ECMAScript Mobile Profile

A Wireless Markup Scripting Language

Version 9-September-2002

Open Mobile Alliance
OMA-WAP-ESMP-v1_0-20020903-C

Continues the Technical Activities
Originated in the WAP Forum

A list of errata and updates to this document is available from the Open Mobile Alliance™ Web site, http://www.openmobilealliance.org/, in the form of SIN documents, which are subject to revision or removal without notice

Document History		
WAP-271-ESMP	27-May-2002	Initial Public Draft
WAP-271-ESMP	30-June-2002	Post Architectural Consistency Review
WAP-271-ESMP	19-July-2002	CR-Wap-271-ESMP-NOKIA-20020630-ARCHCON CR-Wap-271-ESMP-NOKIA-20020715-OWCOMMENTS CR-Wap-271-ESMP-NOKIA-20020718-

		IBMCOMMENTS
WAP-271-ESMP	21-August-2002	Followup Architectural Consistency Review
OMA-WAP-ESMP-v1_0-20020903	9-September-2002	Draft Version
OMA-WAP-ESMPv1_0-20020903-C	9-September-2002	Current Version

Contents

1. Scope

Open Mobile Alliance (OMA) is a consortium of mobile industry companies working to define an industry-wide specification for developing applications that operate over wireless communication networks. The scope for the OMA is to define a set of specifications to be used by service applications. The wireless market is growing very quickly and reaching new customers and services. To enable operators and manufacturers to meet the challenges in advanced services, differentiation and fast/flexible service creation, OMA defines a set of protocols in transport, session and application layers. For additional information on the Wireless Application Environment, refer to [WAESPEC].

This specification describes the OMA wireless markup scripting language known as ECMAScript – Mobile Profile (ESMP). It is strongly based upon ECMAScript Release 3 [ECMA262] from the ECMA TC-39 Working Group. The same ECMA working group has also released a profile of the ECMAScript language for mobile devices [ECMA327]. The language has been modified to better support low bandwidth communication and thin clients. All changes from [ECMA327] are reflected in this specification and noted as static conformance requirements. ESMP should be used together with the OMA profile of the XHTML Markup Language [XHTMLMP] to provide added intelligence to the clients.

The specification will concentrate on three areas:

- Language differences between the ECMAScript Mobile Profile and Wireless Markup Scripting Language [WMLSCRIPT]
- Language syntax differences between [ECMA262] and ECMAScript Mobile Profile.
- The environment in which ECMAScript – Mobile Profile executes.

2. References

2.1. Normative References

CREQ "Specification of WAP Conformance Requirements", WAP Forum™, WAP-221-CREQ, URL: http://www.wapforum.org/

DOM2CORE "Document Object Model (DOM) Level 2 Core Specification", Version 1.0, W3C Recommendation, 13 November, 2000, URL: http://www.w3.org/TR/2000/REC-DOM-Level-2-Core-20001113

DOM2EVENTS "Document Object Model (DOM) Level 2 Events Specification", Version 1.0, W3C Recommendation, 13 November, 2000, URL: http://www.w3.org/TR/2000/REC-DOM-Level-2-Events-20001113

ECMA262 Standard ECMA-262, "ECMAScript Language Specification – Edition 3", December 1999, URL: ftp://ftp.ecma.ch/ecma-st/Ecma-262.pdf

ECMA327 Standard ECMA-327, "ECMAScript 3rd Edition Compact Profile", June 2001, URL: ftp://ftp.ecma.ch/ecma-st/Ecma-327.pdf

HTML401 "HTML 4.01 Specification", W3C Recommendation, 24 December, 1999, URL: http://www.w3.org/TR/html401/

HTTPSM "HTTP State Management Specification", WAP Forum™, WAP-223-HTTPSM, URL: http://www.wapforum.org/

IEEE754 "IEEE Std 754-1985 Revision of Reaffirmed 1990 IEEE Standard for Binary Floating-Point Arithmetic Recognized as an American National Standard (ANSI) IEEE Std 754", URL: http://standards.ieee.org/reading/ieee/std/busarch/754-1985.pdf (restricted access).

RFC2109 "HTTP State Management Mechanism", D. Kristol, L. Montulli, February 1997, URL: http://www.ietf.org/rfc/rfc2109.txt

RFC2119 "Key words for use in RFCs to Indicate Requirement Levels", S. Bradner, March 1997, URL: http://www.ietf.org/rfc/rfc2119.txt

RFC2396 "Uniform Resource Locators (URL)", T. Berners-Lee, R. Fielding, L. Masinter, August 1998, URL: http://www.ietf.org/rfc/rfc2396.txt

UNICODE "The Unicode Standard, Version 3.0", Addison-Wesley, January 2000,ISBN 0-201-61633-5 URL: http://www.unicode.org/unicode/standard/standard.html

WAESPEC "Wireless Application Environment Specification - Version 2.1", Open Mobile Alliance™. OMA-WAP-299-WAESpec-V2_1. URL:http://www.openmobilealliance.org/

WMLSCRIPT "WMLScript Language Specification", WAP Forum™, WAP-193-WMLS, URL: http://www.wapforum.org/

WINA "WAP WINA Process Document", WAP Forum™, WAP-212-WINAProcess, URL: http://www.wapforum.org/

XHTML "XHTML™ 1.0: The Extensible HyperText Markup Language - A Reformulation of HTML 4 in XML 1.0", W3C Recommendation 26 January 2000, URL: http://www.w3.org/TR/2000/REC-xhtml1-20000126

XHTMLMOD "Modularization of XHTML™", W3C Recommendation, 10 April 2001, URL: http://www.w3c.org/TR/2001/REC-xhtml-modularization-20010410

XHTMLMP "XHTML Mobile Profile 1.1", Open Mobile Alliance™. OMA-WAP-XHTMLMP-V1_1. URL: http://www.openmobilealliance.org/

2.2. Informative References

GOODMAN	"JavaScript Bible, 4th Edition", Danny Goodman, 2001, Hungry Minds. Inc., ISBN 0-7645-3342-8
MCFARLANE	"Professional JavaScript", Nigel McFarlane et al., 1999, Wrox Press Ltd., ISBN 1-861002-70-X
MICROSOFT	"Microsoft Developers Network (MSDN) – Jscript", 2002, URL: http://msdn.microsoft.com/library/
NETSCAPE	"Client-side Javascript Reference, v1.3", Netscape Communications Corporation, 1999, URL: http://developer.netscape.com/docs/manuals/js/client/jsref/index.htm
WCSS	"WAP CSS Specification", WAP Forum™, WAP-239-WCSS, URL: http://www.wapforum.org/
WML1	"Wireless Markup Language - Version 1.3", WAP Forum™, WAP-191-WML, URL:http://www.wapforum.org/
WMLSL	"WMLScript Standard Libraries Specification", WAP Forum™, WAP-194-WMLSL, URL:http://www.wapforum.org/
WOOTTON	"JavaScript – Programmer's Reference", Cliff Wootton, 2001, Wrox Press Ltd., ISBN 1-861004-59-1
XHTMLBASIC	"XHTML Basic - W3C Recommendation 19 December 2000", URL: http://www.w3.org/TR/2000/REC-xhtml-basic-20001219

2.3. How to Read this Document

This section is informative.

This specification draws heavily upon a number of existing standards, and assumes familiarity with:

- The ECMAScript Language Specification [ECMA262]
- Document Object Model (DOM) Level 2 Core Specification [DOM2CORE]

This specification is written for two audiences. First and foremost, it is directed towards implementers of ECMAScript Mobile Profile. Secondly, it is written to be a reference for developers. As stated elsewhere in this document ECMA262 is the normative basis for the language syntax. To reduce clutter, normative statements relative to ECMA262 (per RFC2119) are made only when there is a difference between this specification and ECMA262. While this specification is not written as a tutorial, examples may be given, and informative text from relevant specifications may be included. The examples are not exhaustive, and are generally informative only. These examples do not stand as the normative definition, but are there to support normative prose.

In all cases where there may be a question or ambiguity in the specification, source standards always take precedent, unless explicitly noted otherwise.

Where possible reference pointers to base standards are given. They are noted as:

Reference: Reference Document Chapter

Statements which cause the generation of, or are a part of, a static conformance requirement are noted with a boxed, drop shadowed statement:

This statement MUST be construed as a static conformance requirement.

Notes which relate the specification to WMLScript are noted as:

WMLScript Note: A note about wmlscript

References to implementations from Netscape or Microsoft may be referred to by either company name (i.e. Netscape), product name (i.e. Internet Explorer or IE) or scripting name (i.e. Javascript or Jscript). These are used interchangably.

3. Definitions and Abbreviations

3.1. Conventions

The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” in this document are to be interpreted as described in [RFC2119].

All sections and appendixes, except “Scope” (Section 1) and “Introduction” (Section 4) are normative, unless they are explicitly indicated to be informative.

3.2. Definitions

Client - a device (or application) that initiates a request for connection with a server.

Content - subject matter (data) stored or generated at an origin server. Content is typically displayed or interpreted by a user agent in response to a user request.

Content Encoding - when used as a verb, content encoding indicates the act of converting a data object from one format to another. Typically the resulting format requires less physical space than the original, is easier to process or store and/or is encrypted. When used as a noun, content encoding specifies a particular format or encoding standard or process.

Content Format – actual representation of content.

Device - a network entity that is capable of sending and receiving packets of information and has a unique device address. A device can act as both a client and a server within a given context or across multiple contexts. For example, a device can service a number of clients (as a server) while being a client to another server.

ECMAScript Mobile Profile - a scripting language used to program the mobile device. ECMAScript Mobile Profile is a syntactic subset of the ECMAScript scripting language defined in ECMA262. It includes object extensions from both JavaScript and W3C specified functionality.

JavaScript - a *de facto* standard language that can be used to add dynamic behaviour to HTML documents. JavaScript is one of the originating technologies of ECMAScript.

Origin Server - the server on which a given resource resides or is to be created. Often referred to as a Web server or an HTTP server.

Resource - a network data object or service that can be identified by a URL. Resources may be available in multiple representations (e.g. multiple languages, data formats, size and resolutions) or vary in other ways.

Server - a device (or application) that passively waits for connection requests from one or more clients. A server may accept or reject a connection request from a client.

User - a user is a person who interacts with a user agent to view, hear or otherwise use rendered content.

User Agent - a user agent (or content interpreter) is any software or device that interprets markup language such as XHTML, script language, such as ECMAScript or resources. This may include textual browsers, voice browsers, search engines, etc.

Web Server - a network host that acts as an HTTP server.

WML - the Wireless Markup Language is a hypertext markup language used to represent information for delivery to a narrowband device, e.g. a phone.

3.3. Abbreviations

API	Application Programming Interface
DOM	Document Object Model
ECMA	European Computer Manufacturer Association
EFI	External Functionality Interface
ESMP	ECMAScript Mobile Profile
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
IE	Internet Explorer
OMA	Open Mobile Alliance
RFC	Request For Comments
URL	Uniform Resource Locator
UTF	UCS Transformation Format
W3C	World Wide Web Consortium
WAP	Wireless Application Protocol
WINA	WAP Interim Naming Authority
WWW	World Wide Web
XHTMLMP	XHTML - Mobile Profile

4. Introduction

This section is informative.

The OMA has recognized that convergence between the wired Web and wireless devices, as targeted by the WAP architecture, is an important step toward bringing wireless devices into the mainstream. As a part of the convergence process, OMA has redefined the markup language that is to be used by WAP devices [XHTMLMP]. This specification is the redefinition of the scripting language that accompanies that markup.

Goals for this scripting language are:

- The language specified can be executed from within common browsers found on the wired web (i.e. Netscape Communicator and Microsoft Internet Explorer).
- The language must maintain the standard programming paradigms that are associated with wired implementations of script.
- The language must represent a minimal functional implementation, to support small devices.
- The language must be extensible, allowing for the support of future extensions either standard or proprietary.
- The language should support enough runtime discovery mechanisms to support error management through version control, and to support identification of optional, extended or proprietary features.

4.1. Why Scripting?

ECMAScript Mobile Profile is designed to provide general scripting capabilities for the mobile applications environment. Specifically, ECMAScript Mobile Profile is used to complement the XHTML markup language [XHTMLMP]. XHTMLMP is based on the W3C Extensible Hypertext Markup Language [XHTML]. It is designed to be used for specifying application content for small screen, narrowband devices like cellular phones and pagers. This content can be represented with text, images, selection lists etc. Formatting and styling [WCSS] can be used to make the user interfaces more readable as long as the client device used to display the content can support it. However, all this content is *static* and there is no way to extend the language without modifying the markup itself. The following list contains some capabilities that are not supported by XHTML:

- The ability to check the validity of user input (forms validation)
- The ability to apply mathematic and procedural logic locally to document data.
- Providing access to facilities of the device. For example, on a phone, allow the programmer to make phone calls, send messages, add phone numbers to the address book, access the SIM card etc.
- The ability to generate messages and dialogs locally, reducing the need for expensive round-trip for alerts, error messages, confirmations etc.
- The ability to handle events
- The ability to allow the dynamic creation and/or modification of documents on the client.

ECMAScript Mobile Profile is designed to overcome these limitations and to provide programmable functionality that can be used over narrowband communication links and in clients with limited capabilities.

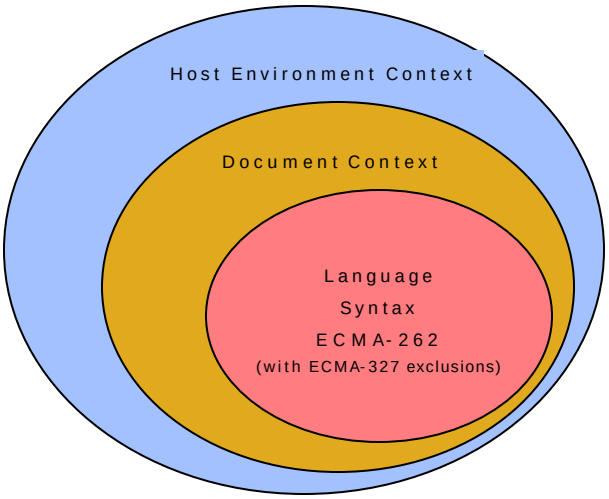
4.2. Language Construction

ECMAScript Mobile Profile (ESMP) is actually a combination of a number of standards and technologies. The basis of ESMP is the scripting language codified by ECMA as ECMA262. This provides the syntax and language semantics. The language is a general purpose scripting language that can be applied in many environments. Since the mobile applications environment is oriented toward small wireless devices, we subset the language by further including

ECMA327 (ECMAScript Compact Profile) as specified by ECMA. This specification allows us to remove a small number of ECMAScript features which are deemed too memory or compute intensive for the wireless environment.

As pointed out in the ECMA262 specification, the language has no context specified in which to execute. So ESMP must provide the execution environment. The context is provided as a set of "host objects" which make available access to the markup document which contains the script, and access to the host environment in which that document executes. Specifically, access to the document is provided by including a subset of the W3C Document Object Model interface DOM2CORE to the markup document. And access to the environment is given by importing four common compatible objects taken from JavaScript (**`history`** object, **`navigator`** object, **`location`** object and the **`document`** object).

Since ECMA262 and DOM2CORE are supported by the major web browsers on the wired web, ESMP represents a proper subset of these browsers. Thus scripts written to run in the ESMP environment should function (taking into account version etc.) on the wired web.



4.3. Language Differences

Specific differences between WMLScript [WMLSCRIPT] and ECMAScript Mobile Profile, and between ECMAScript Mobile Profile and [ECMA262] are detailed in Appendix D and Appendix E.

5. Language Syntax

Unless explicitly stated [ECMA262] is normative for the definition of the syntax used in ECMAScript Mobile Profile.

5.1. Lexical Conventions

5.1.1. Language and Character Set

Reference: ECMA-262 Section 6

ECMAScript Mobile Profile program source text consists of a sequence of code points from the [UNICODE] character set. New code points may be added to the [UNICODE] standard so ECMAScript Mobile Profile implementations must accept any value in the allowed (but possibly unallocated) range specified in the [UNICODE] standard.

At a minimum, ECMAScript Mobile Profile implementations MUST support the UTF-8 and UTF-16 encoding of code points into bit patterns for the source text.

The program source encoding does not imply anything about an ECMAScript Mobile Profile implementation's internal representation of strings. Note: This is a change from [ECMA262] which says it only supports UTF-16 input.

As long as the source text character set and encoding can be mapped to and from UNICODE (e.g., ASCII), other character sets and encodings may be supported. It is intended that documents containing ECMAScript Mobile Profile need not conform to a very limited set of characters and encodings to be acceptable.

An ECMAScript Mobile Profile implementation's internal representation of text (e.g., strings) is not constrained in any way by this specification. However, the specification of certain language features (e.g., **`string.charCodeAt()`**) mandates exposure of a specific 16-bit representation. Note: This is also a change from [ECMA262] which says it only supports 16 bit characters internally.

Implementations MUST support character level indexing for appropriate methods, independent of internal character length.

When "characters" are output, the encoding for output depends on the output destination. For example, output to an existing document usually adopts the existing encoding of the document. Controlling output encoding is not a part of this specification and will generally be implied by the output destination or take on a setting by default.

ECMAScript Mobile Profile is a case-sensitive language. All language keywords, variables, objects, methods and properties must use the proper capitalization of letters.

5.1.2. Whitespace and Line Terminator Characters

Reference: ECMA-262 Section 7.2

Whitespace and line terminator characters are used to improve source text readability and to separate tokens from each other, but are otherwise insignificant. Whitespace may occur within literal strings and is therein considered significant.

Whitespace::

Symbol	Character Represented	Codepoint
<TAB>	Tab	\u0009
<VT>	Vertical Tab	\u000B

<FF>	Form Feed	\u000C
<SP>	Space	\u0020
<NBSP>	Non-breaking Space	\u00A0
<USP>	Unicode space separators	\u2000-\u200A

Reference: ECMA-262 Section 7.3

Line terminators may not occur within literal strings or tokens.

Line Terminator::

Symbol	Character Represented	Codepoint
<LF>	Line Feed	\u000A
<CR>	Carriage Return	\u000D
<ULS>	Unicode Line Separator	\u2028
<UPS>	Unicode Paragraph Separator	\u2029

5.1.3. Semicolon Usage

Unlike ECMAScript [ECMA262], semicolons MUST be included as statement separators, and are required following ECMAScript Mobile Profile language statements.

5.1.4. Comments

Reference: ECMA-262 Section 7.4

The language defines two comment constructs: *line comments* (i.e., start with // and end in the end of the line) and *block comments* (i.e., consisting of multiple lines starting with /* and ending with */).

```
//This is a single line comment example.
/* This is an example of a multi-line block comment that extends for
a few lines. This is typical of a 'C' style comment. */
```

It is illegal to have nested block comments.

5.1.5. Language Tokens

Reference: ECMA-262 Section 7.5

Reserved words, keywords and future reserved words are as defined in [ECMA262].

5.1.6. Identifiers

Reference: ECMA-262 Section 7.6

Identifiers are as defined in [ECMA262].

WMLScript Note: '$'is a valid character in a ECMAScript Mobile Profile identifier, whereas it was illegal in WMLScript. [WMLSCRIPT]

Examples of legal identifiers:

```
aVar $speed NEW_ADDRESS skram_ttenneb myLoopCnt myloopcnt $$BigVar$$
```

Note: **`myLoopCnt`** and **`myloopcnt`** are different variables since identifiers are case sensitive.

Examples of illegal identifiers:

```
while  for  if  4things  35345  dotted.var
```

Note: **`while, for`** and **`if`** are reserved words.

5.1.7. Punctuators

Reference: ECMA-262 Section 7.7

Punctuators are as defined in [ECMA262].

5.1.8. Literals

5.1.8.1. Null

Reference: ECMA-262 Section 7.8.1

Null literals are as defined in [ECMA262].

5.1.8.2. Boolean

Reference: ECMA-262 Section 7.8.2

Boolean literals are as defined in [ECMA262].

Boolean Literal::

true

false

WMLScript Note: In WMLScript these literals were capitalized. [WMLSCRIPT]

5.1.8.3. Numeric

Reference: ECMA-262 Section 7.8.3

Numeric literals are as defined in [ECMA262].

WMLScript Note: Literals referred to as “binary” literals in WMLScript [WMLSCRIPT] are referred to as HexIntegerLiterals in ECMAScript. [ECMA262]

5.1.8.4. String

Reference: ECMA-262 Section 7.8.4

String literals are as defined in [ECMA262].

WMLScript Note: Slash is no longer an escapable character and octal encoding is no longer supported. [WMLSCRIPT

For those characters that are not representable with strings, ECMAScript Mobile Profile supports special escape sequences by which these characters can be represented:

Sequence	Character represented	Code Point	Symbol
\’	Apostrophe or single quote	\u0027	'
\"	Double quote	\u0022	"
\\	Backslash	\u005C	\
\/	Slash	\u002F	/
\b	Backspace	\u0008	<BS>
\f	Form feed	\u000C	<FF>
\n	Newline	\u000A	<LF>
\r	Carriage return	\u000D	<CR>
\t	Horizontal tab	\u0009	<TAB>
\x*hh*	The character with the encoding specified by two hexadecimal digits *hh* (Latin-1 ISO8859-1)		
\u*hhhh*	The Unicode character with the encoding specified by the four hexadecimal digits *hhhh*.		

5.1.8.5. Regular Expression

Reference: ECMA-262 Section 7.8.5

Regular expression Literals are as defined in [ECMA262].

5.2. Variable and Data Types

5.2.1. String Type

Reference: ECMA-262 Section 8.4

The string type is as defined in [ECMA262].

5.2.2. Numeric Types

Reference: ECMA-262 Section 8.5

The numeric data type is a 64-bit IEEE-754 floating number as defined in [ECMA262]. For OMA devices, precision need only be displayed to 14 decimal digits (even though IEEE-754 64-bit floats guarantee ~21 digits).

The language MUST support the display of numbers to a minimum of 14 digits of accuracy.

It is important to note that the [ECMA262] does not specify actual internal numeric representations, only that the numbers stored act as if they are indistinguishable from full IEEE-754 64 bit floats. Hence, the requirement from [ECMA262] is to maintain accuracy, and to provide a uniform external representation, not to mandate internal format. Script language implementers are free to use any appropriate mathematical language optimisations for the purposes of saving space or reducing computing requirements.

5.2.3. Object Type

Reference: ECMA-262 Section 8.6

The Object type is as defined in [ECMA262].

5.2.4. Other Types

5.2.4.1. Undefined

Reference: ECMA-262 Section 8.1

The Undefined type is as defined in [ECMA262].

