[MS-ES5]:

Internet Explorer ECMA-262 ECMAScript Language Specification (Fifth Edition) Standards Support Document

---

Intellectual Property Rights Notice for Open Specifications Documentation

- **Technical Documentation.** Microsoft publishes Open Specifications documentation ("this documentation") for protocols, file formats, data portability, computer languages, and standards support. Additionally, overview documents cover inter-protocol relationships and interactions.
- **Copyrights**. This documentation is covered by Microsoft copyrights. Regardless of any other terms that are contained in the terms of use for the Microsoft website that hosts this documentation, you can make copies of it in order to develop implementations of the technologies that are described in this documentation and can distribute portions of it in your implementations that use these technologies or in your documentation as necessary to properly document the implementation. You can also distribute in your implementation, with or without modification, any schemas, IDLs, or code samples that are included in the documentation. This permission also applies to any documents that are referenced in the Open Specifications documentation.
- **No Trade Secrets**. Microsoft does not claim any trade secret rights in this documentation.
- **Patents**. Microsoft has patents that might cover your implementations of the technologies described in the Open Specifications documentation. Neither this notice nor Microsoft's delivery of this documentation grants any licenses under those patents or any other Microsoft patents. However, a given Open Specifications document might be covered by the Microsoft Open Specifications Promise or the Microsoft Community Promise. If you would prefer a written license, or if the technologies described in this documentation are not covered by the Open Specifications Promise or Community Promise, as applicable, patent licenses are available by contacting iplg@microsoft.com.
- **Trademarks**. The names of companies and products contained in this documentation might be covered by trademarks or similar intellectual property rights. This notice does not grant any licenses under those rights. For a list of Microsoft trademarks, visit www.microsoft.com/trademarks.
- **Fictitious Names**. The example companies, organizations, products, domain names, email addresses, logos, people, places, and events that are depicted in this documentation are fictitious. No association with any real company, organization, product, domain name, email address, logo, person, place, or event is intended or should be inferred.

Reservation of Rights. All other rights are reserved, and this notice does not grant any rights other than as specifically described above, whether by implication, estoppel, or otherwise.

Tools. The Open Specifications documentation does not require the use of Microsoft programming tools or programming environments in order for you to develop an implementation. If you have access to Microsoft programming tools and environments, you are free to take advantage of them. Certain Open Specifications documents are intended for use in conjunction with publicly available standards specifications and network programming art and, as such, assume that the reader either is familiar with the aforementioned material or has immediate access to it.

Revision Summary

Date	Revision History	Revision Class	Comments
9/8/2010	0.1	New	Released new document.
10/13/2010	0.2	Minor	Clarified the meaning of the technical content.
2/10/2011	1.0	Major	Significantly changed the technical content.
3/23/2011	1.1	Minor	Clarified the meaning of the technical content.
2/22/2012	2.0	Major	Significantly changed the technical content.
7/25/2012	2.1	Minor	Clarified the meaning of the technical content.
2/6/2013	2.2	Minor	Clarified the meaning of the technical content.
6/26/2013	3.0	Major	Significantly changed the technical content.
3/31/2014	3.0	None	No changes to the meaning, language, or formatting of the technical content.
1/22/2015	4.0	Major	Updated for new product version.
7/7/2015	4.1	Minor	Clarified the meaning of the technical content.
11/2/2015	4.1	None	No changes to the meaning, language, or formatting of the technical content.
1/20/2016	4.2	Minor	Clarified the meaning of the technical content.
3/22/2016	4.2	None	No changes to the meaning, language, or formatting of the technical content.
7/19/2016	4.3	Minor	Clarified the meaning of the technical content.
11/2/2016	4.3	None	No changes to the meaning, language, or formatting of the technical content.
3/14/2017	4.3	None	No changes to the meaning, language, or formatting of the technical content.

Table of Contents

1 Introduction

The ECMAScript of the Microsoft web browsers is a dialect of the language defined in the *ECMAScript Language Specification (Standard ECMA-262) Fifth Edition* [ECMA-262/5], published December 2009. This document describes the level of support provided by the browsers for that specification.

The [ECMA-262/5] specifications contain guidance for authors of webpages, browser users, and user agents (browser applications). This conformance document considers only normative language from the related specifications that applies directly to user agents.

1.1 Glossary

MAY, SHOULD, MUST, SHOULD NOT, MUST NOT: These terms (in all caps) are used as defined in [RFC2119]. All statements of optional behavior use either MAY, SHOULD, or SHOULD NOT.

