***Minutes of the:*** ***Ecma TC39-TG1***

***held in:*** ***Newton, MA (Adobe)***

***on:*** ***18th – 20th April 2007***

## Attendees

- Douglas Crockford, Yahoo!
- Jeff Dyer, Adobe Systems
- Brendan Eich, Mozilla Foundation
- Cormac Flanagan, Univ. of California, Santa Cruz
- Lars T Hansen, Adobe Systems
- Chris Pine, Opera Software
- Graydon Hoare, Mozilla Foundation
- Allen Wirfs-Brock, Microsoft
- Pratap Lakshman, Microsoft
- Adam Peller, IBM (attending for Wayne Vicknair)
- Dave Herman, Northeastern University

## Agenda

- Pending proposals
  - ES3.1 Proposal Working Draft, as a work in progress
  - `T~` syntax and semantics
    - The semantic issue is whether `T~` should imply `= undefined` defaulting
  - JSON discussion?

- Detail issues / clarifications / bug fixes
  - Exact meaning of “better eval” / “resurrected eval”
    - strict mode selects different runtime `eval`
    - `eval` as downward funarg
  - `RegExp.$`*n* – to condone or not to condone?
  - `intrinsic::hashcode` and operator `===` overriding
  - `intrinsic::hashcode` and NaNs
  - catchalls pending update

- Reference implementation work

- Administration
  - Bug tracking system

- Runtime type checking (Dave, Cormac)
  - Our support for gradual typing (ie the type *) via the 1-bit semantics requires that the evaluator know a ‘static type’ for certain expressions. For example, on a function call e(), where e is some expression, the evaluator needs to evaluate e to a function pointer, but if the safety bit is set on that function, then the evaluator needs to know the static type of e (such as function():Foobar) so that it can check that the value returned by e has the

appropriate type Foobar. (See minutes for the Feb 27 2007 meeting). Three options to implement gradual typing are:

- 1-bit semantics, where the evaluator computes static types as it evaluates
- 1-bit semantics, where the static types are computed by a pre-pass (perhaps a variant of the verifier)
- Wrapper semantics, where each function could have a wrapper function that provides a different type, and performs run-time checks on calls and returns.

## Discussion - Day 1

### Bug tracking

We will set up a Trac and evaluate that within the next few days or at most couple of weeks.

Bugzilla seems a little heavyweight both to set up an instance and to use, plus it doesn't do as much as Trac; and it's most notably hosted on mozilla.org, but we shouldn't jam ECMA TG1 bugs into that instance.

### ES3.1 proposals

Some issues with these proposals: hard to access with all the subpages.

But on the positive side: they point out some missing background in the current ES4 work.

Jeff: what is the status of these proposals? When should the group start looking at them seriously?

Allen: The proposals are two-level, conceptual and detailed. The current proposals are (generally) conceptual, to create discussion in the group. Right now the main points are what the group thinks about ES3.1 and its relationship (compatibility-wise) with ES3 and ES4.

Allen: Though we can patch the ES3 spec, it's more attractive to use the proposed form of ES4. That said, we don't know yet what that spec will look like.

Brendan: The ES4 spec will be excerpted code from the reference implementation surrounded by prose, literal-programming style. The RI will not itself be normative. But the `goto` style of ES3 is no good.

Allen: If we look at ES3.1 as a maintenance version of ES3, is it reasonable to expect ES3 implementors to have to relate to a whole new style of specification for ES3.1?

Pratap: Given that ES3 uses a specification format accepted by ECMA, why not use the same format to ES3.1

Brendan: Given the existence of errata for ES3, just patching the ES3 spec is a fairly empty exercise.

Doug: Can ES3.1 be published as an appendix of some sort?

Brendan: The key aspect of the RI seems to be that it's testable; this has the most value for the implementors. A "delta document" to ES3 is hard work and is of limited use to implementors.

Allen: Seems to argue for updating the ES3 spec.

Brendan: But that is of limited use to implementors; they are already synced to the errata. Testability of the spec is important.

Allen: If the implementors are synced, then let's capture the sync points in a spec.

Doug: ES3.1 needs to be informative and carry authority in helping to deprecate some features.

Brendan: Deprecation is not very helpful; people don't care. And old code continues to exist. The carrot for people is strict mode in ES4.

Allen: Microsoft is very interested in a strong ES3.1 spec, not much at all in ES4 as a future language. ES3 is the standard for the web and will continue to be so; resolving open issues will be worthwhile. Big changes are less valuable.