The Undefined type has exactly one value, called "**undefined**". Any variable that has not been assigned a value has the value "**undefined".**

5.2.4.2. Null

Reference: ECMA-262 Section 8.2

The Null type is as defined in [ECMA262].

The Null type has exactly one value, called "**null".**

5.2.4.3. Boolean

Reference: ECMA-262 Section 8.3

The Boolean type is as defined in [ECMA262].

The Boolean type represents a logical entity having two values, called **"true"** and "**false"**.

5.3. Type Conversions

ECMAScript Mobile Profile is a weakly typed language and like both WMLScript [WMLSCRIPT] and ECMAScript [ECMA262] performs automatic type conversion as needed. Variables may contain values of any standard type:

- Undefined
- Null
- Boolean
- String
- Number
- Object

Reference: ECMA-262 Section 9

Operators described in ECMAScript conversions operate as specified in [ECMA262].

Legal conversions of data types are summarized in the following table.

From				
	To Boolean	To Number	To String	To Object
Undefined	‘false’	NaN (not-a-number)[IEEE754]	“undefined”	Error Exception
Null	‘false’	+0 (zero)	“null”	Error Exception
Boolean ‘true’		1.	“true”	Creates a new boolean object with the value ‘true’
Boolean ‘false’		+0. (zero)	“false”	Creates a new boolean object with the value ‘false’
Number ‘0’ (zero)	‘false’		“0”	Creates a new number object with the value +0 (zero)
Number (non-zero)	‘true’	Floating point value with at least 14 digits of total precision.	See ECMA-262 Section 9.8.1	Creates a new number object with the value of the starting number.
Empty string	‘false’	+0 (zero)		Creates a new **`String`** object with the value of “” (the null string)
Non-empty string	‘true’	See ECMA-262 Section 9.3.1		Creates a new **`String`** object with the value of the starting string.
Object	‘true’	Object dependant	Object dependant	

5.4. Execution Contexts

Reference: ECMA-262 Section 10

Execution context is inherited from the XHTML context. The top-level execution context in which the ECMAScript Mobile Profile code operates is defined as the context of the calling document. ECMAScript Mobile Profile cannot be called from outside an existing document context.

5.4.1. Variable Context

Reference: ECMA-262 Section 10.1.3

Variable instantiation is as defined in [ECMA262].

5.4.2. Dynamically Created Code

Reference:	ECMA-262 Section 10.1.2
Reference:	ECMA-327 Section 5.1

As defined by the ECMAScript Compact Profile [ECMA327] support for **`eval()`** is optional, and is not required for conformance to ECMAScript – Mobile Profile. In those cases where the **`eval()`** method is not supported, an **`EvalError`** exception shall be thrown as per [ECMA327].

As defined by the ECMAScript Compact Profile [ECMA327] support for the dynamic creation of functions is optional, and is not required for conformance to ECMAScript Mobile Profile.

Support for the **`eval()`** function either directly or in support of dynamic function compilation is OPTIONAL ("NOT REQUIRED for implementation" per [ECMA327]) in ECMAScript Mobile Profile.

If **`eval()`** is not supported ECMAScript – Mobile Profile MUST throw an **`EvalError`** exception when eval() is called.

5.5. Language Syntax and Semantics

Reference:	ECMA-262 Section 11

Operator semantics are as described in [ECMA262] section 11. The language semantics are defined in terms of expressions and operators:

- Expressions
  - primary expressions
  - left-hand-side expressions
  - postfix expressions
- Operators
  - unary operators
  - mulplicative operators
  - additive operators
  - shift operators
  - relational operators
  - equality operators
  - binary bitwise operators

- binary logical operators
- conditional operator
- Assignment operators
- comma operator

5.5.1. Expressions

Reference: ECMA-262 Sections 11.1,11.2,11.3

Expressions are as defined in [ECMA262].

5.5.2. Operators

Reference: ECMA-262 Sections 11.4,11.5,11.6,11.7,11.8,11.9,11.10,11,12,11.13,11.14

Operators are as defined in [ECMA262].

The following section contains a summary of the language operators available and possible return values.

Unary Operators

Operator	Result Type	Description	Note
delete	Boolean	deletes referent	ECMA-262 Section 11.4.1
void	undefined	converts unary operand to undefined type.	ECMA-262 Section 11.4.2
typeof	string	returns a description of the referent type	WMLScript Note: WMLScript returned integers ECMA-262 Section 11.4.3
++	number	increments value of referent by 1	ECMA-262 Section 11.4.4
--	number	decrements value of referent by 1	ECMA-262 Section 11.4.5
+ (unary)	number	converts unary operand to a number.	ECMA-262 Section 11.4.6
- (unary)	number	negates numeric referent	ECMA-262 Section 11.4.7
~	number (integer)	bitwise complement of integer	ECMA-262 Section 11.4.8
!	boolean	reverses Boolean sense of referent	ECMA-262 Section 11.4.9

Additive and Multiplicative Operators

Operator	Result Type	Description	Note
*	number	multiplication	ECMA-262 Section 11.5.1
/	number	division	ECMA-262 Section 11.5.2
%	number	remainder	ECMA-262 Section 11.5.3
+	number	addition	ECMA-262 Section 11.6.1,11.6.3
-	number	subtraction	ECMA-262 Section 11.6.2,11.6.3

Bitwise Shift, Relational and Equality Operators

Operator	Result Type	Description	Note
<<	number (integer)	bitwise left shift	ECMA-262 Section 11.7.1
>>	number (integer)	signed bitwise right shift	ECMA-262 Section 11.7.2
>>>	number (integer)	unsigned bitwise right shift	ECMA-262 Section 11.7.3
<	Boolean, undefined	relational less-than comparison	ECMA-262 Section 11.8.1
>	Boolean, undefined	relational greater-than comparison	ECMA-262 Section 11.8.2
<=	Boolean, undefined	relational less-than or equal comparison	ECMA-262 Section 11.8.3
>=	Boolean, undefined	relational greater-than or equal comparison	ECMA-262 Section 11.8.4
instanceof	Boolean	tests whether left-hand variable is an object of the specified type	ECMA-262 Section 11.8.6
in	Boolean	tests whether a property is in an object	WMLScript Note: Th is was not in WMLScript ECMA-262 Section 11.8.7
==	Boolean	tests for equality (allows conversions)	ECMA-262 Section 11.9.1,11.9.3
!=	Boolean	tests for inequality (allows conversions)	ECMA-262 Section 11.9.2,11.9.3
===	Boolean	tests for “strict” equality (type must match, else failure)	ECMA-262 Section 11.9.4,11.9.6
!==	Boolean	tests for “strict” inequality (type must match, else failure)	ECMA-262 Section 11.9.5,11.9.6

Logical, Conditional and Simple Assignment

Operator	Result Type	Description	Note
&	number (integer)	bitwise ‘AND’ of two integers	ECMA-262 Section 11.10
^	number (integer)	bitwise ‘XOR’ of two integers	ECMA-262 Section 11.10
|	number (integer)	bitwise ‘OR’ of two integers	ECMA-262 Section 11.10

&&	boolean	logical 'AND' of two expressions	ECMA-262 Section 11.11
||	boolean	logical 'OR' of two expressions	ECMA-262 Section 11.11
? :	any	conditional expression	ECMA-262 Section 11.12
=	any	simple assignment	ECMA-262 Section 11.13
,	any	comma – multiple evaluation	ECMA-262 Section 11.14

Compound Assignment

Operator	Result Type	Description	Note
*=	number	assignment after multiply	ECMA-262 Section 11.13.2
/=	number	assignment after division	ECMA-262 Section 11.13.2
%=	number	assignment after modulo	ECMA-262 Section 11.13.2
+=	number	assignment after addition	ECMA-262 Section 11.13.2
-=	number	assignment after subtraction	ECMA-262 Section 11.13.2
<<=	number (integer)	assignment after bitwise right shift	ECMA-262 Section 11.13.2
>>=	number (integer)	assignment after bitwise left shift	ECMA-262 Section 11.13.2
>>>=	number (integer)	assignment after bitwise left shift with zero fill	ECMA-262 Section 11.13.2
&=	number (integer)	bitwise 'AND' with assignment	ECMA-262 Section 11.13.2
^=	number (integer)	bitwise 'XOR' with assignment	ECMA-262 Section 11.13.2
|=	number (integer)	bitwise 'OR' with assignment	ECMA-262 Section 11.13.2

5.5.3. Grammar and Syntax

Reference: ECMA-262 Section 12

Statements are as defined in [ECMA262].

Reference: ECMA-327 Section 5.3

In accordance with the ECMAScript Compact Profile [ECMA327] conforming implementations support for the **`with`** statement (ECMA-262 section 12.10) is OPTIONAL. If **`with`** is not supported, use of a `with` statement results in a syntax error.

6. Native (Built-in) Objects

Native or built-in objects are those facilities built into the language that support basic language constructs and mechanisms. All of the native ECMAScript Mobile Profile objects are defined in [ECMA262].

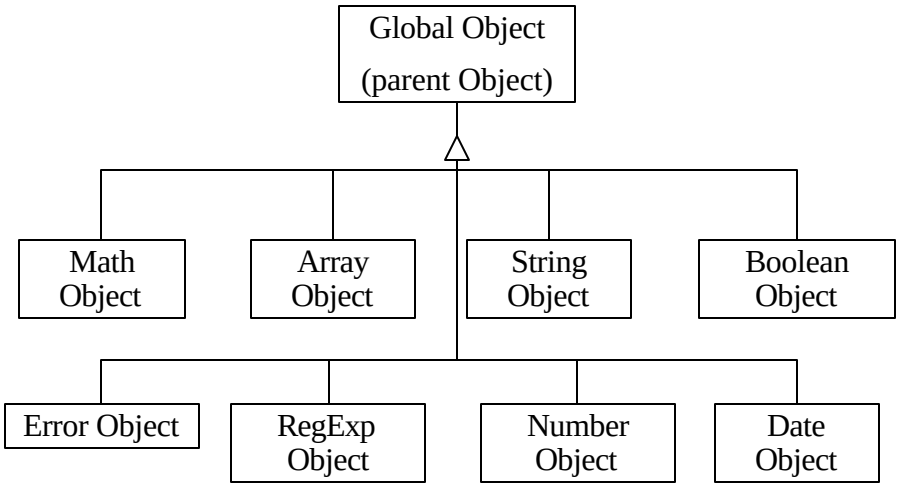
Reference: ECMA-327 Section 5.2

As specified in [ECMA327] a conforming implementation of ECMAScript Mobile Profile SHOULD NOT support addition, deletion or assignment to the properties of built-in objects, other than the global object. If the implementation does not allow modifications to built-in objects, then it SHALL throw a **`ReferenceError`** exception (See section 6.11) when evaluating such modifications.

6.1. Object Relationships

Reference: ECMA-262 Section 10.1.5, 15.1

All objects are the properties of the unique global object, which is instantiated prior to entering any execution context. In this implementation of ECMAScript the global object may be referred to as 'self', 'top', 'parent', or 'window'.



Note: Unless object constructors have "interesting" or "unique" syntax, their existence will be taken for granted and not described as part of any object method descriptions.

6.2. Object Management

6.2.1. version property

All native (built-in) objects MUST contain an enumerable, read-only version property.

This property is a string. The version property string will be of the following format: M.m.I.i

Where:

M = Major release number, corresponding to the scripting major release number. For this document the major release number is 1. The owner of this value is OMA and WINA [WINA].

m = minor release number, corresponding to documented changes or bug fixes mandated by OMA. Objects will start with the value 1 and increment for any change documented by OMA. The owner of this value is OMA and WINA [WINA].

I = Implementers Release number, corresponding to the implementation version of the object. Implementers are encouraged to start with the value 1 and increment whenever syntax and/or semantics of an object are changed. The owner of this value is the implementer of the script specification.

i = Implementers minor number. Implementers should increment this value whenever making a change to an object implementation. The owner of this value is the implementer of the script specification. This value may be used for relating behaviours to specific implementation builds. Actual utilization of this field is up to the implementer.

OMA controlled values (M.m) will only be updated if the above-mentioned criteria are met. Thus an object that is initially released as 1.1[.x.x] will remain 1.1 unless there is either a major new release of the language or there is a documented change in the functioning of the object in question.

Current defined versions for each object are specified in this document, along with a history of version changes for each object. They can be found in the chapters for each object and look like:

Version	Affected	Comment
1.1		ECMAScript initial version
1.2	fooMethod(), property1	This is an example of a change from OMA

Note: This is an extension to [ECMA262].

An implementation SHOULD make best effort to deal with versioning issues.

The actual support, in the field, for multiple versions and the identification of situations where there are either version incompatibilities or cause for execution failure based upon version is the responsibility of the implementation.

6.2.2. Object Enumeration

All non-native objects MUST be enumerable.

Note: This is an extension to [ECMA262]. This allows any object extension to be visible and its existence to be testable using standard ECMAScript constructs.

All standard ECMAScript Mobile Profile objects (objects defined in this specification) SHOULD be enumerable. This is an extension to [ECMA262].

Sample code:

```
<html>
 <head>
```

```
  <script type="text/ecmascript">
  function showContainedObjects(obj,objName) {
     document.write("-- " + objName,".version=",obj.version," --<br/>");
     for  ( var i in obj) {
       if(typeof(obj[i]) == "object" && obj[i] != null )
       document.write( objName + "." + i + " = " + obj[i] + "<br/>");
     }
  }
  </script>
 </head>
 <body>
  <p>
  <b>Current window(top) properties:</b>
  <p></p>
  <script type="text/ecmascript">
           showContainedObjects(top, "top");
  </script>
  </p>
 </body>
</html>
```

6.3. Global Object (parent Object)

Reference: ECMA-262 Section 10.1

All utility built-in functions are properties of the global object. Built-in properties of the global object are **READ-ONLY** and DONTDELETE. Properties of the global object introduced by declarations (ECMA262 section 10.1.3) may be modified but not deleted. The global object may be referred to directly as “top”, “parent”, “self”, or “window” (for compatibility with IE and Netscape).

6.3.1. Version History

Version	Affected	Comment
1.1		ECMAScript initial version

6.3.2. Properties

There is no constructor property for the global object. The global object is not directly callable.

6.3.3. Methods

A number of utility methods are associated directly with the global object.

6.3.3.1. encodeURI()

Syntax:	**encodeURI(URI)**
Argument List:	URI – a complete URI
Description:	encodes a complete URI, with knowledge of what parts of the URI to encode, and what parts, such as the "://" not to encode, as defined by RFC2396. In general the special URI characters, when used as such are not encoded. These include: ; / ? : @ & = + $, This function is I18N aware. It replaces the **escape()** function.
Return Value Type:	a URI encoded string, with appropriate characters encoded into escaped octet sequences
Errors or Exceptions:	throws a **URIError** exception if code point is invalid
Example(s):	**var encStr=encodeURI("http://www.foo.skram/card?dest=#ben dave");** **//encStr="http://www.foo.skram/card?dest=#ben%20dave"**
Reference	ECMA-262 Section 15.1.3.3

6.3.3.2. encodeURIComponent()

Syntax:	encodeURIComponent(pieceOfURI)
Argument List:	pieceOfURI – a string fragment to be encoded
Description:	encodes a string according to URI encoding rules [RFC2396], but unaware of the URI separators
Return Value Type:	a UTF-8 encoded string
Errors or Exceptions:	throws a **URIError** exception if code point is invalid
Example(s):	**var encString = encodeURIComponent("http://www.foo.skram/card?dest=#ben");** **//encString =http%3A%2F%2Fwww.foo.skram%2Fcard%3Fdest%3D%23ben**
Reference	ECMA-262 Section 15.1.3.4

6.3.3.3. decodeURI()

Syntax:	**decodeURI(URI)**
Argument List:	URI – a URI string encoded via **encodeURI()**
Description:	Unescapes encoded URI strings according to the URI separator rules in RFC2396. It replaces

	the **unescape()** function.
Return Value Type:	an unencoded URI
Errors or Exceptions:	throws a **URIError** exception if the code points defined by %XXX are out of bounds
Example(s):	**var decStr=decodeURI(http://www.foo.skram/card?dest=#ben%20dave); //decStr="http://www.foo.skram/card?dest=#ben dave"**
Reference	ECMA-262 Section 15.1.3.1

6.3.3.4. decodeURIComponent()

Syntax:	**decodeURIComponent(aString)**
Argument List:	aString – a string with UTF-8 octet encodings embedded in it
Description:	blindly decodes UTF-8 octet encoded strings
Return Value Type:	a string with the UTF-8 octet encodings (%XXX) restored to the character that it represents
Errors or Exceptions:	throws a **URIError** exception if the code points defined by %XXX are out of bounds
Example(s):	**var decStr=decodeURLComponent("http%3A%2F%2Fwww.foo.skram%2Fcard%3Fdest%3D%23ben%20dave"); // decStr = "http://www.foo.skram/card?dest=#ben dave"**
Reference	ECMA-262 Section 15.1.3.2

6.3.3.5. isFinite()

Syntax:	**isFinite(value)**
Argument List:	value – input to which toNumber is applied.
Description:	built-in function that checks for the infinity value.
Return Value Type:	Boolean - true if the value is a valid number, false if value is NaN or one of the infinity values.
Errors or Exceptions:	none
Example(s):	**testVal = isFinite(12.5); //returns true testVal = isFinite(NaN); //returns false**
Reference:	ECMA-262 15.1.2.5

6.3.3.6. isNaN()

Syntax:	**`isNaN(value)`**
Argument List:	value – input to which **`isNaN()`** is applied.
Description:	checks value for the Not-a-Number value
Return Value Type:	Boolean true if value = = the Not-a-Number value or if the value is not a number primitive, or undefined . false if value is a valid number, infinity, a boolean or null
Errors or Exceptions:	none
Example(s):	**`testVal = isNaN(“53”);  // returns false`** **`testVal = isNAN(“34GF2”); //returns true`** **`testVal = isNaN(false); //return false`** **`testVal=isNaN(“false”); //returns true`** **`testVal=isNaN(23.01E+12); //returns false`**
Reference:	ECMA-262 Section 15.1.2.4

6.3.3.7. parseInt()

Syntax:	**`myNumb = parseInt(InputString, radix)`**
Argument List:	InputString – string to be converted radix – radix applied to the conversion string. If radix is undefined then the string is converted according the leading characters, '0'=octal, '0x,0X'=hexadecimal. Otherwise assume decimal.
Description:	parses an input string into an integer value. **`parseInt()`** will scan only until it encounters an invalid integer number character
Return Value Type:	number primitive or NaN if the string cannot be resolved
Errors or Exceptions:	If the string cannot be resolved as an integer then NaN is returned
Example(s):	**`anInt = parseInt("654.321"); //anInt = 654`** **`anInt = parseInt(9A4,16); //anInt = 2468`** **`anInt = parseInt("321 oz."); //anInt = 321`** **`anInt = parseInt("hi there); //anInt = NaN`**
Reference:	ECMA-262 Section 15.1.2.2

WMLScript Note: The WMLScript version of this function did not support a radix parameter.

6.3.3.8. parseFloat()

Syntax:	**`myFloat = parseFloat(InputString)`**
Argument List:	InputString – string to be converted to a floating point number
Description:	parses an input string into a float point value. **`parseFloat()`** will scan only until it encounters an invalid floating point number character. White space at the beginning of the input string is stripped.
Return Value Type:	number primitive or NaN if the string cannot be resolved
Errors or Exceptions:	If the input string cannot be resolved as a floating point number then NaN is returned
Examples(s):	**`aFloat = parseFloat("+987.0123"); /* aFloat = 987.012 (assume only 6 digits of accuracy) */`** **`afloat=parseFloat("  -9.342JUNK"); //afloat=-9.342`** **`afloat = parseFloat("M213213.1"); //afloat=NaN`**
Reference:	ECMA-262 Section 15.1.2.3

6.3.3.9. toString()

The toString method is inherited by all native object types. Built-in class toString handlers may be different then the generic global method.

Syntax:	**`mystr = Object.toString()`** [See also **`String()`**, **`array.toString()`**]
Argument List:	
Description:	returns a string primitive representation of the receiving Object
Return Value Type:	string primitive
Errors or Exceptions:	none
Examples(s):	**`var a; var b;`** **`b=a.toString(); //b="undefined"`** **`a=345;`** **`b=a.toString(); //b="345"`** **`a=true;`** **`b=a.toString(); //b="true"`**
Reference:	ECMA-262 Section 9.8 Note: this is similar to the **`String()`** method of the **`String`** object, which is really a **`String`** object constructor.

6.3.3.10. eval()

Reference: ECMA-262 Section 10.1.2, 15.1.2.1

Support for the **`eval()`** method is OPTIONAL ("NOT REQUIRED to be implemented per [ECMA327]) in ECMAScript Mobile Profile. See Section 5.4.2.

6.4. Array Object

Reference: ECMA-262 Section 15.4

An array is an indexed collection of references to other objects or values. It is a set of properties, owned by an object, that can be accessed by name or by index position in the containing array.

An array may be constructed either by the function call **`Array(….)`** or by the explicit array object creation **`new Array(….)`**. Both are equivalent.

6.4.1. Version History

Version	Affected	Comment
1.1		ECMAScript initial version

6.4.2. Properties

6.4.2.1. length

Syntax:	**array.length**
Type:	Number (integer)
Description:	The length property of an array is 1+ the largest index assigned to a property in that array. Array indexing is zero based.
Errors or Exceptions:	none
Example(s):	**var months = new Array("Jan","Feb",Mar","Apr","May","Jun","Jul","Aug","Sep","Oct","Nov","Dec");** **cnt = months.length; //cnt = 12**
Reference	ECMA-262 Section 15.4.3, 15.4.5.2

6.4.3. Methods

6.4.3.1. concat()

Syntax:	**bigArray =firstArray.concat([array1 [, array2 [, …]]])**
Argument List:	array1 – the array to be concatenated to the firstArray object array2 etc. – 2nd, 3rd etc. arrays also to be concatenated.
Description:	Allows the joining of two or more arrays into a new, combined array object (bigArray – resultant array object). If no array parameters are specified, then the method creates a new duplicate array.
Return Value Type:	Array Object
Errors or Exceptions:	none
Example(s):	**var arrayNums = new Array(23.5,345.2,55.5);** **var arrayChars = new Array("x","y","z");** **var mixedArray = arrayNums.concat(arrayChars);** **//mixedArray = 23.5,345.2,55.5, "x","y","z"**
Reference	ECMA-262 15.4.4.4

6.4.3.2. join()

Syntax:	**newVar = myArray.join([delimiterString])**
Argument List:	delimiterString – the separator string used to separate array elements in the output string. If the string is not specified (undefined), it is set to the comma character (“,”).
Description:	Joins the contents of array entries into a single string
Return Value Type:	a String or null
Errors or Exceptions:	none
Example(s):	**myArray = new Array("This", "is", "a", "collection", "of", "words");** **str1 =myArray.join();//str1=”This,is,a,collection,of,words”;** **str=myArray.join(“ ”);//str=”This is a collection of words”;**
Reference	ECMA-262 Section 15.4.4.5

6.4.3.3. toString()

Array.toString() is identical to Array.join(“,”).

