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

held in: ***Phone conference***

on: ***22 January 2009***

1 Roll call and logistics

1.1 Participants

Pratap Lakshman (Microsoft), Mark Miller (Google), Rob Sayre (Mozilla) and Allen Wirfs-Brock (Microsoft)

2 Agenda

Not circulated ahead of time.

3 Minutes

eval

Indirect eval is always non-strict independent of the strictness of its caller - to adopt any other rule will have pervasive implementation burden because a call to an indirect eval can look like just any call - also, if we consider indirect eval as just any function call, then we can think of it as being a non-strict function and like all non-strict functions its would not inherit the strictness of its caller.

Direct eval operator gets a new declarative environment - should distinction between the eval operator and eval function be based upon the scope resolution; i.e if a var named eval resolves to a reference whose base is the global object? - what about the following: a user function receives a parameter named eval; somewhere in the body he invokes it; user might not even know that an eval function exists - this could be a security hole - those who get surprised get attacked - so, should strict mode prohibit a local variable named eval?

direct eval is a reference that resolves at the global object; whose name is eval and value is the original eval - should indirect eval be sensitive to the value you are evaluating or to the scope? - could be a breaking change to ES3 - IE's eval is always scoped locally; only Opera does it as specified.

No static analysis required to detect direct eval? - in order to support the distinction we want treat all direct eval as indirect eval and dynamically decide that it is the direct eval - basically, does it syntactically look like eval? And then is the value bound to eval in that scope that of the original eval - lets bring it up on the lists.

Arguments object

Freezing the arguments object severs all joining - strict mode arguments object is frozen, and therefore all its joining is lost - this is almost identical to what we agreed to in Kona - frozen, array-like object that inherits from Array.prototype; can be used where array-like objects are expected - concat will not implicitly spread a frozen arguments object - JSON will serialize arguments and frozen arguments as objects - isArray should return false.

Arrayness test and using [[Class]] Host objects should not be allowed to use the [[Class]] names Array, Date, RegExp, Function - then, we don't need an isArray test - we do actually, since we do not want to depend on names - does that mean we need an isDate, isRegExp and isFunction tests? - where we test the [[Class]] property we can test with these instead -

actually, in all of the Array methods we need to see what it means when we test for whether the this value is an array; the array methods are meant to be generic - that's an action item.

Meeting adjourned.