Minutes of the: ***Ecma TC39, ES3.1WG***

Phone conference

Held on: ***14 August 2008***

1 Roll call and logistics

1.1 Participants

Doug Crockford (Yahoo!), Pratap Lakshman (Microsoft), Mark Miller (Google), Adam Peller (IBM), Sam Ruby (IBM) and Allen Wirfs-Brock (Microsoft)

2 Agenda

Decimal: the behaviour of '==' and '==='

Decimal: handling mixed mode arithmetic

3 Minutes

the behaviour of '==' and '==='

"EQ school" says: ignore the 2 irregularities we cannot fix (NaN and -0) , for all other cases if a === b they are computational indistinguishable;, put coercion behaviour into '=='. "typeof school" says: algebraic property holds in JS that if typeof a is typeof b, and a == b, then a === b. '==' between 2 numbers should compare as if they denote the same abstract number; dec 1.0 is different from 1.0; ' ==' between 2 numbers should compare as if they represent the same point on the real number line - need to have EQ operator; can be on Decimal but might move it to Number - move it to Object - we need a function to ask 'are these two values computationally indistinguishable ?" - would even allow to compare against NaN, and between -0 and +0 - an object hash/identity hash would go hand in hand with this - has historic precedent, it has the right meaning, and the name is short in length too.

handling mixed mode arithmetic

If Number is used to represent both binary FP and decimal forms, then instead of type-testing we can check if the Number is in a decimal form - such a test can be used to determine the arithmetic mode to use - a Form() function can be introduced as a helper function for specification purposes - so, if the 'form' is decimal then we do decimal arithmetic, else we fall back to using double precision - do we need a ToDouble() helper function too ? - introduce these helpers and convert all existing operators that have decimal support to start using these helpers - and, eventually we might fold-in the Decimal class into the Number class too.

Meeting adjourned.