6.4.3.4. pop()

Syntax:	**anArray.pop()**
Argument List:	
Description:	Gets the last array value in an array (array[length]), deletes the value in the array and returns the value. Decreases array.length by 1
Return Value Type:	Any
Errors or Exceptions:	If the array is empty “undefined” is returned.
Example(s):	**var bytecodeArray = new Array(“nop”,”lda 1”,”mult”,”dup”);** **var nextcode = bytecodeArray.pop(); //nextcode = “dup”** **//bytecodeArray=“nop”,”lda 1”,”mult”**
Reference	ECMA-262 Section15.4.4.6 See also **push(), shift() and unshift()**

6.4.3.5. push()

Syntax:	**anArray.push(value)**
Argument List:	value – the value to be added to the end of an array
Description:	appends a value to the end of an array, increases the array.length by 1
Return Value Type:	The new length of the array is returned
Errors or Exceptions:	none
Example(s):	**var bytecodeArray = new Array("nop","lda 1","mult","dup");** **len = bytecodeArray.push("b_and"); //len=5** **//bytecodeArray="nop","lda 1","mult","dup","b_and"**
Reference	ECMA-262 Section15.4.4.7 See also **pop(), shift() and unshift()**

6.4.3.6. reverse()

Syntax:	**anArray.reverse()**
Argument List:	
Description:	The elements of an array are rearranged, so as to reverse their order
Return Value Type:	The object is returned
Errors or Exceptions:	none
Example(s):	**var ordArray = new Array(1,2,3,4,5,6);** **ordArray.reverse(); //ordArray= 6,5,4,3,2,1**
Reference	ECMA-262 Section15.4.4.8

6.4.3.7. shift()

Syntax:	**anArray.shift()**
Argument List:	
Description:	Removes the first value of an array, shifting down all indexes
Return Value Type:	Any – the value shifted out of the array
Errors or Exceptions:	returns undefined if the array is empty
Example(s):	**var bytecodeArray = new Array("nop","lda 1","mult","dup");** **var nextcode = bytecodeArray.shift(); //nextcode = "nop"** **//bytecodeArray="lda 1","mult","dup"**
Reference	ECMA-262 Section15.4.4.9

6.4.3.8. slice()

Syntax:	**newArray = anArray.slice(startIndex [, endIndex])**
Argument List:	startIndex – number (integer) endIndex – optional number (integer), if not included it defaults to the full array
Description:	takes an array and returns a new array with the elements at startIndex up to but not including the optional element pointed to by endIndex. If startIndex, or endIndex is a negative number (integer) it is treated as length+index
Return Value Type:	Array
Errors or Exceptions:	none
Example(s):	**var tArray = new Array(0,1,2,3,4,5,6);** **var slArray= tArray.slice(3); //slArray=3,4,5,6** **var slArray= tArray.slice(1,3); //slArray=1,2** **var slArray= tArray.slice(-4); //slArray=3,4,5,6**
Reference	ECMA-262 Section15.4.4.10

6.4.3.9. sort()

Syntax:	**`anArray.sort([compareFunc]);`**
Argument List:	**`compareFunc(elem1,elem2)`** – an optional function that returns a value, either negative, zero, or positive when comparing two elements (elem1,elem2) of the source array. If **`compareFunc()`** is not specified, sorting occurs based on character string value sort order.
Description:	a general array sort method where the comparison function may be supplied by the developer, based upon the contents of the array to be sorted.
Return Value Type:	the initial array, with the contents sorted according to the presented comparison function
Errors or Exceptions:	A **`ReferenceError`** exception is thrown if a specified compareFunc cannot be found. (This is an extension to ECMA-262.)
Example(s):	**`var myArray = new Array(0,5,6,2,1);`** **`function myCompare(a,b) {return a-b;};`** **`myArray.sort(myCompare);  //myArray= 0,1,2,5,6`**
Reference	ECMA-262 Section15.4.4.11

6.4.3.10. splice()

Syntax:	**newArray = someArray.splice(startIndex,deleteCount [, item1 [, item2 [, …]]])**
Argument List:	startIndex – number(integer) deleteCount – number (integer) itemX – optional data elements of any type which replace deleted elements
Description:	Splice deletes the array elements inside an array, optionally replacing them with supplied items, if given, putting them into an assigned resultant array (newArray – the resulting array) Note: the starting array is untouched .
Return Value Type:	A new array object with appropriate elements removed, and optionally replaced by specified items
Errors or Exceptions:	none
Example(s):	**var stArray = new Array;** **(“hello”,1,”goodbye”,2,”why”,3,”not”,4);** **myArray=stArray.splice(2,2); //myArray = “hello”,1 ,”why”,3,”not”,4** **myArray=stArray.splice(6,2,”should”,5); //myArray=“hello”,1,”goodbye”,2,”why”,3,”should”,5** **myArray=stArray.splice(0,4); //myArray =,”why”,3,”not”,4**
Reference	ECMA-262 Section15.4.4.12

6.4.3.11. unshift()

Syntax:	**someArray.unshift([item$_1$ [,item$_2$ [, …]]]))**
Argument List:	item$_n$ = elements of any type
Description:	zero or more items are prepended to the start of an array such that their order within the array is the same as the argument list order. Note: No new array is created.
Return Value Type:	returns the count of items prepended to the target
Errors or Exceptions:	none
Example(s):	**var bytecodeArray = new Array("nop","lda 1","mult","dup");** **cnt = bytecodeArray.unshift("incr"); //cnt=1** **//bytecodeArray="incr","nop","lda 1","mult","dup"**
Reference	ECMA-262 Section15.4.4.13

6.5. String Object

Reference:	ECMA-262 Section 15.5

The **String** object in ECMAScript Mobile Profile provides the storage and methods for manipulating strings. All string functions that use indexes into strings assume that the indexes are to codepoints (characters), independent of the internal encoding used.

6.5.1. Version History

Version	Affected	Comment
1.1		ECMAScript initial version

6.5.2. Properties

6.5.2.1. length

Syntax:	**someString.length**
Type:	Number(integer)
Description:	– Value that is the character length of the target string

Errors or Exceptions:	the target must be a string, or else the string specific property will not be returned, and the result can be surprising
Example(s):	**“WAPForum”.length; //an 8 is returned** **“”.length // a zero is returned**
Reference	ECMA-262 Section 15.5.4

6.5.3. Methods

6.5.3.1. toString()

This is a description of the string specific instantiation of the toString method. Note: every native object has an instantiation of the toString method.

Syntax:	**strObj.toString()**
Argument List:	
Description:	returns the string value of the **String** object.
Return Value Type:	string value
Errors or Exceptions:	A **TypeError** exception is thrown if the target object cannot be converted to a string.
Example(s):	**var strObj = new String(“This is an Object!”);** **val=strObj.toString(); //val = “This is an Object!”; [but its not, its just a string value]**
Reference	ECMA-262 Section 15.5.4.2

6.5.3.2. valueOf()

This section describes the **String** object specific version of the **valueOf()** method. It should not be confused with other instantiations of this method.

Syntax:	**myStrObj.valueOf()**
Argument List:	
Description:	returns the string value of the **String** object.
Return Value Type:	string value
Errors or Exceptions:	the argument MUST be a **String** object for the method to return a string.
Example(s):	**var theStringObj = new String(“Yo Ho Ho”);** **var chant = theStringObj.valueOf();**

	`//chant="Yo Ho Ho"; (which is a simple string value)`
Reference	ECMA-262 Section 15.5.4.3

Note: In the case of strings the **`valueOf()`** method and the **`toString()`** method are identical.

6.5.3.3. charAt()

Syntax:	**`someString.charAt(pos)`**
Argument List:	pos – number (integer) which is the index into the target string.
Description:	returns a single character string pointed to by the position argument. Index is zero based.
Return Value Type:	a single character string value
Errors or Exceptions:	if pos < 0 or greater then the length of the target string then the empty string is returned
Example(s):	**`"Help me".charAt(3); // returns "p"`**
Reference	ECMA-262 Section 15.5.4.4

6.5.3.4. charCodeAt()

Syntax:	**`targetStr.charCodeAt(position)`**
Argument List:	position – number (integer)
Description:	returns the code point value of the character in the target string, pointed to by the index (position) argument. Index is zero based.
Return Value Type:	number (integer in the range 0 to 2^{16})
Errors or Exceptions:	If there is no Unicode character in the target, pointed to by the index, or pos< 0 then NaN is returned
Example(s):	**`val = "WAPâ".charCodeAt(3);  //returns the Unicode encoding value (2122)`**
Reference	ECMA-262 Section 15.5.4.5

6.5.3.5. concat()

Syntax:	**`targetStr.concat([str1 [,str2 [, …]]])`**
Argument List:	str1-strN – zero or more **`String`** objects or values
Description:	concatenates zero or mo re string values to the target string
Return Value Type:	string value
Errors or Exceptions:	none
Example(s):	**`var speech =“For score”.concat(“and ”, “seven years ”, “ago”);`** **`//speech=”For score and seven years ago”;`** **`var strObj = new String(“Three blind mice, ”);`** **`strObj.concat(“see how they run.”);`** **`//strObj = “Three blind mice, see how they run.” (an object)`**
Reference	ECMA-262 Section 15.5.4.6

Note: The string “+” operator is more often used for the same purpose.

6.5.3.6. indexOf()

Syntax:	**`targetStr.indexOf(searchString [,position] )`**
Argument List:	searchString – a **`String`** object or string value position – number (integer), optional
Description:	searches for the first substring match of the searchString in the targetStr, optionally starting at the position defined by the second argument. If the second argument is not specified (“undefined”), then the search starts from index 0 (zero).
Return Value Type:	-1 if there is no match number(integer) - index of the first match found. Index is zero based.
Errors or Exceptions:	none
Example(s):	**`var sstr=”big”;`** **`ret = “It’s Big, really big. Big,big,big”.indexOf(sstr);`** **`// ret = 17`**
Reference	ECMA-262 Section 15.5.4.7

6.5.3.7. lastIndexOf()

Syntax:	**`targetStr.lastIndexOf(searchString [,position] )`**
Argument List:	searchString – a **`String`** object or string value position – number (integer), optional
Description:	searches for the last substring match of the searchString in the targetStr, optionally starting at the position defined by the second argument, searching backwards toward index 0. If the second argument is not specified ("undefined"), then the search starts from the end of the string.
Return Value Type:	-1 if there is no match number(integer) - index of the first match found. Index is zero based.
Errors or Exceptions:	none
Example(s):	**`var sstr="big";`** **`ret = "It's Big, really big. Big,big,big".lastIndexOf(sstr);`** **`// ret = 30`**
Reference	ECMA-262 Section 15.5.4.8

6.5.3.8. localeCompare()

Syntax:	**`thisString.localeCompare(thatString)`**
Argument List:	thatString – a [converted] string or **`String`** Object
Description:	compares two strings, in a locale dependant way, to determine sort order. thisString comes before thatString ; return a negative value thisString is equivalent to thatString; return zero thisString comes after thatString; return a positive value
Return Value Type:	number (integer) negative, zero or positive to denote comparison 0 denotes canonical equivalence by the Unicode [UNICODE] standard positive (usually 1) denotes "thisString" comes after "thatString" in sort order negative (usually –1) denotes "thatString" comes after "thisString" in sort order
Errors or Exceptions:	none
Example(s):	**`"this".localeCompare("that"); //returns a postive value`**

Reference	ECMA-262 Section 15.5.4.9

6.5.3.9. match()

Syntax:	**myString.match(regExp)**
Argument List:	myString – target string value or **String** object regExp – a pattern or **RegExp** Object representing a valid regular expression
Description:	Find all matches in the target string, as defined by the regular expression, and return them in an array whose length is equal to the number(integer) of matches found.
Return Value Type:	Array with its length set to the number(integer) matches returned in the array
Errors or Exceptions:	Matches with the empty string (“”) are legal.
Example(s):	**var resArray = new Array;** **resArray = "Do da ditty do ".match(/\bd.\b/ig);** **//resArray[0] = “Do”** **//resArray[1] = “da”** **//resArray[2] = “do”** **//resArray.length = 3**
Reference	ECMA-262 Section 15.5.4.10

6.5.3.10. replace()

ECMAScript Mobile Profile does not support the use of a function as the second parameter, even though it is supported in [ECMA262].

<table>
<tr><td>Syntax:</td><td><code>targetString.replace(searchRegExp, replaceStr)</code>
<code>targetString.replace(searchStr,replaceStr)</code></td></tr>
<tr><td>Argument List:</td><td>searchRegExp – a regular expression object

searchStr – a [converted] string or <code>String</code> object

replaceStr – a string value with special character meanings as defined below</td></tr>
<tr><td>Description:</td><td>find the substring match(es) defined by the search expression or string and replace it with the string or expression defined by replaceStr. Certain $ (dollar) sequences have special meaning in the replacement string
<table>
<tr><td>Dollar Sequence</td><td>Replacement text</td></tr>
<tr><td>$$</td><td>Escape for dollar sign</td></tr>
<tr><td>$&</td><td>The matched substring</td></tr>
<tr><td>$`</td><td>The portion of the string that precedes the matched string.</td></tr>
<tr><td>$′</td><td>The portion of the string that follows the matched string.</td></tr>
<tr><td>$n</td><td>the nth parentheses captured string</td></tr>
<tr><td>$nn</td><td>the nnth parentheses captured string where nn is a two digit decimal number 01-99</td></tr>
</table></td></tr>
<tr><td>Return Value Type:</td><td>a String value</td></tr>
<tr><td>Errors or Exceptions:</td><td>none</td></tr>
<tr><td>Example(s):</td><td></td></tr>
<tr><td>Reference</td><td>ECMA-262 Section 15.5.4.11</td></tr>
</table>

6.5.3.11. search()

Syntax:	**aString.search(aRegExp)**
Argument List:	aRegExp – a regular expression object
Description:	searches for the first occurance of the string specified by the regular expression within the target string, and returns an index to that substring.
Return Value Type:	number (integer)
Errors or Exceptions:	-1 if no pattern match was found
Example(s):	**val = "$200.88 97% John Smith".search(/\b\d*%\b/); //val=8**
Reference	ECMA-262 Section 15.5.4.12

6.5.3.12. slice()

Syntax:	**targetString.slice(start,[end])**
Argument List:	start – number(integer) end – number(integer) or "undefined"
Description:	returns the substring defined by indexes 'start' and 'end'. If 'start' is negative it indexes back from the end of the target string. If 'end' is negative, it indexes from the beginning of the target string. If 'end' is not specified ("undefined") the index is through the end of the target string.
Return Value Type:	string value
Errors or Exceptions:	If 'start' is undefined, the result may be surprising.
Example(s):	**"slice-of-life".slice(6,8); //returns "of"**
Reference	ECMA-262 Section 15.5.4.13

6.5.3.13. split()

Syntax:	**aString.split(regExpSeparator [,limit])** **aString.split(strSeparator [, limit])**
Argument List:	regExpSeparator – a regular expression object, which is to be interpreted as a regular expression orstrSeparator – a string which is to be interpreted as a literal string. limit – optional number(integer)
Description:	returns an array into which are stored substrings from the result of splitting the target string. Substrings are determined by searching from left to right for occurances of the separator, as defined by a regular expression or string value. The number of array elements may be optionally limited by setting the 'limit' parameter.
Return Value Type:	Array object
Errors or Exceptions:	If 'limit' equals 0, an empty array is returned.
Example(s):	**resArray = new Array;** **resArray= "Hola".split(""); //resArray has one character at each array location** **resArray= "This,is weird".split(/[,]/);** **//resArray[0]="This" resArray[1]="is" resArray[2]="weird"**
Reference	ECMA-262 Section 15.5.4.14

6.5.3.14. substring()

Syntax:	**`targetStr.substring(start [,end])`**
Argument List:	start – number(integer) end – number(integer) or "undefined" if not specified
Description:	returns a string defined by the indexes start and end. If either start or end is negative or NaN, it is set to zero (0). If start > end then they are swapped. If end is "undefined" it is set to string.length
Return Value Type:	a string value
Errors or Exceptions:	none
Example(s):	**`"sadlkjfouiewr".substring(4,8); //returns "kjfo"`** **`"sadlkjfouiewr".substring(8,4); //returns "kjfo"`**
Reference	ECMA-262 Section 15.5.4.15

6.5.3.15. toLowerCase(), toLocaleLowerCase()

[ECMA262] defines two string methods **`toLowerCase()`** and **`toLocaleLowerCase().`** The method **`toLocaleLowerCase()`** is the same as **`toLowerCase()`** except that it is I18N aware. In ECMAScript Mobile Profile, both these methods are synonyms and are I18N aware (act as **`toLocaleLowerCase()`**).

Syntax:	**`myString.toLowerCase()`** **`yourString.toLocaleLowerCase()`**
Argument List:	MyString, yourString – a **`String`** object
Description:	Takes the **`String`** object, or any object that can evaluate to a string and converts the string to all lowercase characters as defined by the Unicode Character Database [UNICODE] for the host environment's current locale.
Return Value Type:	string value
Errors or Exceptions:	none Results may be language dependant
Example(s):	
Reference	ECMA-262 Section15.5.4.16, 15.5.4.17

6.5.3.16. toUpperCase(), toLocaleUpperCase()

[ECMA262] defines two string methods **`toUpperCase()`** and **`toLocaleUpperCase()`.** The method **`toLocaleUpperCase()`** is the same as **`toUpperCase()`** except that it is I18N aware. In ECMAScript Mobile Profile, both these methods are synonyms and are I18N aware (act as **`toLocaleUpperCase()`**).

Syntax:	**`someStringObject.toUpperCase();`** **`aStringObject.toLocaleUpperCase();`**
Argument List:	SomeStringObject, aStringObject – a **`String`** object
Description:	Takes the **`String`** object, or any object that can evaluate to a string and converts the string to all uppercase characters as defined by the Unicode Character Database [UNICODE] for the host environment's current locale.
Return Value Type:	String value
Errors or Exceptions:	none Results may be language dependant
Example(s):	**`myStr = "TempestB".toUpperCase(); //myStr = "TEMPESTB"`**
Reference	ECMA-262 Section15.5.4.18, 15.5.4.19

6.6. RegExp Object

Reference:	ECMA-262 Section 15.10

Regular expressions as defined in [ECMA262] are supported in ECMAScript Mobile Profile.

6.6.1. Version History

Version	Affected	Comment
1.1		ECMAScript initial version

6.6.2. Pattern Summary

Patterns are denoted as "/ pattern string /".

The following metacharacters can be used as elements in the pattern string:

Matching characters

Character	Meaning
\b	word boundary

\B	word non-boundary
\d	numeral (as defined by Unicode definition of current charset)
\D	any character except numeral (as defined by Unicode definition of current charset)
\s	a white space
\S	a non-white space
\w	letter, numeral or underscore (as defined by Unicode definition of current charset)
\W	any character that is not a letter, numeral or underscore (as defined by Unicode definition of current charset)
.	any character except a newline
^	Pattern is anchored to the beginning of a line.
$	Pattern is anchored to the end of a line.
\	Escapes the meaning of a subsequent special pattern character when it precedes one. May be used to escape the meaning of: . \ $ ^ * + ? ( ) [ ] { } |

Count and range modfiers

Character	Meaning
[…]	range or set delimiter
[^ …]	negation range or set
*	zero or more times modifier for previous character
?	zero or one time modifier for previous character
+	one or more times modifier for previous character
[n]	exactly ‘n’ times modfier for previous character
[n,]	‘n’ or more times modifier for previous character
[n,m]	at least ‘n’, but not more than ‘m’ times modifier for previous character

regular expression modifiers

Character	Meaning
g	global search modifier
i	ignorecase modifier
m	multiline modifier – not supported

6.6.3. Pattern Semantics

Reference: ECMA-263 Section 15.10.2

Pattern semantics are as defined in [ECMA262].

6.6.4. Properties

Each instance of a regexp object inherits the following properties from the regexp prototype:

6.6.4.1. source

Syntax:	**`aRegExpObj.source`**
Type:	String value, read-only
Description:	This is the original pattern string of the the regular expression object
Errors or Exceptions:	none
Example(s):	
Reference	ECMA-262 Section 15.10.7.1

6.6.4.2. global

Syntax:	**`aRegExpObj.global`**
Type:	Boolean, read-only
Description:	–Property that shows the state of the global modifier (“g”) of this object
Errors or Exceptions:	none
Example(s):	
Reference	ECMA-262 Section 15.10.7.2

6.6.4.3. ignoreCase

Syntax:	**`aRegExpObj.ignoreCase`**
Type:	Boolean, read-only
Description:	This is the state of the ignoreCase modifier (“i”) of this object
Errors or Exceptions:	none
Example(s):	
Reference	ECMA-262 Section 15.10.7.3

6.6.4.4. lastIndex

Syntax:	**`aRegExpObj.lastIndex`**
Type:	Number (integer)
Description:	–Property that specifies the index in the target string, at which to start the next match.
Errors or Exceptions:	none
Example(s):	
Reference	ECMA-262 Section 15.10.7.5

6.6.4.5. multiline

The property ‘multiline’ and associated modifier ‘m’ are not supported in ECMAScript Mobile Profile .

6.6.5. Methods

6.6.5.1. exec()

Syntax:	**`regExpObj.exec(targetString)`**
Argument List:	targetString – [converted] **`String`** object or string value against which the regular expression match is applied
Description:	Performs a regular expression match of “targetString” against the regular expression and returns an array object containing the results of the match.
Return Value Type:	Array Object
Errors or Exceptions:	“null” is returned if there is no match
Example(s):	**`var myRExp = /\b\d*\b/g;`** **`var myArray = new Array;`** **`myArray = myRExp.exec(“joe 100 fred 106 ralph 97 greg 102 “);`** **`//myArray[0]=100 myArray[1]=106 myArray[2]=97 myArray[3]=102`**
Reference	ECMA-262 Section 15.10.6.2

6.6.5.2. test()

Syntax:	**`aRegExpObj.test(someString)`**
Argument List:	someString – a [converted] string or **`String`** object
Description:	Tests whether a match exists or not. This is equivalent to **`RegExp.exec("someString") != null;`**
Return Value Type:	Boolean
Errors or Exceptions:	none
Example(s):	
Reference	ECMA-262 Section 15.10.6.3

6.6.5.3. toString()

This is the description of **`toString()`** method specific to the regular expression object.

