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

held in: *Phone conference*

on: *05 June 2008*

1 Roll call and logistics

1.1 Participants

Doug Crockford (Yahoo!), Pratap Lakshman (Microsoft), Mark Miller (Google), Allen Wirfs-Brock (Microsoft) and Kris Zyp (The Dojo Foundation)

2 Agenda

Enumeration order

JSON

3 Minutes

Enumeration order

Original language was intended allow for enumeration order to be implementation dependent - any implementation will have an order; that it is observable does not mean that it was intentional - should not incorporate into spec unless there is a strong precedent - arguably there is such a precedent for non-numeric properties - but do we want to treat numeric properties on arrays as a special case? And arrays themselves as a special case of object? - how real is this requirement of testing? - order should be preserved and reproducible - any place where you iterate over a hash-based data structure you need to provide your own canonicalization for testing - what if you want to represent an object as a B-tree; spec language should not indirectly prohibit it - what would be the fastest implementation's natural order - that would be pure hash tables, or arrays - implementations can have another layer to impose the ordering - what if we introduce a new statement to enforce enumeration order - who wants it? - Apple and Mozilla have changed iteration order from what it has been - JScript too is using optimized representation starting IE8 Beta1 - properties on the prototype, and the possibilities of shadowing can make this complex to pin down in spec language - first do no harm - unless we have language that we believe is correct, we should not put it in the spec - push to next revision - we are not doing anything that prevents others from doing anything in the future - Object.keys method can be used as a mitigation - by default it will enumerate enumerable owned properties in insertion order - it can optional advisory parameter 'fast' that may not preserve any ordering.

JSON

Using the term 'keys'; is that conforming with the rest of the spec language? - are they keys always strings? - yes, if you pass in something that is not a string, it will be stringified first - change its name then - not yet precise enough for a spec of this kind - examples not the best way to convey the format - better to rework the spec that try providing examples - the key problem to address is how to describe the translation - provide a translator who behavior is normative - the ES3.1 spec language calls out that we can use ECMAScript itself as an alternative notation to present a reference implementation - or, perhaps we can use some tabular form - exactly which properties get stringified? -what is the criteria on each property that determines what the default translation would be - now what happens if you have a replacer function - need to clarify interaction of toJSON method with getters of the same name - what constitutes a toJSON method? - we should consider a function valued property as a method - rephrase spec language to say “if it is a function member”

Do we need to distinguish between property fetches and invocation? - perhaps for the next revision of the spec - current API is Ok with regard to secure sub languages - good to have a list of proposals that we will want to consider for the next revision.

Meeting adjourned.