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

held in: ***Phone conference***

on: ***6 January 2009***

1 Roll call and logistics

1.1 Participants

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

2 Agenda

Not circulated ahead of time.

3 Minutes

strict mode

"use strict" directive - no issue with the meaning or intent - intent-wise it looks like a string literal - however, there is still a lexical ambiguity between string tokens and syntax that specifies strict directive - currently specified as a sequence of string tokens, but the string literal is itself an atomic token! – need to consider how it interacts with semicolon insertion as well - may end up having to specify it as a static semantic restriction (a prose restriction).

“this” coercion

if you extend Number.prototype with a strict function (say, a strict factorial function ‘fact’), then when you call 3.fact(), it would be nice if the this in fact were bound to 3, and not to a wrapper on 3 - the base needs to be bound to 3 - programmers must be able to ignore the existence of wrappers - 3.fact() should do likewise as 3.fact.call(3) - need to maintain the algebraic equivalence of a direct function invocation, and invocations through call and apply - if 'x' is a var and x.foo is a function, then x.foo(a), x.foo.call(x, a) and x.foo.apply(x, [a]) are equivalent - but this is currently broken! - ES3.1 evaluates 3.fact() to a reference whose property name is fact and whose base is got from the operation ToObject(3); the evaluation of a '.' coerces its base to an object - non-strict functions coerce their this argument, but strict functions do not - agreed, Function.prototype.call/appy do not coerce the first argument, but the receiver function coerces it according to its own strictness.

what about the case 3.foo = 9 - [[PutValue]] whose base is a primitive object, no observable difference between strict/nonstrict code; the wrapper cannot escape the [[PutValue]] - the property could be a setter (only inherited) - independent of strictness, the setter is invoked and this is bound to 3 (non-strict setters will promote the 3 to a wrapper) - at the minimum you have wrap in order to look up what kind of property you got - we can know if references are strict or not - a strict reference to a primitive value is always an error? - determination is happening in [[ThrowablePut]] - do we need a separate [[ThrowablePut]] for primitives? [[PrimitiveThrowablePut]] as an internal function instead of an internal method - if there is no accessor, then in ES3.1 nonstrict, this remains a no-op; non-strict failed assignments are silent - indeed, we can treat this as a failed assignment, and hence throw an exception in strict mode.

we are changing the contract on GetBase since it can now return a primitive value! - need to make sure we don't break indexing into strings - the contract on GetValue remains unchanged - still, we need to look at [[GetValue]], [[PutValue]], function calls, and the delete operator - change the spec to do an object coercion on the base only to look up the property, however, treat the coerced object as ephemeral since this wrapper (coerced object) cannot escape and ToObject has no side effects - if it cannot escape perhaps we can avoid allocating it as well (perf gain too); and purely strict programmers can ignore the possibility of wrappers.

Need to think about this some more - need to take a look at all the places where we are wrapping primitives - also need to consider impact on implementations - probably doable - but lets check on the discuss list first.

Meeting adjourned.