Minutes of the: Ecma TC39-TG1

held in: San Francisco (Adobe)

on: 16th March 2006

Attendees

- Francis Cheng, Adobe Systems
- Jeff Dyer, Adobe Systems
- Brendan Eich, Mozilla Foundation
- Cormac Flanagan, UC Santa Cruz
- Gary Grossman, Adobe Systems
- Lars Hansen, Opera Software
- Dave Herman, Northeastern University
- Graydon Hoare, Mozilla Foundation
- Blake Kaplan, Mozilla Foundation
- Dan Smith, Adobe Systems
- Edwin Smith, Adobe Systems
- Rok Yu, Microsoft

Agenda

Thursday

- Lunch & Logistics
- Next meetings
- Scope of edition 4
- Review proposals
- Dinner

Friday

- Review proposals (continued)

Notes (March 16th)

- Jeff proposed that we look for ways to constrain the scope of Edition 4. Edition 4 has been in progress since 1999, and we should figure out how to get something out the door!
- Could we intentionally do a two stage release, with Edition 5/6 being what people really implement?
- There is a risk if we rush of releasing something that has serious problems. Rule 1 is don't screw it up.
- Our concern should be soundness, not surface. Type system is soundness, generators is surface.
- Some long-standing bug fixes should be considered for that short list.

- Lars is concerned about the language being too complicated to implement in the constraints that Opera has to work with, such as mobile devices. Adobe has some of the same constraints. For instance, do namespaces incur an unreasonable hit on property lookup performance?

Bang and Tilde

Dynamic languages can have very little type analysis, figure out as you go.

Problems encountered when doing Bang

Reviewed the namespace shadowing proposal namespace_shadowing

Reviewed the namespace shadowing problem

Member hijacking: No ability for subclasses to take over names

```
class A {
   static var x = 1
   function f() {
     print(x)             // 1
     with(this) print(x)  // 2
   }
 }

class B extends A {
   var x = 2
}
var a = new B()
a.f()

scope chain
values [ global, A, B, f ]
types [Object, class, A, f]
```

Improving type system

- Graydon thinks Edition 4 as currently implemented in AS3 sounds like Java 1.4 and he'd like to see it pushed a little towards Java 1.5.

Versioning using Namespaces

- Adobe experimented with namespaces as a mechanism for versioning and didn't get very far.
- Waldemar's model doesn't seem very scalable for versioning. Decorating declarations with 10+ namespace attributes doesn't seem acceptable.
- Removal is rare and nonexistent on the Web. Renaming means you probably made a mistake but you should live with it. Mainly additions are the only thing you worry about. Both JS and Flash have had to worry about conflicts introduced by additions.
- "The highways are littered with the bodies of those who have tried to solve the versioning challenge, so don't try to solve that one"
- Do we end up with versioning metadata bits?
- Improved namespace mechanism that lets one redefine public?

- Do interfaces help? You could have concrete classes hidden away in factory pattern and use only interfaces. Factory pattern is kind of clunky, one of the reasons people find this approach unattractive.

Multiple Compilation Units

- Reviewed Lars's proposal multiple_compilation_units
- It's a "big bug" in Edition 3 that it doesn't talk about multiple compilation units. There are complexities, like if you merge two Edition 3 scripts, there won't be a compilation error, but the program may no longer behave correctly.
- Adobe implementation implements packages as namespaces. Ambiguous binding error is reported if a public name and a package-imported name conflict.

Foundational Issues

- Type inference is on the chopping block for Edition 4, but we should discuss it a bit further.
- Cormac has talked about more unified approach to type system so that static and dynamic are not two different dialects.
- Desire to not have to annotate everything. Possibility of method-local type inference. C# has initialization-based type inference.
- What is our compatibility goal?
- Is the goal that all code continues to run?
- Compatibility kept ECMAScript Edition 1 and 2 from getting second-system syndrome.
- Now there is a pile of code on the Web and you'd like to tell people not to have to rewrite.
- There is a high premium on compatibility. You can try deprecating things. Maybe that means 10 years from now they'll be gone, maybe not!
- Lars thinks that type annotations are mostly useful for programming in the large, so that function signatures have types. But then type inference local to methods doesn't seem very useful.

