ECMA/TC39/2002/00

Draft Standard ECMA-999
September 2002

ECMA International

Standardizing Information and Communication Systems

ECMAScript for XML (E4X) Specification

Draft 01 – September 2002

Phone: +41 22 849.60.00 - **Fax:** +41 22 849.60.01 - **URL:** http://www.ecma.ch - **Internet:** helpdesk@ecma.ch

Draft Standard ECMA-999
September 2002

ECMA International

Standardizing Information and Communication Systems

ECMAScript for XML (E4X) Specification

Phone: +41 22 849.60.00 - **Fax:** +41 22 849.60.01 - **URL:** http://www.ecma.ch - **Internet:** helpdesk@ecma.ch

AWB doc00010.doc 26-09-19 16,23

Brief History

TBD.

This ECMA Standard has been adopted by the ECMA General Assembly of ....

Table of contents

1 Scope

This standard defines the syntax and semantics of ECMAScript for XML (E4X), a set programming language extensions adding native XML support to ECMAScript.

2 Status of this Document

This is a working draft produced to motivate and facilitate discussions related to E4X with the goal of creating a general purpose, cross platform, vendor neutral E4X standard. Comments and suggestions are solicited and encouraged.

3 Motivation

3.1 The Rise of XML Processing

Developing software to create, navigate and manipule XML data is a significant part of every Internet developer's job. Developers are inundated with data encoded in the eXtensible Markup Language (XML). Web pages are increasingly encoded using XML vocabularies, including XHTML and Scalable Vector Graphics (SVG). On mobile devices, data is encoded using the Wireless Markup Language (WML). Web services interact using the Simple Object Access Protocol (SOAP) and are described using the Web Service Description Language (WSDL). Deployment descriptors, project make files and configuration files and now encoded in XML, not to mention an endless list of custom XML vocabularies designed for vertical industries. XML data itself is even described and processed using XML in the form of XML Schemas and XSL Stylesheets.

3.2 Current XML Processing Approaches

Current XML processing techniques require ECMAScript programmers to learn and master a complex array of new concepts and programming techniques. The XML programming models often seem heavyweight, complex and unfamiliar for ECMAScript programmers. This section provides a brief overview of the more popular XML processing techniques.

3.2.1 The Document Object Model (DOM)

One of the most common approaches to processing XML is to use a software package that implements the interfaces defined by the XML DOM (Document Object Model). The XML DOM represents XML data using a general purpose tree abstraction and provides a tree-based API for navigating and manipulating the data (e.g., getParentNode(), getChildNodes(), removeChild(), etc.).

This method of accessing and manipulating data structures is very different from the methods used to access and manipulate native ECMAScript data structures. ECMAScript programmers must learn to write tree navigation algorithms instead of object navigation algorithms. In addition, they have to learn a relatively complex interface hierarchy for interacting with the XML DOM. The resulting XML DOM code is generally harder to read, write, and maintain than code that manipulates native ECMAScript data structures. It is more verbose and often obscures the developer's intent with lengthy tree navigation logic. Consequently, XML DOM programs require more time, knowledge and resources to develop.

3.2.2 The eXtensible Stylesheet Language (XSLT)

XSLT is a language for transforming XML documents into other XML documents. Like the XML DOM, it represents XML data using a tree-based abstraction, but also provides an expression language called XPath designed for navigating trees. On top of this, it adds a declarative, rule-based language for matching portions of the input document and generating the output document accordingly.

From this description, it is clear that XSLT's methods for accessing and manipulating data structures are completely different from those used to access and manipulate ECMAScript data structures. Consequently, the XSLT learning curve for ECMAScript programmers is quite steep. In addition to learning a new data model, ECMAScript programmers have to learn a declarative programming model, recursive decent processing model, new expression language, new XML language syntax, and a variety of new programming concepts (templates, patterns, priority rules, etc.). These differences also make XSLT code harder to read, write and maintain for the ECMAScript programmer. In addition, it is not possible to use familiar development environments, debuggers and testing tools with XSLT.

