relevant for things like bind which must distinguish between length indicating the preferred number of arguments and length being an unrelated size of something.

Thursday high level items:
Concerns about ES3.1:
Object.keys(fast): need second argument ?
Object constructor method name conflicts
ES3.1 opt-in
Compatibility with future const
Reflection leakage
Chapter 16
this binding for callbacks (array comparator etc.)
Object.getPrototypeOf
Exposition of chapters 8 and 10
Arguments array in strict mode
Strict mode ambiguities
isArray(arguments)
Statement grammar
Webfoot
[[class]] "function" bind

The Kona draft needs to be patched with the email update to the statement grammar and strict mode restrictions sent out by pratapL

Agreed to remove the "fast" parameter of Object.keys and take out the sort. If an implementation defines a specific order for for-in then Object.keys must return the same order.

15.4.4.11: sort is broken by getters that return inconsistent values, setters, read-only properties, non-configurable properties if there are holes, non-extensible objects, etc. Agreed to fix this somehow; it won't necessarily be easy. Will also need to verify that all of the other algorithms in chapter 15 still work in the presence of getters, setters, read-only properties, non-configurable properties, and non-extensible objects.

With "const" missing, the changes to chapter 8 for attribute description become premature standardization and should be cut. The problem is that these changes are likely to be incompatible with ES-Harmony due to the same logic that cut "const". Without "const" we have no way of testing this.

Allen says that there is no conflict between the reflection API and const. We'll likely approach it in Harmony by not creating any properties in the global object until the const is initialized. All agreed to verify that const (as planned for Harmony) is not broken by ES3.1.

Reflection leakage: For Harmony we'll look at ways to seal abstraction leaks (interposing hidden levels in prototype hierarchies of user-defined classes, etc.).

Chapter 16: Extend list of errors that can be signaled early to include some strict mode violations (duplicate parameter names, etc.).

Chapter 8 and 10: Get rid of hidden state that's implicit in the algorithms but not exposed in the data structure.
Introduce concepts before using them.
Go through Waldemar's list of comments from before the meeting.

bind behavior: Should bind create only a call or both a call and a construct bound property ? We decided to stick to last meeting's decision of creating both a call and a construct bound property. (If it were just call, then the argument for adding bind to the language at all weakens since it would do duplicate what the frameworks already do but perhaps slightly less compatibly.)

Harmony hour:
wiki:strawman:strawman

- classes
- const / let
- decimal
- lambda
- lexical scope mode (pragma) vs. module {}
- names
- return to label
- types
- webfoot

What is webfoot ? Performance optimization for Valija-like things. Whitelisting flag, interceptors, catchalls ?

How is lambda useful except for code gemerators? It's hazardous because it's too easy to leak completion values that were not intended to be returned. Also it's hard to refactor lambdas if it's not clear whether their return values are intentional or accidental.

classes:
(class and instance) * (const and mutable variables, methods, and getters/setters) * (public and private) + constructor ?
instance private vs. class private
class private would require a different syntax for accessing the private value "length" vs. the length of some unrelated object that was passed in in an argument.

pratap

4.2 Next steps

Pratap will complete the final draft by the end of the year or into January to get final clean-up.

5 ES4 draft proposal discussion

Subjects for discussion:

Catchalls

Classes Static or Instance members (var or methods), Public static or writable static, Public or Private by default (preference is Private by default),

Const

Decimal

Lambda

Lexical Scope mode

Names

Return to label

Webfoot

It has been suggested that written proposals be provided on each of the above topic areas so that the group can have something concrete to discuss at the January meeting

6 Interoperability testing

6.1 Concept

Two Browser based implementations will be provided by Mozilla and Microsoft and the test set will be provided by Mozilla.

6.2 Actions

7 Progress reports on the "Secure ECMAScript" Plan

8 Any other business

None

9 Date and place of the next meeting(s)

- January (Mountain View [28-29])
- March (Sunnyvale/San Jose Area [25-26])

10 Closure