Dave Herman's Contracts Proposal

- Reviewed Dave's contracts proposal contracts
- Dave: We should worry about maintaining two different dialects, basically ending up with two different specs. We should avoid having a cliff between the two, where it's hard to make the leap between small apps written in one to large apps written in the other. Can we define one core language that both modes map to? Further, can we align them even more?
- There will always be split between uses of JS. Traditional development environments where people are developing and releasing. In ActionScript, people use a compiler and development tools and can release something already typechecked. With the Web, you've got this source release and any static checking you do is already too late... if it finds an error, it's essentially a runtime compile error. In a sense, those are conflicting requirements.
- This idea of having a dynamic interpretation of type annotations in that environment has a lot of sense to it, but as long as we're doing that, we probably have to consider that the meaning of programs is slightly different in those two different modes.
- We might be stuck with some notion of modes, but avoiding a total split in the language is a good objective. We'd like to be able to have static types where we can, and also more precise contracts that might not be statically decidable and not entirely predictable by a compile-time type checker.
- We'd like to expose both of those, but also have those in a world where we don't have such a heavyweight compile process. If we have both static and dynamic type annotations and have those expressed in terms of contracts in the core language, that is attractive. Dave and Cormac are working on formulating this proposal.

- We will have issues with compatibility with old code. What are the boundaries between typed and untyped code, or statically-typed code and dynamically-typed code?
- Contracts can be compiled into code that doesn't have the contracts... basically assertions and blame locations.
- A contract error will be an exception, and since there are stack traces in the language, there will be a stack trace as well. Message will be "C screwed up because C screwed up its contract."
- A new buzzword is "optional type systems". A type annotation may be enforced differently. A particular implementation of JavaScript may test the assertion earlier on, but not required of the semantics. In a Web browser, it may not catch the errors early, but in an ActionScript IDE, it would. It's just a matter of when you catch the errors, which is what Jeff has been saying all along.
- There are usage modes as well: Use Bang when you compile ahead, use Tilde when you're loading a Web page.
- For programs that pass the static checker, programs should mean the same thing when executed. Static checker catches errors earlier... that's it. When you run in the web, it doesn't give you those early, but it does catch them when they happen.
- Dave: I would argue that using contracts is better than having a hard distinction between Tilde and Bang. Right now if we load Tilde code and then it interfaces with Bang code, it's not going to work well because Bang has a very rigid type system. With contracts, we can put contracts at the boundaries between the strongly and weakly typed code, and improve it incrementally.
- Lars: Perhaps we should impose some restrictions on things like with and eval, because with/eval basically bypass safety mechanisms. with and eval basically turn the static language back into a dynamic language.
- Idea of safe/unsafe module. What if you can't use with or eval unless you say the class or module is unsafe?
- Still can be problems with something crawling unsafe up the hierarchy and affecting something safe, and altering something that the type checker blessed as safe.
- Our stipulation should be that unsafe code cannot undermine the type safety of safe code.
- In Adobe's implementation, if one writes

