TC39 TG1 E4X Meeting Notes 2003-06-06

Date	June 6, 2003 9:30 AM – 4:30 PM
Location	Netscape Communications Corp. Miami Vice Conference Room Building 21, 1st Floor 466 Ellis Street Mountain View, CA 94043
Convener	Rok Yu (Microsoft)
Editor	John Schneider (BEA/AgileDelta)
Participants	Jeff Dyer (Macromedia) John Schneider (BEA/AgileDelta) Rok Yu (Microsoft) Waldemar Horwat (Netscape)

Agenda

Agenda was adopted as written.

Discussion re: Notes from 2003-05-23 Conference Call

Waldemar sent the following comments on previous set of meeting notes.

11.3.1: "He adds that XMLList can contain primitives or data of type XML." We revisited this issue later in the teleconference after realizing that nothing would actually generate an XMLList of numbers, booleans, or maybe even strings. Now there appears to be no reason to permit non-XML values in XMLLists. (You mention this briefly at the end of the minutes; it would be good to cross-reference that decision here since it contravenes the earlier decision.)

11.3.3.5: After coming to the methods that could potentially return a list of booleans or strings we revisited what childIndex should do.
Because a list of numbers is not useful in the common case of getting a single result and doing arithmetic on it, I recommended that childIndex return a scalar and not a list. I thought we got agreement on that, and this is a prerequisite to the agreement on XMLLists not containing non-XML values that is stated at the end of the minutes.

We revisisted these issues and have agreed that:

11.3.1: Wording makes it sound like decision that XMLList can contain primitives is decided. Working group agrees it was not.

11.3.3.5: John agrees that notes are consistent with his recollection that we did not finalize this decision. John was going to look at how this change would impact other scenarios before we finalized this decision. (43).

ECMA-323 Use Case

John asks whether we should send the converted use case to ECMA-323 that was discussed at the 2003-05-09 meeting.

Rok thinks and working group agrees that we should wait until the spec is stable before sending.

Status

The group is still about one meeting behind. Goal continues to be complete by September. Members agree that all major design decisions need to be finalized by July.

Review of Action Items

Rok will begin tracking all issues and action items on E4X issues spreadsheet. For items tracked by the spreadsheet, notes will include corresponding issue number in parentheses.

Action Items From Last Face to Face Meeting

(22) Look for cases where String is special cased in E3 spec to understand consequences of changing ToString on XML fragment to not insert commas
Calls to ToString comes up in property name lookup, concatenation, argument conversion. John reviewed all instances and believes that there is no issues.

Rok and Waldemar believe that a decision on this item cannot be made until a decision is made on whether primitives can be stored in XMLList because for the primitive case, not having some sort of delimeter will produce unexpected results. This is tracked in (43) Review algorithms and propose guideline for algorithms that return lists of non-XML objects.

Working group agrees to postpone discussion of this until (43) has been resolved.

(23) Modify ToXML to parse string as escaped document fragment
Not done

(24) Review XML 1.1 spec to determine whether to reference or embed productions associated with characters.
The productions are small enough to embed. Change is tracked by work items (44) and (45). The second is to determine the state of the various W3C specifications.

(25) Ask Markus on how Cf characters are used and whether we can get away with removing them when processing XML literals.
Inherit E3 rules for pre-processing source to remove Cf. These are not not significantly more important in XML literals than they are in string literals. Cf characters are still legitimate and can be found in XML documents read from external sources.

(26) Modify algorithms to test for primitive values rather than use [[Class]] internal property
Made global changes throughout document.

(27) Move W3C DOM element references to non-normative section
Not done

(28) Add algorithm that defines deep-copy
Completed. Details to be reviewed during “Review New Sections” agenda item.

(29) Consider if there is a better place to store XML settings
No better place found than the XML object.

(30) Determine why proposal for settings is stored in an XML object
No good reason. Settings object is nuked. Properties will live directly off of XML constructor. Work item (46) tracks this.

(31) Define how built-in object model is called on XML data types

Prototype objects will exist for XML, XMLList. [[prototype]] members on these will be set to Object.prototype. Method lookup looks at contained objects first. E.g. When a method is invoked on an instance of XMLList, method look up is done on prototype objects in the following order: XMLList.prototype, XML.prototype, String.prototype, Object.prototype. Prototypes will be mutable.

(32) Rework insertAfter, insertBefore to insertChildAfter, insertChildBefore
Not done

(33) Update all calls to [[Get]] and [[Put]] to use wording consistent with E3
Made global changes throughout document.

(34) Move domNode, domNodeList sections to non-normative appendix
Not done

(35) Define algorithm for toCanonicalString
Not done

Other Issues

(7) Should the list of XMLList (or List) methods be immutable?
Agreement reached that behavior consistent with E3 where methods are in general mutable.

