

INTERNATIONAL
STANDARD

ISO/IEC
16262

Second edition
2002-06-01

**Information technology — ECMAScript
language specification**

Technologies de l'information — ECMAScript spécifications du langage

iTeh STANDARD PREVIEW
(standards.iteh.ai)

[ISO/IEC 16262:2002](https://standards.iteh.ai/catalog/standards/sist/f38a2d8a-09e0-443a-97d3-462c8a94ea08/iso-iec-16262-2002)

<https://standards.iteh.ai/catalog/standards/sist/f38a2d8a-09e0-443a-97d3-462c8a94ea08/iso-iec-16262-2002>



Reference number
ISO/IEC 16262:2002(E)

© ISO/IEC 2002

PDF disclaimer

This PDF file may contain embedded typefaces. In accordance with Adobe's licensing policy, this file may be printed or viewed but shall not be edited unless the typefaces which are embedded are licensed to and installed on the computer performing the editing. In downloading this file, parties accept therein the responsibility of not infringing Adobe's licensing policy. The ISO Central Secretariat accepts no liability in this area.

Adobe is a trademark of Adobe Systems Incorporated.

Details of the software products used to create this PDF file can be found in the General Info relative to the file; the PDF-creation parameters were optimized for printing. Every care has been taken to ensure that the file is suitable for use by ISO member bodies. In the unlikely event that a problem relating to it is found, please inform the Central Secretariat at the address given below.

iTeh STANDARD PREVIEW
(standards.iteh.ai)

[ISO/IEC 16262:2002](#)

<https://standards.iteh.ai/catalog/standards/sist/b38a2d8a-09e0-443a-97d3-462c8a94ea08/iso-iec-16262-2002>

© ISO/IEC 2002

All rights reserved. Unless otherwise specified, no part of this publication may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying and microfilm, without permission in writing from either ISO at the address below or ISO's member body in the country of the requester.

ISO copyright office
Case postale 56 • CH-1211 Geneva 20
Tel. + 41 22 749 01 11
Fax + 41 22 749 09 47
E-mail copyright@iso.ch
Web www.iso.ch

Printed in Switzerland

Contents

	Page
1 Scope	1
2 Conformance	1
3 Normative References	1
4 Overview	2
4.1 Web Scripting	2
4.2 Language Overview	2
4.2.1 Objects	3
4.3 Terms and Definitions	4
4.3.1 Type	4
4.3.2 Primitive Value	4
4.3.3 Object	4
4.3.4 Constructor	4
4.3.5 Prototype	5
4.3.6 Native Object	5
4.3.7 Built-in Object	5
4.3.8 Host Object	5
4.3.9 Undefined Value	5
4.3.10 Undefined Type	5
4.3.11 Null Value	5
4.3.12 Null Type	5
4.3.13 Boolean Value	5
4.3.14 Boolean Type	5
4.3.15 Boolean Object	5
4.3.16 String Value	6
4.3.17 String Type	6
4.3.18 String Object	6
4.3.19 Number Value	6
4.3.20 Number Type	6
4.3.21 Number Object	6
4.3.22 Infinity	6
4.3.23 NaN	6
5 Notational Conventions	6
5.1 Syntactic and Lexical Grammars	6
5.1.1 Context-Free Grammars	6
5.1.2 The Lexical and RegExp Grammars	7
5.1.3 The Numeric String Grammar	7
5.1.4 The Syntactic Grammar	7
5.1.5 Grammar Notation	8
5.2 Algorithm Conventions	10
6 Source Text	11
7 Lexical Conventions	11
7.1 Unicode Format-Control Characters	12
7.2 White Space	12
7.3 Line Terminators	12
7.4 Comments	13
7.5 Tokens	14

