Minutes of the: Ecma TC39-TG1
held in: Phone Conference
on: 24th February 2006

Attendees

- Brendan Eich, Mozilla Foundation
- Dave Herman, Northeastern
- Graydon Hoare, Mozilla Foundation
- Rok Yu, Microsoft
- Ed Smith, Adobe Systems

Agenda

- Introductions, Maciej?
- Go through proposals. Make sure each proposal has an owner.
- Split proposals out as necessary, e.g. syntax invented for builtin classes.
- Review recent changes on the wiki.

Notes

- Brendan to collect small fixes to Edition 3 under clarification issues.
- Dave wonders whether we are going to defer blame and better exception annotation to contracts. Need a proposal that is not deferred.
- This leads to: want a fresh proposal for static vs. dynamic modes. Graydon added.
- Unicode issues, non-BMP code points being indexed and counted. Contentious.
  - rok points out that concatenating invalid surrogate pairs doesn’t throw now, would throw in proposal.
- Catch-alls look ok, but should be possible only on dynamic classes. Ed observes that subclass members override wildcards. Wildcard should apply only when the name lookup would otherwise fail.
- Decimal – IEE754r looking up! Still fighty. We can do a sane decimal in any event, but need to stare down operators.
- hashcodes – want a global hashcode function, deferring value indexing.
  - rok points out that E4X and catchalls both allow property names to be non-string values.
  - We don’t want just one Hash or Dictionary class. More motivation for type parameters.
- Block expressions
  - Can these be translated to the core language without let? No (consider looping over a let-block).
- Standard library issues
  - Debug helpers, specifically a way to get a stack trace for the exception’s construction site.
    - Dave points out that TCO and other optimizations motivate not overspecifying the content of the trace.
    - Ed points out that throw-site stack trace is also sometimes useful. Sometimes want both construct and throw sites, throw and re-throw, general logging of stacks, etc. This leads to:
Graydon and a bunch of us want general reflection of the stack. Errors thrown by the runtime would compute this.

- **Date and time** issues:
  - Locale can-of-worms.
  - Property-based beats method-based year, month, etc. access.
  - Resolution, etc.

- **String formatting** choices:
  - Leave out, defer to the emergent standard library ecology? Then lose type system tie-in opportunities.
  - .NET vs. MSCOM vs. Java vs. others leaves no single obvious choice of what to imitate.
  - Ed points out that strings imply localization, more worms.

- **Foundational issues**
  - **drop traits** still sounds good, but:
    - Ed points out that `[[Prototype]]` (Edition 3), aka `__proto__`, is of type `Object`.
    - Vtables? Horrors. Ed to write as addendum to `drop traits`.
  - **is as to** is ok, but nullability issues remain.
    - AS3 has nullable `Object` and `String`, non-nullable `Boolean, Number, and String`.
    - C# nullability motivated by database integration. Might be relevant motivation for ES4.
    - Given nullable-by-definition `Object`, want anti-nullability notation of some sort, to rule out null using the type system (statically if possible).
    - Nullable types are just a (possibly very important, worth special syntax, etc.) special case of **sum types**.
    - Dave to sweat nullability.
  - **type parameters** seems in good shape, gives typed arrays (yay).

- **meta objects**
  - Does it raise type soundness issues? Not really, just the usual `java.lang.reflect` thing: type safety requires downcasts.
  - Ed notices that `java.lang.reflect` adds runtime costs. Graydon: could be optional.
  - Dave: optionality good to reduce costs, increase optimization opportunities when doing without reflection.
  - Reflective MOP should not be in Edition 4 Compact Profile.

- **builtin classes**
  - Invented syntax – how does `prototype` map to Graydon’s `drop traits` proposal, e.g.? Open issues – need everyone to read and study Ed’s proposal. Particular attention to class `Class`.

- Dave to propose TCO (yay!!)
  - [jodyer] sorry i missed the meeting. what is TCO?
  - [dherman] TCO is **Tail Call Optimization**, whereby evaluation of the last expression in a given context, e.g. the operand of a `return`, should not take extra stack space. Also sometimes known as “proper tail recursion,” though it’s about evaluation of anything in tail position, not just recursive function calls. I sometimes use the name “proper tail calls,” because it suggests a) that it’s not just about recursive calls, and b) that it’s a correctness criterion, not just an optimization.