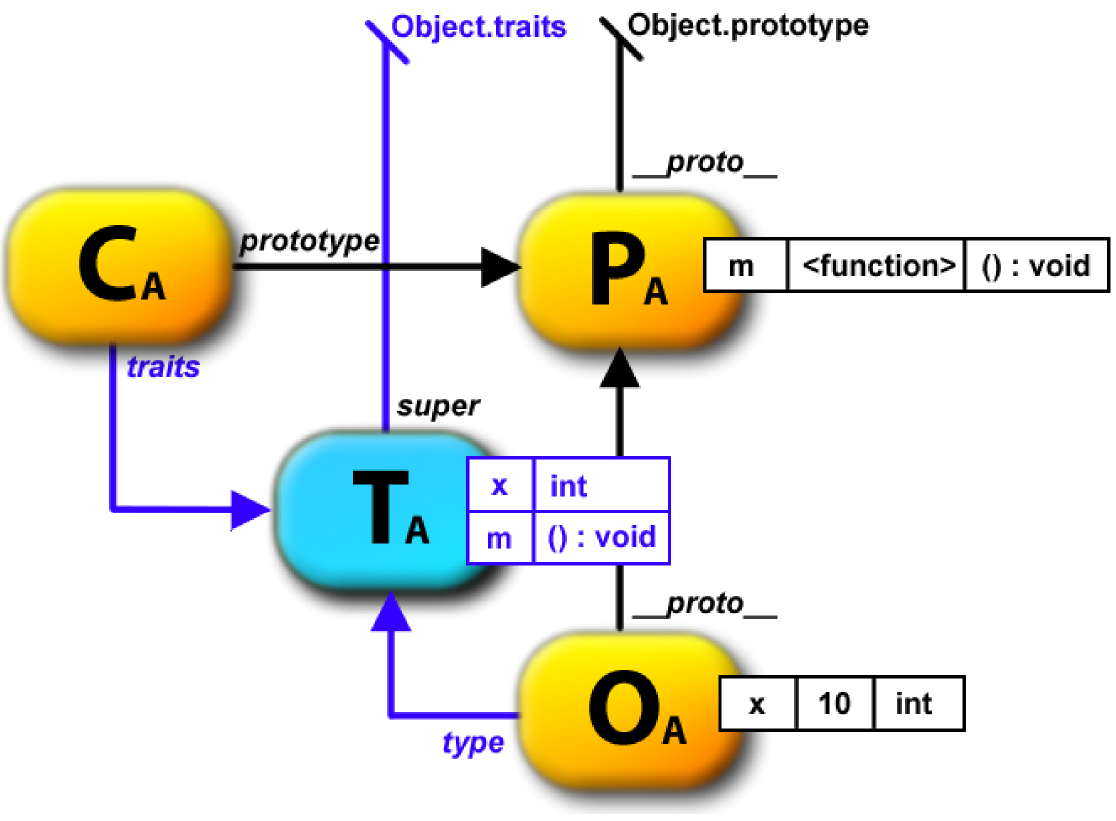
```
class A {
  var x
  public function err() {
    // error because no y in scope
    y = 3
  }
  public function noerr(y) {
    // no error, even in bang, because o might have a y
    with (o) {
      y = 3
    }
  }
 }
```

- Adobe didn't see a need to mark class A "unsafe" in this situation, although the idea could be useful.
- Should eval be an operator instead of a function?

Drop Traits

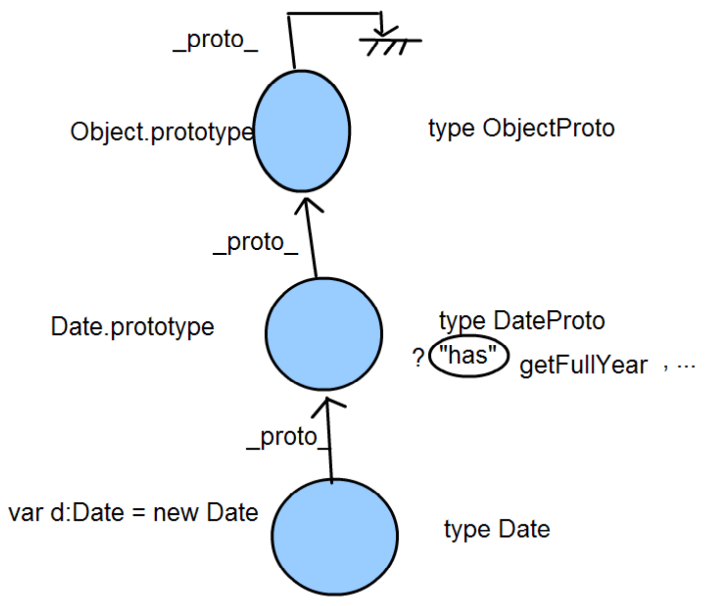
- Reviewing proposal drop_traits
- Jeff: Is it just a spec issue whether we describe the language with traits or without traits?

