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

held in: **Phone conference**

on: **16 October 2008**

1 Roll call and logistics

1.1 Participants

Mark Miller (Google), Doug Crockford (Yahoo!), Allen Wirfs-Brock (Microsoft), Pratap Lakshman (Microsoft)

2 Agenda

Not circulated ahead of time.

3 Minutes

Updates required for the JSON section (refer comments in the 13 Oct draft on the wiki).

Stringify needs to specify what do we need to detect, and what do we need to forbid - how do you specify any restrictions? - the object you are trying to serialize may only be a single object or a tree of objects; the root object must form a tree of objects - it is up to an implementation to make that happen; it is up to an implementation to prevalidate that or incrementally determine it as it is running - how about the following: the replacer maintains a hashtable that remembers every object it serializes; if it sees one it puts in an ibid entry into the hashtable, and proceeds; on the other end the reviver starts with an array of ibids, and progressively clears the array of the ibid entries as it sees duplicated objects in the JSON stream - that won't work, because the reviver is not called in the same order as the replacer was called! - ok, in that case instead of an ibid entry we could use a JSONPath - that won't work either, because the replacer does not have enough context up front to generate the path names - perhaps is best to specify that the object you are trying to serialize may only be a single object or a tree of objects; the root object must form a tree of objects - given that there is a JSON implementation in JavaScript, no matter how we specify it, it should not preclude implementing JSON in JavaScript itself.

What are we trying to prevent really? - prevent the creation of an infinite length string; that is a symptom of failure - problem similar to that of specifying factorial if it were a library function - what bounds would you apply on the input, and how would you specify that bound? - so, in that case what is the symptom of failure? - resource exhaustion - would prefer if we could precisely specify stringify - do we agree on the precondition that the object should be rooting a tree? - that is not a precondition strictly speaking; it is an input validation condition - for implementations that incrementally determine if there is a cycle, what should happen? Do they stringify up to the point where there is a cycle, or, completely abort the stringification? - completely abort the stringification - but, what about side effects that might have been caused by property getters? - those will be observable.

Is there a reason why we want to under-specify this? - as an implementer I want the ability to optimize away un-observable behaviour - can you provide example of specification that prevented making such optimizations? - yes, the memory model in Java - what about the need for exact reproducibility of floating point computation in Java - Java serialization can handle cycles, though - perhaps we should take the position that JSON serializes records and not JavaScript objects - can we specify the order in which the properties get serialized? - the 'fast' flag to Object.keys (used from the stringify method) can be used - if true, the keys are returned as-is, but if false, they keys are returned in sorted order - would prefer if the 'fast' flag can be used to produce the same order as for-in; should be deterministic - but, we have been through this before … we don’t want to specify enumeration order in for-in - why not? - because people have used various internal representations for Arrays and can no longer honour such a constraint (of enumeration order) without jumping through hoops; it goes back to the memory-model over specification of Java forcing implementers to forego optimization opportunities - Ok, but we would like to see some more concrete examples like these.

So, we are not agreed, yet, on how to specify what stringify should detect and what it should forbid - lets revisit at next meeting.

Meeting adjourned.