TC39 coalesces on future direction of Web Programming Language

Executive Summary

The committee has resolved in favour of these tasks and conclusions:

1. Focus work on ES3.1 with full collaboration of all parties, and target two interoperable implementations by early next year.
2. Collaborate on the next step beyond ES3.1, which will include syntactic extensions more modest than current ES4 suggestions in both semantic and syntactic innovation.
3. Remove from consideration the ES4 concepts of “packages,” “namespaces” and “early binding”
4. Rephrase other goals and ideas from ES4 to keep consensus in the committee; these include a notion of classes based on existing ES3 concepts combined with proposed ES3.1 extensions.

Technical Summary

Harmony was proposed to unite the committee into a single focus for the next version of ES3.1. This required intentionally dropping specific ES4 proposals, such as ‘‘namespaces,”

This is good news for those who favour smaller changes to the language and those who advocate ongoing evolution that requires new syntax if not new semantics. It does mean that some of the ideas going back to the first ES4 proposals in 1999, implemented variously in JScript.NET and others, won’t make it into any ES standard. But the benefit is collaboration on unified successor specifications to follow ES3, starting with ES3.1 and continuing after it with larger changes and improved specification techniques.

One of the use-cases for namespaces in ES4 was early binding (use ’namespace’ intrinsic), both for performance and for programmer comprehension -- no chance of runtime name binding disagreeing with any earlier binding. But early binding in any dynamic code loading scenario like the web requires a prioritization or reservation mechanism to avoid early versus late binding conflicts. Plus, as some JS implementers have noted, multiple open ’namespaces impose runtime cost unless an implementation works significantly harder. For these reasons, the ES4 concepts of ‘namespaces’ and early binding (like packages before them) will not be included in ES3.1. This is final, they are not even a future possibility. To achieve harmony, we have to focus not only on nearer term improvements -- on "what's in" or what could be in -- we must also strive to agree on what's out.

Once namespaces and early binding are out, classes can de-sugar to lambda-coding + Object.freeze and friends from ES3.1. There's no need for new runtime semantics to model what we talked about in Oslo as a harmonized class proposal (I will publish wiki pages shortly to show what was discussed).

We talked about de-sugaring classes in some detail in Oslo. During these exchanges, we discussed several separable issues, including classes, inheritance, like patterns, and type annotations. Note that there were clear axes of disagreement and agreement, grounds for hope that the committee could reach consensus on some of these ideas, and general preference for starting with the simplest proposals and keeping consensus as we go.

We may add runtime helpers if lambda-coding is too obscure for the main audience of the spec, namely implementers who aim to achieve interoperation, but who may not be lambda-coding gurus. But we will try to avoid extending the runtime semantic model of the 3.1 spec, as a discipline to guard against complexity.

One possible semantic addition to fill a notorious gap in the language, which was sketched by Brendan Eich, with able help from Mark Miller: a way to generate new Name objects that do not equate as property identifiers to any string. I also showed some sugar, but that is secondary at this point. Many were in favor of this new Name object idea.

There remain challenges, in particular getting off of the un-testable and increasingly unwieldy ES1-3.x spec formalism. There was general agreement, and no objections, about the ES4 approach of using an SML + self-hosted built-ins reference implementation (RI).

ES3.1 standardizes getters and setters that were first implemented at Mozilla and then used by Apple and Opera. More such de-facto standardization is on the table for a successor edition in the harmonized committee.

There was good agreement on "de-facto standard" suggestions for ES3.1. Some favourable comments were discussed about simple changes in JS1.7 and 1.8 that do not require new runtime semantic models.

New changes will require new syntax, which is appropriate for a major post-3.1 "ES-harmony" edition. Syntax is user interface, which should be improved. The intersection semantics of extended ES3 implementations conflict and choke off backward-compatible *semantics* for syntax that may even parse in all top four browsers (e.g., functions in blocks).

Both the appropriateness of new syntax, and the need to make incompatible (with ES3 extensions) semantic changes, motivate opt-in versioning of harmonized successor edition. It is believed that past concerns about opt-in versioning requiring server file suffix to MIME type mapping maintenance were assuaged (browsers in practice, and HTML5 + RFC 4329, do not consider server-sent Content-Type -- the web page author can write version parameters directly in script tag type attributes).

Some expressed interest in an in-language pragma to select version; this would require immediate version change during parsing. It's a topic for future discussions.

The committee has a vision for extending the language syntactically, not trying to fit new semantics entirely within some combination of existing "three of four top browsers" syntax and standard library extensions.