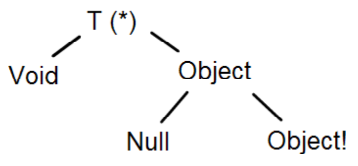
- Brendan: Motivation of the drop_traits proposal was to simplify the spec and make it more like Edition 3, to apply Occam's Razor.
- Jeff: I liked the idea of dropping traits but if it makes the spec more complicated, it's not worth it.
- Graydon: I now feel ambivalent about what I wrote in the proposal; my first instinct on reading about traits in the spec was to see if we could simplify things by doing without it. But we should do whatever is simplest.
- What are traits?



- Does it make it easier if we put the traits object in the scope chain?
- Meta-slots are probably equally required as traits for certain situations. Which is a simpler data model for the spec algorithms?
- The big question is the object model used by the specification. Affects the model of scope for the language.
- We should be able to write the specification in a way such that the implementation can be traits-based or meta-object-based.
- Prototype presents a complexity in the traits model. It was not possible to make the prototype a subtype of the instance type. Can it be the same type? That breaks down because you can't go into Date.prototype and get its proto and expect to get to a Date. Inheritance and polymorphism: If I'm 3 or 4 levels down the inheritance tree, I have to hop 3-4 to get to the function pointer, but I don't know how many hops to make.
- Edwin: VTables are purely an optimization. VTables are only needed if you want to do early binding. If you actually want to do early binding, the runtime needs vtables.
- Brendan: If you do what Lars does and do a lightweight dynamic implementation, is there a problem?

- Edwin: If the lookup is dynamic, then it works.
- In Edition 3, Date.prototype is of type Date. Class Date has an internal Time property. Date.prototype has the internal Time property too. The only difference is that Date.prototype has a proto pointing to Object.prototype, not to Date.prototype.
- If the type of proto is Object, in the model of using prototypes to encode type information, then what that means is... we gave up and let proto be the type loosest enough to work for all the cases. We couldn't do reasoning about what proto can point to.
- VTable as implementation technique will
- As informative exercise, take proposal and implement it directly and try to do early binding. Edwin thinks early binding cannot be implemented without vtables. VTables cannot be simulated using type-constrained slots. They can't be simulated with traits objects, except you can find the associated vtable from the traits. To support early binding, you need to go from proto to traits, then from proto to vtable.
- In Adobe implementation, object points to vtable, vtable points to traits. As opposed to object points to traits, traits points to vtable. Just because we consult vtable more often than traits.
- To close out this discussion, we need a study group to go off and investigate the hard cases to figure out where the model breaks. Jeff, Graydon.
- Does the spec on the wiki contain algorithms that use traits? Jeff: Yes.





Type	_proto_
DP	Object
Date	DP

Spec Notation

- Overall question of how we write the spec. There will have to be prose regardless. How much do we want of rigorous provable notation?
- Right now, we have Dave's CEKS notation, but that isn't typographically unambiguous and isn't complete. Is that the right road to go down? There is a risk of us getting blocked on that.
- How are Waldemar's semantics?
- Edwin: Perhaps it's best to simply do a reference implementation?
- Lars: In Java?

