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

held in: ***Phone conference***

on: ***04 September 2008***

1 Roll call and logistics

1.1 Participants

Pratap Lakshman (Microsoft), Mark Miller (Google), Sam Ruby (IBM) and Allen Wirfs-Brock (Microsoft)

2 Agenda

Decimal

What other changes do we expect between now and Redmond ? (based on that we need some more writing/editing to happen).

3 Minutes

Decimal

typeof 1m should return “decimal”; not “number” - no distinction between new Decimal(’1m’) and Decimal(’1m’); its typeof would still be “decimal” - ES3 has a clear separation between primitive values and objects; Decimal seems to be a part of both - should 1m instanceof Decimal return ‘true’ or ‘false’ ? - its should return ‘false’; same as 1m instanceof Number .

named methods of Decimal, should they be static or instance methods ? - if Decimal is going to be a fully integrated type, put them on instances - also, look at the operators; operators modeled as functions should go as static functions.

What other changes do we expect between now and Redmond ? (based on that we need some more writing/editing to happen).

Use subset cautious needs to be revised - get rid of the subset term - more useful to leave in the possibility that there are pragmas that change the meaning of a compilation unit; for now though we will have only one such pragma which would be the strict mode.

‘this’ coercion - the question is specific to strict mode; if a strict function that internally uses “this” as an rvalue, what should happen when it is called as a function ? - we have 3 options:

(1) in an execution context in which ‘this’ is bound to null or undefined, evaluating ‘this’ as an rvalue in strict code throws (in non-strict code it returns the global object, just as in ES3)

(2) if the strict function mentions ‘this’ freely, then an attempt to call it with ‘this’ bound to null or undefined throws rather than entering the function

(3) no coercion is done at all - ‘this’ is bound to whatever it was bound to.

When invoked using a “Call” with a primitive, function gets entered with ‘this’ bound to the corresponding wrapper; every time you get a new wrapper ! - where in spec language should this be controlled ? - this can be done in call/apply, in function entry, or in the section introducing strict mode - hardly seems containable in the spec for call/apply - checking on function entry might require that every function prologue check if its ‘this’ is bound to a primitive value - but, that tax needs to be borne only by strict code - regarding ‘this’ coercion, what is the benefit to strict code again ? - the function simply obtain what the caller passed; just like any other function parameters - keep wrapping in the spec in call/apply, and make it

conditional on strictness of the function being called - what should happen to 1.foo() ? - coerce only if foo is strict - but isn't this a non mainline scenario ? - not really; it is common to augment String and then invoke such methods through string literals.

Decision:

(1) All coercion from null/undefined to the global object gets migrated to section 11.1.1 in the spec, and is made conditional on strictness.

(2) Wrapping of primitive values - raise this on the discuss lists

Meeting adjourned.