7.5.1 Reserved Words	14
7.5.2 Keywords	14
7.5.3 Future Reserved Words	14
7.6 Identifiers	15
7.7 Punctuators	16
7.8 Literals	16
7.8.1 Null Literals	16
7.8.2 Boolean Literals	17
7.8.3 Numeric Literals	17
7.8.4 String Literals	19
7.8.5 Regular Expression Literals	21
7.9 Automatic Semicolon Insertion	22
7.9.1 Rules of Automatic Semicolon Insertion	22
7.9.2 Examples of Automatic Semicolon Insertion	23
8 Types	24
8.1 The Undefined Type	24
8.2 The Null Type	24
8.3 The Boolean Type	24
8.4 The String Type	24
8.5 The Number Type	25
8.6 The Object Type	26
8.6.1 Property Attributes	26
8.6.2 Internal Properties and Methods	26
8.7 The Reference Type	29
8.7.1 GetValue (V)	30
8.7.2 PutValue (V, W)	30
8.8 The List Type	30
8.9 The Completion Type	30
9 Type Conversion	30
9.1 ToPrimitive	30
9.2 ToBoolean	31
9.3 ToNumber	31
9.3.1 ToNumber Applied to the String Type	31
9.4 ToInteger	34
9.5 ToInt32: (Signed 32 Bit Integer)	34
9.6 ToUint32: (Unsigned 32 Bit Integer)	35
9.7 ToUint16: (Unsigned 16 Bit Integer)	35
9.8 ToString	35
9.8.1 ToString Applied to the Number Type	36
9.9 ToObject	37
10 Execution Contexts	37
10.1 Definitions	37
10.1.1 Function Objects	37
10.1.2 Types of Executable Code	37
10.1.3 Variable Instantiation	38
10.1.4 Scope Chain and Identifier Resolution	38
10.1.5 Global Object	38
10.1.6 Activation Object	39
10.1.7 This	39
10.1.8 Arguments Object	39
10.2 Entering An Execution Context	39
10.2.1 Global Code	39
10.2.2 Eval Code	40
10.2.3 Function Code	40
11 Expressions	40

iTech STANDARD PREVIEW
(standards.iteh.ai)

<https://standards.iteh.ai/catalog/standards/sist/b8a2d8a-09e0-443a-97d3-462c8a94ca08/iso-iec-16262-2002>

11.1 Primary Expressions	40
11.1.1 The <code>this</code> Keyword	40
11.1.2 Identifier Reference	40
11.1.3 Literal Reference	40
11.1.4 Array Initialiser	40
11.1.5 Object Initialiser	42
11.1.6 The Grouping Operator	43
11.2 Left-Hand-Side Expressions	43
11.2.1 Property Accessors	44
11.2.2 The <code>new</code> Operator	44
11.2.3 Function Calls	45
11.2.4 Argument Lists	45
11.2.5 Function Expressions	45
11.3 Postfix Expressions	46
11.3.1 Postfix Increment Operator	46
11.3.2 Postfix Decrement Operator	46
11.4 Unary Operators	46
11.4.1 The <code>delete</code> Operator	46
11.4.2 The <code>void</code> Operator	47
11.4.3 The <code>typeof</code> Operator	47
11.4.4 Prefix Increment Operator	47
11.4.5 Prefix Decrement Operator	47
11.4.6 Unary <code>+</code> Operator	47
11.4.7 Unary <code>-</code> Operator	48
11.4.8 Bitwise NOT Operator (<code>~</code>)	48
11.4.9 Logical NOT Operator (<code>!</code>)	48
11.5 Multiplicative Operators	48
11.5.1 Applying the <code>*</code> Operator	49
11.5.2 Applying the <code>/</code> Operator	49
11.5.3 Applying the <code>%</code> Operator	49
11.6 Additive Operators	50
11.6.1 The Addition operator (<code>+</code>)	50
11.6.2 The Subtraction Operator (<code>-</code>)	51
11.6.3 Applying the Additive Operators (<code>+</code> , <code>-</code>) to Numbers	51
11.7 Bitwise Shift Operators	51
11.7.1 The Left Shift Operator (<code><<</code>)	51
11.7.2 The Signed Right Shift Operator (<code>>></code>)	52
11.7.3 The Unsigned Right Shift Operator (<code>>>></code>)	52
11.8 Relational Operators	52
11.8.1 The Less-than Operator (<code><</code>)	53
11.8.2 The Greater-than Operator (<code>></code>)	53
11.8.3 The Less-than-or-equal Operator (<code><=</code>)	53
11.8.4 The Greater-than-or-equal Operator (<code>>=</code>)	53
11.8.5 The Abstract Relational Comparison Algorithm	54
11.8.6 The <code>instanceof</code> operator	54
11.8.7 The <code>in</code> operator	54
11.9 Equality Operators	55
11.9.1 The Equals Operator (<code>==</code>)	55
11.9.2 The Does-not-equals Operator (<code>!=</code>)	55
11.9.3 The Abstract Equality Comparison Algorithm	56
11.9.4 The Strict Equals Operator (<code>===</code>)	57
11.9.5 The Strict Does-not-equal Operator (<code>!==</code>)	57
11.9.6 The Strict Equality Comparison Algorithm	57
11.10 Binary Bitwise Operators	57
11.11 Binary Logical Operators	58
11.12 Conditional Operator (<code>?:</code>)	59
11.13 Assignment Operators	60
11.13.1 Simple Assignment (<code>=</code>)	60