Syntax:	**`myRegExpObj.toString()`**
Argument List:	myRegExpObj – a regular expression object
Description:	returns a string which is in the form of a regular expression pattern generated from the pattern which is part of the regular expression object. "/" chars are concatenated at the beginning and end of the pattern, and the appropriate modifiers ("g","i") are appended if true.
Return Value Type:	a string value
Errors or Exceptions:	none
Example(s):	
Reference	ECMA-262 Section 15.10.6.4

6.6.5.4. compile()

The **`compile()`** function is not supported.

6.7. Boolean Object

Reference:	ECMA-262 Section 15.6

6.7.1. Version History

Version	Affected	Comment
1.1		ECMAScript initial version

6.7.2. Methods

6.7.2.1. toString()

This is a description of the boolean object specific instantiation of the **`toString()`** method. Note: Every native object has an instantiation of the toString method.

Syntax:	**`aBool.toString()`**
Argument List:	ABool – an object of the type “Boolean”
Description:	converts the Boolean value or object to the string “true” or “false” based on the input value
Return Value Type:	string value
Errors or Exceptions:	none
Example(s):	**`var mySwitch = new Boolean(1);`** **`aStr = mySwitch.toString(); //aStr = “true”`**
Reference	ECMA-262 Section 15.6.4.2

6.7.2.2. valueOf()

This is a description of the boolean object specific instantiation of the **`valueOf()`** method.

Syntax:	**`myBool.valueOf()`**
Argument List:	myBool – an object of the type “Boolean”
Description:	coerces a Boolean object into a Boolean value
Return Value Type:	Boolean value
Errors or Exceptions:	none
Example(s):	
Reference	ECMA-262 Section 15.6.4.3

6.8. Number Object

Reference: ECMA-262 Section 15.7

The number object is a container for useful constants and functions that may need to be constructed.

6.8.1. Version History

Version	Affected	Comment
1.1		ECMAScript initial version

6.8.2. Constants

6.8.2.1. MAX_VALUE

Reference: ECMA-262 Section 15.7.3.2

64-bit Double Precision Floating Point	$\sim 2^{1024}$ (~1.797693134862316e+308)

6.8.2.2. MIN_VALUE

Reference: ECMA-262 Section 15.7.3.3

64-bit Double Precision Floating Point	$\sim 2^{-1024}$ (~5.562684646268e-309)

6.8.2.3. NaN

Reference: ECMA-262 Section 15.7.3.4

The value of aNumber.NaN is NaN. The assignment of a particular bit pattern representing NaN is implementation specific.

6.8.2.4. NEGATIVE_INFINITY

Reference: ECMA-262 Section 15.7.3.5

The value of aNumber.NEGATIVE_INFINITY is - . The assignment of a particular bit pattern representing - is implementation specific.

6.8.2.5. POSITIVE_INFINITY

Reference: ECMA-262 Section 15.7.3.6

The value of aNumber.POSITIVE_INIFINITY is + . The assignment of a particular bit pattern representing + is implementation specific.

6.8.3. Properties

No special properties are specified for the Number object.

6.8.4. Methods

6.8.4.1. toExponential()

Note: The fractionDigits value in [ECMA262] is defined as being between 0 and 20 inclusive. In this implementation, the language is only required to support a fractionDigits value between 0 and 13 inclusive.

Syntax:	**`aNumber.toExponential([fractionDigits])`**
Argument List:	fractionDigits – a number (integer) [range is implementation dependant]
Description:	converts a number to display in exponential format with 'fractionDigits' to the right of the decimal point. Missing digits are zero filled.
Return Value Type:	string value
Errors or Exceptions:	If the input number is NaN then "NaN" is returned Only 14 digits of output accuracy are guaranteed across implementations. If 'fractionDigits' < 0 or 'fractionDigits > supported implementation (at least 13) a RangeError exception is thrown.
Example(s):	
Reference	ECMA-262 Section 15.7.4.6

6.8.4.2. toFixed()

Note: The fractionDigits value in [ECMA262] is defined as being between 0 and 20 inclusive. In this implementation, the language is only required to support a fractionDigits value between 0 and 13 inclusive.

Syntax:	**`aNumber.toFixed([fractionDigits])`**
Argument List:	fractionDigits – a number (integer) [range is implementation dependant]
Description:	converts a number to display in fixed format placing 'fractionDigits' digits after the decimal point. Missing digits to the right of the decimal point are zero filled. Rounding accuracy is not guaranteed beyond 14 decimal digits total. If **`fractionDigits`** is not specified, 0 (zero) is assumed
Return Value Type:	string value
Errors or Exceptions:	If the input number is NaN then "NaN" is returned A **`fractionDigits`** value outside of the supported range (0-13 as a minimum) will generate a **`RangeError`** exception. Only 14 digits of output string accuracy are guaranteed across implementations.
Example(s):	**`var aNum = 6789912;`** **`dispNum=aNum.toFixed(2); //dispNum=6789912.00`** **`var bNum = 67899.12;`** **`dispNum=bNum.toFixed(2); //dispNum=67899.10 (possible loss of 7th digit of precision)`** **`var cNum=987.44`** **`dispNum=cNum.toFixed(10); //dispNum=987.4400000000`**
Reference	ECMA-262 Section 15.7.4.5

6.8.4.3. toLocaleString()

Reference: ECMA-262 Section 15.7.4.3

Implementations are encouraged to implement I18N locale specific conversions for numbers. In the absence of a true locale specific implementation, **`toFixed(2)`** and **`toLocaleString()`** shall be considered synonyms.

6.8.4.4. toString()

This is a description of the number object specific instantiation of the **`toString()`** method. Note: Every native object has an instantiation of the **`toString()`** method.

Syntax:	**`aNumberObj.toString([radix])`**
Argument List:	radix – optional radix for conversion. If radix is "undefined" it is assumed to be 10.
Description:	Convert a number into a printable string.
Return Value Type:	string value
Errors or Exceptions:	If number is NaN then **`NaN`** is returned. Only 14 digits of decimal accuracy can be guaranteed.
Example(s):	
Reference	ECMA-262 Section 15.7.4.2

6.8.4.5. valueOf()

Syntax:	**`aNumberObj.valueOf()`**
Argument List:	ANumberObj – a number object
Description:	returns a number value from a number object
Return Value Type:	number value
Errors or Exceptions:	none
Example(s):	
Reference	ECMA-262 Section 15.7.4.4

6.8.4.6. toPrecision()

Note: The precision value in [ECMA262] is defined as being between 1 and 21 inclusive. In this implementation, the language is only required to support a precision value between 1 and 14 inclusive.

Syntax:	**`aNumberObj.toPrecision(precision)`**
Argument List:	precision – a number (integer) [range is implementation dependant]
Description:	returns a string containing the number represented either in exponential notation with one digit before the significand's decimal point and 'precision' total digits, or in fixed notation with 'precision' total digits. The dis played value is rounded with respect to undisplayed digits.
Return Value Type:	String value
Errors or Exceptions:	if "precision" is "undefined" then the function simply calls the toString method for numbers. A precision value outside of the supported range (1-14 as a minimum) will generate a **`RangeError`** exception. Only 14 digits of accuracy are guaranteed across implementations.
Example(s):	**`var aNum= 123.4567`** **`dispNum = aNum.toPrecision(2); //dispNum=1.2E2`** **`dispNum = aNum.toPrecision(3); //dispNum=123`** **`dispNum = aNum.toPrecision(4); //dispNum=123.5  Note rounding`** **`dispNum = aNum.toPrecision(5); //dispNum=123.46`** **`dispNum = aNum.toPrecision(6); //dispNum=123.457`** **`dispNum = aNum.toPrecision(7); //dispNum=123.456x where the values of x could vary from implementation to implementation`**
Reference	ECMA-262 Section 15.7.4.7

6.9. Math Object

Reference: ECMA-262 Section 15.8

The Math object is a single object, owned by the global object, which is the container for some useful constants and mathematical functions. This object cannot be instantiated.

6.9.1. Version History

Version	Affected	Comment
1.1		ECMAScript initial version

6.9.2. Properties and Constants

Useful constants are contained in the Math object as properties. All constants are guaranteed to support at least 14 total digits of accuracy. The following constants are supported:

Constant	Value
Math.E	2.7182818284590…
Math.LN2	0.6931471805599…
Math.LN10	2.3025850929940…
Math.LOG10E	0.4342944819035…
Math.LOG2E	1.4426950408890…
Math.PI	3.1415926535897…
Math.SQRT1_2	0.7071067811865…
Math.SQRT2	1.4142135623731…

6.9.3. Methods

6.9.3.1. Integerizing Methods

Any of the following methods, if applied to numbers that are already integers, simply returns the number itself.

Method	Description	Examples	Reference
`Math.round(x)`	Returns the number value that is closest to the given value and is an integer	`a= Math.round(3.5); //a=4` `b= Math.round(-3.5);//b= -3`	ECMA-262 Section 15.8.2.15
`Math.ceil(x)`	Returns the smallest number value that is not less than x and is an integer.	`a= Math.ceil (3.4); //a=4` `b= Math.ceil(-3.4);//b= -3`	ECMA-262 Section 15.8.2.6
`Math.floor(x)`	Returns the greatest number value that is not greater than x and is an integer.	`a= Math.floor(3.4); //a=3` `b= Math.floor(-3.4);//b= -4`	ECMA-262 Section 15.8.2.9

6.9.3.2. General Mathematical Methods

Method	Description	Reference
`Math.abs(x)`	Returns the absolute value of 'x'.	ECMA-262 Section 15.8.2.2
`Math.exp(x)`	Returns an implementation dependent approximation of the exponential function of 'x' ('e' raised to the power of 'x', where 'e' is the base of the natural logarithms).	ECMA-262 Section 15.8.2.8
`Math.log(x)`	Returns an implementation dependent approximation of the natural logarithm of 'x'	ECMA-262 Section 15.8.2.10
`Math.pow(x,y)`	Returns an implementation dependent approximation of the result of raising 'x' to the 'y' power.	ECMA-262 Section 15.8.2.13
`Math.sqrt(x)`	Returns an implementation dependent approximation of the square root of 'x'	ECMA-262 Section 15.8.2.17

6.9.3.3. Trigonometric Methods

Method	Description	Reference
`Math.acos(x)`	Generates the arc-cosine of number x Range: +0 to +ð	ECMA-262 Section 15.8.2.2
`Math.asin(x)`	Generates the arc-sine of number x Range:- ð/2 to + ð/2	ECMA-262 Section 15.8.2.3
`Math.atan(x)`	Generates the arc-tangent of numberx Range:- ð/2 to + ð/2	ECMA-262 Section 15.8.2.4
`Math.atan2(y,x)`	Generates the arc-tangent of the quotient y,x Range:- ð to + ð	ECMA-262 Section 15.8.2.5
`Math.cos(x)`	Generates the cosine, in radians, of number x	ECMA-262 Section 15.8.2.7
`Math.sin(x)`	Generates the sine, in radians, of number x	ECMA-262 Section 15.8.2.16
`Math.tan(x)`	Generates the tangent, in radians, of number x	ECMA-262 Section 15.8.2.18

6.9.3.4. max()

Reference: ECMA-262 Section 15.8.2.11

Syntax:	**`Math.max([value1[,value2 [,…]]])`**
Argument List:	value1 – any value, which is then converted with **`toNumber()`** value2 – any value, which is then converted with **`toNumber()`**
Description:	return the largest of the resulting values
Return Value Type:	number
Errors or Exceptions:	If any of the values is NaN the result is NaN
Example(s):	**`Math.max("3.12E12",36); //returns the number 3.12E12`**
Reference	ECMA-262 Section 15.8.2.11

WMLScript Note: ECMAScript allows zero or more arguments, returning the maximum. WMLScript specified exactly two arguments. [WMLSCRIPT]

6.9.3.5. min()

Reference: ECMA-262 Section 15.8.2.12

Syntax:	**`Math.min([value1 [,value2 [,…]]])`**
Argument List:	value1 – any value, which is then converted with **`toNumber()`** value2 – any value, which is then converted with **`toNumber()`**
Description:	returns the smallest of the resulting values
Return Value Type:	number
Errors or Exceptions:	If any of the values is NaN the result is NaN
Example(s):	**`Math.min("3.12E12",36); //returns the number 36`**
Reference	ECMA-262 Section 15.8.2.11

WMLScript Note: ECMAScript allows zero or more arguments, returning the minimu m. WMLScript specified exactly two arguments. [WMLSCRIPT]

6.9.3.6. random()

Syntax:	`Math.random([intMaxVal])`
Argument List:	intMaxVal – number(integer), optional
Description:	returns a random (or pseudorandom) number. If the input parameter is "undefined" (not specified) then the function returns a floating point value between 0 and 1. If the input parameter is specified, then the method returns an integer value >= 0 and < to the input value.
Return Value Type:	a Number
Errors or Exceptions:	if no input parameter is specified, and the implementation supports only integers then "NaN" is returned. It is an error to specify a negative argument.
Example(s):	`aVal = Math.random(); //returns .789032 (example floating val 0<aVal<1.0)` `bVal= Math.random(100); //returns 89 (integer val 0<=bVal<100)`
Reference	ECMA-262 Section 15.8.2.14

Note: This function is an enhancement to the the ECMAScript 3rd Edition method, to support integers as well as floating point values. Effort should be made to use a random number algorithm that generates truly randomly distributed numbers.

6.10.Date Object

Reference:	ECMA-262 Section 15.9

The date object provides support for date and time handling within ECMAScript Mobile Profile.

6.10.1. Version History

Version	Affected	Comment
1.1		ECMAScript initial version

6.10.2. Time Range

Reference:	ECMA-262 Section 15.9.1.1

Time range is as specified in [ECMA262], approximately ±285,616 years from January 1st 1970 (the epoch).

6.10.3. Day Number and Time within Day

A given time value 't' belongs to day number

Day('t') = floor('t'/secPerDay)

where the number of milliseconds per day is

1000 * 60 * 60 * 24 = 86400000

The remainder is called the time within the day:

TimeWithinDay('t') = 't' modulo secPerDay

Reference: ECMA-262 Sections 15.9.1

Implementations are encouraged to support the persistent storage of local timezone, and to provide a mechanism for the user to access the timezone as well as other date and time settings. If there is no support for timezone in a particular implementation, then the UTC version of a method MUST return the same value as the local version of the method.

Actual implementations may not be accurate to the millisecond, and may be rounded.

6.10.4. Properties

No special properties are specified for the Date object.

6.10.5. Methods

6.10.5.1. getTime()

Syntax:	**`aDateObj.getTime()`**
Argument List:	
Description:	returns the current time into the object.
Return Value Type:	sets the time of the target object
Errors or Exceptions:	if the target is not a Date object a **`TypeError`** exception is generated.
Example(s):	**`var now= new Date;`** **`now.getTime(); // now holds the current time; (right now)`**
Reference	ECMA-262 Section 15.9.5.9

6.10.5.2. getFullYear(), getUTCFullYear()

Syntax:	**`aDateObj.getFullYear()`** – local timezone based **`aDateObj.getUTCFullYear()`** – UTC timezone based
Argument List:	
Description:	returns the year contained in the date object
Return Value Type:	a number(integer) in the range 1970 ±~238,000
Errors or Exceptions:	if the date is NaN then “NaN” is returned if the target is not a Date object a **`TypeError`** exception is generated.
Example(s):	**`now = new Date;`** **`document.write("Current year is " + now.getFullYear());`**
Reference	ECMA-262 Section 15.9.5.10, 15.9.5.11

6.10.5.3. getMonth(), getUTCMonth()

Syntax:	**`aDateObj.getMonth()`** – local timezone based **`aDateObj.getUTCMonth()`** – UTC timezone based
Argument List:	
Description:	returns the month contained in the date object
Return Value Type:	a number(integer) in the range [0-11]
Errors or Exceptions:	if the date is NaN then “NaN” is returned. if the target is not a Date object a **`TypeError`** exception is generated.
Example(s):	**`now = new Date;`** **`document.write("Current month is " + now.getMonth());`**
Reference	ECMA-262 Section 15.9.5.12, 15.9.5.13

6.10.5.4. getDate(), getUTCDate()

Syntax:	**`aDateObj.getDate()`** – local timezone based **`aDateObj.getUTCDate()`** – UTC timezone based
Argument List:	
Description:	returns the date of the month contained in the date object
Return Value Type:	a number(integer) in the range [1-31]
Errors or Exceptions:	if the date is NaN then “NaN” is returned. if the target is not a Date object a **`TypeError`** exception is generated.
Example(s):	**`now = new Date;`** **`document.write("Current date is " + now.getUTCDate());`**
Reference	ECMA-262 Section 15.9.5.14, 15.9.5.15

6.10.5.5. getDay(), getUTCDay()

Syntax:	**`aDateObj.getDay()`** – local timezone based **`aDateObj.getUTCDay()`** – UTC timezone based
Argument List:	
Description:	returns the day of the week contained in the date object
Return Value Type:	a number(integer) in the range [0-6] where 0 is Sunday and 6 is Saturday
Errors or Exceptions:	if the date is NaN then “NaN” is returned. if the target is not a Date object a **`TypeError`** exception is generated.
Example(s):	**`now = new Date;`** **`document.write("Current day is " + now.getUTCday());`**
Reference	ECMA-262 Section 15.9.5.16, 15.9.5.17

6.10.5.6. getHours(), getUTCHours()

Syntax:	**`aDateObj.getHours()`** – local timezone based **`aDateObj.getUTCHours()`** – UTC timezone based
Argument List:	
Description:	returns the hour of the day contained in the date object
Return Value Type:	a number(integer) in the range [0-23]
Errors or Exceptions:	if the date is NaN then “NaN” is returned. if the target is not a Date object a **`TypeError`** exception is generated.
Example(s):	**`now = new Date;`** **`document.write("Current hour is " + now.getHours());`**
Reference	ECMA-262 Section 15.9.5.18, 15.9.5.19

6.10.5.7. getMinutes(), getUTCMinutes()

Syntax:	**`aDateObj.getMinutes()`** – local timezone based **`aDateObj.getUTCMinutes()`** – UTC timezone based
Argument List:	
Description:	returns the minute within the hour contained in the date object
Return Value Type:	a number(integer) in the range [0-59]
Errors or Exceptions:	if the date is NaN then “NaN” is returned. if the target is not a Date object a **`TypeError`** exception is generated.
Example(s):	**`now = new Date;`** **`document.write("Current minute is " + now.getUTCMinutes());`**
Reference	ECMA-262 Section 15.9.5.20, 15.9.5.21

6.10.5.8. getSeconds(), getUTCSeconds()

Syntax:	**`aDateObj.getSeconds()`** – local timezone based **`aDateObj.getUTCSeconds()`** – UTC timezone based
Argument List:	
Description:	returns the second within the minute contained in the date object
Return Value Type:	a number(integer) in the range [0-59]
Errors or Exceptions:	if the date is NaN then “NaN” is returned. if the target is not a Date object a **`TypeError`** exception is generated.
Example(s):	**`now = new Date;`** **`document.write("Current second is " + now.getUTCSeconds());`**
Reference	ECMA-262 Section 15.9.5.22, 15.9.5.23

6.10.5.9. getMilliseconds(), getUTCMillieconds()

Syntax:	**`aDateObj.getMilliseconds()`** – local timezone based **`aDateObj.getUTCMilliseconds()`** – UTC timezone based
Argument List:	
Description:	returns the millisecond within the second contained in the date object
Return Value Type:	a number(integer) in the range [0-999]
Errors or Exceptions:	if the date is NaN then “NaN” is returned. if the target is not a Date object a **`TypeError`** exception is generated. If milliseconds are not supported for a given device then 0 (zero) is returned
Example(s):	**`now = new Date;`** **`document.write("Current millisecond is " + now.getUTCMilliseconds());`**
Reference	ECMA-262 Section 15.9.5.24, 15.9.5.25

6.10.5.10. getTimezoneOffset()

Syntax:	**`aDateObj.getTimezoneOffset()`**
Argument List:	

Description:	returns the value in minutes which is the difference between the current timezone and UTC
Return Value Type:	a number(integer) in the range [0-1410]
Errors or Exceptions:	If timezone is not supported on a particular device this method MUST return zero (0). if the target is not a Date object a **`TypeError`** exception is generated.
Example(s):	`var someDate = new Date;` `val = someDate.getTimezoneOffset();` `//In the Eastern U.S, at certain times, val would = 300`
Reference	ECMA-262 Section 15.9.5.26

6.10.5.11. parse()

The **`parse()`** method is not supported for the **`Date`** object, because textual dates are not specified in a standard way, and the cost of parsing freeform date syntax is deemed too expensive for small devices.