Brendan: But the question is still how to specify these resolutions, and a testable RI is the way to go.

Allen: It would seem that many successful languages do not have testable RIs.

Brendan: But a lot of uses of those languages are single-implementation, nonportable, not directly connected to those language standards.

Graydon: We choose between overspecifying with an RI or underspecifying with a prose spec. Ambiguities in the latter may not be discovered in the spec process at all.

Allen: No quibbles with that; the issue is whether that's the way to go for a maintenance release of ES3?

Jeff: The group is split; the people interested in ES3.1 should probably continue working on that and clean up and flesh out the proposals. But if this work goes on in the framework of ES3 it's probably not of much interest to the rest of the group.

Brendan: The Ajax implementors are not very interested in proposals on the level of `typeOf`, which they can implement themselves; it's larger scale problems – the power of the language – that concerns them. Failure to upgrade the language will drive implementors to competing technologies (Apollo, Flash, Silverlight, ...). And smaller devices can't support multiple runtimes.

Pratap: Just as a data point Flash v9 ships with 2 VMs (for ActionScript2 and ActionScript3). (Jeff clarification: this is primarily because AS2 was less concerned with ES3 compatibility than AS3. We made a choice in AS3 to prefer ES3 compatibility over AS2 compatibility. Because ES4 is compatible by design with ES3, this particular motivation does not exist for an implementation of ES3 and ES4.)

Allen: Some of us believe the scope and breadth of changes in ES4 will not be helpful on the web or to implementors, and ES4 will not be adopted widely.

### ''T~''

- Controversy about the implied `= undefined` behavior in the parameter list
- Brendan argues for the useful shorthand of `~`; Allen worries about syntactic overload for the programmer
- Jeff advocates doing nothing
- **Resolved**: we withdraw this

### JSON

- Issue from the public mailing list: should there be a `toJSON` rather than / in addition to `toJSONString`
- Arguments for: avoid accidents, more concise
- Arguments against: Doug has had no complaints about the existing API
- **Resolved**: we keep it as `toJSONString`

- **Action**: Doug follows up on the mailing list
- **Action**: Doug/Brendan extend the spec to deal with shared structure

### Resurrected eval

- Proposal not in good shape, needs to be cleaned up (a number of things are in the minutes)
- We clarify that strict eval is a compile-time rewriting, so strict and standard modes have the same run-time behavior
- There are problems on the web we've not previously considered, but which we will need to contend with:
  - `var evil=eval`
  - `o.map(eval)`
- ...
- **Action**: Chris will provide some examples based on recent investigations

### ''RegExp.$n''

- Not defined in ES3
- Perl-based
- The `RegExp` object is dynamically scoped in Firefox
- **Resolution**: Leave it out of the spec

### ''hashcode''

- Which `===` are we talking about?
- Should we be talking about `===` at all?
- **Resolved**: `hashcode` uses `intrinsic::===`
- **Resolved**: values that are not `intrinsic::===` to themselves hash to 0

## Discussion - Day 2

### 3.1 Wiki

We discussed options for using the wiki to record the 3.1 work. We agreed to create a new namespace (directory) and that the structure of that namespace should mirror the existing proposals, discussions, and clarifications structure.

### ES 3.1 discussion

Douglas Crockford, Allen Wirfs-Brock, Adam Peller, Pratap Lakshman participating.

[I was there briefly, to try to find agreement on compatibility principles that were not consistent in ES3.1 as proposed, before I had to leave due to a family emergency. I am recording **dissenting notes (DN)** here. — *Brendan Eich 2007/04/21 22:05*]

Discussion to pin down a set of principle to guide ES 3.1 reached the following general points of consensus among the participants

Overarching principle #1: "Scripts written to take advantage of ES3.1 functionality must be able to load and parse on an ES 3 implementation."