11.13.2 Compound Assignment (<code>op=</code>)	60
11.14 Comma Operator (<code>,</code>)	60
12 Statements.....	61
12.1 Block.....	61
12.2 Variable statement	62
12.3 Empty Statement.....	63
12.4 Expression Statement	63
12.5 The <code>if</code> Statement	64
12.6 Iteration Statements	64
12.6.1 The <code>do-while</code> Statement.....	65
12.6.2 The <code>while</code> statement.....	65
12.6.3 The <code>for</code> Statement.....	65
12.6.4 The <code>for-in</code> Statement.....	66
12.7 The <code>continue</code> Statement	67
12.8 The <code>break</code> Statement.....	67
12.9 The <code>return</code> Statement.....	68
12.10 The <code>with</code> Statement	68
12.11 The <code>switch</code> Statement.....	68
12.12 Labelled Statements	70
12.13 The <code>throw</code> statement	70
12.14 The <code>try</code> statement.....	71
13 Function Definition	72
13.1 Definitions	73
13.1.1 Equated Grammar Productions	73
13.1.2 Joined Objects.....	73
13.2 Creating Function Objects.....	73
13.2.1 <code>[[Call]]</code>	75
13.2.2 <code>[[Construct]]</code>	75
14 Program	75
15 Native ECMAScript Objects	76
15.1 The Global Object	77
15.1.1 Value Properties of the Global Object	77
15.1.2 Function Properties of the Global Object	77
15.1.3 URI Handling Function Properties.....	79
15.1.4 Constructor Properties of the Global Object.....	83
15.1.5 Other Properties of the Global Object	84
15.2 Object Objects.....	84
15.2.1 The Object Constructor Called as a Function	84
15.2.2 The Object Constructor	84
15.2.3 Properties of the Object Constructor	84
15.2.4 Properties of the Object Prototype Object.....	85
15.2.5 Properties of Object Instances	86
15.3 Function Objects	86
15.3.1 The Function Constructor Called as a Function	86
15.3.2 The Function Constructor	86
15.3.3 Properties of the Function Constructor.....	87
15.3.4 Properties of the Function Prototype Object	87
15.3.5 Properties of Function Instances.....	88
15.4 Array Objects.....	88
15.4.1 The Array Constructor Called as a Function	89
15.4.2 The Array Constructor	89
15.4.3 Properties of the Array Constructor.....	89
15.4.4 Properties of the Array Prototype Object.....	90
15.4.5 Properties of Array Instances	97

iTeh STANDARD PREVIEW
 (standards.iteh.ai)
<https://standards.iteh.ai/catalog/standards/sist/08a2d8a-09e0-443a-97d3-462c8a94ea08/iso-iec-16262-2002>

15.5 String Objects.....	98
15.5.1 The String Constructor Called as a Function	98
15.5.2 The String Constructor	98
15.5.3 Properties of the String Constructor	99
15.5.4 Properties of the String Prototype Object.....	99
15.5.5 Properties of String Instances	106
15.6 Boolean Objects.....	107
15.6.1 The Boolean Constructor Called as a Function	107
15.6.2 The Boolean Constructor	107
15.6.3 Properties of the Boolean Constructor	107
15.6.4 Properties of the Boolean Prototype Object.....	107
15.6.5 Properties of Boolean Instances	108
15.7 Number Objects	108
15.7.1 The Number Constructor Called as a Function.....	108
15.7.2 The Number Constructor	108
15.7.3 Properties of the Number Constructor.....	108
15.7.4 Properties of the Number Prototype Object	109
15.7.5 Properties of Number Instances.....	112
15.8 The Math Object.....	112
15.8.1 Value Properties of the Math Object	112
15.8.2 Function Properties of the Math Object.....	113
15.9 Date Objects	118
15.9.1 Overview of Date Objects and Definitions of Internal Operators	118
15.9.2 The Date Constructor Called as a Function	122
15.9.3 The Date Constructor	122
15.9.4 Properties of the Date Constructor.....	123
15.9.5 Properties of the Date Prototype Object.....	124
15.9.6 Properties of Date Instances	131
15.10 RegExp (Regular Expression) Objects	131
15.10.1 Patterns	131
15.10.2 Pattern Semantics.....	133
15.10.3 The RegExp Constructor Called as a Function	145
15.10.4 The RegExp Constructor	145
15.10.5 Properties of the RegExp Constructor.....	146
15.10.6 Properties of the RegExp Prototype Object	146
15.10.7 Properties of RegExp Instances.....	147
15.11 Error Objects.....	148
15.11.1 The Error Constructor Called as a Function.....	148
15.11.2 The Error Constructor.....	148
15.11.3 Properties of the Error Constructor	148
15.11.4 Properties of the Error Prototype Object	148
15.11.5 Properties of Error Instances.....	149
15.11.6 Native Error Types Used in This Standard.....	149
15.11.7 <i>NativeError</i> Object Structure	149
16 Errors	151
Annex A (informative) Grammar Summary.....	152
A.1 Lexical Grammar	152
A.2 Number Conversions	156
A.3 Expressions	157
A.4 Statements.....	161
A.5 Functions and Programs.....	163
A.6 Universal Resource Identifier Character Classes.....	163
A.7 Regular Expressions.....	164
Annex B (informative) Compatibility	166
B.1 Additional Syntax	166
B.1.1 Numeric Literals	166

