ECMA Technical Report TR/00

ECMA 2000

ECMA/TC39-TG1/01/3

Standardizing Information and Communication Systems

ECMAScript 3rd Edition Compact Profile

Draft 0.6 January 2001

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

MB 2001tg1-003.doc 19.01.01 13:50

Brief history

This technical report defines ECMAScript Compact Profile as a subset of ECMA-262 (ECMAScript).

ECMAScript is based on several originating technologies, the most well known being JavaScript (Netscape) and JScript (Microsoft). The language was invented by Brendan Eich at Netscape and first appeared in that company's Navigator 2.0 browser. It has appeared in all subsequent browsers from Netscape and in all browsers from Microsoft starting with Internet Explorer 3.0.

The development of an ECMA standard began in November 1996. The first edition of ECMA-262 was adopted by the ECMA General Assembly of June 1997.

That ECMA Standard was submitted to ISO/IEC JTC1 for adoption under the fast-track procedure, and approved as international standard ISO/IEC 16262, in April 1998. The ECMA General Assembly of June 1998 approved the second edition of ECMA-262 to keep it fully aligned with ISO/IEC 16262. Changes between the first and the second edition are editorial in nature.

The third edition of ECMA-262 includes significant enhancements, for instance support for regular expressions, and improvements in regards to internationalisation. It was adopted by the ECMA General Assembly in December, 1999. This technical report defines the ECMAScript Compact Profile as a strict subset of the 3rd edition of ECMA-262. The Compact Profile is intended to meet the needs of resource-constrained environments, and has been developed with the help of Hewlett-Packard, IBM, Microsoft, Netscape, Nokia, Openwave, and Pixo.

1 Introduction

1.1 Scope

This Technical Report defines the ECMAScript Compact Profile (ES-CP) scripting language.

1.2 Conformance

A conforming implementation of ES-CP must provide and support all the types, values, objects, properties, functions, and program syntax and semantics described or normatively referenced in this specification.

A conforming implementation of ES-CP is permitted to provide additional types, values, objects, properties, and functions beyond those described in this specification. In particular, a conforming implementation of ES-CP is permitted to provide properties not described in this specification, and values for those properties, for objects that are described in this specification.

2 Normative References

ECMA-262, 3rd Edition

3 Definitions

ES3	ECMAScript Language Specification (ECMA-262), 3rd Edition
ES-CP	ECMAScript 3rd Edition Compact Profile

4 Overview

This section contains a non-normative overview of the ECMAScript Compact Profile language.

ECMAScript 3rd Edition is an object-based programming language for performing computations and manipulating computational objects within a host environment.

ECMAScript Compact Profile is a subset of ECMAScript 3rd Edition tailored to resource-constrained devices such as battery-powered embedded devices. Therefore, special attention is paid to constraining ECMAScript features that require proportionately large amounts of system memory (both for storing and executing the ECMAScript language features) and continuous or proportionately large amounts of processing power.

Operations which can potentially consume lots of memory may exhaust the available memory in a resource constrained device. It is recommended that implementations of ES-CP provide a means for scripts to determine the amount of available memory, to allow the appropriate action to be taken when memory is in short supply.

Implementations of ES-CP are also recommended to provide a means for scripts to determine the version of host objects (see ES3 section 4.3.8).

5 Language

This section contains a normative description of ECMAScript 3rd Edition Compact Profile. Unless specifically noted, ES-CP adopts all requirements of ECMA-262 3rd Edition.

5.1 Runtime Compilation

A conforming implementation of ES-CP SHALL support the global built-in object, but is NOT REQUIRED to support the global `eval()` method (ES3 section 15.1.2.1). A conforming implementation is NOT REQUIRED to support calling `Function` as a function (ES3 section 15.3.1.1) or as a constructor (ES3 section 15.3.2.1).

An implementation that does not support global `eval`() or calling `Function` as a function or constructor SHALL throw an `EvalError` exception whenever global `eval`() (ES3 section 15.1.2.1), `Function`(*p*$_1$, *p*$_2$, ..., *p*$_n$, *body*) (ES3 section 15.3.1.1), or `new Function`(*p*$_1$, *p*$_2$, ..., *p*$_n$, *body*) (ES3 section 15.3.2.1) is called.

NOTE

Runtime compilation is one of the most resource intensive features of ECMA-262, and many target environments for ES-CP might not support it. Global `eval()` requires runtime compilation because the source text might not appear in the program (ECMA-262 sections 15.1.2.1 and 10.1.2).' Runtime compilation is required to support the `Function` constructor and the `Function` function when the body is not a string literal.

5.2 Dynamic Modification of Built-in Objects

A conforming implementation of ES-CP is NOT REQUIRED to allow addition, deletion or assignment to the properties of built-in objects, other than the global object. Built-in properties of the global object are `READONLY` and `DONTDELETE`. Properties of the global object introduced by declarations (ES3 section 10.1.3) may be modified but not deleted.

If the implementation does not allow modifications to built-in objects, then it SHALL throw a `ReferenceError` exception when evaluating such modifications.

NOTE

The rationale for not requiring support for the modification of built-in objects is to allow more efficient implementations of ES-CP based upon static compilation of built-in objects without risking that the objects are mutated or shadowed by dynamically added properties.

5.3 The `with` Statement

A conforming implementation is NOT REQUIRED to support the `with` statement (ES3 section 12.10). If `with` is not supported, use of a `with` statement results in a syntax error.

NOTE

The `with` statement makes access to named references inefficient, because the scopes for such access cannot be computed until runtime.