For any new revision of ECMAScript to be of any utility to web application developers that revision has to be implemented by the most commonly used web browsers. However, even after browsers implementing the new version start to be widely available a vast installed base of browser that only implement the prior language definition will

continue to exist for a long period of time. Hence application developers must continue to provide scripts that will load and operate upon implementations of the older version of the language. However, they would also like the opportunity to utilize the functionality of the new version if it is available in a user's browser. The technique of accomplishing this that is currently preferred by many web developer

[**DN** Web developers do not "prefer" this, as in "like it" – they are forced by the IE6 installed base to cope this way instead of using other means – but not all developers agree and some in fact author for only specific browsers, or use server-side code generators that do their own compatibility handling. It's misleading to argue from a contingent state of affairs without saying why the contingency (IE failing to update to a newer version for over four years) should be repeated again — *Brendan Eich 2007/04/21 22:05*]

is to author a single script file which can load into implementations of either the old or new version of the language and to then use explicit conditional execution based upon version identification to exclude the execution of code that requires the new version when running upon am old version implementation.

[**DN** This is a straw man. Many Ajax apps load multiple files, including "compatibility" shims. Atlas is an example. So it is **not** required by the design of ES3 that all code be loaded in one file. Therefore it is possible to load different files, some containing incompatible syntax, depending on version detection. See versioning. — *Brendan Eich 2007/04/21 22:05*]

This approach works as long as there are no syntax differences between the two language versions and all new functionality is implemented via function definitions. If there are syntax differences then any scripts that used the new syntax could not be loaded into an implementation of the old language version (a syntax error would be detected) and hence will not be used by developers that need to support the universal web.

[**DN** Or they'd factor the new code into a separately loaded file, and load it conditionally. Enough with the straw men, please. — *Brendan Eich 2007/04/21 22:05*]

We recognize that in some cases new syntactic forms would be quite desirable, but the reality is that any new functionality that depends upon the use of such new syntax will not achieve any near term adoption by web developers. A corollary of this is that in order to achieve adoption, new syntax needs to appear in web browsers several years before significant adoption actually begins to occur (examples: ES 3 try/catch syntax, property getters/setters).

New syntax is a long term investment with little short term return. Hence, any investment in such new syntax should be carefully considered and focused on features that are expected to have a very high value to developers.

[**DN** Like, for instance, ES4. Or did you have a substantive criticism of ES4 to make? — *Brendan Eich 2007/04/21 22:05*]

This leads to the following design guidelines for ES 3.1:

Slogan: "No new syntax"

(a) Try to provide a non-syntactic form of any new feature that is syntactically backwards compatible with existing ES3 implementation

- Separate function from syntactic sugar
- code written using ES3.1 functionality loads without syntax errors into ES3
- allows for the possibility of introducing ES3.1 functionality in ES3 implementations via compatibility packages.

[**DN** In which case, new syntax could be used. The "no new syntax" slogan is thus not a valid requirement. — *Brendan Eich 2007/04/21 22:05*]

(b) Only introduce new syntax into ES3.1 if it offers extremely high programmer convenience

- Programmers who use the new syntax implicitly choose to run only on ES3.1 implementations
- Broad usage of the new syntax won't occur until browser that only support ES3 are not longer in broad usage (5+ years into the future)

[**DN** 5+ years is not based on what actually happened last time, or the time before (ES2) when there actually was still browser competition, and this time we have self-updating browsers and Windows Update. Please stop the exaggeration (the cute + after the 5, even). We should be arguing about things that can be determined now, not asserting certain knowledge of the future, even if minimum number of years until some event. — *Brendan Eich 2007/04/21 22:05*]

Overarching principle #2: ES 3 scripts must continue to function correctly when executed upon an ES 3.1 implementation.

In general, we expect ES3.1 implementation to be deployed in browsers as replacements for ES 3 implementations. There are 100's of millions of pre-existing ES 3 scripts on the web. Browser supporting ES3.1 will only be deployed and used if they can continue to execute essentially all existing ES 3 scripts. While most syntactic additions (other than adding new reserved words) do not present forward compatability problems (by definition ES 3 programs will not use the new syntax) the additions of new functions does. This is because, the behavior of existing ES 3 scripts can depend upon the non-existence of functions newly defined in ES3.1.