B.1.2 String Literals	166
B.2 Additional Properties.....	167
B.2.1 escape (string).....	167
B.2.2 unescape (string)	168
B.2.3 String.prototype.substr (start, length).....	168
B.2.4 Date.prototype.getYear ().....	169
B.2.5 Date.prototype.setYear (year).....	169
B.2.6 Date.prototype.toGMTString ().....	169

iTeh STANDARD PREVIEW
(standards.iteh.ai)

[ISO/IEC 16262:2002](#)

<https://standards.iteh.ai/catalog/standards/sist/f38a2d8a-09e0-443a-97d3-462c8a94ea08/iso-iec-16262-2002>

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 3.

The main task of the joint technical committee is to prepare International Standards. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

Attention is drawn to the possibility that some of the elements of this International Standard may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

ISO/IEC 16262 was prepared by ECMA (as ECMA-262) and was adopted, under a special "fast-track procedure", by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 22, *Programming languages, their environments and system software interfaces*, in parallel with its approval by national bodies of ISO and IEC.

This second edition cancels and replaces the first edition (ISO/IEC 16262:1998), which has been technically revised.

[https://standards.iteh.ai/catalog/standards/sist/b38a2d8a-09e0-443a-97d3-](https://standards.iteh.ai/catalog/standards/sist/b38a2d8a-09e0-443a-97d3-462c8e94ea08/iso-iec-16262-2002)

[462c8e94ea08/iso-iec-16262-2002](https://standards.iteh.ai/catalog/standards/sist/b38a2d8a-09e0-443a-97d3-462c8e94ea08/iso-iec-16262-2002)
Annexes A and B of this International Standard are for information only.

iTeh STANDARD PREVIEW
(standards.iteh.ai)

ISO/IEC 16262:2002

<https://standards.iteh.ai/catalog/standards/sist/f38a2d8a-09e0-443a-97d3-462c8a94ea08/iso-iec-16262-2002>

Information technology — ECMAScript language specification

1 Scope

This International Standard defines the ECMAScript scripting language.

2 Conformance

A conforming implementation of ECMAScript must provide and support all the types, values, objects, properties, functions, and program syntax and semantics described in this specification.

A conforming implementation of this International Standard shall interpret characters in conformance with the Unicode Standard, Version 2.1 or later, and ISO/IEC 10646-1 with either UCS-2 or UTF-16 as the adopted encoding form, implementation level 3. If the adopted ISO/IEC 10646-1 subset is not otherwise specified, it is presumed to be the BMP subset, collection 300. If the adopted encoding form is not otherwise specified, it is presumed to be the UTF-16 encoding form.

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

A conforming implementation of ECMAScript is permitted to support program and regular expression syntax not described in this specification. In particular, a conforming implementation of ECMAScript is permitted to support program syntax that makes use of the "future reserved words" listed in section 7.5.3 of this specification.

3 Normative References

The following normative documents contain provisions which, through reference in this text, constitute provisions of this International Standard. For dated references, subsequent amendments to, or revisions of, any of these publications do not apply. However, parties to agreements based on this International Standard are encouraged to investigate the possibility of applying the most recent editions of the normative documents indicated below. For undated references, the latest edition of the normative document referred to applies. Members of ISO and IEC maintain registers of currently valid International Standards.