- Dave: The reduction semantics, single-step notation can be written and actually result in a free interpreter. So it can actually serve as a reference implementation.
- Edwin: Having a reference implementation based on a formal notation is definitely money and can find really hard bugs in the spec. This was very helpful at one stage for Java.
- Dave is going to try and actually write a real formal semantics for the language, with Cormac's help. But we don't need to make it the critical path for the spec.
- Jeff: What does the spec look like that, then? If we have formal notation, can we write English prose for the spec and have that be enough?
- Dave: If the formal language is clear enough (the devil is in the details, some wiggle room), then specifying each piece of the language can be a process of: "Section 8.9, evaluation of throw." Then, a picture of the formal rule, and English discussion under it that explains it in prose. That is close to the way Edition 3 works, except the "formal" language was somewhat informal... the line-numbered pseudocode. That general approach, however, is very useful for implementers.
- Brendan: We found bugs in the E4X spec where the steps say one thing and the prose says another.
- Dave: I had a professor who referred to it as Biblical exegesis. One column has the Bible verse and the other has the explanation.
- Jeff: My hope is that with the formal notation backing us up, we could then have English prose for the description that ordinary people could read.
- Edwin: I wouldn't expect that the formal notation is any more readable than our mangled-up C++ code
- Graydon: Well, the fact is people CAN read mangled C++ code, and are fine with picking through the code in reference implementations. The operational semantics shouldn't be harder.
- Dave: One of the best things about operational semantics is that you find corner cases just in the act of writing out the notation.
- Brendan: The value is in the journey, not the end.
- Edwin: A real easy way for the process to work is to treat the formal notation as just another implementation of JavaScript.
- Graydon: But it still leaves open the question of, is there anything other than English prose that goes into the ECMA specification that we submit? Suppose the formal spec stalls for 3 years. Do we have pseudocode, scratch diagrams, anything?
- Jeff: I think we'd have to have pseudocode.
- Graydon: An important test is whether Dave is the only one who can write the formal notation. If Dave is the only one who can do it, then we've probably chosen the wrong notation. If we can do it, then idelaly I'd love to submit the spec as a bunch of fragments of Scheme code.
- Dave: I don't think we'd actually want to use s-expressions; I would play with the visual presentation.
- Graydon: At least an s-expression can be typed, talked in a wiki, talked about. A big problem is characters that nobody even knows how to typeset. They are a big impediment.

is/as/to

- Reviewing proposal is_as_to
- The user-defined "to" conversion doesn't work in the static profile. The static mode will reject pretty much any program that makes use of the user-defined "to".
- Putting type annotations on an API is very valuable for informing the tool chain, giving it the ability to do things like code hints, and can be helpful for performance. But it would be very unfriendly if you are forced to write String casts.

- In the ECMA spec, the as operator throws. In C#, it does not throw. In Adobe's implementation, we originally were not throwing.
- Edwin proposes that we leave the as operator out of the spec. We need a downcast operator. How about cast?
- The cast operator would cast, and throw if failure. Removes the ambiguity that we currently have with T(x) as the cast operator, with builtins like Array.
- If "as" was the downcast operator, "as" was considered harmful because people probably often wouldn't check for a null result.
- Why do people use dynamic_cast in C++? Sometimes it's defensive coding, unwillingness to use exceptions. A lot of people write that out longhand in Java code too.
- What if switch class became an expression instead of a statement?
- Do we need cast and to?
- cast is a cast through the lattice. to is a conversion, which is not modeled by the lattice.

Nullability

- Boolean? is higher on the type lattice than Boolean, because it supports more values.
- Object! is lower on the lattice than Object, because it doesn't support null.
- There is a proposal to put the ? FIRST in the type name, like ?Boolean instead of Boolean?. This would have to be done to eliminate ambiguity with the conditional ?: operator. But Rok says he will dig up the C# rules on how they avoid this problem.
- Capitalization of names: Edwin would like to go with Dave's proposal but spell int lowercase. Brendan concurs, we come from a C tradition and int is lowercase there and if we add double later we'd lowercase that too.

Parameter Types

- Still making a case for parameterized types in the next edition.
- Polymorphic arrays are the big one. Saving that for tomorrow morning for a deep-dive discussion.

Meta-Objects (Reflection)

- A java.lang.reflect type API. Needs to be fleshed out more.

Built-in Classes

- We want to define the built-ins as classes, must figure out loose ends on that.
- The Adobe implementation has a "native" attribute. It probably makes sense to propose that as part of the spec.
- "dynamic static" looks weird but makes sense. We're not about to change "static" to "class" and break 3 decades of C tradition. The word static is horribly misused but it's pretty much stuck now.
- Ed is on the hook to finish the table of specifiers.