6.10.5.12. UTC()

Syntax:	**Date.UTC(year, month [, date [, hours [, minutes [, seconds [, ms]]]]])**
Argument List:	year - number – a full year value month – number (0-11) date – number (1-31) hours – number (0-23) minutes – number (0-59) seconds – number (0-59) ms – number (0-999)
Description:	Returns a number(integer) which corresponds to the date described in milliseconds from the epoch
Return Value Type:	number
Errors or Exceptions:	If an argument is not within the valid ranges given above, the result is implementation-dependent.
Example(s):	**`aVal = Date.UTC(1954,3,4,0,0,0,0);`** **`// aVal = -496886400000`**
Reference	ECMA-262 Section 15.9.4.3

6.10.5.13. setTime()

Syntax:	**`aDateObj.setTime(time)`**
Argument List:	time – number(signed integer)
Description:	assigns the input value ‘time’ as the time for the target date object
Return Value Type:	integer ‘time’
Errors or Exceptions:	if the target is not a Date object a **`TypeError`** exception is generated.
Example(s):	**`var newTime = new Date;`** **`newTime.setTime(990807301);  //newTime = May 25th 12:15:01 PM 2001`**
Reference	ECMA-262 Section 15.9.5.27

6.10.5.14. setFullYear(), setUTCFullYear()

Syntax:	**aDateObj.setFullYear(year [,month [, date]])** **aDateObj.setUTCFullYear(year [,month [, date]])**
Argument List:	year – number(integer) month – number(integer) date – number(integer)
Description:	sets the year (and optionally the month and date) of the target date object
Return Value Type:	returns the value of the newly calculated date
Errors or Exceptions:	if any of the input numbers is out of range of a signed 64 bit integer, or the resultant date cannot be expressed within a signed 64 bit integer, the date is set to NaN
Example(s):	**aDateObj.setFullYear(2003); // would change the year but no other part of the date**
Reference	ECMA-262 Section 15.9.5.40, 15.9.5.41

6.10.5.15. setMonth(), setUTCMonth()

Syntax:	**aDateObj.setMonth(month [, date])** **aDateObj.setUTCMonth(month [, date])**
Argument List:	month – number(integer) date – number(integer)
Description:	sets the month (and optionally the date) of the target date object
Return Value Type:	returns the value of the newly calculated date
Errors or Exceptions:	if any of the input numbers is out of range of a signed 64 bit integer, or the resultant date cannot be expressed within a signed 64 bit integer, the date is set to NaN
Example(s):	**aDateObj.setMonth(-22); // would set the date back to the 2nd month (modulo 12)**
Reference	ECMA-262 Section 15.9.5.38, 15.9.5.39

6.10.5.16. setDate(), setUTCDate()

Syntax:	**aDateObj.setDate(date)** **aDateObj.setUTCDate(date)**
Argument List:	date – number(integer)
Description:	sets the date of the target date object
Return Value Type:	returns the value of the newly calculated date
Errors or Exceptions:	if any of the input numbers is out of range of a signed 64 bit integer, or the resultant date cannot be expressed within a signed 64 bit integer, the date is set to NaN
Example(s):	**aDateObj.setDate(5); // would set the date to be the 5th of the month**
Reference	ECMA-262 Section 15.9.5.36, 15.9.5.37

6.10.5.17. setHours(), setUTCHours()

Syntax:	**aDateObj.setHours(hour [,min [,sec]])** **aDateObj.setUTCHours(hour [,min [,sec]])**
Argument List:	hour – number(integer) min – number(integer) sec – number(integer)
Description:	sets the hour (and optionally the minutes and seconds) of the target date object
Return Value Type:	returns the value of the newly calculated date
Errors or Exceptions:	if any of the input numbers is out of range of a signed 64 bit integer, or the resultant date cannot be expressed within a signed 64 bit integer, the date is set to NaN
Example(s):	**aDateObj.setHours(24); // would set the hour to be 0 and increment the day count**
Reference	ECMA-262 Section 15.9.5.34, 15.9.5.35

6.10.5.18. setMinutes(), setUTCMinutes()

Syntax:	**`aDateObj.setMinutes(min [,sec])`** **`aDateObj.setUTCMinutes(min [,sec])`**
Argument List:	aDateObj – a Date Object min – number(integer) sec – number(integer)
Description:	sets the minutes (and optionally seconds) of the target date object
Return Value Type:	returns the value of the newly calculated date
Errors or Exceptions:	if any of the input numbers is out of range of a signed 64 bit integer, or the resultant date cannot be expressed within a signed 64 bit integer, the date is set to NaN
Example(s):	**`aDateObj.setMinutes(-1); // would set the minutes to 59 and decrement the hour`**
Reference	ECMA-262 Section 15.9.5.32, 15.9.5.33

6.10.5.19. setSeconds(), setUTCSeconds()

Syntax:	**`aDateObj.setSeconds(sec)`** **`aDateObj.setUTCMinutes(sec)`**
Argument List:	sec – number(integer)
Description:	sets the seconds of the target **`Date`** object
Return Value Type:	returns the value of the newly calculated date
Errors or Exceptions:	if any of the input numbers is out of range of a signed 64 bit integer, or the resultant date cannot be expressed within a signed 64 bit integer, the date is set to NaN
Example(s):	**`aDateObj.setSeconds (3602); // would set the seconds to 2 and the hours to 1`**
Reference	ECMA-262 Section 15.9.5.30, 15.9.5.31

6.10.5.20. setMilliseconds(), setUTCMilliseconds

Syntax:	**aDateObj.setMilliseconds(msec)** **aDateObj.setUTCMilliseconds(msec)**
Argument List:	msec – number(integer)
Description:	sets the milliseconds of the target **Date** object.
Return Value Type:	returns the value of the newly calculated date
Errors or Exceptions:	if any of the input numbers is out of range of a signed 64 bit integer, or the resultant date cannot be expressed within a signed 64 bit integer, the date is set to NaN
Example(s):	**aDateObj.setMilliseconds (10002); // would set the seconds to 1 and the milliseconds to 2**
Reference	ECMA-262 Section 15.9.5.28, 15.9.5.29

6.10.5.21. toString(), toLocaleString(), toUTCString()

This **toString()** method is specific for **Date** objects.

Syntax:	**aDateObj.toString()** **aDateObj.toLocaleString()** **aDateObj.toUTCString()**
Argument List:	
Description:	creates an implementation dependant string, that is a human readable full date
Return Value Type:	a string value
Errors or Exceptions:	if the **Date** object is NaN then “NaN” is returned
Example(s):	**document.write("Date String = " + now.toString());** **//Date String = Wed May 30 00:36:34 EDT 2001** **document.write("Date String is " + now.toUTCString());** **//Date String is Wed, 30 May 2001 04:36:34 UTC**
Reference	ECMA-262 Section 15.9.5.2, 15.9.5.42

Note: toLocaleString may return a specially localized date string, otherwise it is a synonym for toString when applied to date. The exact format of the human readable string is implementation dependant, and cannot be used for standard string comparisons.

6.10.5.22. toDateString(), toLocaleDateString()

Syntax:	**aDateObj.toDateString()** **aDateObj.toLocaleDateString()**
Argument List:	
Description:	creates an implementation dependant string, that is the human readable date part of the target date object. **toLocaleDateString()** optionally expresses the date in a manner that represents an I18N localized version. If **toLocaleDateString()** is not explicitly implemented, it should be a synonym for **toDateString().**
Return Value Type:	a string value
Errors or Exceptions:	if the date object is NaN then “NaN” is returned
Example(s):	**document.write("Date String = " + now.toDateString());** **// Date String = Wed May 30 2001** **document.write("Date String is " + now.toLocaleDateString());** **//Date String is Wednesday,May 30,2001**
Reference	ECMA-262 Section 15.9.5.3, 15.9.5.6 Note: This is not compatible with IE or Netscape.

6.10.5.23. toTimeString(), toLocaleTimeString()

Syntax:	**aDateObj.toTimeString()** **aDateObj.toLocaleTimeString()**
Argument List:	
Description:	creates an implementation dependant string, that is the human readable time part of the target date object. **toLocaleTimeString** optionally expresses the time in a manner that represents an I18N localized version. If **toLocaleTimeString()** is not explicitly implemented, it should be a synonym for **toTimeString()**
Return Value Type:	a string value
Errors or Exceptions:	if the date object is NaN then “NaN” is returned
Example(s):	**document.write("Time String = " + now.toTimeString());** **//Time String = 00:57:31 EDT** **document.write("Date String is " + now.toLocaleTimeString());** **//Time String is 12:57:31 AM**
Reference	ECMA-262 Section 15.9.5.4, 15.9.5.7

6.10.5.24. valueOf()

Syntax:	**`aDateObj.valueOf()`**
Argument List:	
Description:	returns a number(signed integer) which represents the date contained in the date object.
Return Value Type:	number(signed integer)
Errors or Exceptions:	none
Example(s):	
Reference	ECMA-262 Section 15.9.5.8

6.11. Error (Exception) Object

Reference: ECMA 262 Section 15.11

`Error` Objects can be used to capture and report errors that are either part of the native implementation (native errors) or are generated explicitly by the application.

6.11.1. Version History

Version	Affected	Comment
1.1		ECMAScript initial version

6.11.2. Constructor

Syntax:	**`myError = new Error([aMessage]);`**
Argument List:	aMessage – the optional message string that is to be associated with the message property of this **`Error`** object
Description:	Constructs an instance of an **`Error`** object, and optionally sets the message property with the supplied parameter.
Return Value Type:	returns an **`Error`** object instance
Errors or Exceptions:	none
Example(s):	**`//setting up a user defined error`** **`killerError = new Error("You should not have done that!");`** **`killerError.name = "KillerError";`** **`// force this error`** **`try {`** **`throw killerError;`** **`}`** **`catch(killerError) {`** **`document.write(killerError.message);`** **`}`** **`finally {`** **`alert("Done with this thing.");`** **`}`**
Reference	ECMA-262 Section 15.11.1

6.11.3. Properties

6.11.3.1. name

Syntax:	**myError.name**
Type:	string
Description:	A property containing a one word error title (i.e. “TypeError”). The initial value of a newly constructed **Error** object name is “Error”.
Errors or Exceptions:	none
Example(s):	
Reference	This is an extension to ECMA262 Section 15.11.7

6.11.3.2. message

Syntax:	**myError.message**
Type:	string
Description:	A property containing an implementation dependant description of the error captured in the **Error** object.
Errors or Exceptions:	none
Example(s):	**varmyError = new Error;** **document.write(“Error message is “ + myError.message);**
Reference	ECMA262 Section 15.11.7.2

6.11.3.3. code

Syntax:	**`myError.code`**
Type:	number(integer)
Description:	A property that contains the value of the exception thrown. The initial value of a newly constructed **`Error`** object code is NaN.
Errors or Exceptions:	none
Example(s):	
Reference	This is an extension to ECMA262 Section 15.11.7

Note: Assignment of fixed error code constants is an extension to ECMA262.

6.11.4. Native Error Types (Constants)

Reference: ECMA-262 Section 15.11.7

The following native error types MUST be supported:

Name	Code Constant	Description	Reference
`RangeError`	101	A numeric value has exceeded its range.	ECMA-262 Section 15.11.6.2
`ReferenceError`	102	An invalid reference has been detected.	ECMA-262 Section 15.11.6.3
`SyntaxError`	103	A parsing error has occurred.	ECMA-262 Section 15.11.6.4
`TypeError`	104	The actual type of an operand is different than the expected type.	ECMA-262 Section 15.11.6.5
`URIError`	105	A global URI handling function was used in an inconsistent or incompatible way.	ECMA-262 Section 15.11.6.6
`EvalError`	106	The global **`eval()`** function was used in a way that is incompatible with its definition, or attempts were made to use **`eval()`** when it isn't present. This error value is required whether or not an implementation	ECMA-262 Section 15.11.6.1 ECMA-327 Section 5.1

		supports **eval()**.	
MemoryError	99	The method was unable to obtain the memory resources required to complete the request.	

This means that the implementation MUST be able to throw instantiations of any of the above **Error** objects, as defined by their name. To catch and process any of the native errors a try / catch {} clause must be used. The catch parameter will be set to reference the instantiation of the native error. The instantiated **Error** object MUST set the 'name' and the 'code' properties to the proper value.

```
function myException()
{
    myErr = new Error;
    var anInt;
    var aVal = 4.3;
    document.write("Entering routine.");
    try {

        anInt = aVal.Garbage(); //will generate an exception
        document.write("Shouldn't get here");
    }
    catch(myErr)
    {
        document.write("<br/>An error was thrown. <br/>");
        document.write("<br/> myErrName=" + myErr.name + "<br/>message=" +
myErr.message);
    }  //myErr.name = "TypeError"
}
```

6.12.Unsupported Native Objects

Because of the separation between compilation and execution support for the dynamic creation of new object and function prototypes is not required for language conformance. [ECMA327]

6.12.1. Object object

Reference: ECMA-262 Section 15.2

Dynamic creation of new Object objects SHOULD NOT be supported in ECMAScript – Mobile Profile.

Creation of dynamic objects is a large memory burden upon a client.

6.12.2. Function object

Reference: ECMA-262 Section 15.3

Because support for the **`eval()`** function (runtime compilation) is not required, the **`Function`** object is not required to support a constructor supporting dynamic creation of runtime functions. [ECMA327]. See Section 5.4.2.

The Function constructor SHOULD NOT support the ability to construct dynamic functions requiring runtime compilation.

When **`eval()`** is not supported, attempts to compile a dynamically created function MUST throw an **`EvalError`**.

7. The Language Environment

7.1. Reference Programming Model

The reference programming model described here is concerned only with the entry to and exit from script. When inside script the reference programming model is as described in [ECMA262].

7.1.1. Script Context

Script context is inherited at the time of script invocation. It is an aggregation of the current browser context and the document context (including, but not limited to, current location, form variable state, event state, history stack, etc.) from which the script was called. Event state is defined as the set of events (0 or more) and their associated attribute values, which are pending at the time the script is invoked.

7.1.2. Generic Browser Context

The generic browser context is composed of the history, location and event state associated with the currently executing document [XHTMLMP]. These data are accessible from within script through the **`history`** and **`location`** host objects (See sections 9.3 and 9.4). **`history`** and **`location`** objects are instantiated at the time of script invocation, and go out of scope upon script exit.

7.1.3. Document Context

Document context is as defined by [DOM2CORE] and is accessed via the Browser **`Document`** Object (See Section 9.5). Document context is instantiated at the time a markup document is entered. Document-level state (such as form variable values) remain in scope until the document is exited. Hence when invoking script multiple times from within a single markup document, document form variables remain in scope.

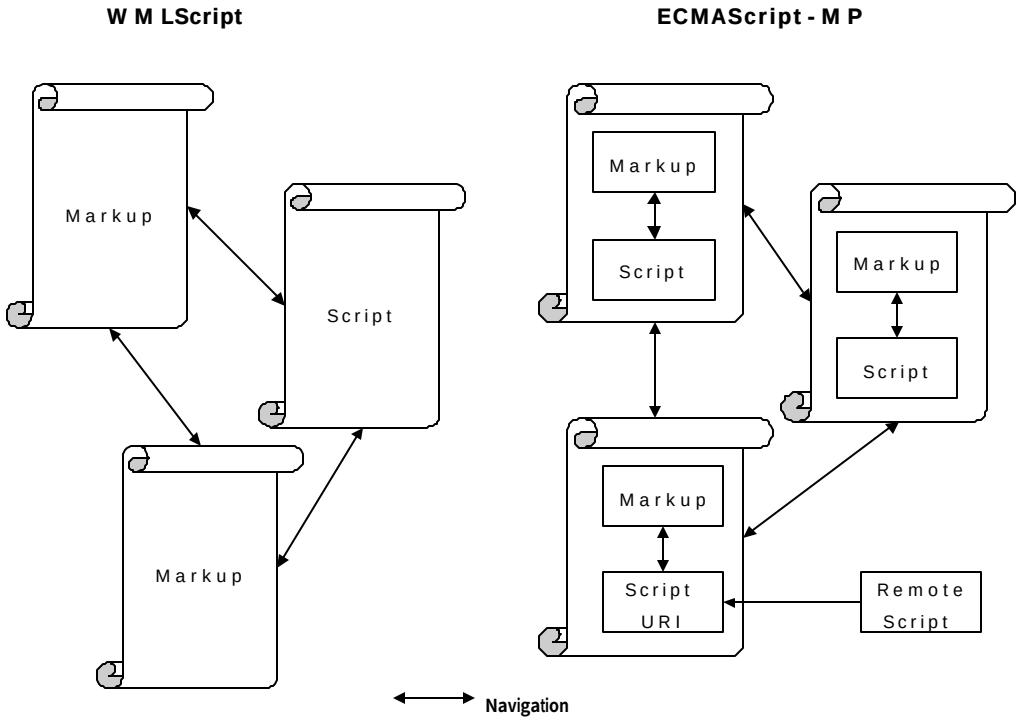
7.2. Script Invocation Mechanisms

7.2.1. Invocation via Navigation

All script invocations in WMLScript [WMLSCRIPT] are via navigation to a script document. This method is not supported in ECMAScript Mobile Profile, where scripts are contained within an XHTML Mobile Profile document and script invocation occurs during the loading of the document or as a result of events.

7.2.2. **<script>** Element Definition

When invoking script directly from a markup document, the script is considered to be part of the invoking document. This is the markup/script relationship that exists between Javascript and markup in the “wired web”. This method of invocation is supported by ECMAScript Mobile Profile. This method of invocation is also referred to as child invocation. The context of the invoking markup document is available to the script. Script may be defined in three ways as defined in sections 7.2.2.1,7.2.2.2, 7.2.2.3.



7.2.2.1. Inline Execution

Script segments may be defined directly inline as embedded script text. This occurs when the inline script is not enclosed in a function body.

```
<script type="text/ecmascript">
          document.write("I will be invoked.<br/>");
</script>
```

Multiple direct embedded script segments MUST be supported. Scripts defined in this way are said to be directly invoked.

Directly invoked script segments MUST be executed in the order in which they appear.

Directly invoked script segments MUST be executed prior to any event invoked script segments.

7.2.2.2. Event-based (Deferred) Execution

The most common way to define a script is as a response to an event. (See section 8). Scripts written to be executed as a result of an event are said to have event-based, or deferred invocation. Event-based invocation requires the use of event assignment attributes in XHTML [XHTMLMP] markup.

```
<html>
 …
          <script type="text/ecmascript">
                    someFunc() {
```

```
                        //do something
                    }
            </script>
 …
 <body>
 …
            <form onsubmit="someFunc()">
            …
            </form>
 …
 </body>
</html>
```

Deferred execution MUST be supported.

The script engine MUST allow deferred execution scripts to be located in either the **<body>** or the **<head>** of a document.

7.2.2.3. File-based Execution

Script may be defined by referencing a URL that contains the source for the script to execute.

Script files defined by reference SHOULD contain only script statements and SHOULD be text only.

Failure to meet these criteria may lead to rejection of the file by the implementation. Remotely referenced script files are processed by replacing the contents of the script element with the contents of the file.

The script engine MUST allow script elements representing references to remote scripts to be located in either the <**body**> or the <**head**> of a document.

After loading, remote scripts are subject to the invocation rules described in 7.2.2.1 and 7.2.2.2.

```
            <script type="text/ecmascript" src="aFileofESMP.es"/>
```

Remotely referenced script files MUST be supported.

The **type** attribute on the **<script>** element SHOULD be supported.

7.2.2.4. Scheme-based execution

Scheme-based invocation, (using a scheme syntax such as **href="ecmascript:<immediate statements>"**) which is a syntactic shortcut, is not supported.

7.3. Script Completion Mechanisms

7.3.1. Normal Completion

Unlike WMLScript, ECMAScript completion follows a very simple model. Irrespective of the state in which a script execution completes, control is returned to the context of the parent document. This is true for abnormal and successful

completions as well as completions following explicit navigation to a new document via the **`history`** or **`location`** objects (See section 9.3 and 9.4).This means that successive calls to a script segment may be initiated by the parent context if so directed. The script invocation mechanism is considered to be stateless. All script completions return to the parent document as if the execution was successful. It is implementation dependent what, if any errors, from abnormal termination are reported to the user.

The user agent MAY report to the user, errors from the abnormal termination of scripts.

In the case of completion following an explicit navigation to a new document, the navigation request is to be considered asynchronous and cancellable like any other document navigation. Should the script request a navigation, complete and return to the parent document context and the user the immediately selects another navigation object or invokes the navigation cancel command, the script's navigation request will be cancelled. In the case of the user selecting the cancel command, the user is of course free to re-invoke the script.

7.3.2. Aborted Completion

The script engine MUST provide an abort event to allow the initiation of abnormal script execution.

There MUST be a user input mechanism, such as a key or button, available to provide access to the abort functionality to the end user.

Completion of the abort processing MUST guarantee that the browser is capable of continuing, or returning to a known state.

8. Events

Events are the primary method of invoking scripts. The event language bindings, management and flow mechanisms specified in ECMAScript Mobile Profile are based on the attributes defined in section 5.14 (Intrinsic Events Module) of [XHTMLMOD] plus the Events attributes defined in section 5.1 (Attribute Collections) of [XHTMLMOD]. The semantics for the event types are defined in [XHTMLMP]

In ESMP event bindings are only supported through the markup interface, although they should be considered compatible with the procedural interface as defined in DOM2 events [DOM2EVENTS].

8.1. XHTML Event Types

XHTML Mobile Profile 1.1 [XHTMLMP] defines events and a mechanism for binding event handlers to those events.

The XHTML – Mobile Profile user agent MUST support XHTML Events as a prerequisite for script support.

Note: These events are also referred to "DOM0" events in [DOM2EVENTS] chapter 1.6.5. For a list of XHTML Event attributes, their targets, and semantics see [XHTMLMP].

To guarantee interoperability between legacy events and any future DOM2 event semantics, implementers SHOULD guarantee that legacy events operate as a special case of the broader DOM2 event model.

The user agent MUST ignore event handlers registered for event types that are not supported.

8.2. Event Binding

An event handler is bound to an event by assigning the value of an event handler attribute. The event handler attribute specifies the type of the event and its value specifies the handler. The target element is the element to which the event handler attribute is attached.

The user agent MUST operate as if the value of the event handler attribute is the body of an anonymous function that is registered as an event handler.

The prototype of the function contains a single argument, an **`Event`** object (See Section 8.3)

```
function <anonymous>(Event event)
{
 ...
}
```

The binding is equivalent to the W3C DOM Level 2 Events registration method **`addEventListener()`** invoked on the element to which the attribute is attached, with **`useCapture`** specified as **`false`**.

Any attempt to modify the value of an event handler attribute via the DOM interfaces MUST result in a deregistration of any existing event handler. Only a single event handler is supported for a given event type on a given target element.

8.3. Event Object

When the user agent handles an event, it makes available to the event listener an **`Event`** object. The **`Event`** object defined in ECMAScript Mobile Profile is a small intersecting subset of the W3C DOM Level 2 Events **`Event`** object and the Netscape Navigator **`Event`** object. The static **`Event`** object, as defined in the IE implementation, is not available in this implementation.

8.3.1. Event Properties

All events are considered to be "non-bubbling", since the full W3C DOM event binding interface is not supported.

All events are considered to "non-capturing", since the full W3C DOM event binding interface is not supported.

Because all events are considered non-bubbling and non-capturing, events fired without a specified handler are ignored.

8.3.2. Properties

8.3.2.1. keyCode

Syntax:	**`anEvent.keyCode`**
Type:	number (integer), read-only
Description:	For keyboard events only (onkeyup, onkeydown, onkeypress) this property is an integer corresponding to the following: For KeyPress – the Unicode [UNICODE] value of the character pressed by the user For KeyDown, KeyUp – the Unicode [UNICODE] value of the keyboard key pressed
Errors or Exceptions:	Keys which do not have a Unicode equivalent mapping are ignored. Note: keyCode semantics are not an exact match for either IE or Netscape. Netscape uses two properties **`keyCode`** and **`charCode`** to represent the same thing. IE uses a different property name.
Example(s):	
Reference	

8.3.2.2. target

Syntax:	**`anEvent.target`**
Type:	**`Node`** reference, read-only
Description:	This property is a reference to the original markup element that is the target of the event. It can be used to reference nodes (See section 10.2) and attributes relative to the event originator.
Errors or Exceptions:	none
Example(s):	
Reference	DOM2EVENTS Section 1.4

.