ISO/IEC 9899:1999 *Programming languages – C*, including Technical Corrigendum 1

ISO/IEC 10646-1:2000 *Information technology – Universal Multiple-Octet Coded Character Set (UCS) – Part 1: Architecture and Basic Multilingual Plane*, including Amendment 1

Unicode Inc. (1996), *The Unicode Standard™*, Version 2.0. ISBN: 0-201-48345-9, Addison-Wesley Publishing Co., Menlo Park, California.

Unicode Inc. (1998), *Unicode Technical Report #8: The Unicode Standard™*, Version 2.1.

Unicode Inc. (1998), *Unicode Technical Report #15: Unicode Normalization Forms*.

ANSI/IEEE Std 754-1985: *IEEE Standard for Binary Floating-Point Arithmetic*. Institute of Electrical and Electronic Engineers, New York (1985).

4 Overview

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

ECMAScript is an object-oriented programming language for performing computations and manipulating computational objects within a host environment. ECMAScript as defined here is not intended to be computationally self-sufficient; indeed, there are no provisions in this specification for input of external data or output of computed results. Instead, it is expected that the computational environment of an ECMAScript program will provide not only the objects and other facilities described in this specification but also certain environment-specific *host* objects, whose description and behaviour are beyond the scope of this specification except to indicate that they may provide certain properties that can be accessed and certain functions that can be called from an ECMAScript program.

A **scripting language** is a programming language that is used to manipulate, customise, and automate the facilities of an existing system. In such systems, useful functionality is already available through a user interface, and the scripting language is a mechanism for exposing that functionality to program control. In this way, the existing system is said to provide a host environment of objects and facilities, which completes the capabilities of the scripting language. A scripting language is intended for use by both professional and non-professional programmers. To accommodate non-professional programmers, some aspects of the language may be somewhat less strict.

ECMAScript was originally designed to be a **Web scripting language**, providing a mechanism to enliven Web pages in browsers and to perform server computation as part of a Web-based client-server architecture. ECMAScript can provide core scripting capabilities for a variety of host environments, and therefore the core scripting language is specified in this document apart from any particular host environment.

Some of the facilities of ECMAScript are similar to those used in other programming languages; in particular Java™ and Self, as described in:

- Gosling, James, Bill Joy and Guy Steele. The Java™ Language Specification. Addison Wesley Publishing Co., 1996.
- Ungar, David, and Smith, Randall B. Self: The Power of Simplicity. OOPSLA '87 Conference Proceedings, pp. 227–241, Orlando, FL, October 1987.

4.1 Web Scripting

A web browser provides an ECMAScript host environment for client-side computation including, for instance, objects that represent windows, menus, pop-ups, dialog boxes, text areas, anchors, frames, history, cookies, and input/output. Further, the host environment provides a means to attach scripting code to events such as change of focus, page and image loading, unloading, error and abort, selection, form submission, and mouse actions. Scripting code appears within the HTML and the displayed page is a combination of user interface elements and fixed and computed text and images. The scripting code is reactive to user interaction and there is no need for a main program.

A web server provides a different host environment for server-side computation including objects representing requests, clients, and files; and mechanisms to lock and share data. By using browser-side and server-side scripting together, it is possible to distribute computation between the client and server while providing a customised user interface for a Web-based application.

Each Web browser and server that supports ECMAScript supplies its own host environment, completing the ECMAScript execution environment.

4.2 Language Overview

The following is an informal overview of ECMAScript—not all parts of the language are described. This overview is not part of the standard proper.

ECMAScript is object-based: basic language and host facilities are provided by objects, and an ECMAScript program is a cluster of communicating objects. An ECMAScript **object** is an unordered collection of **properties**

each with zero or more **attributes** that determine how each property can be used—for example, when the `ReadOnly` attribute for a property is set to **true**, any attempt by executed ECMAScript code to change the value of the property has no effect. Properties are containers that hold other objects, **primitive values**, or **methods**. A primitive value is a member of one of the following built-in types: **Undefined**, **Null**, **Boolean**, **Number**, and **String**; an object is a member of the remaining built-in type **Object**; and a method is a function associated with an object via a property.

