ECMA TC39 TG1 Meeting
Held 13th June 2002,
Netscape, Mountain View, CA, USA

Attendees:

Waldemar Horwart (Netscape)
Jeff Dyer (Compiler Company)
John Schneider (BEA)
Rok Yu, Herman Venter, Peter Torr (Microsoft)

Next Meeting:

Wednesday 7th August (ECMAScript) and Thursday 8th August (XML)
Microsoft, Redmond, WA, USA

Agenda:

XML Presentation (John Schneider)
Incorrectly reported bug in Edition 3
"abstract" keyword
Constants
Namespaces and classes
Nested namespaces and classes
Cutting interfaces, class extensions, units, and operator overloading
Reflection API
Idempoten attributes

XML Presentation:

John Schneider gave a very informative and compelling presentation on some of the thinking he has been doing (along with other interested parties) to better integrate the manipulation of XML data into ECMAScript. The presentation will be available on the ECMA web site.

The presentation generated a lot of discussion based on issues with the proposed syntax, but John made it clear that he was interested in getting across the concepts rather than the exact syntax (which is subject to change). The proposed timeline for standardising the XML+ECMAScript work is tentatively set at one year, and it is not explicitly targeted at browser support, but rather for any kind of application that uses ECMAScript to manipulate XML data.

The group decided that the best course of action in the short term would be to create a subgroup of TG 1 to continue reviewing the proposal and develop standardised extensions to ECMAScript. John Schneider will act as editor, and Peter Torr will act as convenor until his term ends in October. Not all of the companies involved in the initial discussions with John are currently ECMA members, although some of them are interested in becoming members. The WAP members are also interested in using the XML + ECMAScript work in their "mobile profile" in the future.

Incorrectly reported bug in Edition 3:

Waldemar presented a bug in Edition 3 whereby local references to global functions did not correctly pick up the activation object. For example:

```
var x = "global"
function show() { return this.x }
function g()
{
  var x = "local"
  var y = show
  print(y()) // should be "local", but prints "global"
}
g()
```

After some discussion, it was agreed that Edition 3 (and both the Microsoft and Netscape implementations) are indeed correct and that the code above should print "global" and not "local". When the call to "y" is made, there is no explicit "this" object, so the "this" object is the global scope, not the activation frame of "g".

"abstract" keyword:

John queried the value of "abstract", given that classes with abstract members could still be instantiated. Waldemar agreed that the usefulness of abstract classes as currently specified was limited, and the group decided to consider it for removal from the standard. Unless anyone has objections to the removal of abstract, it will be officially removed at the next meeting, and Microsoft's implementation of "abstract" classes and methods will be an extension to the standard.

Constants:

In the previous meeting, the committee decided to cut the "compile" attribute, but this leaves no way for the author to explicitly distinguish between compile-time and run-time constants. Waldemar would like to maintain this ability to enable forward-references to compile-time constants.

```
const a = 5 // 'a' is a compile-time constant
const b = a + c // unknown; 'c' is not yet defined
const c = 2 // 'c' is compile-time, and now 'b' is also compile-time
const d = e // unknown; 'e' is not yet defined
var e = 2 // 'd' is a runtime constant, value is "undefined"
const f = g // error; 'g' is never defined
```

The decision was made that compile-time constants would be evaluated if and only if all their RHS values are resolvable to constants within the current environment.

Defining classes and namespaces:

Class and namespace definitions can only be placed at the global scope, class scope, package scope, or within a block inside one of these scopes. They can never appear inside a function, method, or statement. This rule was implicit before, but has now been made explicit.

Nested classes and namespaces:

Microsoft's implementation supports both instance nested classes (the default) and static nested classes. This makes development in environment such as ASP.NET much simpler. Currently Netscape only supports static nested classes. It was decided that the standard will require all nested classes to be marked as `static` (resulting in static nested classes), but Microsoft will extend the standard to allow classes without the `static` modifier. The same rules will apply to namespaces.

Cutting interfaces, class extensions, Units, and operator overloading:

It was decided to consider postponing these features until Edition 5. The members present at the meeting believed that these features did not provide sufficient additional benefit to further risk the schedule, and that barring any objections by other members they will be officially postponed at the next meeting. The core features of Edition 4 that enable "programming in the large" are classes, types, and namespaces, and whilst these features provide additional assistance to programmers, they do so at considerable cost to the standard.

Reflection API:

The group again discussed the simple reflection API and how enumeration works with various objects (types, collections, arrays, expando objects, etc.). Rok Yu will incorporate the details into his Reflection proposal for the next meeting. The basic idea is that `for..in` will always enumerate keys if possible (arrays and collections) or whatever enumeration is provided by the object in other cases. We will consider introducing a new syntax to enumerate the values of collections or arrays, for example:

```
vay myArray = ['a', 'b', 'c']

for (var i in myArray)
  print(i) // prints 0, 1, 2

for (var i in values(myArray))
  print(i) // prints a, b, c
```

Waldemar would like to keep backwards compatibility and maintain the equivalence between `ob['x']` and `ob.x` for all objects. Herman pointed out that there are differences between old-style expando objects (which are essentially property bags) and classes. Waldemar would like a way to upgrade property bags to classes so that reflection still worked as in Edition 3. Rok will include this requirement in his reflection proposal.

Idempotent attributes:

It is not yet clear whether arbitrary attributes are idempotent. Namespaces need to be legal to specify a namespace more than once, and other attributes such as `enumerable` will imply `public`, so `public` will need to be specifiable more than once to make declarations such as `puclic enumerable var x` legal.