8.3.2.3. timeStamp

Syntax:	**`anEvent.timeStamp`**
Type:	number(integer), read-only
Description:	A property which represents the time the event was created. The format of the timestamp is defined by DOM2 Events. It is in a form that is compatible with the **`Date`** object (See section 6.10)
Errors or Exceptions:	none
Example(s):	
Reference	DOM2EVENTS Section 1.4

8.3.2.4. type

Syntax:	**`anEvent.type`**
Type:	string, read-only
Description:	A property that describes the kind of event that generated this object. Strings are lower case and are the same as XHTML event attribute, with the "on" removed. Valid strings are:

Source XHTML Event Attribute	Type String
onclick	click
ondblclick	dblclick
onload	load
onunload	unload
onreset	reset
onsubmit	submit
onkeypress	keypress
onkeydown	keydown
onkeyup	keyup
onblur	blur
onfocus	focus
onchange	change
onselect	select
onwap-<evt>-event	wap-<evt>-event

Errors or Exceptions:	Unrecognized event types will set **`type`** to **`null`**.
Example(s):	
Reference	DOM2EVENTS Section 1.4

8.4. Reference Processing Model

The event model for EcmaScript Mobile Profile is the same model as that of W3C DOM Level 2 Events.

When an event supported by a user agent occurs, the user agent MUST create a read-only instance of an **Event** object, as specified in section 8.3. It must then invoke the event handler associated with that event, passing the **Event** object instance to the handler as described in section 8.2.

The user agent MUST ignore any event bindings for event types that are not supported.

Note: The event binding mechanism specified for ECMAScript Mobile Profile does not permit an event handler (event listener) to specify whether it gets invoked in the capture or bubbling phase. Because of this, all events are considered "non-bubbling" and "non-capturing". All event handlers are registered on the event's target element and are triggered when the event reaches the target.

8.5. Sample Code

This subsection is informative. For the definition of event binding in markup see [XHTMLMP].

The following is an example of handling simple events, using a single script function to support multiple event triggers:

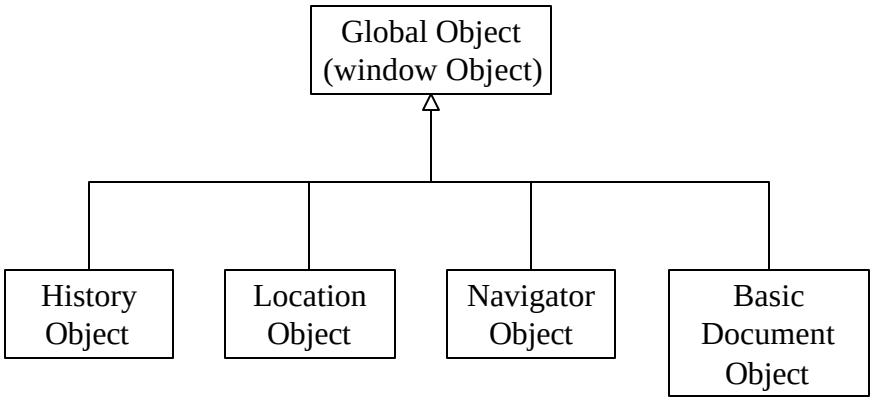
```
<html>
 <head>
  <script type="text/ecmascript">
  function doEvtThing(evt){
    document.write("In handler<br/>");
    if (evt.type == "click")
      document.write("Click received<br/>");
    if (evt.type == "submit")
      document.write("Submit received<br/>");
    if (evt.type == "keydown")
      document.write("Key pressed =" + evt.keyCode +"<br/>");
    return;
  }
  </script>
 </head>
 <body>
  <h1>Event Tester</h1>
  <form>
   <input type="button" value="OnClickTest" onclick="doEvtThing(event)"/>
  </form>
```

```
  <form method="post" onsubmit="doEvtThing(event)" onreset="doEvtThing()">
    Type a word:
    <input type="text" name="text" value="" onkeydown="doEvtThing(event)"/>
    <input type="submit" name="Submit" value="Submit"/>
    <input type="reset" />
  </form>
 </body>
</html>
```

Note: In some cases the event parameter is specified in markup, and where the event instance is not explicitly used, it is not required.

9. Browser Host Objects

In addition to the native objects that encapsulate language semantics, a set of host objects are defined to interface to the host environment in which the scripting language resides. This section describes these objects. Section 10 describes the DOM host objects.



9.1. Global Object (window object)

Note: The version history for this object can be found in section 6.3.1. The **`global`** object may be referred to as "top", "parent", "self" or "window" for backwards compatibility with the wired web. (See section 6.3)

9.1.1. Properties

The **`history`** object, **`location`** object, **`navigator`** object and **`document`** object are contained in the **`global`** object and are hence properties of the **`global`** object. There are no pre-defined read-only properties directly attributable to the **`global`** object.

9.1.1.1. history

Syntax:	**`[top.]history`**
Type:	object, read-only, enumerable
Description:	A property which is the **`history`** object described in detail in section 9.3.
Errors or Exceptions:	none
Example(s):	
Reference	Common object IE, Netscape

9.1.1.2. navigator

Syntax:	**`[top.]navigator`**
Type:	object, read-only, enumerable
Description:	A property which is the **`navigator`** object described in detail in section 9.2.
Errors or Exceptions:	none
Example(s):	
Reference	Common object IE, Netscape

9.1.1.3. location

Syntax:	**`[top.]location`**
Type:	object, read-only, enumerable
Description:	A property which is the **`location`** object described in detail in section 9.4.
Errors or Exceptions:	none
Example(s):	
Reference	Common object IE, Netscape

9.1.1.4. document

Syntax:	**[top.]document**
Type:	object, enumerable
Description:	A property which is the **document** object described in detail in section 9.5 and section 10
Errors or Exceptions:	none
Example(s):	
Reference	Common object IE, Netscape

9.1.2. Methods

9.1.2.1. prompt()

Syntax:	**prompt(aMessage [,defaultReply]);**
Argument List:	aMessage – string to be used as a prompt defaultReply – string that will be returned if there is no input. If not specified, it is set to "undefined".
Description:	Displays the given message as a dialog and prompts for user input. If the user does not enter a reply the defaultReply is returned as the user input string. If the user replies, the user reply is returned. This method is modal, and blocks waiting for a response.
Return Value Type:	String or "undefined"
Errors or Exceptions:	none
Example(s):	**prompt("What is your sex?","M");**
Reference	Common function IE, Netscape

9.1.2.2. confirm()

Syntax:	**`confirm(aMessage [,OKReply [,CancelReply]])`**
Argument List:	aMessage – string OKReply – optional string for labeling positive reply CancelReply – optional string for labeling negative reply
Description:	Displays message and waits for user to select positive or negative selection choice. This method is modal, and blocks waiting for a response.
Return Value Type:	**`true`** is returned upon selection of positive selection choice. **`false`** is returned upon selection of negative selection choice.
Errors or Exceptions:	none
Example(s):	**`confirm("Do you want to exit?","YES","NO");`**
Reference	Common function IE, Netscape

9.1.2.3. alert()

Syntax:	**`alert(aMessage);`**
Argument List:	aMessage – a string to be displayed
Description:	Displays the given message to the user, and waits for user confirmation of the message. This method is modal, and blocks waiting for a response.
Return Value Type:	none
Errors or Exceptions:	none
Example(s):	**`if (confirm("Time to panic?", "YES","NO"))`** **`{ alert("PANIC"); }`** **`else`** **`{ alert("No problem."); }`**
Reference	Common function IE, Netscape

9.1.2.4. setTimeout()

Syntax:	**`setTimeout(funcRef[()],milliSecs [,funcArg`**$_1$**`, …, funcArg`**$_N$**`]);`**
Argument List:	funcRef – the name of a script function ie. myTimerFunc milliSecs – number, that represents duration of the timeout as specified in milliseconds funcArg$_X$ – optional arguments which are passed to the function referenced in parameter 1.
Description:	This global function specifies the duration to wait before calling the requested function. It acts as if a delay function were inserted at the end of the currently executing function. Execution of the current script does not stop. Rather, the requested function is called as soon as possible after the duration requested has passed. The function reference MUST be specified as <funcname>, without quotes. This is so that the function reference is not interpreted as a string or expression. Inclusion of the parentheses after the function reference is optional (and allowed for compatibility with the wired web). Evaluation of the first parameter as an expression (such as a quoted string) may throw an **`EvalError`** if **`eval()`** is not supported (See section 5.4.2). An implementation MUST support a minimum of a single timeout. The user agent is not rquired to implement a timer with resolution of 1 millisecond. Authors should note that a user agent may not honor the exact value of a timeout expressed in milliseconds.
Return Value Type:	returns an opaque handle (a number) representing the timer. This ID can be used with the **`clearTimeout()`** method to stop a timer. Note: A return of zero denotes an error.

Errors or Exceptions:	If a request for a timeout is made, when there are no more timeout slots available, the function returns 0. If the number of function arguments do not match the number of parameters specified by the requested function, unspecified arguments are set to "undefined", extra arguments are ignored. If a function reference is specified that cannot be found a **`ReferenceError`** exception is thrown.
Example(s):	`function loopMe(sflg) {` `var tID;` `var index;` `var specialflg = "false";` `if(sflg){` `// do something special` `}` `if(index < 10) {` `index++;` `if (index == 6) specialflg = "true";` `//do something` `tID = setTimeout(loopMe, 5*1000, specialflg); //wait 5 secs` `// wait for awhile` `}` `else {` `clearTimeout(tID);` `}` `}`
Reference	Common function IE, Netscape Note: This method of passing parameters (while avoiding an **`eval())`** is taken from Netscape and may not correctly pass parameters in IE. To assure compatibility with all web implementations, do not use parameters.

Note: This function is a change from the Javascript standard, in that only the function name (without parameters) may be used as the function input. This is because the use of parameters could require the invocation of an **`eval()`** operation, which is not a mandatory requirement of ESMP.

9.1.2.5. clearTimeout()

Syntax:	`clearTimeout(timeoutID);`
Argument List:	timeoutID – a number representing an opaque handle to a timeout. timeoutID is obtained as the return of the **`setTimeout()`** method.
Description:	unsets and clears an existing timeout
Return Value	none

Type:	
Errors or Exceptions:	none
Example(s):	**`See above`**
Reference	Common function IE, Netscape

9.2. Navigator Object

The **`navigator`** object is a read-only object that contains a set of properties that help to identify the browser and user agent that reside in a particular device.

9.2.1. Version History

Version	Affected	Comment
1.1		ECMAScript initial version

9.2.2. Properties

9.2.2.1. appName

Syntax:	**`navigator.appName`**
Type:	string, read-only
Description:	A property that contains a descriptor of the browser application and device. Manufacturers are free to insert any string here, but the following guidelines should be followed: A string identifying the browser application manufacturer SHOULD appear first. A string identifying the browser application name SHOULD appear next. Optionally, any qualifiers SHOULD be inside parentheses.
Errors or Exceptions:	none
Example(s):	**`document.write("Appname = " + navigator.appName + " ");`** **`//displays  Appname = MobileCompany Ubrowser (test version)`**
Reference	Common IE and Netscape object property

9.2.2.2. appVersion

Syntax:	**navigator.appVersion**
Type:	string, read-only
Description:	A property that contains a version number for the browser application and device. Manufacturers are free to insert any string here, but the following guidelines should be followed: A string identifying the manufacturers application version number SHOULD appear first. Optionally, any qualifiers SHOULD be inside parentheses.
Errors or Exceptions:	none
Example(s):	**document.write("App Version = " + navigator.appVersion + " ");** **//displays App Version = 3.4 (Build 6 Dynamo 080902)**
Reference	Common IE and Netscape object property

9.2.2.3. userAgent

Syntax:	**navigator.userAgent**
Type:	string, read-only
Description:	A property that contains a string that uniquely describes the running user agent. Manufacturers are free to insert any string here, but the following guidelines should be followed: A string identifying the user agent name SHOULD appear first. Optionally, any qualifiers SHOULD be inside parentheses.
Errors or Exceptions:	none
Example(s):	**document.write("App User Agent = " + navigator.appuserAgent + " ");** **//displays App User Agent = Mozilla/5.0 (Windows; U; Windows NT 5.0; en-US; rv:0.9.4) Gecko/20011128 Netscape6/6.2.1**
Reference	Common IE and Netscape object property

9.3.

The **history** object maintains a list of the URLs most recently visited by the browser. These URLs are available to scripts for controlling navigation.

Direct access to the list of URL strings is a security concern and MUST NOT be supported.

9.3.1. Version History

Version	Affected	Comment
1.1		ECMAScript initial version

9.3.2. Properties

9.3.2.1. length

Syntax:	**history.length**
Type:	number(integer) , read-only
Description:	The number of history entries currently in the history list is 'length'.
Errors or Exceptions:	none
Example(s):	
Reference	Common IE and Netscape object property

9.3.3. Methods

9.3.3.1. back()

Syntax:	**history.back()**
Argument List:	
Description:	This method executes a browser "navigate to history entry" to the previous entry in the history list (if there is one).
Return Value Type:	none

Errors or Exceptions:	If there are no entries in the history list, then the method does nothing. Note: Actual navigation to the requested URL is not guaranteed until script exit. Use of this method, as with all methods that cause URL navigation, should be tied to script exit.
Example(s):	`history.back(); // reloads the previous URL on the history list`
Reference	Common IE and Netscape object property

9.3.3.2. forward()

Syntax:	`history.forward()`
Argument List:	
Description:	This method executes a browser "navigate to history entry" action with next URL entry in the history list.
Return Value Type:	none
Errors or Exceptions:	If there are no subsequent entries in the history list, then the method does nothing. Note: Actual navigation to the requested URL is not guaranteed until script exit. Use of this method, as with all methods that cause URL navigation, should be tied to script exit.
Example(s):	`history.forward(); // reloads the next URL on the history list`
Reference	Common IE and Netscape object property

9.3.3.3. go()

Syntax:	**history.go(index)**
Argument List:	index – a number(integer) which is the index to an ordered list of history entries. 0 (zero) is the index to the current entry.
Description:	Requests the browser to navigate to a URL using the indexed entry in the history list. Form variable state SHOULD be preserved when loading a document requested by the **history.go()** method. Preservation of variable state is dependent upon cache availability. Note: A **history.go(0)** is NOT the same as a **location.reload()**. **history.go(0)** is more "gentle", preserving the state of all form variables whenever possible. **history.go(X)** loads the 'X' entry in the history list and navigates there if available. There is no notion of navigation to any intervening entries on the history list. Negative numbers represent previous entries in the history list, positive numbers represent subsequent entries (i.e. when one has already navigated backwards) Note: Actual navigation to the requested URL is not guaranteed until script exit. Use of this method, as with all methods that cause URL navigation, should be tied to script exit.
Return Value Type:	none
Errors or Exceptions:	If there are no entries at the requested index in the history list, then the method does nothing.
Example(s):	**history.go(0); // reloads the current URL on the history list, preserving all form variable and context state – beware a loop!** **history.go(-1);//same as history.back()** **history.go(1);//same as history.forward()**
Reference	Common IE and Netscape object protperty

9.4. Location Object

The **location** object allows a script to query and manipulate the URL of the current document. When the **location** object is assigned to (used as a left-hand value), the script engine silently applies the **location.assign(URL)** (See section 9.4.3.1) method to the current context. URL is the complete constructed resource locator of the **location** object.

9.4.1. Version History

Version	Affected	Comment
1.1		ECMAScript initial version

9.4.2. Properties

9.4.2.1. hash

Syntax:	**`location.hash`**
Type:	string
Description:	A property that is the fragment (anchor) part of the current URL (including the # [hash]) if it exists. Setting the **`hash`** property (using location.hash as an lvalue) navigates to the named anchor without reloading the document. The hash symbol may be included, or not, in an lvalue assignment. Note: Actual navigation to the requested URL is not guaranteed until script exit. Use of property assignment, as with all methods that cause URL navigation, should be tied to script exit.
Errors or Exceptions:	An HTTP error may be generated if attempts to navigate to the anchor specified in an assignment cannot be found. Attempts to construct an illegal URI will throw a **`URIError`**. This is an extension to ECMA262.
Example(s):	**`location.hash = "#nextanchor"; //will navigate to the nextanchor tag in the document`**
Reference	Common IE and Netscape object property, RFC2396

9.4.2.2. host

Syntax:	**location.host**
Type:	string
Description:	The **host** property specifies a portion of the current URL. The **host** property is the concatenation of the **hostname** (See section 9.4.2.4) and **port** (see section 9.4.2.6) properties, separated by a colon. When the **port** property is null, the **host** property is the same as the **hostname** property. Assignment to the **location.host** property may be used to initiate navigation. Note: Actual navigation to the requested URL is not guaranteed until script exit. Use of property assignment, as with all methods that cause URL navigation, should be tied to script exit.
Errors or Exceptions:	Attempts to navigate to non-existent URLs will generate standard HTTP errors. Attempts to construct an illegal URI will throw a **URIError**. This is an extension to ECMA262.
Example(s):	**document.write("Host = " + location.host + " ");** **//returns Host = www.mysite.com:8080**
Reference	Common IE and Netscape object property, RFC2396

9.4.2.3. href

Syntax:	**location.href**
Type:	string
Description:	The **href** specifies the entire URL currently pointed to by the **location** object. Assignment to the **location.href** property may be used to initiate navigation. Note: Actual navigation to the requested URL is not guaranteed until script exit. Use of property assignment, as with all methods that cause URL navigation, should be tied to script exit.
Errors or Exceptions:	Attempts to navigate to non-existent URLs will generate standard HTTP errors. Attempts to construct an illegal URI will throw a **URIError**. This is an extension to ECMA262.
Example(s):	**Location.href = "http://www.wapforum.org"; //will navigate to the WAP Forum website**
Reference	Common IE and Netscape object property, RFC2396

9.4.2.4. hostname

Syntax:	**location.hostname**
Type:	string
Description:	A property that contains a string, which is the hostname part of the current URL. Assignment to the **location.hostname** property may be used to initiate navigation. Note: Actual navigation to the requested URL is not guaranteed until script exit. Use of property

	assignment, as with all methods that cause URL navigation, should be tied to script exit.
Errors or Exceptions:	Attempts to navigate to non-existent URLs will generate standard HTTP errors. Attempts to construct an illegal URI will throw a **`URIError`**. This is an extension to ECMA262
Example(s):	
Reference	Common IE and Netscape object property, RFC2396

9.4.2.5. pathname

Syntax:	**`location.pathname`**
Type:	string
Description:	A property which is the pathname component of the current URL. This includes the document name, but not the server name. Assignment to the **`location.pathname`** property may be used to initiate navigation. Note: Actual navigation to the requested URL is not guaranteed until script exit. Use of property assignment, as with all methods that cause URL navigation, should be tied to script exit.
Errors or Exceptions:	Attempts to navigate to non-existent URLs will generate standard HTTP errors. Attempts to construct an illegal URI will throw a **`URIError`**. This is an extension to ECMA262.
Example(s):	
Reference	Common IE and Netscape object property, RFC2396

9.4.2.6. port

Syntax:	**`location.port`**
Type:	string
Description:	A property which is the specified port number of the current URL. The colon that separates the hostname from the port is not included. . Assignment to the **`location.port`** property may be used to initiate navigation. Note: Actual navigation to the requested URL is not guaranteed until script exit. Use of property assignment, as with all methods that cause URL navigation, should be tied to script exit.
Errors or Exceptions:	Attempts to navigate to non-existent URLs will generate standard HTTP errors. Attempts to construct an illegal URI will throw a **`URIError`**. This is an extension to ECMA262.
Example(s):	
Reference	Common IE and Netscape object property, RFC2396

9.4.2.7. protocol

Syntax:	**`location.protocol`**
Type:	string, read-only
Description:	A property that contains a string, which is the first component of a URL, and specifies the protocol used for retrieving the document. Typically this would be “http:”, or “file:”, or “ftp:” etc. The colon is returned as part of the string.
Errors or Exceptions:	none
Example(s):	
Reference	Common IE and Netscape object property, RFC2396

9.4.2.8. search

Syntax:	**`location.search`**
Type:	string
Description:	A property which is the search component of the current URL. The search component is defined as that substring of the URL that comes after the first question mark URI delimiter. The string returned includes the question mark. Assignment to the **`location.search`** property may be used to initiate navigation. Note: Actual navigation to the requested URL is not guaranteed until script exit. Use of property assignment, as with all methods that cause URL navigation, should be tied to script exit.
Errors or Exceptions:	Attempts to navigate to URLs that cannot process the appended search string may generate standard HTTP errors. Attempts to construct an illegal URI will throw a **`URIError`**. This is an extension to ECMA262.
Example(s):	
Reference	Common IE and Netscape object property, RFC2396

9.4.3. Methods

9.4.3.1. assign()

Syntax: **`location.assign(URL)`**

Argument List: URL – a string

Description: Loads the document pointed to by the parameter **`URL`** into the current context. The history stack is pushed with the requested URL.

Note: Actual navigation to the requested URL is not guaranteed until script exit. Use of this method, as with all methods that cause URL navigation, should be tied to script exit.

Return Value Type:	none
Errors or Exceptions:	Attempts to navigate to non-existent URLs will generate standard HTTP errors. Attempts to use an illegally formed URI will throw a **`URIError`**. This is an extension to ECMA262.
Example(s):	`//location.href = http://somewhere/somedoc` `location.assign("http://somewhere/somedoc");` `//loads the document somedoc`
Reference	Common IE and Netscape object property, RFC2396

9.4.3.2. reload()

Syntax:	`location.reload(fromServer)`
Argument List:	fromServer – Boolean, which if **`true`** forces a reload of the current URL from the origin server, bypassing the client cache. If 'fromServer' is **`false`**, reload will attempt to use local client cache before making a network request.
Description:	This method reloads the current document. Normally the current document is reloaded from cache when available. When the 'fromServer' Boolean is set to **`true`**, cache is ignored, and the document is always reloaded from the server. A new presentation document is generated. Form variable data MUST be reinitialised when navigating to a document using this method. Note: This causes all form variable data to be re-initialised (as opposed to history.go(0) which maintains current variable state). Note: Actual navigation to the requested URL is not guaranteed until script exit. Use of this method, as with all methods that cause URL navigation, should be tied to script exit.
Return Value Type:	none
Errors or Exceptions:	none
Example(s):	`//location.href = http://somewhere/somedoc` `location.reload(true);  //forces a reload from the origin` `//server. Form variables are reinitialised.`
Reference	Common IE and Netscape object property, RFC2396

9.4.3.3. replace()

Syntax: **`location.replace(URLString)`**

Argument List: URLString – a string

Description: This method loads the document pointed to by the specified URL, while at the same time replacing the current entry in the history list with the specified URL. The over-written URL history reference is no longer available for backwards navigation.