1.2 References

Links to a document in the Microsoft Open Specifications library point to the correct section in the most recently published version of the referenced document. However, because individual documents in the library are not updated at the same time, the section numbers in the documents may not match. You can confirm the correct section numbering by checking the Errata.

1.2.1 Normative References

We conduct frequent surveys of the normative references to assure their continued availability. If you have any issue with finding a normative reference, please contact dochelp@microsoft.com. We will assist you in finding the relevant information.

[ECMA-262/5] ECMA International, "Standard ECMA-262 ECMAScript Language Specification", 5th Edition (December 2009), http://www.ecma-international.org/publications/files/ECMA-ST-ARCH/ECMA-262%205th%20edition%20December%202009.pdf

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997, http://www.rfc-editor.org/rfc/rfc2119.txt

1.2.2 Informative References

[MS-ES3EX] Microsoft Corporation, "Microsoft JScript Extensions to the ECMAScript Language Specification Third Edition".

[MS-ES3] Microsoft Corporation, "Microsoft JScript ECMA-262-1999 ECMAScript Language Specification Standards Support Document".

[MS-ES5EX] Microsoft Corporation, "Internet Explorer Extensions to the ECMA-262 ECMAScript Language Specification (Fifth Edition)".

1.3 Microsoft Implementations

The following Microsoft web browser versions implement some portion of the [ECMA-262/5] specification:

- Windows Internet Explorer 9
- Windows Internet Explorer 10
- Internet Explorer 11

- Internet Explorer 11 for Windows 10
- Microsoft Edge

Each browser version may implement multiple document rendering modes. The modes vary from one to another in support of the standard. The following table lists the document modes in each browser version that support the [ECMA-262/5] specification.

Browser Version	Document Modes Supported
Internet Explorer 9	IE9 Mode
Internet Explorer 10	IE9 Mode IE10 Mode
Internet Explorer 11	IE9 Mode IE10 Mode IE11 Mode
Internet Explorer 11 for Windows 10	IE9 Mode IE10 Mode IE11 Mode
Microsoft Edge	EdgeHTML Mode

For each variation presented in this document there is a list of the document modes and browser versions that exhibit the behavior described by the variation. All combinations of modes and versions that are not listed conform to the specification. For example, the following list for a variation indicates that the variation exists in four document modes in all browser versions that support these modes:

IE9 Mode, IE10 Mode, IE11 Mode, and EdgeHTML Mode (All Versions)

1.4 Standards Support Requirements

To conform to [ECMA-262/5], a user agent must provide and support all the types, values, objects, properties, functions, and program syntax and semantics described in the specification (See [ECMA-262/5] section 2, Conformance). Any optional portions that have been implemented must also be implemented as described by the specification. Normative language is usually used to define both required and optional portions. (For more information, see [RFC2119].)

The following table lists the sections of [ECMA-262/5] and whether they are considered normative or informative.

Sections	Normative/Informative
1	Informative
2-3	Normative
4	Informative
5-15	Normative
Annex A–Annex E	Informative

Relationship to Standards and Other Extensions

The following documents describe variations and extensions from versions 3 and 5 of the ECMAScript Language:

Document Type	Reference	Title
Variations	[MS-ES3]	Internet Explorer ECMA-262 ECMAScript Language Specification Standards Support Document
Extensions	[MS-ES3EX]	Microsoft JScript Extensions to the ECMAScript Language Specification Third Edition
Extensions	[MS-ES5EX]	Internet Explorer Extensions to the ECMA-262 ECMAScript Language Specification (Fifth Edition)

1.5 Notation

The following notations are used in this document to differentiate between notes of clarification, variation from the specification, and points of extensibility.

Notation	Explanation
C####	This identifies a clarification of ambiguity in the target specification. This includes imprecise statements, omitted information, discrepancies, and errata. This does not include data formatting clarifications.
V####	This identifies an intended point of variability in the target specification such as the use of MAY, SHOULD, or RECOMMENDED. (See [RFC2119].) This does not include extensibility points.
E####	Because the use of extensibility points (such as optional implementation-specific data) can impair interoperability, this profile identifies such points in the target specification.

For document mode and browser version notation, see also section 1.3.

2 Standards Support Statements

This section contains all variations and clarifications for the Microsoft implementation of [ECMA-262/5].