ECMAScript defines a collection of **built-in objects** that round out the definition of ECMAScript entities. These built-in objects include the **Global** object, the **Object** object, the **Function** object, the **Array** object, the **String** object, the **Boolean** object, the **Number** object, the **Math** object, the **Date** object, the **RegExp** object and the Error objects **Error**, **EvalError**, **RangeError**, **ReferenceError**, **SyntaxError**, **TypeError** and **URIError**.

ECMAScript also defines a set of built-in **operators** that may not be, strictly speaking, functions or methods. ECMAScript operators include various unary operations, multiplicative operators, additive operators, bitwise shift operators, relational operators, equality operators, binary bitwise operators, binary logical operators, assignment operators, and the comma operator.

ECMAScript syntax intentionally resembles Java syntax. ECMAScript syntax is relaxed to enable it to serve as an easy-to-use scripting language. For example, a variable is not required to have its type declared nor are types associated with properties, and defined functions are not required to have their declarations appear textually before calls to them.

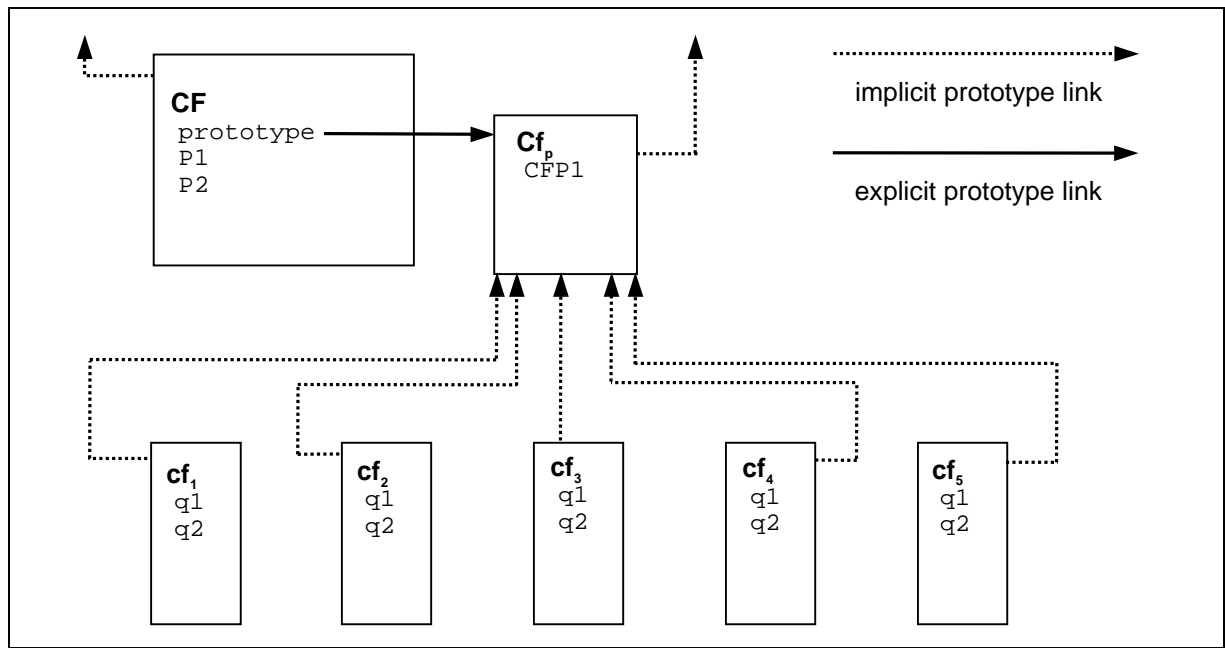
4.2.1 Objects

ECMAScript does not contain proper classes such as those in C++, Smalltalk, or Java, but rather, supports **constructors** which create objects by executing code that allocates storage for the objects and initialises all or part of them by assigning initial values to their properties. All constructors are objects, but not all objects are constructors. Each constructor has a **Prototype** property that is used to implement **prototype-based inheritance** and **shared properties**. Objects are created by using constructors in **new** expressions; for example, `new String("A String")` creates a new `String` object. Invoking a constructor without using **new** has consequences that depend on the constructor. For example, `String("A String")` produces a primitive string, not an object.

ECMAScript supports **prototype-based inheritance**. Every constructor has an associated prototype, and every object created by that constructor has an implicit reference to the prototype (called the *object's prototype*) associated with its constructor. Furthermore, a prototype may have a non-null implicit reference to its prototype, and so on; this is called the *prototype chain*. When a reference is made to a property in an object, that reference is to the property of that name in the first object in the prototype chain that contains a property of that name. In other words, first the object mentioned directly is examined for such a property; if that object contains the named property, that is the property to which the reference refers; if that object does not contain the named property, the prototype for that object is examined next; and so on.