Design Guideline: Slogan: "Don't break ES 3 scripts" (a) Don't add any new reserved words

[**DN** We went through this for ES4, see reserved words. We even shipped an implementation of that proposal in Firefox 2. The only way to add reserved words is with opt-in versioning, as in versioning – but that **is** a way to add reserved words without breaking ES3 scripts, so the Guideline is misstated here. And let me gripe once again: it was not good form, or helpful to ES3.1, to ignore our experience which we documented in the wiki. — *Brendan Eich 2007/04/21 22:05*]

(b) Add new functions/methods by using names that will not conflict with existing programs"

- solution: use namespace style naming conventions for all new functions and access them using bracket notation..
- Even if awkward or ugly

[**DN** This is no guarantee of anything. Existing scripts could use such names. This part of the Guideline is also not correctly stated ("solution"), although if one were to suspect that the idea was to vandalize ES3.1 so that no one would want to use it, it would make sense. — *Brendan Eich 2007/04/21 22:05*]

**Example: Function/Method Namespace Conventions**

Here is how the above guidelines might be applied to ES3.1 function/namespace naming:

We need to introduce new functions and members into ES3.1 without creating naming conflicts with names that already exist in the vast corpus of ES3 programs. ES4 provides for explicit namespace qualification of all function or member identifiers. Syntactically, in ES4 a member reference can be written as

```
object.namespace::name
```

For example: `obj.JSON::toString` (ES4 syntax)

[**DN** But no one has proposed putting the JSON methods in such a namespace in ES4; see json encoding and decoding. — *Brendan Eich 2007/04/21 22:05*]

By choosing universally unique namespace names (eg, use domain naming conventions for namespaces) and namespace qualified member references, new names can be introduce that will not conflict with exist member names.

[**DN** Again, this "unversally unique" assertion is a false claim. There is no restriction in ES1-3 on property naming, so someone could have written a script in 1999 that object-detects and conditionally binds its own "JSON::toString". — Brendan Eich *2007/04/21 22:05*]

However, ES4 depends upon new syntax of namespace qualification to accomplish this. This violates the "No new syntax" rule.

[**DN** ES4 violates an ES3.1 slogan, er, rule – so what? Is this impeaching ES4 by pointing to more recent, questionable ES3.1 rule-making? That's silly. — Brendan Eich *2007/04/21 22:05*]

However, there is a way to approximate this style of namespace qualification in ES3 using bracket notation and quoted strings. For example:

`obj['JSON::toString']` (ES3 compatible syntax)

If all new ES3.1 functions are named and accessed using this style then scripts using them will be syntactically compatible with ES3.

[**DN** And ugly enough to be unusable, hard to write and read for newcomers to the language, and hard to use in comparison to all the other standard methods.

It's ironic that previous proposals along the ES3.1 lines entertained adding the "array extras" shipped in JS1.6 in Firefox 1.5 (e.g., `map`, `forEach`). We added these without any regressions on the web being reported, and they indeed were then emulated by Dean Edwards on other browsers. So why the sudden fetish for ugly names, which again do not guarantee non-collision? — Brendan Eich *2007/04/21 22:05*]

However, this style access is clearly not the preferred style of member access in EcmaScript and if ES3/31 backward/forward compatibility was not an issue we would not consider using it.

[**DN** The compatibilty reasoning above is flawed, so I recommend you do not consider using it. — Brendan Eich *2007/04/21 22:05*]

When ES3.1 is probably adopted we would prefer to use the ES 4 syntax for namespace qualification.

[**DN** This must be misstated. Do you mean, "If ES3.1 is standardized and finally is adopted by enough browsers, and had we included namespace support in it, then **users** would prefer to use namespaces instead of ugly bracketed string names"? Then why not include namespaces, and indeed all of ES4 via opt-in versioning? You seem to be assuming your conclusion and then contradicting yourself. — Brendan Eich *2007/04/21 22:05*]

For this reason we believe that we are justified in incorporating the ES4 based namespace qualification syntax into ES 3.1.

[**DN** Note that namespaces and the `::` operator are in E4X, ECMA-357, already – but that they **do** entail "new syntax" in ES4 (`namespace N`, `use namespace N`). So how would you incorporate them without violating your "no new syntax" slogan/rule/guideline/whatever? You are not arguing consistently. — Brendan Eich *2007/04/21 22:05*]

This allows a script programmer who knows that he will run only on a ES3.1 implementation to use the more natural syntax.

(however, note that this approaches may force ES 4 to adopt an internal representation convention for namespace qualified properties that would be compatible with the proposed ES 3 compatible convention. The implications of this need to be further investigated.)

[**DN** No, the implications of this are clear: no namespace integrity, since namespace prefixes are just strings and anyone can forge a namespace-qualified reference using a computed property name. This idea shows a complete misunderstanding of namespaces in ES4.

Also, backward incompatibility with existing scripts that use `::` in property identifiers, since those scripts do **not** mean to segregate the part of the id after the `::` (the local name) from the part before (the namespace). So this idea shows a misunderstanding of property identifiers in ES1-3.

