Minutes of the: Ecma TC39-TG1
held in: Phone Conference
on: 27th January 2006

Attendees

- Brendan Eich, Mozilla Foundation
- Ed Smith, Adobe Systems
- Graydon Hoare, Mozilla Foundation
- Jeff Dyer, Adobe Systems

Agenda

- Make it up as we go
- Will get one together in advance of next meeting

Process Stuff

- Kudos to Graydon for setting up the wiki
- ECMA wants “visibility”, meaning:
  - Minutes for each meeting
  - Word doc updates occasionally
- Why not make part of the wiki readable to the world?
  - We could blog about it
- Graydon asks about reserving new identifiers
  - Can’t in web embeddings without new version selection
  - We are trying not to reserve if keyword-in-context suffices
- Jeff: how do we call out what is agreed on vs. not
  - Graydon: categories such as proposals: vs. spec:
  - But the spec: was imported wholesale, yet not all agreed on
  - Proposals are good for small, readable straw-men
  - As they become agreed upon they move into the spec and may cause sweeping changes to the spec
  - Spec therefore will not be frozen to the same degree everywhere
  - So we want a way to call out less-agreed-upon parts of the spec

Type Annotations

- Graydon’s proposal: is as to
- Brendan proposed operator “to” for explicit conversions
- We affirmed that “as” is not the right operator
- Ed proposed: “switch class (x) {case C1: ... case C2: ... etc.}”
  - Order of matching is order of cases, not sorted by <:
- Need a way to write non-nullable T: T!
  - Ed: T! means don’t call “to T” conversion, just do <: T
Jeff: what about “: T! means call ‘to T’ but throw if null results”
Graydon: “x to T” should result in T, not ?T – T has to include null if that’s what you wanted
Agreed that we want something like T!

Do we want ?T to complement T! for symmetry (syntax is placeholder/strawman, don’t panic)
Jeff: overcomplicating the language for little return
Ed: maybe add string, boolean, etc. as non-nullable counterparts boxed by String, Boolean...
Brendan: we don’t want boxing, so make boolean <: Boolean & Null <: Boolean
Graydon and Brendan: keep symmetry
Jeff: but names are asymmetric as to nullability: Boolean, Double vs. Object, String
Brendan: want notational symmetry – ability to be symmetric in User types (Color, Complex)
as well as in built-ins, with nullability or not according to pragmatics
Graydon: Boolean! === boolean
Ed: Edition 3 Boolean is not the same, however
Brendan: True, calculated incompatibility – we agreed several times to get rid of boxing
(mutable primitive type wrappers), let’s not go backward

User-defined “to” could preempt annotations from checking <:?
Brendan: this loses something valuable – ability to cast rather than convert – and splits User
from non-User
Jeff: if compiling in bang you get <: but in tilde you get “to”
Graydon: so do you want to remove implicit conversions in bang?
Jeff: no, need implicit when converting from unannotated slot, and among numeric types,
and anything to Boolean – based on user feedback
Graydon: if User type has “to” conversion, will it be invoked in bang?
Jeff: yes, if statically sound
This is different from Graydon’s proposal as written before the meeting, so he revised it
immediately and updated the wiki

Static mode tends to make users over-annotate
Brendan says this makes migration hard, proposes we at least think about some kind of
inference
Interface for unannotated slots breaks duck typing in bang, so that’s a problem
Graydon says this conceptually brings in interfaces again

Compact Profile

Ed brought up lack of complete method name-to-slot optimizability in CP
CP doesn’t say “can’t shadow prototype functions” – should it?
CP do this without breaking method extraction meaning this-binding?

Should we do a CP for Edition 4
CP for Edition 4 could be done by subsetting/restricting only
Graydon: this binds to type and value, or one of type and value, or neither?
To value implies by type, backward compatibility requires neither, so the question is do we
need this binding to type? Probably not.

Host Objects

I want to restrict how “host objects” might be different
Especially for callable objects
  e.g. apply is not an operator, host callable doesn’t delegate to Function.prototype