- Section 2.1 describes normative variations from the MUST requirements of the specification.
- Section 2.2 describes clarifications of the MAY and SHOULD requirements.
- Section 2.3 considers error handling aspects of the implementation.
- Section 2.4 considers security aspects of the implementation.

2.1 Normative Variations

The following subsections describe normative variations from the MUST requirements of [ECMA-262/5].

2.1.1 [ECMA-262/5] Section 7.6, Identifier Names and Identifiers

V0043:

The specification states:

```
ECMAScript implementations may recognise identifier characters defined in later editions of
the Unicode Standard. If portability is a concern, programmers should only employ identifier
characters defined in Unicode 3.0.
```

IE9 Mode, IE10 Mode, and IE11 Mode (All Versions)

Unicode 2.1 is supported. Later Unicode standards are not supported.

2.1.2 [ECMA-262/5] Section 10.1.1, Strict Mode Code

V0001:

The specification states:

```
Strict Mode Code
An ECMAScript Program syntactic unit may be processed using either unrestricted or strict
mode syntax and semantics.
```

IE9 Mode, IE10 Mode, and IE11 Mode (All Versions)

No code is interpreted as strict mode code. For more information about strict mode, see Annex C of [ECMA-262/5].

2.1.3 [ECMA-262/5] Section 11.4.3, The typeof Operator

V0002:

The specification states:

```
Table 20 – typeof Operator Results lists the strings returned when the production
UnaryExpression : typeof UnaryExpression is evaluated. When the type of value is:
```

```
Object (host and does not implement [[Call]])

The result is:

Implementation-defined except may not be "undefined", "boolean", "number", or "string".
```

IE9 Mode, IE10 Mode, IE11 Mode, and EdgeHTML Mode (All Versions)

"object" is returned for all host objects, including those objects that do not implement **[[Call]]**.

2.1.4 [ECMA-262/5] Section 12.15, The debugger statement

V0003:

The specification states:

```
1.  If an implementation defined debugging facility is available and enabled, then
      a.  Perform an implementation defined debugging action.
      b.  Let result be an implementation defined Completion value.
```

IE9 Mode, IE10 Mode, IE11 Mode, and EdgeHTML Mode (All Versions)

If a debugger is attached to the currently executing program, step 1.a suspends execution and passes control to the debugger. If the debugger resumes execution of the program, step 1.b produces the Completion value of (normal, empty, empty).

2.1.5 [ECMA-262/5] Section 15.1, The Global Object

V0005:

The specification states:

```
The values of the [[Prototype]] and [[Class]] internal properties of the global object are
implementation-dependent.
```

IE9 Mode, IE10 Mode, IE11 Mode, and EdgeHTML Mode (All Versions)

The **[[Class]]** internal property of the global object is "WindowPrototype" and the **[[Prototype]]** internal property is an implementation-provided prototype object.

2.1.6 [ECMA-262/5] Section 15.2.2.1, newObject ([value])

V0006:

The specification states:

```
1.  If value is supplied, then
    a.  If Type(value)  is Object, then
        i.  If the value is a native ECMAScript object, do not create a new object but simply
return value.
        ii. If the value is a host object, then actions are taken and a result is returned in
an implementation-dependent manner that may depend on the host object.
```

IE9 Mode, IE10 Mode, IE11 Mode, and EdgeHTML Mode (All Versions)

`value` is returned if the `value` is a host object.

2.1.7 [ECMA-262/5] Section 15.2.4.4, Object.prototype.valueOf ()

V0007:

The specification states:

```
2.  If O is the result of calling the Object constructor with a host object (15.2.2.1), then

    a.  Return either O or another value such as the host object originally passed to the
constructor. The specific result that is returned is implementation-defined.
```

IE9 Mode, IE10 Mode, IE11 Mode, and EdgeHTML Mode (All Versions)

When O is the result of calling the Object constructor with a host object, the **this** value is returned.

2.1.8 [ECMA-262/5] Section 15.3.4.2, Function.prototype.toString ()

V0008:

The specification states:

```
An implementation-dependent representation of the function is returned. This representation
has the syntax of a FunctionDeclaration. Note in particular that the use and placement of
white space, line terminators, and semicolons within the representation String is
implementation-dependent.
```

IE9 Mode, IE10 Mode, IE11 Mode, and EdgeHTML Mode (All Versions)

The following variations apply:

- The implementation-dependent representation has the syntax of a **FunctionExpression** function object.
- The representation of a function that is implemented by using ECMAScript code is the exact sequence of characters that is used to define the function. The first character of the representation is the letter "f" of function, and the final character is the final closing brace ("}") of the function definition. However, if the function is defined by using a **FunctionExpression** function object that is immediately surrounded by one or more levels of grouping operators ([ECMA-262/5] section 11.1.6), the first character of the representation is the opening parenthesis ("(") of the innermost such grouping operator and the final character is the closing parenthesis (")") of the innermost such grouping operator.