And just **using** a prefix in a string-identified property does not and should not create and bind (in what scope or scopes?) a new ES4 namespace object. This idea is neither compatible forward or backward, nor sane as a language design element.

See name objects for more if you don't understand property identifiers in ES3 and ES4. — *Brendan Eich 2007/04/21 22:05*]

**Example ES3.1 benefits from ES3 syntax compatibility**

(a) multiversion scripts that "light up" in ES3.1

The following will load an parse on an ES3 implementation, and will light up on ES3.1 implememtation.

```
if (typeof (Object["org.ecma::ECMAScriptVersion"]) == "undefined") {

    // use ES3 syntax to access only ES3 features
} else {
   // use ES3 compatible syntax to access ES3.1 features
   // "lights up"!
}
```

(b) backwards compatible libraries that provide ES3.1 functions in ES3

```
if (typeof (Object['org.ecma::ECMAScriptVersion']) == 'undefined') {
   // load up a set of methods ES 3.1 compatibility methods
   // (that are accessible through ES3 syntax)
}
// rest of the program does not have to worry about
// whether it is running in ES3 or ES3.1
```

[**DN** Didn't I write something like this in versioning? Failure to cite precedent is bad form in standards bodies and academic literature alike. Indeed, my example from that page shows how to load out-of-line scripts, which can contain incompatible syntax. Which shows the falsity of the "no new syntax" slogan/guideline/rule. — *Brendan Eich 2007/04/21 22:05*]

We also talked about the following and still need to elaborate on the discussion.

**Decimal**

We discussed how the IBM decimal arithmetic proposal could be accommodated in a ES3.1 context. One way ES3 is different from ES4 is that it doesn't have the concept of multiple numeric types and overloaded numeric operators. In ES4 adding decimal support is largely a matter of adding an additional numeric type to a design that already has

multiple numeric types. If we want to support decimal arithmetic types in ES3.1 does it require that ES 3.1 also adopt ES 4's design for multiple numeric types? Is there a way to incorporate decimal support without also introducing multiple numeric types.

One possibility would be to simply change the required representation of numeric values in ES 3.1 to IEEE decimal from ES 3's IEEE binary floating point. Could this be done with sufficient forward compatibility for existing ES 3 programs? Douglas said he believed that this would be Mike Cowlishaw (IBM) preference.

[**DN** Please, let's have Mike speak for himself.

Given Douglas's concern about backward and forward compatibility, we need to be careful here, at least as careful as with adding new property bindings without opt-in versioning. I know of numeric scripts on the web that depend on IEEE754 double precision, including precision limits and non-finite values. So just switching to decimal **will** break some pages and web apps, e.g., those Excel spreadsheet and Doom2 game knock-offs. — *Brendan Eich 2007/04/21 22:42*]

Making decimal arithmetic the default would be highly beneficial to the large body of naive ECMAScript programmers who don't understand the subtitles of binary floating arithmetic.

[**DN** which was the point made repeatedly in the development of ES4, see decimal. Yet we anticipated the slowdown that such a change would cause. This is all water under the bridge. — *Brendan Eich 2007/04/21 22:42*]

However, a major concern about this approach is the performance implications. For the foreseeable future, IEEE decimal arithmetic would have to implemented via a software library. Douglas speculated that such packages are on the order of 100 times slower than hardware IEEE FP for basic operations. While numeric computation is not a core usage scenario for ECMAScript there is concern that this would be too much of a performance hit for the simple arithmetic needed for basic computational control such as looping, indexing, etc.

[MFC: I don't think you need worry about performance too much. Although it is true that a worst-case add (with 34 digits in each operand, unaligned exponents, and rounding required) is of the order of 200-300 cycles, the more typical cases can be very much faster. For example, to increment an integer (x=x+1, as in the loop control or indexing case) requires two tests to check that x is an integer and after that is very fast – aboout 10 cycles in all. And if the interpreter/compiler can determine that x is always an integer it can of course be optimized to use a binary integer as noted below. — *Mike Cowlishaw 2007/04/23 01:13*]

Allen pointed out that a implementation would be free to optimize via alternative representations (eg native integers) numeric computations (and numeric values) that were dynamically determined to be integral. This is similar to the manner in which Lisp and Smalltalk implementations transparently support indefinite precision integers but use optimize representations for the most common cases involving "small" integers (small meaning less then approximately 32 bits of precisions).