3.2.3 Object Mapping

Several have also tried to navigate and manipulate XML data by mapping it to and from native ECMAScript objects. The idea is to map XML data onto a set of ECMAScript objects, manipulate those objects directly, then map them back to XML. This would allow ECMAScript programmers to reuse their knowledge of ECMAScript objects to manipulate XML data.

This is a great idea, but unfortunately it does not work. Native ECMAScript objects do not preserve the order of the original XML data and order is significant for XML. Not only do XML developers need to preserve the order of XML data, but they also need to control and manipulate the order of XML data. In addition, XML data contains artifacts that are not easily represented by the ECMAScript object model, such as attributes, comments and mixed element content.

3.3 The E4X Approach

ECMAScript for XML was envisioned to address these problems. E4X extends the ECMAScript object model with native support for XML data. It reuses familiar ECMAScript operators for creating, navigating and manipulating XML, such that anyone who has used ECMAScript is able to start using XML with little or no additional knowledge. The extensions include a native XML data type, XML literals and a small set of new operators useful for common XML operations, such as searching and filtering.

E4X applications are smaller and more intuitive to ECMAScript developers than comparable XSLT or DOM applications. They are easier to read, write and maintain requiring less developer time, skill and specialized knowledge. The net result is reduced code complexity, tighter revision cycles and shorter time to market for Internet applications. In addition, E4X is a lighter weight technology enabling a wide range of mobile applications.

4 Design Principles

The following design principles are used to guide the development of E4X and encourage consistent design decisions. They are listed here to provide insight into the design rational and to anchor discussions on desirable E4X traits

- **Simple:** One of the most important objectives of E4X is to simplify common programming tasks. Simplicity should not be compromised for interesting or unique features that do not address common programming problems.

- **Consistent:** The design of E4X should be internally consistent so developers can anticipate its behaviour. To the extent possible, the logical data model for XML data should mirror that of native ECMAScript objects. Developers already familiar with ECMAScript objects should be able to begin using XML objects with minimal surprises.

- **Familiar:** Common operators available for manipulating ECMAScript objects should also be available for manipulating XML data. The semantics of the operators should not be surprising to those familiar with ECMAScript objects.

- **Minimal:** Where appropriate, E4X defines new operators for manipulating XML that are not currently available for manipulating ECMAScript objects. This set of operators should be kept to a minimum to avoid unnecessary complexity. It is a non-goal of E4X to provide, for example, the full functionality of XPath.

- **Complementary:** E4X should integrate well with other languages designed for manipulating XML, such as XPath, XSLT and XML Query. Specifically, E4X should be able to invoke complementary languages when additional expressive power is needed without compromising the simplicity of the E4X language itself. In addition, invoking E4X code from other host languages should be possible.

- **Loose Coupling:** To the degree practical, E4X operators will enable applications to minimize their dependencies on external data formats. For example, E4X applications should be able to extract a value deeply nested within an XML structure, without specifying the full path to the data. Thus, changes in the containment hierarchy of the data will not require changes to the application.

- **Secure [Edition 4]**: E4X for ECMAScript Edition 4 should support ECMAScript Edition 4 security mechanisms, such as access control and visibility. In addition, E4X for Edition 4 should support verification of data integrity of XML data types using a Schema.

This list is evolving and may be revised as the specification progresses.

Free printed copies can be ordered from:
ECMA
114 Rue du Rhône
CH-1204 Geneva
Switzerland

Fax: +41 22 849.60.01
Email: documents@ecma.ch

Files of this Standard can be freely downloaded from the ECMA web site (www.ecma.ch). This site gives full information on ECMA, ECMA activities, ECMA Standards and Technical Reports.

ECMA
114 Rue du Rhône
CH-1204 Geneva
Switzerland

See inside cover page for obtaining further soft or hard copies.