    If the function is created by the **Function** constructor ([ECMA-262/5] section 15.3.2.1), the representation of the function consists of the following elements in this order:

    1. The string "`function anonymous(`"
    2. The value of P that is used in step 16 of the [ECMA-262/5] section 15.3.2.1 algorithm that created the function
    3. The string "`) {`"
    4. A <LF> character
    5. The value of **body** that is used in step 16 of the algorithm

6. A <LF> character and a closing brace ("}").

If the function is not implemented by using ECMAScript code (that is, it is a built-in function or a host object function), the **FunctionBody** function object of the generated representation does not conform to ECMAScript syntax. Instead, the **FunctionBody** function object consists of the string "[native code]".

The format of the representation that is generated has the syntax of a standard ECMAScript, Fifth Edition **FunctionExpression** function object rather than a **FunctionDeclaration** function object. For anonymous functions that are created through a **FunctionExpression** function object that does not include the optional Identifier, the generated syntax does not include the optional Identifier and does not conform to the base standard's definition of **FunctionExpression**.

2.1.9 [ECMA-262/5] Section 15.4.4.3, Array.prototype.toLocaleString ()

V0010:

The specification states:

```
8.   Else

    …

    d.   Let R be the result of calling the [[Call]] internal method of func
providing elementObj as the this value and an empty arguments list.
…
10.   Repeat, while k < len

    …

    d.   Else

         …

         iv.   Let R be the result of calling the [[Call]] internal method of func
providing elementObj as the this value and an empty arguments list.
```

IE9 Mode, IE10 Mode, IE11 Mode, and EdgeHTML Mode (All Versions)

For the steps that are described in steps 8.d and 10.d.iv, if a recursive call to **toLocaleString** would cause a non-terminating recursion, the empty string is used as the result.

V0011:

The specification states:

```
When the toString method is called, the following steps are taken:
1.  Let array be the result of calling ToObject on the this value.
2.  Let func be the result of calling the [[Get]] internal method of array with argument
"join".
3.  If IsCallable(func) is false, then let func be the standard built-in method
Object.prototype.toString (15.2.4.2).
4.  Return the result of calling the [[Call]] internal method of func providing array as the
this value and an empty arguments list.
```

IE9 Mode, IE10 Mode, IE11 Mode, and EdgeHTML Mode (All Versions)

In step 4, the separator character is determined by using the Microsoft Windows **GetLocaleInfo** system function and requesting the `LOCALE_LIST` value for the current user locale.

2.1.10 [ECMA-262/5] Section 15.5.4.9, String.prototype.localeCompare (that)

V0032:

The specification states:

```
The actual return values are implementation-defined to permit implementers to encode
additional information in the value, but the function is required to define a total ordering
on all Strings and to return 0 when comparing Strings that are considered canonically
equivalent by the Unicode standard.
```

IE9 Mode, IE10 Mode, IE11 Mode, and EdgeHTML Mode (All Versions)

The returned value is determined as follows:

1. Call the Microsoft Windows **CompareString** system function, passing `S`, `that`, and the current locale information as arguments.
2. Pass the value `0` as the **dwCmpFlags** argument.
3. Return result (`1`).

2.1.11 [ECMA-262/5] Section 15.7.4.5, Number.prototype.toFixed (fractionDigits)

V0033:

The specification states:

```
An implementation is permitted to extend the behaviour of toFixed for values of
fractionDigits less than 0 or greater than 20. In this case toFixed would not necessarily
throw RangeError for such values.
```

IE9 Mode, IE10 Mode, IE11 Mode, and EdgeHTML Mode (All Versions)

If any value of the **fractionDigits** function is converted to an integer and it is equal to +∞ or −∞, this value is treated as if it is the value `0`.

2.1.12 [ECMA-262/5] Section 15.7.4.6, Number.prototype.toExponential (fractionDigits)

V0034:

The specification states:

```
An implementation is permitted to extend the behaviour of toExponential for values of
fractionDigits less than 0 or greater than 20. In this case toExponential would not
necessarily throw RangeError for such values.
```

IE9 Mode, IE10 Mode, IE11 Mode, and EdgeHTML Mode (All Versions)

If any value of the **fractionDigits** function is converted to an integer and it is equal to +∞ or −∞, this value is treated as if it is the value `0`.