Note: Actual navigation to the requested URL is not guaranteed until script exit. Use of this method, as with all methods that cause URL navigation, should be tied to script exit.

Return Value Type: none

Errors or Exceptions: none

Attempts to use an illegally formed URI will throw a **`URIError`**. This is an extension to ECMA262.

Example(s):

```
location.replace("http://www.wapforum.org");
//would navigate to and display the WAPF home page while
//replacing the top of the history stack with the current URL,
//rather then stacking a new location onto the stack
```

Reference Common IE and Netscape object property, RFC2396

9.5. Basic Document Object

9.5.1. Version History

Version	Affected	Comment
1.1		ECMAScript initial version

9.5.2. Properties

9.5.2.1. cookie

Syntax:	**`document.cookie`**
Type:	string
Description:	The **`cookie`** property provides access to the cookies associated with the current document domain, as specified by [RFC2109]. Operations performed with the 'cookie' property SHOULD use the client's cookie store. A cookie is read by reading the value of the **`cookie`** property, then parsing it for the particular cookie. When read, the **`cookie`** property value MUST be a semi-colon-separated list of the cookies for the domain of the document containing the script. For each cookie list, the format is NAME"="VALUE[";"$NAME_2$"="$VALUE_2$[";"$NAME_N$"="$VALUE_N$]] where NAME and VALUE are as defined in [RFC2109]. Cookie attributes are not provided. A cookie is written by setting the value of the property equal to a string representing the name-value pair. The string must match the "cookie" production rule from [RFC2109]. If the string does not match, the cookie must be ignored. Only single name/value pairs, along with acceptable attributes may be written. Cookies may have attributes associated with them. The legal attribute values are defined by the "cookie" production rule. Any other values result in the value string not matching the production rule; such a cookie must be ignored. Legal attributes that MUST be supported are:

Attribute	Description
`Domain`	Defaults to the request-host. Note that there is no dot (“.”) at the beginning of request-host.
`Max-Age`	Defines the lifetime of the cookie in seconds. A value of zero means the cookie SHOULD be discarded immediately. The default behavior is to discard the cookie when the user agent exits.
`Path`	Defaults to the path of the request URL that generated the Set-Cookie response, up to, but not including, the right-most slash (“/”).
`Secure`	If absent, the user agent may send the cookie over an insecure channel. The definition of whether a channel is secure or not is left to the implementation, but MAY include use of HTTPS, WTLS or other end-to-end encryption schemes.

Note: The **`Expires`** attribute, which is a “Netscape Cookie” feature, is not supported, as it requires date synchronization between client and server.

Attributes are included in a cookie by appending them to the name/value cookie pair, separated by semicolons.

The rules for rejecting cookies from [RFC2109 Section 4.3.2] apply for cookies created by setting the value of the 'cookie' property. For example, it MUST NOT be possible to set a cookie for another domain.

Note: This property is only able to access cookies which are local to the terminal on which the script interpreter resides. Cookie storage may be delegated to network proxies (as defined in [HTTPSM]). Cookies stored in network proxies are not visible to the **`document`** object.

Errors or Exceptions: Creation of cookies using property assignment is subject to all domain security and syntax checks defined in RFC2109 and will not create a new cookie entry if these rules are broken.

Example(s):

```
function getCookie(NameOfCookie){
var begin;
var end;
  if (document.cookie.length > 0) {
  begin = document.cookie.indexOf(NameOfCookie+"=");
  if (begin != -1) {
    begin += NameOfCookie.length+1;
    end = document.cookie.indexOf(";", begin);
    if (end == -1) end = document.cookie.length;
      return decodeURIComponent(document.cookie.substring(begin,
end));
    }
  }
  return null;
}

function setCookie(NameOfCookie, value, expirehours) {
  var seclength = expirehours * 3600;
  document.cookie = NameOfCookie + "=" + encodeURIComponent(value)
  + ((expirehours == null) ? "" : "; Max-Age=" + seclength);
}

function delCookie (NameOfCookie) {
  if (getCookie(NameOfCookie)) {
    document.cookie = NameOfCookie + "=" +"; Max-Age=0";
  }
}
```

Reference: Common IE and Netscape object property, RFC2109

9.5.2.2. domain

Syntax:	**`document.domain`**
Type:	string
Description:	A property that contains the current document domain. Writing to this property (using it as a left side assignment) will attempt to change the document domain, according to specified security rules in RFC2396. A script may not change the origin domain of a document. Two domains are considered to have the same origin if the protocol, port (if given), and host (as defined in RFC2396) are the same for both pages.
Errors or Exceptions:	Attempts to change the origin domain will generate an exception.
Example(s):	Example: If documents are needed from both www.wapforum.org and www1.wapforum.org, we may broaden the domain from the initial "www.wapforum.org" to "wapforum.org". Changing the major domain root is illegal. Example: `//if current connected domain = www1.wapforum.org` `document.domain="www.wapforum.org";  //legal` `document.domain="wapforum.org";  //legal` `document.domain="ebay.com"; //illegal-will generate an exception`
Reference	Common IE and Netscape object property, RFC2396

9.5.2.3. referrer

Syntax:	**`document.referrer`**
Type:	string, read-only
Description:	A property that is the URL of the document that linked to the current document. The **`referrer`** property only contains a value when the user reaches the current document via a link. In all other cases the property is set to the empty string (""). If **`referrer`** is not supported by the browser, or is turned off for security reasons, this property will be an empty string.
Errors or Exceptions:	none
Example(s):	`if (document.referrer && document.referrer != "")` `document.write("You came in from "+ document.referrer + " ");`
Reference	Common IE and Netscape object property

9.5.2.4. title

Syntax:	**`document.title`**
Type:	string, read-only
Description:	The property that is set to the text in a **`<title>`** element, if present.
Errors or Exceptions:	none
Example(s):	**`var myTitle = document.title;`**
Reference	Common IE and Netscape object property

9.5.3. Methods

9.5.3.1. clear()

Syntax:	**`document.clear()`**
Argument List:	none
Description:	The current document is cleared from the browser window. It is identical to **`document.write("")`**. Note: Neither the cached version of the document, nor the history is affected.
Return Value Type:	none
Errors or Exceptions:	none
Example(s):	**`document.clear();`**
Reference	Common IE and Netscape object property

9.5.3.2. open()

Syntax:	**`document.open([MIMEType])`**
Argument List:	optional MIMEType – a hint to the browser that the content that will be written into a document by a subsequent **`document.write()`** statement will be of a MIME type other than text/html. (An XHTML-MP document is assumed if the parameter is missing). This is for compatibility only, and SHOULD be ignored by ECMAScript Mobile Profile implementations, as there is no way to modify the MIME type of the existing document. This parameter MAY be used as input to document validation if a validating parser is directly supported. This parameter MAY also be used as input to the implementation to direct the document to be interpreted as another supported MIME type.
Description:	Logically opens a path to the current document for writing. The document DOM tree is cleared. Note: It is often not necessary to specify a **`document.open()`** as **`document.write()`** automatically opens the current document for writing.
Return Value Type:	none
Errors or Exceptions:	none
Example(s):	(see section 9.5.3.3)
Reference	Common IE and Netscape object property

9.5.3.3. close()

Syntax:	**`document.close()`**
Argument List:	none
Description:	Closes the write path to the current document, and flushes all output to the document.
Return Value Type:	none
Errors or Exceptions:	none
Example(s):	**`var now = new Date;`** **`document.open();`** **`document.write(" The current time is " + now.getTime());`** **`document.close();`**
Reference	Common IE and Netscape object property

9.5.3.4. write(), writeln()

Syntax:	**`document.write( outputString)`** **`document.writeln(outputString)`**
Argument List:	outputString – an expression, converted to a string
Description:	writes the outputString, as part of a stream to the current document. If the document is closed, the write issues an automatic **`document.open()`** and whatever was previously written is discarded and replaced with the new outputString. **`document.writeln()`** is the same as **`document.write()`** except that it appends a newline to the end of the written string. **`document.write()`** statements in inline script segments are interpreted such that they replace the DOM nodes of the **`document.write()`** statements with the results of the **`document.write().`** Note: **`document.write()`** or equivalent statements in intrinsic event handlers create and write to a reset document rather than modifying the current one. This is reflected in the DOM tree, which is reset. (In future releases of W3C DOM this will be changing. Rather then a new DOM tree being created, a sibling branch will be created in the existing tree.)
Return Value Type:	none
Errors or Exceptions:	none Note: Subsequent interpretation of documents created using this method are subject to the same constraints and restrictions as any other markup document.
Example(s):	**`document.write("  Protocol= " + myURL.protocol);`**
Reference	Common IE and Netscape object property

9.6. Host Object Extension Mechanism

Other objects besides those defined in this section may be a part of the host object set specified for a particular device. Objects such as an EFI object, a crypto object etc. may be included in the **`global`** object property set. The definition of these extended objects is out of scope for this specification.

For details regarding the process associated with the creation and addition of new objects see [WAESPEC].

The addition of new host objects may follow two paths:

- The addition of new standard host objects – MUST follow standard OMA processes as independent specifications.

New standard built-in object names MUST be registered with WINA [WINA] according to the WINA process.

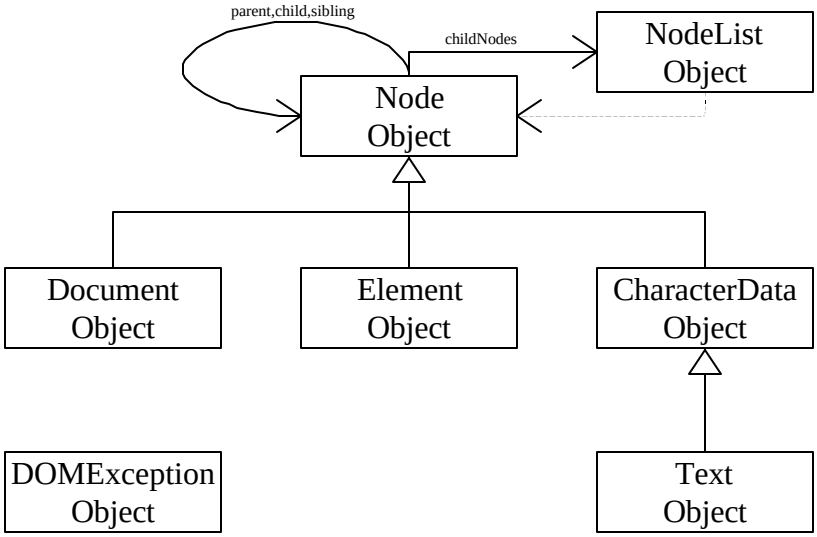
- The addition of new private host objects – no process is defined.

New object names SHOULD be registered with WINA [WINA] to guarantee that there will not be any name conflicts.

In all cases new objects MUST support the read-only version property, and MUST be enumerable as properties of the "parent" object. See section 6.2.2.

10. Browser DOM2 Core Objects

The **`document`** object set specified for ECMAScript Mobile Profile is a subset of the **`document`** object interface ECMAScript language bindings and definitions of the Document Object Model (DOM) found in [DOM2CORE]



Namespace aware versions of the object functions are not supported.

The API described in this section is partitioned into mandatory and optional functions and properties. Any methods that cause the document to change are labelled Mutation methods. Mutation is further sub-categorized into methods and properties that are oriented toward changing document data (data mutation) vs. methods and properties that are specifically oriented toward modifying the structure of the document (structural mutation). Methods and properties that are listed as causing structural mutation are optional.

Note: There are cases where methods that are primarily data mutation methods will cause structural changes to the document, i.e., a call to **`setAttribute()`** when there is no existing attribute.

Function and Property Grouping	Status	Requirement
Document Interrogation (DI)	Mandatory	
Data Mutation (DM)	Mandatory	DI
Structural Mutation (SM)	Optional	DI and DM

Methods are labelled as [DI],[DM],[SM] as appropriate.

10.1.DOMException Object

A **`DOMException`** object is derived from the ECMAScript **`Error`** object (Section 6.11).

10.1.1. Version History

Version	Affected	Comment
1.1		ECMAScript initial version

10.1.2. Properties

The properties of the **`DOMException`** Object are the same as the ECMAScript **`Error`** Object (Section 6.11).

The setting of the **`name`** and **`message`** properties of the **`exception`** object SHOULD be done when throwing a DOMException.

If the **`name`** property is set, it SHOULD be set to the error name listed in the table in section 10.1.3.

10.1.3. Constants

The DOM exception, when thrown may return one of the following constant number values for the name property:

Value	Error Name
1	INDEX_SIZE_ERR
2	DOMSTRING_SIZE_ERR
3	HIERARCHY_REQUEST_ERR
4	WRONG_DOCUMENT_ERR
5	INVALID_CHARACTER_ERR
6	NO_DATA_ALLOWED_ERR
7	NO_MODIFICATION_ALLOWED_ERR
8	NOT_FOUND_ERR
9	NOT_SUPPORTED_ERR
10	INUSE_ATTRIBUTE_ERR
11	INVALID_STATE_ERR
12	SYNTAX_ERR

13	INVALID_MODIFICATION_ERR
14	NAMESPACE_ERR
15	INVALID_ACCESS_ERR

Note: To maintain compatibility between DOM Exceptions and ECMAScript exceptions the naming of the errors appears inconsistent, with DOM exceptions using only upper case and ECMAScript using mixed case. This is intentional.

10.2.Node Object

The **`Node`** object is the base DOM object from which the **`Document`**, **`Element`** and **`CharacterData`** objects are derived.

10.2.1. Version History

Version	Affected	Comment
1.1		ECMAScript initial version

10.2.2. Properties

10.2.2.1. nodeName

nodeName is not supported. The value of the **nodeName** property is accessible through the element's **tagName** property.

10.2.2.2. nodeValue

nodeValue is not supported. The content of a **Text Node** is available through the **Text** object's **data** property.

10.2.2.3. nodeType

Syntax:	**someNode.nodeType**
Type:	number(integer), read-only
Description:	A property that describes the type of node being referenced. Values for **nodeType** are specified in [DOM2CORE], and valid values for supporting the DOM subset aresummarized in the following table:

Value	NodeType
1	**Element** Node
3	**Text** Node
9	**Document** Node

Other values (described in DOM2CORE) are not supported

Errors or Exceptions:	Unsupported **Node** types return 0.
Example(s):	
Reference	DOM2CORE Section 1.2, Appendix E.

10.2.2.4. parentNode

Syntax:	**someNode.parentNode**
Type:	**Node**, read-only
Description:	A property which points the the parent of the target node.
Errors or Exceptions:	If this is an unconnected **Node**, or there is no parent **Node** the property is set to null.
Example(s):	**parentnode=myNode.parent;**
Reference	DOM2CORE Section 1.2, Appendix E.

10.2.2.5. childNodes

Syntax:	**someNode.childNodes**
Type:	**NodeList**, read-only
Description:	A property that points to an array of one or more child nodes.
Errors or Exceptions:	If this is an unconnected **Node**, or there is are no child nodes the property is set to null.
Example(s):	**children=myNode.childNodes;**
Reference	DOM2CORE Section 1.2, Appendix E.

10.2.2.6. firstChild

Syntax:	**someNode.firstChild**
Type:	**Node**, read-only
Description:	–A property that points to the first child of the target **Node**.
Errors or Exceptions:	If this is an unconnected **Node**, or there is no child **Node** the property is set to null.
Example(s):	**Kid1=myNode.firstChild;**
Reference	DOM2CORE Section 1.2, Appendix E.

This read-only property is a **Node** object.

10.2.2.7. lastChild

Syntax:	**someNode.lastChild**
Type:	**Node**, read-only
Description:	–A property that points to the last child of the target **Node**.
Errors or Exceptions:	If this is an unconnected **Node**, or there is no child **Node** the property is set to null.
Example(s):	**Kidlast=myNode.lastChild;**
Reference	DOM2CORE Section 1.2, Appendix E.

10.2.2.8. previousSibling

Syntax:	**someNode.previousSibling**
Type:	**Node**, read-only
Description:	–A property that points the **Node** immediately preceding this **Node**.
Errors or Exceptions:	If this is an unconnected **Node**, or there is no preceding **Node** the property is set to null.
Example(s):	**aBrother=myNode.previousSibling;**
Reference	DOM2CORE Section 1.2, Appendix E.

10.2.2.9. nextSibling

Syntax:	**someNode.nextSibling**
Type:	**Node**, read-only
Description:	–A property that points the **Node** immediately following this **Node**.
Errors or Exceptions:	If this is an unconnected **Node**, or there is no following **Node** the property is set to null.
Example(s):	**anotherBrother=myNode.nextSibling;**
Reference	DOM2CORE Section 1.2, Appendix E.

10.2.2.10. attributes

The **attributes** property is not supported. To view and manipulate attributes use the **Element** functions **getAttribute()**,**setAttribute()**, and **hasAttribute().**

10.2.2.11. ownerDocument

ownerDocument is not supported.

10.2.2.12. namespaceURI

namespaceURI is not supported.

10.2.2.13. prefix

prefix is not supported.

10.2.2.14. localName

localName is not supported.

10.2.3. Methods

10.2.3.1. hasAttributes() [DI]

Syntax:	**<refNode>.hasAttributes()**
Argument List:	<refNode> - a **Node** object reference
Description:	returns a Boolean describing whether there are any attributes associated with this **Node**.
Return Value Type:	Boolean true – if there are child attributes under the referenced element false – it the referenced element has no child attributes
Errors or Exceptions:	None
Example(s):	**if(document.getElementByID("myNode").hasAttributes())** **{** **//handle the attributes** **}**
Reference	DOM2CORE Appendix E.

10.2.3.2. hasChildNodes() [DI]

Syntax:	**<refNode>.hasChildNodes()**
Argument List:	<refNode> - a **Node** object reference
Description:	returns a Boolean describing whether there are any child nodes of this element
Return Value Type:	Boolean true – if there are child nodes under the referenced element false – it the referenced element has no child nodes
Errors or Exceptions:	None
Example(s):	**if(document.getElementByID("myNode").hasChildNodes())** **{** **//process the children** **} else** **// no children, so we are done**
Reference	DOM2CORE Appendix E.

10.2.3.3. insertBefore() [SM]

Syntax:	**<refNode>.insertBefore(newChild [,refChild]);**
Argument List:	newChild – a **Node** object refChild – a **Node** object <refNode> - a **Node** object reference
Description:	finds the referenced **Node** <refNode> and insert the newChild **Node** as a child of that referenced **Node**. If refChild is specified the newChild **Node** is inserted before it. If not specified, the newChild **Node** is appended as the last child of the referenced element.
Return Value Type:	returns a **Node** object
Errors or Exceptions:	NO_MODIFICATION_ALLOWED_ERR is raised if the **Node** is read-only NOT_FOUND_ERR is raised if the refChild is not a child of this **Node**.
Example(s):	**myElem = document.createElement();** **document.getElementByID("anID").insertBefore(myElem);**
Reference	DOM2CORE Appendix E.

10.2.3.4. replaceChild() [SM]

Syntax:	**`<refNode>.replaceChild(newChild, oldChild);`**
Argument List:	newChild – a **`Node`** object refChild – a **`Node`** object <refNode> - a **`Node`** object reference
Description:	swaps the existing child (oldChild) of the referenced element with a new child **`Node`** (newChild).
Return Value Type:	returns a **`Node`** object reference
Errors or Exceptions:	DOMExceptions NO_MODIFICATION_ALLOWED_ERR is raised if the **`Node`** is read-only NOT_FOUND_ERR is raised if oldchild in not a child of this **`Node`**
Example(s):	
Reference	DOM2CORE Appendix E.

10.2.3.5. removeChild() [SM]

Syntax:	**`<refNode>.removeChild(aNode);`**
Argument List:	aNode – a **`Node`** Object <refNode> - a **`Node`** object reference
Description:	removes the reference to the child **`Node`** (aNode) from the referenced parent element
Return Value Type:	a **`Node`** object reference to the removed **`Node`**
Errors or Exceptions:	NO_MODIFICATION_ALLOWED_ERR is raised if the **`Node`** is read-only. NOT_FOUND_ERR is raised if the refChild is not a child of this **`Node`**.
Example(s):	
Reference	DOM2CORE Appendix E.

10.2.3.6. appendChild() [SM]

Syntax:	**`<refNode>.appendChild(newChild);`**
Argument List:	newChild – a **`Node`** object <refNode> - a **`Node`** object reference
Description:	finds the referenced **`Node`** <refNode> and insert the newChild node as a child of that referenced node.The newChild node is appended as the last child of the referenced element.
Return Value Type:	returns a **`Node`** object
Errors or Exceptions:	NO_MODIFICATION_ALLOWED_ERR is raised if the node is read-only
Example(s):	**`myElem = document.createElement();`** **`document.getElementByID("anID").appendChild(myElem);`**
Reference	DOM2CORE Appendix E.

10.2.3.7. cloneNode() [SM]

Syntax:	**`<refNode>.cloneNode(deep);`**
Argument List:	deep – Boolean <refNode> - a **`Node`** object reference
Description:	makes a copy of the node referenced by refElement. This copy is not attached to the document until explicitly connected using **`appendChild()`**, **`replaceChild()`** or **`insertBefore()`.** When the Boolean 'deep' is set to true, the element and all of its child nodes are copied.
Return Value Type:	a **`Node`** object reference to the just cloned node, or node subtree
Errors or Exceptions:	DOMExceptions
Example(s):	
Reference	DOM2CORE Appendix E.

10.3.DOM2 Document Object

The DOM2 **`document`** object is an extended definition of the interface of the basic **`document`** object (Section 9.5) to include a set of DOM2 objects, properties and methods for the traversal, and modification of a document.

Creation of arbitrary new documents, of arbitrary types is not supported.

10.3.1. Version History

Version	Affected	Comment
1.1		ECMAScript initial version

10.3.2. Properties

`doctype`, **`implementation`**, and **`documentElement`** properties, as specified in [DOM2CORE], are not supported.