Type Inference

- Moved out of Edition 4

Lexical Scope

- Getting scratched from Edition 4

Contracts

- Needs further development

Numeric Types

- Graydon reports that IEEE 754R is back on track.
- We would like to get decimal in somehow.
- There are a number of tough issues with conversions. How do the operators behave? What gets promoted when?

```
function f(a:dec, b:dec) use decimal
{
   return a + 1.5
}
add.dec(a, 1.5)
```

- Implementing a full numeric tower is a very tricky proposition.
- With decimal, follow the same pattern that we use for conversion between int and Number. Implicit conversions in both directions. But math converts towards Decimal, so if you really want double-precision math, you must cast to double.
- Not exactly the same pattern as today, since if you add int and int, you get double, but if you add double and double, we're saying you get a double, not a decimal.

Notes (March 17, 2006)

Abstract Syntax Tree

- In a lot of language specs, the grammar doesn't actually parse any programs that people write.
- We want to produce a parser that emits abstract syntax trees. It could be written in Edition 3.
- Narcissus parses source, builds trees, and then evaluates the trees. No intermediate representation. Edition 3 plus a few extensions.
- The AS3 compiler has a "show trees" mode which can display the state of the abstract syntax tree.

Type Parameters

- Note that Adobe AS3 supports conditional type expressions.
- The biggest motivation for doing this is for arrays, and containers in general. Want to be able to specify type of array, and have automatic cast on in/out. We want to help people avoid the perils of Arrays.
- We don't want to do something complex; don't want to go down road of Java 1.5 with fancy dispatch rules.
- We could do what Waldemar did with const parameters. That can satisfy the requirement that later type annotations have a constant expression.
- Syntax: Could use ':' followed by some kind of bracket; that would be unambiguous.
- We should be concerned about readability to the common user. Angle brackets don't look too weird but might be unfamiliar with non-C++/Java/C# programmers.
- People generally agree that <> is what people expect, at least C++/Java/C# programmers, but Jeff objects that it alien to ECMAScript customers. Also, it may cause some weird ambiguities.
- Is there a lookahead problem? Do you not know that you're parsing a type identifier until the closing brace? You may need to do things more bottom-up. Other languages distinguish types more so than our language does.
- Parameterized types are IN for Edition 4.

- It would be good to get signoff from our language semanticists (Dave or Cormac), but Dave has read the proposal on the wiki.
- Every C-like language has added these at some point, so it's a common problem and people have achieved solutions.
- Ed wonders whether this is a slippery slope.

```
function f(t) {
   class c(t) {
     ...
   }
 }
 f(String) == ?
 f(String)
```

Enumerations

- We have a reserved enum keyword that we haven't done anything with.
- JScript.NET and haXe both have enum support.
- Graydon likes the haXe support, even better than his original sketch.

```
&[red, green, blue] = enum(3)
```

Compact Profile

- If you look at browser DOM, how many use real methods vs. prototype?
- It depends. Mozilla uses prototype.
- IE is a bit special... alert is not a function object; you cannot apply alert. And there are weird things in IE like assigning a value to alert and then still call it.

Operator Overloading

- Deferred for Edition 4. Interesting to experiment with.

Enumerability

- "dynamic" attribute on declarations implies DontEnum
- See discussion below on Built-in classes.

Built-in Classes

- Any access specifier (public, private, protected) means the definition should not be enumerable.
- Dynamic properties in AS3 are always public. If they supported namespaces, then some complications arise: like, for...in must iterate with QNames.
- DontEnum is clearly a bit, not a namespace, in Edition 3. How to express the bit?
- AS3 has implemented the E4 Netscape Proposal as far as enumerability, with the exception that we didn't actually implement Waldemar's "enumerable" attribute.
- Really, what customers want may be a real reflection API as opposed to for...in and enumerability.

```
function Array.prototype.t() { ... }
```

- This is equivalent to

```
Array.prototype.t = function() { ... }
Array.prototype.t.magicEnum = false
```

- IE can enum the builtins
- Does this mean function can take any lvalue as the function name? Or, just dotted. Probably need some restrictions.
- New Idea: We extend Object.prototype.propertyIsEnumerable to be a write operation, not just a read operation.

```
Object.prototype.propertyIsEnumerable
Usage: obj.propertyIsEnumerable(name, enumerableFlag)
```

- This lets user code explicitly toggle DontEnum.
- This replaces the Array.prototype.t syntax above, which was too subtle.