2.1.13 [ECMA-262/5] Section 15.7.4.7, Number.prototype.toPrecision (precision)

V0035:

The specification states:

```
An implementation is permitted to extend the behaviour of toPrecision for values of precision
less than 1 or greater than 21. In this case toPrecision would not necessarily throw
RangeError for such values.
```

IE9 Mode, IE10 Mode, IE11 Mode, and EdgeHTML Mode (All Versions)

The behavior of the **toPrecision** function is not extended to values of the **precision** property that are less than 1 or greater than 21.

2.1.14 [ECMA-262/5] Section 15.9.1.8, Daylight Saving Time Adjustment

V0036:

The specification states:

```
If the host environment provides functionality for determining daylight saving
time, the implementation of ECMAScript is free to map the year in question to an
equivalent year (same leap-year-ness and same starting week day for the year) for
which the host environment provides daylight saving time information. The only
restriction is that all equivalent years should produce the same result.
```

IE9 Mode, IE10 Mode, IE11 Mode, and EdgeHTML Mode (All Versions)

To determine adjustments for daylight savings time, equivalent years are mapped to the current year by using the following values.

Day of the week for January 1	0 (Sunday)	1 (Monday)	2 (Tuesday)	3 (Wednesday)	4 (Thursday)	5 (Friday)	6 (Saturday)
Non-leap years before 2007	1995	1979	1991	1975	1987	1971	1983
Leap years before 2007	1984	1996	1980	1992	1976	1988	1972
2007 and non-leap years after 2007	2023	2035	2019	2031	2015	2027	2011
Leap years after 2007	2012	2024	2036	2020	2032	2016	2028

2.1.15 [ECMA-262/5] Section 15.9.1.14, TimeClip (time)

V0037:

The specification states:

> The operator TimeClip calculates a number of milliseconds from its argument, which must be an ECMAScript Number value. This operator functions as follows:
>
> 1. If time is not finite, return NaN.
> 2. If abs(time) > 8.64 x 1015, return NaN.
> 3. Return an implementation-dependent choice of either ToInteger(time) or ToInteger(time) + (+0). (Adding a positive zero converts −0 to +0.)

IE9 Mode, IE10 Mode, IE11 Mode, and EdgeHTML Mode (All Versions)

In step 3, **ToInteger**(*time*) is returned.

2.1.16 [ECMA-262/5] Section 15.9.4.2, Date.parse (string)

V0038:

The specification states:

> The parse function applies the ToString operator to its argument and interprets the resulting String as a date and time; it returns a Number, the UTC time value corresponding to the date and time. The String may be interpreted as a local time, a UTC time, or a time in some other time zone, depending on the contents of the String. The function first attempts to parse the format of the String according to the rules called out in Date Time String Format (15.9.1.15). If the String does not conform to that format the function may fall back to any implementation-specific heuristics or implementation-specific date formats. Unrecognizable Strings or dates containing illegal element values in the format String shall cause **Date.parse** to return NaN.

IE9 Mode, IE10 Mode, IE11 Mode, and EdgeHTML Mode (All Versions)

If the argument string for the **parse** function does not conform to the Date Time String Format, the **parse** function tries to parse the string value and it produces a value in accordance with the following grammar and rules. If the string cannot be recognized starting with the **DateString** production, the `NaN` number value is returned.

Date String Syntax

The following lexical grammar defines the tokens that make up date strings.

DateToken ::

Separator

NumericDateToken

AlphaDateToken

DateComment

OffsetFlag

Separator :: one of

, : / <SP>

DateComment ::

(DateCommentBody$_{opt}$ *)*

DateCommentBody ::

DateCommentChars DateComment$_{opt}$

DateComment DateCommentBody$_{opt}$

DateCommentChars ::

DateCommentChar DateCommentChars$_{opt}$

*DateCommentCh*ar ::

DateChar **but not** *(or)*

OffsetFlag :: one of

\+ –

AlphaDateToken ::

AlphaDateComponent period$_{opt}$

AlphaDateComponent ::

WeekDay

Month

TimeZone

MilitaryTimeZone

AmPmFlag

AdBcFlag

period ::

.