In a class-based object-oriented language, in general, state is carried by instances, methods are carried by classes, and inheritance is only of structure and behaviour. In ECMAScript, the state and methods are carried by objects, and structure, behaviour, and state are all inherited.

All objects that do not directly contain a particular property that their prototype contains share that property and its value. The following diagram illustrates this:



CF is a constructor (and also an object). Five objects have been created by using **new** expressions: cf_1 , cf_2 , cf_3 , cf_4 , and cf_5 . Each of these objects contains properties named q_1 and q_2 . The dashed lines represent the implicit prototype relationship; so, for example, cf_3 's prototype is CF_p . The constructor, CF, has two properties itself, named P1 and P2, which are not visible to CF_p , cf_1 , cf_2 , cf_3 , cf_4 , or cf_5 . The property named CFP1 in CF_p is shared by cf_1 , cf_2 , cf_3 , cf_4 , and cf_5 (but not by CF), as are any properties found in CF_p 's implicit prototype chain that are not named q_1 , q_2 , or CFP1. Notice that there is no implicit prototype link between CF and CF_p .

Unlike class-based object languages, properties can be added to objects dynamically by assigning values to them. That is, constructors are not required to name or assign values to all or any of the constructed object's properties. In the above diagram, one could add a new shared property for cf_1 , cf_2 , cf_3 , cf_4 , and cf_5 by assigning a new value to the property in CF_p .

4.3 Terms and Definitions

The following are informal terms and definitions of key terms associated with ECMAScript.

4.3.1 Type

A **type** is a set of data values.

4.3.2 Primitive Value

A **primitive value** is a member of one of the types **Undefined**, **Null**, **Boolean**, **Number**, or **String**. A primitive value is a datum that is represented directly at the lowest level of the language implementation.

4.3.3 Object

An **object** is a member of the type **Object**. It is an unordered collection of properties each of which contains a primitive value, object, or function. A function stored in a property of an object is called a method.

4.3.4 Constructor

A **constructor** is a Function object that creates and initialises objects. Each constructor has an associated prototype object that is used to implement inheritance and shared properties.

4.3.5 Prototype

A **prototype** is an object used to implement structure, state, and behaviour inheritance in ECMAScript. When a constructor creates an object, that object implicitly references the constructor's associated prototype for the purpose of resolving property references. The constructor's associated prototype can be referenced by the program expression `constructor.prototype`, and properties added to an object's prototype are shared, through inheritance, by all objects sharing the prototype.

4.3.6 Native Object

A **native object** is any object supplied by an ECMAScript implementation independent of the host environment. Standard native objects are defined in this specification. Some native objects are built-in; others may be constructed during the course of execution of an ECMAScript program.

4.3.7 Built-in Object

A **built-in object** is any object supplied by an ECMAScript implementation, independent of the host environment, which is present at the start of the execution of an ECMAScript program. Standard built-in objects are defined in this specification, and an ECMAScript implementation may specify and define others. Every built-in object is a native object.

4.3.8 Host Object

A **host object** is any object supplied by the host environment to complete the execution environment of ECMAScript. Any object that is not native is a host object.

4.3.9 Undefined Value

iTeh STANDARD PREVIEW
(standards.iteh.ai)

The **undefined value** is a primitive value used when a variable has not been assigned a value.

4.3.10 Undefined Type

ISO/IEC 16262:2002

The type **Undefined** has exactly one value, called **undefined**.

4.3.11 Null Value

The **null value** is a primitive value that represents the null, empty, or non-existent reference.

4.3.12 Null Type

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

4.3.13 Boolean Value

A **boolean value** is a member of the type **Boolean** and is one of two unique values, **true** and **false**.

4.3.14 Boolean Type

The type **Boolean** represents a logical entity and consists of exactly two unique values. One is called **true** and the other is called **false**.

4.3.15 Boolean Object

A **Boolean object** is a member of the type **Object** and is an instance of the built-in Boolean object. That is, a Boolean object is created by using the Boolean constructor in a **new** expression, supplying a boolean as an argument. The resulting object has an implicit (unnamed) property that is the boolean. A Boolean object can be coerced to a boolean value.