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

Phone conference

held on: ***24 April 2008***

1 Roll call and logistics

1.1 Participants

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

2 Agenda

ES3.1 strict mode

Getters/setters (from last call)

Meta (from last call)

3 Minutes

ES3.1 strict mode

We want that writing to a "readonly" property should throw an exception; does that not change control flow in a manner that could catch developers/testers off guard especially if it happens to be in some rarely traversed code path ? - that is actually good to have from a integrity perspective - don't want execution to continue in the normal code path once such integrity is compromised - it changes control flow in a fail-stop manner - Strict mode defines a fail-stop subset of the language (just like in the case of Caja/JavaScript) - strict mode comes into effect on a per-module basis - but isn't throwing an exception a global characteristic ? - need to worry about the case where writes happen across module boundaries; should we honour the mode of the source, or of the destination ? - also, there is no notion of objects being defined on a per module basis; how would we define that ? - need the concept of a function defined in a module - since functions are defined by special forms using the 'function'.

keyword, they are unambiguous: a function defined in a strict module is strict. Objects don't need to be strict or not. Only operations on objects do. The property assignments

x.foo = y

and

x[f] = y

are strict or not, and therefore throw or not, depending on the strictness of the module this assignment appears in. So this case follows the "honour source" rule.

How do we indicate 'strict mode' ? - multiple proposals being floated - there is the "use" "strict", and the one armed if statement: "if" "(" "false" ")" "use" "(" "strict" ")" - how about using the meta statement; something like //@strictMode - a hint within a comment - kind-of like embedding javadoc comments within source code - but javadoc can be stripped out of source without changing the semantics - meta statements (within comments) won't be handled well by minifiers - wait! Stop thinking about them as comments - minifiers anyway need to handle the

conditional compilation syntax in IE - the one armed if statement is weird - and the "use" "strict" is like two statements that need to be present as if in a prologue position; and that is very unintuitive - how about indicating it with a do-nothing string literal expression-statement ? - how about indicating it as an attribute on the script tag ? - Ian Hickson and HTML folks feel that such conditioning should be done within the material that is being loaded rather than on the page that is loading the material - ok, that might aid code reuse too - need to raise all proposals - or, we need to look at strict mode differently.

ES4 strict mode is strict about arity too - not possible to enforce that in ES3 - Mark to work on a proposal (leaving aside the issue of how one turns on strict mode)

Meta

How about having a "redefinable" attribute instead of a "deletable" attribute ? - if "redefinable" is set, then you can define a property and later change its attributes (so long as it's redefinable, one can also delete it); if it is not set, then you can do neither - is "readonly" but "deletable" even useful ? - no need to factor that out; it is a harmless orthogonality - UNIX File system as an example - can own a file but can make it readonly - agreed that this was weak, how about folding together writable and redefinable into one bit. No one would mourn the loss of readonly-but-deletable. However, we also need to express writable-but-not-deletable, so we still need two bits.

What does "readonly" mean in the presence of getters and setters ? - does it mean the definition cannot be changed ? - whether getters and setters can be replaced must be determined by "redefinable" - property is either a data property (value + writable) or a procedural property (getter + setter functions) - "(getter + optional setter)". If a procedural property has no setter attribute, then it is considered readonly - a property intended to a purely a method must also not be "writable" - that seems like a fuzzy difference - we store functions as data values - defining "readonly" (const) properties could be common - what else can we make the distinction pay for ? - Caja makes strong distinction between simple functions, methods, and constructors - need a more detailed proposal with pros and cons - we should postpone making such distinction until ES3.2.

Can we introduce new attributes in ES3.1 (from the perspective of "redefinable", etc.) - if we can do any, we can do multiple - must pay for themselves, though - may have impact on the low level object representation - what if some implementation has only 1 bit left? - nobody has asked if new attributes are tolerable to existing implementations, and if so, is there a limit to that tolerance - need a list of candidate attributes for future versions of the spec - how about "protected" for ES3.2 ?

1 arg beget or 2 arg beget ? - can tolerate either, but definitely need the first arg - lets think about this some more.

Getters/setters (from last call)

Not discussed.

Meeting adjourned.