Minutes of the: Ecma TC39-TG1

held in: Redmond (Microsoft)

on: 21st April 2006

Attendees

- Jeff Dyer, Adobe Systems
- Rok Yu, Microsoft
- Brendan Eich, Mozilla Foundation
- Graydon Hoare, Mozilla Foundation
- Dave Herman, Northeastern University
- Lars Thomas Hansen, Opera Software

On phone:

- Francis Cheng, Adobe Systems
- Pratap Lakshman, Microsoft
- Blake Kaplan, Mozilla Foundation
- Cormac Flanagan, UC Santa Cruz

Agenda

- date literal syntax
- type system
- type parameters

Discussion

- Dave: packages are only compile time?

```
package p { }
var p = {x:3}
// now an x is added p somehow (how?):
import q
// what does this mean?
p.x
```

- namespaces and packages

```
namespace N;  // makes a compile-time const N = new Namespace(unique)
package p { } // makes a compile-time const package name (could be dotted)
```

- the names are disjoint, so you could have namespace N and package N { }
- Dave: good namespace use cases?

- Jeff: cross-cutting AS3::Object, etc. for early binding
- Brendan: 1) MOP::iterator; 2) builtin::hashcode
- Property id is pair (namespace, string), call it N::S for short

```
I ::= N::S
E ::= ... | E1[E2::E3] | E1.E2::E3
```

- General idea is to have some E known at compile time. The pragma use namespace N requires N to be a compile-time expression.
- Do we need run-time namespaces? They come in through eval and E4X anyway. eval can't open a namespace in its caller.

- Doug Crockford's call for `typeof []` == "array", but is should help avoid that
- What about code that hacks String = Array? Editions 1-3 allow this, but specify "original String.prototype value" be used for automatic constructions, leading to incoherence. Can we make the standard class constructors {RO,DD} in the global object?

- More namespaces
- open namespace priority based on explicit before implicit open
- or using two scopes/objects (package or scripts or prototypes)
- Lars has a clarification issue: should explicitly opened ns shadow implicit

---

- Doug Crockford's call for unreserved words in object literal ids and to right of dot
- counterexample in light of automatic semicolon insertion:

```
  foo = "hi".     if (bar()) ...
```

- is this different because |if| is already reserved? wish to avoid degrading error reporting
- If we can avoid more than one or two tokens of lookahead, and the rules are clear and simple enough, then yes

---

- More namespaces
- AS3 makes different files have different implicitly open internal scopes

```
package p {
  var x = 20
}
```

```
public var x = 10      // public required if this is in a different file
```

```
{
  import p.x
  print(x)             // want 10, but we get 20 if p is in a different file
}
```

- builtin classes
- Discussion about prototype – does it allow static shadowing checking? yes and type checking, if the type annotation

- Only if DontDelete is included
- Dynamic prototype defeats static shadowing and type checking. So dynamic doesn't make sense as a declaration qualifier, and we should avoid it, even though that incompatibly would make ES3 builtin prototype properties DontDelete. We think we can get away with this change.
- Still need dynamic for class, but only for class.
- Lars proposes

```
deletable / fixed
writable / readonly
enumerable / nonenumerable
```

- But if we boot dynamic from member qualifiers, we have

```
const
prototype
static
```

- and the combinations, we think, make sense now (Graydon is gleefully editing the table to remove `dynamic static`).

- Type system for initialisers. Idea #1 (link to it here) seems best.
- Dave: lacking a single default value for non-nullable user-defined types, runtime error to refer to a member var before it has been set.
- New proposal to capture heterogenous array type:
  - Array[[int]] is array of zero or more ints.
  - Array[[`int, String, *]]` is array of two or more elements, the first

an int, the second a String, and any at index 2 or greater of top type.

- Rok objects, arguing that Arrays should be typed homogeneously.
- Dave raises non-nullability: are holes in sparse arrays undefined or null?
- You can have a homogenous array of ints, but if it is sparse the holes can't have a value in the type's value set.
- Do we need tuples? Arrays can be used like tuples but you don't get length constancy and length checking. Examples:

```
Array[[int, int, int]] <: Array[[*]]
Array[[int]] is a subtype of Object[[#:int, length:uint]]
```

- Using more concise notation where # is 0..(2^32-1):

```
[int] <: {#:int, length:uint}
```

- Rationalize Array and other objects as Records. What about structural vs. nominal typing?
- Discussion of bang again – var i : int = foo() where foo returns * is ok. Another example: if (foo()) ... is ok, var b : Boolean = foo(); if (b) ... is too, but some people want the latter to be an error. Lars points out that a correct re-factoring would be var b : * = foo(); if (b) ....
- Destructuring review: still wondering if there is a less backward-looking form of object destructuring.

21 April 2006

* Move weekly teleconference to Wednesday 10:00-12:00 Pacific. * Lars asks about Opera hosting June face-to-face? Sentiment favors keeping Opera-hosted meeting in July.

Agenda

- More type stuff.
- `enum` come-back?
- Michael's operators proposal.

Discussion

Lars presents results for structural typing of initialisers vs. named types.

Controversy: implicit vs. explicit contextual type narrowing in initialisers.

Three options:

- Must use `: contextually` after `{f: 37}`
- Must use `: T` for some record type
- In contexts where type of left-hand side is known, contextually is implicit
- Lars: evil, because E3 vs. E4 mixtures change incompatibly from E3 only
  - General agreement after some discussion
  - Type inference in general, at scale, requires static typing
- Can `let` and `var` use initialiser context implicitly?
  - Perhaps, but we don't implicitly type `x` as `int` in `var x = 42`
- Record types can be supertypes of classes
  - How much type-checking pain is this in default dynamic language?
  - Lack of inference leads back to nominal typing

Talk evolved through discussion of the necessity of structural typing for many poorly advocated "duck typing" use-cases, a review of switch class, and a grand unification of structural types as follows:

```
type U = (A, B, C)          // Sum of A, B, and C
type R = {p:int, q:String}  // Object with at least p and q of given types
type A = [int,,String,*]    // Array of int, *, String, and 0 or more *
type F = function(int):int  // Function mapping int to int with this:*
```

These can be composed. They're finite by outlawing `type T = (int, T)` or more obscure recursion across packages.

Lars writing this up at type system.

Would we benefit from `(type T)` as a unary expression (parenthesized for clarity)? We are deprecating `typeof` with a bug fix for null.

Constructor restrictions against this mutation: they are not allowed to call methods, just statics and global functions; not allowed to pass this or super anywhere. Dave is updating nullability. Jeff will vet Flex SDK against this if possible.

Michael's operators stuff, hashed out with counter-proposal and agreement on public static (slightly magic) functions.

Unicode spec: Lars suggests specify both UCS-2 and UTF-16 indexing, and let the browser implementation choose. The market will sort.

Brendan solicits a JScript.NET compatible `enum` proposal (see bottom of switch class for msdn2.microsoft.com doc link).

Bugfixing the spec: get the wrong bits removed, link to proposals where spec gaps remain that we are not ready to fill.