WeekDay ::

Sunday

Monday

Tuesday

Wednesday

Thursday

Friday

Saturday

Month ::

January

February

March

April

May

June

July

August

September

October

November

December

TimeZone ::

`est`

`edt`

`cst`

`cdt`

`mst`

`mdt`

`pst`

`pdt`

`gmt`

`utc`

MilitaryTimeZone ::

`a` [lookahead ∉ {`.m m d .d p u`}]

`p` [lookahead ∉ {`.m m d s`}]

`b` [lookahead ∉ {`.c c`}]

`f` [lookahead ∉ {`e i`}]

`m` [lookahead ∉ {`a d o s`}]

`s` [lookahead ∉ {`a e u`}]

o [lookahead ≠ c]

n [lookahead ≠ o]

d [lookahead ≠ e]

t [lookahead ∉ {h u}]

w [lookahead ≠ e]

e [lookahead ∉ {d s}]

c [lookahead ∉ {d s}]

g [lookahead ≠ m]

u [lookahead ≠ t

UniqueMilitaryTimeZone

UniqueMilitaryTimeZone :: **one of**

z y x v r q h i k l

AmPmFlag ::

am

a.m

pm

p.m

AdBcFlag ::

ad

a.d

bc

b.c

NumericDateToken ::

NumericDateComponent -

NumericDateComponent [lookahead ≠ –]

NumericDateComponent ::

DateDigit [lookahead ∉ *DateDigit*]

DateDigit DateDigit [lookahead ∉ *DateDigit*]

DateDigit DateDigit DateDigit [lookahead ∉ *DateDigit*]

DateDigit DateDigit DateDigit DateDigit [lookahead ∉ *DateDigit*]

DateDigit DateDigit DateDigit DateDigit DateDigit [lookahead ∉ *DateDigit*]

DateDigit DateDigit DateDigit DateDigit DateDigit DateDigit [lookahead ∉ *DateDigit*]

DateDigit :: **one of**

0 1 2 3 4 5 6 7 8 9

Sunday ::

su

sun

sund

sunda

sunday

Monday ::

mo

mon

mond

monda

monday

Tuesday ::

tu

tue

tues

tuesd

tuesda

tuesday

Wednesday ::

we

wed

wedn

wedne

wednes

wednesd

wednesda

wednesday

Thursday ::

```
th
thu
thur
thurs
thursd
thursda
thursday
```

Friday ::

```
fr
fri
frid
frida
friday
```

Saturday ::

```
sa
sat
satu
satur
saturd
saturda
saturday
```

January ::

```
ja
jan
janu
januar
january
```

February ::

```
fe
feb
```

```
febr
febru
februa
februar
february
```

March ::

```
ma
mar
marc
march
```

April ::

```
ap
apr
apri
april
```

May ::

```
ma
may
```

June ::

```
jun
june
```

July ::

```
ju
jul
july
```

August ::

```
au
aug
augu
augus
august
```

September ::

se

sep

sept

septe

septem

septemb

septembe

september

October ::

oc

oct

octo

octob

octobe

october

November ::

no

nov

nove

novem

novemb

novembe

november

December ::

de

dec

dece

decem

decemb

decembe

```
december
```

Parsing Rules for Date.parse Date Strings

1. The string to be parsed is converted to lowercase and then these rules are applied.
2. The preceding grammar syntax uses **NumericDateToken** literals or **AlphaDateToken** literals to define the following components of a date object: weekday, year, month, date, hours, minutes, seconds, time zone, AD/BC flag, and AM/PM flag.
3. Any date string must define at least year, month, and date components. No component can be defined multiple times.
4. Components can be in any order, except for cases that are explicitly specified otherwise.
5. The following rules apply to the **OffsetFlags** literal:
    - The plus sign (+) and minus sign (-) are offset classifiers, when they do not follow a number. The next numeric component that follows an offset classifier is classified as an offset value. The numeric component does not have to follow immediately after the plus sign (+) or minus sign (-).
    - The + offset and the – offset cannot be specified before the year field. + or - offsets refer to the UTC time zone and set the time zone to UTC. A time zone component cannot follow a + or - offset.
6. The colon (:) separator char acts as a time classifier for numeric components:
    - A colon (:) that follows a number classifies the previous numeric component as hours.
    - A colon (:) that follows a number that is classified as an hour classifies the next numeric component as minutes. The next numeric component does not have to immediately follow the colon.
    - A colon (:) that follows a number that is classified as a minute classifies the next numeric component as seconds. The next number does not have to immediately follow the colon.
7. The following rules define date classification for numeric components:
    - A number that is not classified and that has a value that is greater than or equal to 70 is always classified as years. Even when such a number is followed by a colon (:) and could be classified as hours, the number is classified as years. In this case, the colon (:) is a simple separator.
    - A number that is not classified by a classifier is always classified as a date.
    - Forward slash (/) and hyphen (-) separator chars can act as classifiers in the following ways:
        - A forward slash (/) or hyphen (-) that follows a numeric component classifies that numeric component as months.
        - A forward slash (/) or hyphen (-) that follows a numeric component that is classified as a month classifies the next numeric component as a date. The next numeric component does not have to immediately follow the forward slash or the hyphen.
        - A forward slash (/) or hyphen (-) that follows a numeric component that is classified as a date classifies the next numeric component as a year. The next numeric component does not have to immediately follow the forward slash or the hyphen.
8. The week day is ignored regardless of whether it is correct or incorrect.