10.3.3. Methods

10.3.3.1. createElement() [SM]

Syntax:	**`document.createElement(aTagName)`**
Argument List:	aTagName – a case sensitive string which is the element type to instantiate.
Description:	creates an element of the type specified.
Return Value Type:	returns a new **`Element`** object.
Errors or Exceptions:	INVALID_CHARACTER_ERR is raised if the specified tag name contains an invalid character.
Example(s):	
Reference	DOM2CORE Appendix E.

10.3.3.2. createTextNode() [SM]

Syntax:	**`document.createTextNode(textData)`**
Argument List:	textData – string
Description:	creates a **`Text`** node given the specified string.
Return Value Type:	The new **`Text`** object is returned.
Errors or Exceptions:	none
Example(s):	
Reference	DOM2CORE Appendix E.

10.3.3.3. getElementsByTagName() [DI]

Syntax:	**`document.getElementsByTagName(aTag);`**
Argument List:	aTag – a string which specifies an element name
Description:	returns a list of all objects in the document which have the specified element name
Return Value	a **`NodeList`**

Type:	
Errors or Exceptions:	Null is returned if there are no matches.
Example(s):	
Reference	DOM2CORE Sec. 1.2 ; DOM2CORE Appendix E.

10.3.3.4. getElementById() [DI]

Syntax:	**`document.getElementById(anID)`**
Argument List:	anID – string that specifies the value of an XML id attribute.
Description:	
Return Value Type:	returns the first object with a matching id attribute. If the id is associated with a collection, then the first object in that collection is returned.
Errors or Exceptions:	returns null if there is no match.
Example(s):	**`<script type="text/ecmascript">`** **`function gettarg() {`** **`  var targetelement = document.getElementById(“targ1”);`** **`}`** **`</script>`** **`<div id=”targ1”>This is the target. </div>`**
Reference	DOM2CORE Sec. 1.2 ; DOM2CORE Appendix E.

10.4.NodeList Object

DOM2 specifies the return from certain methods to be objects of type **`NodeList`**.

10.4.1. Version History

Version	Affected	Comment
1.1		ECMAScript initial version

10.4.2. Properties

10.4.2.1. length

Syntax:	**`ANodeList.length`**
Type:	number (unsigned integer), read-only
Description:	A property that is the number of nodes in the list.
Errors or Exceptions:	none
Example(s):	
Reference	DOM2CORE Sec. 1.2 ; DOM2CORE Appendix E.

10.4.3. Methods

10.4.3.1. item() [DI]

Syntax:	**`<refNodeList>.item(N) or <refNodeList>[N]`**
Argument List:	N – the index of the item in the **`NodeList`**
Description:	returns the nth indexed **`Node`** from the referenced **`NodeList`**. Note: The index is zero based.
Return Value Type:	returns a **`Node`** object
Errors or Exceptions:	null is returned if the index is not a valid index into the **`NodeList`**.
Example(s):	
Reference	DOM2CORE Sec. 1.2 ; DOM2CORE Appendix E.

10.5.Element Object

10.5.1. Version History

Version	Affected	Comment
1.1		ECMAScript initial version

10.5.2. Properties

10.5.2.1. tagName

Syntax:	**anElement.tagName**
Type:	string, read-only
Description:	A property that is the value of the **id** attribute in the referenced **Element**
Errors or Exceptions:	none
Example(s):	**<table id="bigtable" cellspacing="0" cellpadding="0">** **….** **</table>** **elemRef = document.getElementByID("bigtable");** **myID = elemRef.tagName; //myID=="bigtable"**
Reference	DOM2CORE Sec. 1.2 ; DOM2CORE Appendix E.

10.5.3. Methods

10.5.3.1. getAttribute() [DI]

Syntax:	**<refElement>.getAttribute(attrName [,caseSensitive])**
Argument List:	attrName – string, which is the name of a child attribute caseSensitive – optional Boolean, if set to true then the search for the attribute is case sensitive
Description:	returns a string which is the current value of the requested attribute. Note: This returns the value that is the running state value, not the initial document value.
Return Value Type:	string
Errors or Exceptions:	returns the empty string if it cannot find the requested attribute
Example(s):	
Reference	DOM2CORE Sec. 1.2 ; DOM2CORE Appendix E.

Note: The optional parameter specifying case sensitivity is an extension to the W3C DOM 2 Core specification.

10.5.3.2. setAttribute() [DM],[SM*]

Syntax:	**`<refElement>.setAttribute(attrName,attrValue [,caseSensitive])`**
Argument List:	attrName – string, which is the name of a child attribute attrValue – a string caseSensitive – optional Boolean, if set to true then the search for the attribute is case sensitive
Description:	sets the value of the requested attribute to the specified value. If the attribute doe not exist, it is created. * Note: using **`setAttribute()`** to set an attribute that does not exist will cause structural mutation
Return Value Type:	none
Errors or Exceptions:	INVALID_CHARACTER_ERR is raised if the specified name contains an illegal character. NO_MODIFICATION_ALLOWED_ERR is raised if the attribute is read-only. Note: If **`setAttribute()`** is called where an attribute does not exist, and structural mutation is not supported, this error is raised.
Example(s):	
Reference	DOM2CORE Sec. 1.2 ; DOM2CORE Appendix E.

Note: The optional parameter specifying case sensitivity is an extension to the W3C DOM 2 Core specification.

10.5.3.3. removeAttribute() [DM]

Syntax:	**`<refElement>.removeAttribute(attrName [,caseSensitive])`**
Argument List:	attrName – string, which is the name of a child attribute caseSensitive – optional Boolean, if set to true then the search for the attribute is case sensitive
Description:	removes the attribute from the referenced element
Return Value Type:	none
Errors or Exceptions:	Removing an attribute which is required by the DTD forces the creation of a new instance of that attribute, whose value is the default set by the DTD. NO_MODIFICATION_ALLOWED_ERR is raised if the attribute is read-only.
Example(s):	
Reference	DOM2CORE Sec. 1.2 ; DOM2CORE Appendix E.

Note: The optional parameter specifying case sensitivity is an extension to the W3C DOM 2 Core specification.

10.5.3.4. getElementsByTagName() [DI]

Syntax:	**`<refElement>.getElementsByTagName(aTag);`**
Argument List:	aTag – a string that matches the requested element.
Description:	returns a list of elements which are children of the reference element, whose element tag matches the input parameter.
Return Value Type:	a **`NodeList`**
Errors or Exceptions:	Null is returned if there are no matches.
Example(s):	
Reference	DOM2CORE Sec. 1.2 ; DOM2CORE Appendix E.

10.5.3.5. hasAttribute() [DI]

Syntax:	**`<refElement>.hasAttribute()`**
Argument List:	
Description:	returns a 'true' if the referenced element has any attributes associated with it. **`false`** is returned if there are no attributes associated with this element.
Return Value Type:	boolean
Errors or Exceptions:	none
Example(s):	
Reference	DOM2CORE Sec. 1.2 ; DOM2CORE Appendix E.

10.6.Text (CharacterData) Object

Properties of the **`Text`** object are inherited from the DOM2 **`CharacterData`** object.

10.6.1. Version History

Version	Affected	Comment
1.1		ECMAScript initial version

10.6.2. Properties

10.6.2.1. data

Syntax:	**`aTextNode.data[DM]`**
Type:	string
Description:	A property that is the current value of the text in the **`Text`** node. Using this property as a left-side assignment (lvalue) it is possible to reassign the value. Implementations MUST support the ability to deal with 32 character data strings as a minimum.
Errors or Exceptions:	NO_MODIFICATION_ALLOWED_ERR is raised if the **`Node`** is read-only DOMSTRING_SIZE_ERR is raised if the string to be returned is larger the implementation maximum.
Example(s):	
Reference	DOM2CORE Sec. 1.2 ; DOM2CORE Appendix E.

10.6.2.2. length

Syntax:	**`aTextNode.length`**
Type:	number(unsigned integer), read-only
Description:	A property that is the count of characters available through the **`data`** (see section 10.6.2.1) attribute.
Errors or Exceptions:	none
Example(s):	
Reference	DOM2CORE Sec. 1.2 ; DOM2CORE Appendix E.

10.6.3. Methods

10.6.3.1. appendData() [DM]

Syntax:	**`<nodeRef>.appendData(appendString)`**
Argument List:	appendString – a string to be appended
Description:	Appends a string to the end of any character data associated with the referenced **`Node`**.
Return Value Type:	none
Errors or Exceptions:	A NO_MODIFICATION_ALLOWED_ERR is raised if the **`Node`** is read-only
Example(s):	
Reference	DOM2CORE Sec. 1.2 ; DOM2CORE Appendix E.

10.6.3.2. deleteData() [DM]

Syntax:	**`<nodeRef>.deleteData(offset, count)`**
Argument List:	offset – the starting offset in the character data for removing characters. Offsets are zero (‘0’) based. count – the number of character units to remove
Description:	removes a number of characters from the character data associated with a **`Node`**
Return Value Type:	None Side effect: Upon success, the ‘data’ and ‘length’ properties will reflect the change.
Errors or Exceptions:	INDEX_SIZE_ERR is raised if the offset into the character data is not valid. NO_MODIFICATION_ALLOWED_ERR is raised if the **`Node`** is read-only
Example(s):	
Reference	DOM2CORE Sec. 1.2 ; DOM2CORE Appendix E.

10.6.3.3. insertData() [DM]

Syntax:	**`<nodeRef>.insertData(offset, string)`**
Argument List:	offset – the starting offset in the character data for adding characters. Offsets are zero (‘0’) based. string – the string to insert
Description:	adds a number of characters to the character data associated with a **`Node`**
Return Value Type:	None Side effect: Upon success, the ‘data’ and ‘length’ properties will reflect the change.
Errors or Exceptions:	INDEX_SIZE_ERR is raised if the offset into the character data is not valid. NO_MODIFICATION_ALLOWED_ERR is raised if the **`Node`** is read-only
Example(s):	
Reference	DOM2CORE Sec. 1.2 ; DOM2CORE Appendix E.

10.6.3.4. replaceData() [DM]

Syntax:	**`<nodeRef>.replaceData(offset, count, aString)`**
Argument List:	offset – the starting offset in the character data for removing characters, and for replacement. Offsets are zero (‘0’) based. count – the number of character units to replace aString – the replacement string
Description:	removes a number of characters from the character data associated with a **`Node`**, and replaces them with the supplied string parameter. This function is a combination of **`removeData()`** and **`insertData()`**.
Return Value Type:	None Side effect: Upon success, the ‘data’ and ‘length’ properties will reflect the change.
Errors or Exceptions:	INDEX_SIZE_ERR is raised if the offset into the character data is not valid. NO_MODIFICATION_ALLOWED_ERR is raised if the **`Node`** is read-only
Example(s):	
Reference	DOM2CORE Sec. 1.2 ; DOM2CORE Appendix E.

10.6.3.5. substringData() [DI]

Syntax:	**`<nodeRef>.substringData(offset, count)`**
Argument List:	offset – the starting offset in the character data for extracting characters. Offsets are zero (‘0’) based. count – the number of character units to extract
Description:	extracts a number of characters from the character data associated with a **`Node`**, and returns them. The character data is not effected.
Return Value Type:	A string is returned
Errors or Exceptions:	INDEX_SIZE_ERR is raised if the offset into the character data is not valid. DOMSTRING_SIZE_ERR is raised if the string to be returned is larger the implementation maximum.
Example(s):	
Reference	DOM2CORE Sec. 1.2 ; DOM2CORE Appendix E.

Appendix A. Static Conformance Requirements

This appendix is normative.

The notation used in this appendix is specified in [CREQ].

A.1 Encoder/ Compiler Conformance

Client conformance ("-C-") vs. Server ("-S-") conformance may be a misnomer in the context of a script interpreter or compiler. In this case all static conformance is listed as client, even though there may be a physical separation of the encoder/compiler function from the virtual machine execution.

Item	Function	Reference	Status	Requirement
-C-001	ECMAScript Type Support	5.2	M	
-C-002	UTF-16 Code point support	5.1.1	M	
-C-003	UTF-8 Code point support	5.1.1	M	
-C-004	Character indexing support	5.1.1,6.5	M	
-C-005	Semicolon support at end of statements	5.1.3	M	
-C-006	IEEE 754 64 Bit Float Support with accuracy to at least 14 digits	5.2.2	M	
-C-008	ECMAScript Language Syntax and semantics	5.5	M	
-C-009	Support for **`eval()`** ECMA327	5.4.2	O	
-C-010	Support for **`EvalError`** Exception	5.4.2	M	
-C-011	Support for dynamic function creation ECMA327	5.4.2, 6.12.2	O	-C-019
-C-012	Support for the **`with`** statement ECMA327	5.5.3	O	
-C-013	Support for dynamic modification of built-in objects ECMA327	6	O	
-C-014	Version properties for all native objects	6.2.1	M	
-C-015	All non-native, built-in objects are enumerable	6.2.2	M	
-C-016	All native, built-in objects are enumerable	6.2.2	O	
-C-017	Native Object Support – Global Object	6.3	M	
-C-018	Native Object Support – Array Object	6.4	M	
-C-019	Native Object Support –	6.5	M	

Item	Function	Reference	Status	Requirement
	`String` Object			
-C-020	Native Object Support – Regular Expression Object	6.6	M	
-C-021	Native Object Support – **`Boolean`** Object	6.7	M	
-C-022	Native Object Support – **`Number`** Object	6.8	M	
-C-023	Native Object Support – **`Math`** Object	6.9	M	
-C-024	Native Object Support – **`Date`** Object	6.10	M	
-C-025	Native Object Support – **`Error`** Object	6.11	M	
-C-026	Native Object Support – Object Object Creation ECMA327	6.12.1	O	
-C-027	Native Object Support – Function Object - dynamic function construction ECMA327	6.12.2	O	
-C-028	Support for **`EvalError`**	6.12.2	M	
-C-029	Support for Inline Script Execution	7.2.2.1	M	
-C-030	Support for Deferred Script Execution	7.2.2.2	M	
-C-031	Support for File Based Script Execution	7.2.2.3	M	
-C-032	Abnormal termination error reporting	7.3.1	O	
-C-033	Support for Aborted Script completion	7.3.2	M	
-C-034	Support for XHTML Events	8.1	M	XHTMLMP:MCF
-C-035	Support for DOM2 compliant event binding	8.2	M	
-C-036	Host Object Support – parent **`global`** Object	9.1	M	
-C-037	Host Object Support – **`navigator`** Object	9.2	M	
-C-038	Host Object Support – **`history`** Object	9.3	M	
-C-039	Host Object Support – **`location`** Object	9.4	M	
-C-040	Host Object Support – Basic **`document`** Object	9.5	M	

Item	Function	Reference	Status	Requirement
-C-041	Object Extension and addition must follow the WINA process	9.6	M	
-C-042	DOM Object Support – **`Exception`** Object	10.1	M	
-C-043	DOM Object Support – **`Node`** Object – Property Support	10.2	M	
-C-044	DOM Object Support – **`Node`** Object – Data Interrogation Methods	10.2	M	
-C-045	DOM Object Support – Structural Mutation	10	O	ESMP-C-045 AND ESMP-C-047 AND ESMP-C-052
-C-046	DOM Object Support – **`Node`** Object – Structural Modification Mutation Methods	10.2	O	
-C-047	DOM Object Support – **`document`** Object - Data Interrogation Methods	10.3	M	
-C-048	DOM Object Support – **`document`** Object – Structural Modification Mutation Methods	10.3	O	
-C-049	DOM Object Support – **`NodeLlist`** Object	10.4	M	
-C-050	DOM Object Support – **`Element`** Object – Property Support	10.5	M	
-C-051	DOM Object Support – **`Element`** Object – Data Interrogation Methods	10.5	M	
-C-052	DOM Object Support – **`Element`** Object – Data Modification Methods	10.5	M	
-C-053	DOM Object Support – **`Element`** Object – Structural Modification Mutation Methods	10.5	O	
-C-054	DOM Object Support – **`Text`** Object	10.6	M	

Appendix B. Change History

This appendix is informative.

Incorporated SCDs and SINs

SCD/SIN Document Identifier	Comments
	This is the initial version of the document.

Editorial changes

Location	Description
	This is the initial version of the document.

Appendix C. Mapping WMLScript Libraries to ECMAScript Mobile Profile Objects

This appendix is informative.

Library	Call	Object	Method/Constant	Comment
Lang	abs	Math	`abs()`	ECMA-262 Section 15.8.2.1
Lang	min	Math	`min()`	ECMA-262 Section 15.8.2.12
Lang	max	Math	`max()`	ECMA-262 Section 15.8.2.11
Lang	parseInt	Global	parseInt	
Lang	parseFloat	Global	parseFloat	
Lang	isInt			All numbers are 64 bit float and require no explicit conversion.
Lang	isFloat			All numbers are 64 bit float and require no explicit conversion.
Lang	maxInt		MAX_VALUE	These are actually floating point numbers in ECMAScript.
Lang	minInt		MIN_VALUE	These are actually floating point numbers in ECMAScript.
Lang	float			All numbers are 64 bit float and require no explicit conversion.
Lang	exit			
Lang	abort			
Lang	random	Math	random	random
Lang	seed			
Lang	characterSet			

Library	Call	Object	Method	Comment
Float	int			
Float	floor	Math	floor	floating point only ECMA-262 Section 15.8.2.9
Float	ceil	Math	ceil	floating point only ECMA-262 Section 15.8.2.6
Float	pow	Math	pow	floating point only ECMA-262 Section 15.8.2.13
Float	round	Math	round	floating point only ECMA-262 Section 15.8.2.15
Float	sqrt	Math	sqrt	floating point only ECMA-262 Section 15.8.2.17
Float	maxFloat	Number	MAX_VALUE	constant ECMA-262 Section 15.7.3.2
Float	minFloat	Number	MIN_VALUE	constant ECMA-262 Section 15.7.3.3

Library	Call	Object	Method/Property	Comment
String	length	String	.length property	a call in WMLScript vs.a property of the **`String`** object in ESMP
String	isEmpty			
String	charAt	String	**`charAt()`**	ECMA-262 Section 15.5.4.4
String	subString	String	**`subString()`**	Semantics are different
String	find	String	**`indexOf()`**	ECMA-262 Section 15.5.4.7 parameters are different
String	replace			See Note 1
String	elements			See Note 1
String	elementAt			See Note 1
String	removeAt			See Note 1
String	replaceAt	String	**`replace()`**	Object version works on itself ECMA-262 Section 15.5.4.11
String	insertAt			See Note 1
String	squeeze			
String	trim			
String	compare	String	**`localeCompare()`**	ESMP is Unicode sensitive ECMA-262 Section 15.5.4.9
String	toString	Global	**`toString()`**	ESMP returns a string value
String	format			

Note 1: these functions can be emulated with a combination of string.split, and array methods

Library	Call	Object	Method/Property	Comment
URL	getScheme	location	protocol property	
URL	getHost	location	host property	
URL	getPort	location	port property	
URL	getPath	location	hostname property	
URL	getParameters	location	href property	
URL	getQuery	location	search property	
URL	getFragment	location	hash property	
URL	getBase			
URL	getReferer	document	referrer property	
URL	resolve			
URL	escapeString	Global	**`encodeURI()`**, **`encodeURIComponent()`**	
URL	unescapeString	Global	**`decodeURI()`**, **`decodeURIComponent()`**	
URL	loadString			

Library	Call	Object	Method/Property	Comment
WMLBrowser	getVar			DOM access to the document replaces this function
WMLBrowser	setVar			DOM access to the document replaces this function
WMLBrowser	go	history	**`go()`**	
WMLBrowser	prev	history location	**`back()`** **`reload()`**	history.back retains variable state location.reload reinitialises state
WMLBrowser	newContext			
WMLBrowser	getCurrentCard	location	href property	return is absolute, not relative
WMLBrowser	refresh	location history	**`reload()`** **`go(0)`**	

Library	Call	Object	Method	Comment
Dialogs	prompt	Global	`prompt()`	part of the global object
Dialogs	confirm	Global	`confirm()`	part of the global object
Dialogs	alert	Global	`alert()`	part of the global object

Appendix D. Differences between WMLScript and ECMAScript Mobile Profile

This appendix is informative.

- WMLScript does not support objects. It supports native libraries. ECMAScript Mobile Profile supports objects.
- ECMAScript Mobile Profile supports 64 IEEE-754 numbers as the sole external numeric type, WMLScript supported integers and optionally 32-bit floats
- All objects in ECMAScript Mobile Profile must be versioned. WMLScript libraries were not versioned.
- WMLScript doesn't support HTML commenting.
- WMLScript doesn't support the usage of $ character in any position of the name. ECMAScript allows it.
- WMLScript requires lower case Boolean literals. Both upper and lower case characters are supported for boolean literals in ECMAScript.
- WMLScript supports local assignment of global variables with access to WML globals through a library call. There are no global variables in [XHTMLMP] or ECMAScript Mobile Profile
- WMLScript requires explicit variable declaration. ECMAScript allows the creation of variables, simply by referencing them.
- Native support for strings in WMLScript was via library calls. It is now available through the **`String`** object.
- Arrays are not supported in WMLScript. They are now available through the array object.
- **`Typeof()`** operators now return a string, not a value, consistent with standard ECMAScript. WMLScript **`typeof()`** returned a number.
- Variable length parameter lists are supported, per standard ECMAScript.
- Support for **`throw/catch`** and **`in`** is included in ECMAScript support.
- There is support for the 'NULL' literal in ECMAScript Mobile Profile .
- The *Invalid* literal is no longer supported in ECMAScript.

Appendix E. Differences between ECMAScript Mobile Profile and ECMA-262

This appendix is informative.

- All objects must maintain an enumerated version property.
- ECMAScript Mobile Profile support both UTF-8 and UTF-16 as input, where [ECMA262] specifies only UTF-16.
- As well as escape characters, ECMAScript Mobile Profile supports non-escape characters preceded by a backslash.
- The global method **`eval()`** is not supported in ECMAScript Mobile Profile. [ECMA327]
- Support for the **`with`** statement is not required for conformance. [ECMA327]
- The function object is not required to support dynamic function construction. [ECMA327]
- Dynamic modification of built-in (native and host) object properties (addition, deletion or assignment), other than the global object is not required for conformance. [ECMA327]
- ECMAScript Mobile Profile defines a set of "host" objects to interface to the XHTML browser and documents.
- There is no automatic insertion of missing semicolons at logical statement end in ECMAScript Mobile Profile.
- **`toLowerCase()`** is a synonym for **`toLocaleLowerCase()`**.
- **`toUpperCase()`** is a synonym for **`toLocaleUpperCase()`**.