Unicode

- No opinion presently, as Michael Daumling is not present at the discussion. Will take it up next time.

Extend Regexps

- Brendan has action item to figure this out.

Catchalls

- This is the proposal to specify "catchall" get, set, and method-call accessors.
- There is some concern about interaction with type system, and questions about how identifiers are represented (notably for E4X), but mostly acceptable.

One-off getters

```
o = { p:42,
      "q!":43,
      0:44,
      x getter:function () { ... ret ... }
      x setter:function (v) { ... }
get x() { ... }
set x(nx) { ... }

o.__defineGetter(name, fun)
getter = fun
```

- Brendan feels that we should have a mechanism for doing one-off getters/setters in the language. Classes shouldn't have all the fun. Forcing people to write a class to do accessors is against the opportunistic spirit of the language.

Decimal

- decimal, the proposal to add IEEE754r-style decimal floating point to the language.
- Proposal needs revision.
- Status: really only depends on deciding how to do operators, otherwise accepted. Needs proposal from MFC@uk.ibm.com and/or Graydon
- Some info on Microsoft's System.Decimal type: http://www.cs.berkeley.edu/~ejr/projects/754/email/msg01909.html

Over a decade ago Microsoft released a numeric format (OLE Decimal) which is in common use and is similar in its mathematical properties to that being proposed for the IEEE 754R standard. As part of finalizing the ISO/IEC 23271:2003(E) standard, IBM lead an effort to harmonize the mathematical properties of an updated version of that format so that they would match those of IEEE 754R (as it existed at that time). This updated type, known as "System.Decimal", was adopted into ISO/IEC 23271:2003(E). Microsoft released its implementation in 2003 as part of the .NET Framework version 1.1. The type has been widely adopted by users of the Microsoft programming platform on systems ranging from cell phones to personal digital assistants to laptops to desktops to servers. The successful customer adoption of System.Decimal relies in no small part to the fact that, while some of the mathematical operations have subtly changed their behavior, the stored data format is unchanged from its predecessor.

(ISO 23271:2003 is the Microsoft CLI standard specification.)

Hashcodes

- Reviewed proposal hashcodes, the proposal to provide objects with a mechanism for producing integer ids.
- :-) Proposal accepted.
- Definitely will accept hashcode() function, deferring / ignoring "indexing by value".

Block Expressions

- Reviewed proposal block_expressions, the proposal to add a lexically scoped expression form that is not hoisted.
- :-) Proposal accepted.
- Block expressions are an attractive thing to have. It's a powerful construct. It has the potential to make code within methods more modular. We think we want this in Ed 4.

With-Bound Variables

- Reviewed proposal with-bound_variables
- This one can be retracted. It is backwards compatible and doesn't genuinely fix the issue.
- We probably won't want to do this ever, so terminated and not deferred.

Destructuring Assignment

- Reviewed proposal destructuring assignment, a proposal for syntax to pick apart arrays and structures (and subsume group assignment).
- :-) Proposal accepted.

Bug Fixes

- Reviewed proposal bug fixes, a proposal to make a few small, "easy" fixes to ECMA-262 Edition 3.
- :-) Proposal accepted.

Standard library issues

- (!) Proposal needs revision. date_and_time, or, Date is so 1995, we can do better.
  - **status:** need nanosecond()-ish facility in std. library, leave remainder to user ecology?
- (!) Proposal needs revision. string formatting, some string formatting (printf-like) function or class.
  - **status:** lars has a proposal, needs to be spec'ed more.

- (!) Proposal needs revision. containers, some standard container types (hashtables, red-black trees or skip-lists, sets, queues).
    - **status:** desires hashcodes, type parameters; otherwise can be done by ecosystem.
- (!) Proposal needs revision. networking, some network-access classes.
    - **status:** filesystem and network seem like host object issues
- debug helpers, some facilities for standardized logging, assertions, debugging.
    - **status:** brendan thinks very important, graydon will produce a concrete proposal.