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

Phone conference

held on: ***08 May 2008***

1 Roll call and logistics

1.1 Participants

Pratap Lakshman (Microsoft), Allen Wirfs-Brock (Microsoft) and Kris Zyp (The Dojo Foundation)

2 Agenda

Getters and Setters

Improvements to the Function object including richer reflection capability

A standard name and conventions for a top level “Host Extensions” object

3 Minutes

Getters and Setters

What about inheritance - said object has a setter but not a getter; some other objects on its prototype chain has a getter; does said object inherit that getter? - currently in FF it can be inherited only as a pair - they can also only be overridden as a pair - how are things going on the defineProperty? are they being accepted for ES4? - we want to push it through - need to reconcile with ES4, though.

Same level of expressiveness can be got using the catchall - yes, they can be used to catch all side effects to an object - even if you have them and want to use them, you don't want to use them in the normal scenarios; for e.g. accessing a property in a loop - yes, could be very expensive - not good if you have properties mixed with getter/setter properties - well, catchalls are the only primitives you need! - use cases for catchalls? data binding - when does the catchall come into place? like a "filter" (before the access happens? or, like a "backstop" (after all normal mechanisms of access have been tried and failed?

Getter/setter; several motivations - simplified notation for function invocation that looks like variable access, provide the ability to make instance variable access act polymorphically, add additional processing as part of a variable access; like security checks - yes, like a unified notation for any kind of message you want to pass to an object - catchalls give an object full control over its message processing - if defineProperty is going to be problematic then catchalls could be useful - except that catchalls seem more experimental in nature than getters/setters - also, they are no substitute for getters/setters - use case for catchalls? dynamically creating a façade and intercepting all property accesses.

Given the existing implementation of getters/setters in the 3 browsers, can we change the definition so that they are independently inherited? - how close are the 3 browsers' semantics?; could such a change be tolerated? how impactful would that be? - how does ES4 support them? - defines them as methods on the class - independent of each other syntactically - Kris to check on the discuss lists.

Also how about using the object literal notation to create "pristine" objects? - that seems like a new proposal - yes, separate proposal required.

Improvements to the Function object including richer reflection capability

Lets wait till we have Doug, Mark and others on the call. Move to next meeting.

A standard name and conventions for a top level “Host Extensions” object

Standard way for hosts to introduce names without worrying about conflicts - in ES4 you might use namespaces, and the ES3 you might use objects as namespaces - IE might introduce, say "MSIE", and FF might introduce something like "Mozilla", etc. but that is not good - instead, how about introducing something like "ECMAScriptHost"; implementations can define properties within this, and user code could check for the existence of specific properties - instead of making it normative, why not leave it as a guidance or recommendation? - need to reserve the name so that you can depend on it - doesn't Dojo need to do lots of such detections? - this could be a standardized way to do sniffing - need to balance between consistency and leaving room for browsers to experiment on things that can get standardized.

Meeting adjourned.