(11) Methods defined on XMLList and XML may hide methods added to String
This problem is only slightly worse than that with prototype based inheritance.

The hiding behavior is fine as long as the author of the new methods write their code so that they special case the single element list case and delegate when appropriate.

Discussion of New Sections

Namespaces

John introduced the working group to the concept of XML namespaces and the approach he is has started taking in integrating them into E4X.

The model John is taking is based on a Namespace object which stores a prefix. The prefix is significant in the data model and operations explicitly manage them, inserting them as necessary to preserve a tree with consistent XML namespace declarations and references.

John proposes that an initial assignment of a Namespace object to a variable sets up the prefix.

```
   var ns = new Namespace(“xyz.com”);  → ns contains { uri:“xyz.com”, prefix:”ns” }
   x = <ns:foo xmlns:ns=”xyz.com”><a/></ns:elem>
   x.ns::bar = “test?”;
  print(x)
```

Both Waldemar and Rok strongly oppose this idea as data from the context effects the meaning of the expression. They both believe that if (47) the prefix is part of the namespace, it should be explicit in the constructor. e.g. var ns = new Namespace(“ns”, “abc.com”)

John notes that in the model where namespace declarations are automatically added, adding elements is safe because prefix declarations can be added to ensure a document is consistent with respect to namespace declarations.

Waldemar asks what occurs (48) if an attribute is added with a conflicting namespace. e.g.

```
   var ns = new Namespace(“abc.com”);
   x.@ns:test = 4;
```

John recognizes this as an issue. This would have to be an error on assignment, or a new prefix would have to be generated.

Waldemar asks (49) what determines if URI’s in a namespace are equal. For instance are URI’s case insensitive? Rok will look up the answer.

Rok asks how XML DOM and XQuery handles namespaces.

John states that XML DOM punts on the issue and requires DOM client to explicitly managed XML declarations. The data model allows state that is inconsistent with respect ot namespaces. On serialization, namespaces are controlled by the prefix, not the URI because the prefix is captured by the namespace declaration regardless of the URI.

John is not sure how XQuery handles this and will (50) lookup the answer.

During the meeting's discussion a model was suggested wherein a namespace is not bound to any prefix when the namespace is created. Furthermore, namespace references in XML element tags and attributes refer to the full URI's. The namespace prefixes are found upon XML serialization by searching for the namespace URI in the set of xmlns:prefix="URI" definitions defined in the outer XML. A normalization method would be available to construct these in if needed.

Waldemar notes that for the case where XML is being constructed bottom up, having the runtime manage declarations results many unnecessary declarations.

The working group agrees that significant additional thought is needed before the model can be decided on. Issues to think about:

- (51) When should prefixes be relevant?
- (52) Should data model allow partially defined state?
- (53) Should E4X automatically add prefixes to data model to ensure consistency?

To help drive discussions, Rok will (54) look for use cases using namespaces.

7.4 ToQualifiedName

Waldemar notes that the algorithm for ToQualifiedName captures the closure. In particular it walks the scope chain to resolve the namespace portion of the identifier. This has the problem that it forces implementations to preserve names of variables in the scope chain (i.e. locals), in order to resolve.

The problem exists for eval, but eval is a special form that can be detected. Functions and methods that contain eval end up carrying around significantly more runtime state. The same requirement in ToQualifiedName is significantly worse because this means the state is needed whenever there is a bracket “[]” member access or any other namespace use.

Waldemar adds that another problem with its current definition is that, since it assumes that a namespace prefix is also defined as a variable in scope, it will fail if called on a prefix extracted from a piece of XML or a prefix that happens to alias into a local or global variable.

John will (55) consider ToQualifiedName and how it can work without capturing variables in scope chain.

6.1.1.4 DeepCopy

Following comments were made on the algorithm:
Step 2: Use of term copy is ambiguous.

Step 5 and 7: Set parent property

These changes are already made in the 2003-06-06 E4X draft.

9.5.1 Equality Operators

A question was raised regarding step 3 where two values x and y of type XML compare equal if ToString(x) == ToString(y).

Rok and Waldemar note that there are some cases where XML is not structurally equivalent even though their textual representations are equivalent. e.g.

```
  x = <>&lt;foo&gt;&lt;bar/&gt;&lt;/foo&gt;<>
  y = <foo><bar/></foo>

print(String(x));
  → <foo><bar/></foo>
print(String(y));
  → <foo><bar/></foo>
```

Waldemar believes that users generally want a comparison test for XML that only looks at the structure of the XML, or only looks at the text representation of the XML, not both.

John states that the textual comparison was added primarily for the primitive case and that he is okay to (56) modify the algorithm to add a check so the test is only made for this case.