[**DN** This is not a matter of conjecture. Real implementations already so optimize, but nevertheless: many real graphics apps on the web depend on IEEE double. — *Brendan Eich 2007/04/21 22:42*]

We also briefly discussed the possible of using a simpler (and faster) alternative decimal representation. Most likely some sort of scaled decimal representation implemented as an object encapsulating two native integers. (note that this is probably not enough precisions for large currency values). Our guess is that numeric experts such as Mike would not look favorably upon this approach but is does seem to be an alternative that may be worth some further investigation.

[MFC: The IEEE 754 decimal formats are exactly that – two integers. The decimal encoding uses a kind of compression to maximise precision in the space available, but this is very fast to encode or decode (2-3 gate delays in hardware, or a one-cycle table lookup per declet in software). There is also a binary encoding, but that's a lot slower for conversion to string, etc., so may not help at all. There would be many disadvantages in introducing a new format, notably different

ranges and scales causing overflow or underflow when exchanging values with other programs. — *Mike Cowlishaw 2007/04/23 01:27*]

We also discussed the possibility of adding decimal support via an encapsulating object type that could not be used with the numeric operators. All arithmetic would have to be accomplished via method invocation. We speculated that while this might be acceptable for expert users, it would probably not be used by the naive users that most need it.

[**DN** Please read, or re-read, decimal and follow the links to Cameron Purdy's blog. Java crippled itself and even expert users fail; C# did not, kudos to MS. So why would MS here toy with a regression from the C# standard's wisdom? — *Brendan Eich 2007/04/21 22:05*]

**Pragma**

Out of the decimal discussion we also talked about the possible need for ES3.1 "pragmas". For example, a pragma might be used to select between default IEEE binary FP arithmetic and default decimal arithmetic.

There was some discussion of pragmas expressed syntactically or via function invocation syntax. Allen pointed out that some pragmas setting might need to be visible to the compiler so that any function invocation syntactic approach would need to have a static interpretation. We also discussed the possibility of "pragmas" as global environmental settings. For example, switch setting to the Ecmascript engine or something specified in the header section of a web page before the first use of any script tag.

[**DN** But if you are going to add new syntax to the document head, what became of "no new syntax"? Ah, yes: in HTML unknown tags are ignored, their content rendered if they are not point tags. But how is that different in concept from versioning? It's not. — *Brendan Eich 2007/04/21 22:05*]

## Discussion - Day 3

### 'this' binding

Lars, Graydon and Jeff discussed `this` binding rules, since the existing material is a bit haphazard in the reference implementation. The consensus we arrived at is that the RI evaluator should replace its current dynamic state (encapsulated in a `scope` value) with a register set that has both a `scope` entry and *also* a `this` entry (and also a call-stack entry, since we are implicitly propagating one around with push/pop calls already, for debugging).

Then, rather than storing `this` in a scope, we will resolve `this` expressions by looking in the `this` register (and if the register is `null`, the `this` expression evaluates to the global object). Function closures will contain an `OBJ option` field indicating whether they clobber `this` on entry: methods of classes will have the field set to `SOME instance`, other functions will have the field set to `NONE`.

### 3.1 status

- Discussion drilling down into the principles of 3.1
- Feedback is that it is important to parse using 3.0 syntax (no new syntax!)
- Desire to have a quicker (than ES4) path to deploying new features
- Provide functionality in the form of new functions so that there is "no new syntax!"
    - Graydon points out that whether or not the program is syntactically compatible content producers will have to fork their programs to provide code that is compatible with new and old browsers
- Reasonable to add new syntax to 3.1 if that syntax if that syntax provides high programmer value (convenience). Decision based on programmer willingness to use new syntax on 3.1 browsers. Need to standardize the new syntax now so that browsers can implement and users can use at their own time.
- 3.0 scripts must continue to to run

- Specific guidelines:
  - no new reserved words
  - no new function names (see discussion of namespaces below)
- Use ES4 style namespaces to control visibility of new names
- Decimal arithmetic. ES4 has concepts of multiple number types, numeric modes, etc
  - Considering changing the default numeric type from binary to decimal floating point
    - This would break the web and be difficult to implement efficiently on small devices especially
  - Perhaps using a pragma would be the way forward. Need to discuss more
- Would like to articulate more precisely what the principles are and evaluate feature selection