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

held in: ***Phone conference***

on: ***21 October 2008***

1 Roll call and logistics

1.1 Participants

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

2 Agenda

Scoping of ‘const’ and blocks-introduce-scope
Updates required for the JSON section (contd. from last week)
Should SubStatement be a part of LabelledStatement

3 Minutes

scoping ‘const’

scoping ‘const’ is not so much about how we write up the spec; there is a semantic definition problem - wanted const as applicable to vars and not on properties of objects with read barriers - but then what do we do about top level consts? - its base would be a special ‘object environment’ - const declaration treated as the creation of a property whose value was readonly - at the instantiation of the binding it would be set of ‘undefined’, ‘readonly’ and ‘configurable’- but readonly does not create a read barrier - strict mode needs to behave differently - read barrier needs to be unconditional - objection to semantics requiring a read barrier (from a performance perspective) - actually, in ES-H we will want ‘let’ and ‘const’ to behave consistently - looks like the semantics are not representable cleanly as the semantics of a property - two options to address the use-before-initialize problem (a) “dynamic dead zone”: runtime read barrier (b) “static dead zone”: set of static rules that prevent the read - before-initialization, and thereby removing the need for a read barrier - dynamic dead zone is practical, but static dead zone would be better - not sure if we have the time to nail down the rules in time for ES3.1 - so we have the option of either going with the dynamic dead zone, or pulling const out of ES3.1.

on the base to a reference being null

what should happen in strict mode when assigning to an unbound name (e.g. “foo = 8;”) where foo is unbound? - should this get bound to the global object? to window? - what should be the base for an unbound reference? - coming with the notion of an “environment record”, unbound names will have this as their base? - consider ‘foo’ as a top level var, as a property on window, and as used with out being declared; what are its bases in these cases? - if it is unbound, it base is null; for window.foo, the base is the base is the global object; for top level foo, the base is this environment record - what about the case when these are within a ‘with’ block? - same inside a ‘with’ block; doesn’t distinguish between environment records for ‘with’ and the global object - when foo is unbound, strict “foo = 8;” must throw - not sure what non-strict “delete foo;” must do - when top level foo is bound, foo() must call foo with undefined as the “this” value - when window.foo exists, window.foo() must call foo with window as the “this” value - if foo is a top level const or let, then we may decide that its scope is its program unit, but not make it a property of the global object; in any case, we can now postpone that last issue till ES-H - in non-strict mode, “foo = 8;” and “delete foo;” are ok.

Updates required for the JSON section (contd. from last week)

Postponed to next meeting.

Should SubStatement be a part of LabelledStatement

Postponed to next meeting.

Meeting adjourned.