9. The default value for the AD/BC flag is `AD`.

10. When the AM/PM flag is not defined, the default interpretation for hours is 24-hour notation. The AM flag is ignored when the time is greater than 13:00:00. When the PM flag is used, the time must be less than 12:00.

Algorithm for Computing the Time Value

Numeric values are calculated for year, month, date, and time through classification, numeric components, and alpha components. The following adjustments are done because of the flags, offsets, and time zones:

1. If the BC/AD flag is `BC`, `year = -year + 1`.

   Note 1 BC is year 0 and 2 BC is year -1.

2. If the BC/AD flag is `AD` and the year value is less than 100, `year = year+1900`. This rule allows the short form for the year value. For example, 99 stands for 1999.

3. The time value (that is, the time during the day) is calculated in seconds from the hour, minute, and seconds components. The AM/PM flag can change the time value as follows:

   - If no AM/PM flag is present, the time is considered to be in 24-hour notation and no adjustment is done.
   - If the time is greater than or equal to 12 * 3600 and the time is less than 13*3600 and if the AM/PM flag is AM, time = time – 12*3600. For example, 12:45 AM means 0:45.
   - If the AM/PM flag is PM and the time is less than 12*3600, time = time + 12 *3600. For example, 2PM means 14:00.

4. Time zone adjustment. The time value (from rule 3) is adjusted by the zone display values that are specified in the following tables. Check the **TimeZone** and **MilitaryTimeZone** values. If *zone* is the value for a given zone, the time is adjusted by: `time = time - zone * 60`.

5. Offset adjustment. The offset value applies to the time in the UTC zone. Let *nn* be the value of the numeric component that follows an offset. The following formulas define the offset value, in seconds, that then add up to the UTC time:

   - If nn <24: vOffset = 60* nn * 60
   - If nn >=24: vOffset = 60* (nn modulo 100) + (floor (nn / 100)) * 60))
   - time = Result(4) - vOffset * 60;

6. Date adjustment. Set `date = date -1`.

7. Month adjustment. Set `month = (month-1)`.

8. Final time calculation:

   - year = year + floor(month / 12);
   - month = Remainder(month, 12)
   - day = day + DayFromYear(year);
   - day = day + DayNumbersForTheMonthOfALeapYear(month);
   - If month is greater than or equal to 2 and the year is not a leap year, day = day - 1;
   - result = day * 86400000 + time;

9. If no time zone is specified, consider the time to be in the current local time zone and then get the UTC displacement of the time.

TimeZone value	UTC displacement
est	-5
edt	-4
cst	-6
cdt	-5
mst	-7
mdt	-6
pst	-8
pdt	-7
gmt	0
utc	0

MilitaryTimeZone value	UTC displacement
z	0
y	12
x	11
w	10
v	9
u	8
t	7
s	6
r	5
q	4
p	3
o	2
n	1
a	-1
b	-2
c	-3
d	-4
e	-5

MilitaryTimeZone value	UTC displacement
f	-6
g	-7
h	-8
i	-9
k	-10
l	-10
m	12

2.1.17 [ECMA-262/5] Section 15.9.4.3, Date.UTC (year, month [, date [, hours [, minutes [, seconds [, ms]]]]])

V0039:

The specification states:

```
When the UTC function is called with fewer than two arguments, the behaviour is
implementation-dependent.
```

IE9 Mode, IE10 Mode, IE11 Mode, and EdgeHTML Mode (All Versions)

When the UTC function is called with less than two arguments, the following steps are taken:

1. If *year* is supplied, let *y* be **ToNumber(***year***)**; otherwise, let *y* be `0`.
2. If *month* is supplied, let *m* be **ToNumber(***month***)**; otherwise, let *m* be `0`.
3. If *date* is supplied, let *dt* be **ToNumber(***date***)**; otherwise, let *dt* be `1`.
4. If *hours* is supplied, let *h* be **ToNumber(***hours***)**; otherwise, let *h* be `0`.
5. If *minutes* is supplied, let *min* be **ToNumber(***minutes***)**; otherwise, let *min* be `0`.
6. If *seconds* is supplied, let *s* be **ToNumber(***seconds***)**; otherwise, let *s* be `0`.
7. If *ms* is supplied, let *milli* be **ToNumber(***ms***)**; otherwise, let *milli* be `0`.
8. If *y* is not **NaN** and 0 ≤ **ToInteger(***y***)** ≤ 99, let *yr* be 1900+**ToInteger(***y***)**; otherwise, let *yr* be *y*.
9. Return **TimeClip(MakeDate(MakeDay(***yr*, *m*, *dt***)**, **MakeTime(***h*, *min*, *s*, *milli***)))**.

2.1.18 [ECMA-262/5] Section 15.9.5.2, Date.prototype.toString ()

V0040:

The specification states:

```
This function returns a String value. The contents of the String are implementation-
dependent, but are intended to represent the Date in the current time zone in a convenient,
human-readable form.
```

IE9 Mode, IE10 Mode, IE11 Mode, and EdgeHTML Mode (All Versions)

The returned **String** value of the **Date.prototype.toString** method is determined from the following steps:

1. Let *tv* be the time value.
2. If *tv* is **NaN**, return the string "`NaN`".
3. Let *t* be **LocalTime**(*tv*).
4. Using *t*, create a string value that has the following format, according to the items that are defined in the following table:

   DDDbMMMbddbhh:mm:ssbzzzzzzbyyyyy
5. Return *Result(4)*.

The following table defines the variables in the string value that is referenced in the preceding steps.

Variable	Description
DDD	The day of the week abbreviation from the following set: `Sun Mon Tue Wed Thu Fri Sat`.
b	A single space character.
MMM	The month name abbreviation from the following set: `Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec`.
dd	The day of the month as a one-decimal or two-decimal number, from `1` to `31`.
hh	The number of complete hours since midnight as a two-decimal number.
:	The colon character.
mm	The number of complete minutes since the start of the hour, as a two-decimal number.
ss	The number of complete seconds since the start of the minute, as a two-decimal number.
zzz or zzzzzzzz	If the local time offset from UTC is an integral number of hours between -8 and -5 inclusive, this item is the standard abbreviation for the corresponding North American time zone. This time zone is one of the following set: `EST EDT CST CDT MST MDT PST PDT`. Otherwise, this item is the characters **UTC** followed by a plus sign (+) or minus sign (−) character that corresponds to the sign of the local offset from UTC followed by the two-decimal hours part of the UTC offset and the two-decimal minutes part of the UTC offset.
yyyyy	If **YearFromTime(*t*)** is greater than 0, this item is three or more digits from the value of **YearFromTime(*t*)**. Otherwise, this item is the one or more numbers that correspond to the number that is 1-**YearFromTime(*t*)** followed by a single space character and then followed by `B.C.`
,	The comma character.
UTC	The literal characters `UTC`.

2.1.19 [ECMA-262/5] Section 15.10.1, Patterns

V0083:

The specification states:

```
Term ::
  Assertion
  Atom
  Atom Quantifier
Atom ::
  PatternCharacter
  .
  \ AtomEscape
  CharacterClass
  ( Disjunction )
  ( ? : Disjunction )
PatternCharacter ::
  SourceCharacter but not one of
    ^ $ \ . * + ? ( ) [ ] { } |
Assertion ::
  ^
  $
  \ b
```

```
  \ B
  ( ? = Disjunction )
  ( ? ! Disjunction )
AtomEscape ::
  DecimalEscape
  CharacterEscape
  CharacterClassEscape
CharacterEscape ::
  ControlEscape
  c ControlLetter
  HexEscapeSequence
  UnicodeEscapeSequence
  IdentityEscape
IdentityEscape ::
  SourceCharacter but not IdentifierPart
  <ZWJ>
  <ZWNJ>
NonemptyClassRanges ::
  ClassAtom
  ClassAtom NonemptyClassRangesNoDash
  ClassAtom - ClassAtom ClassRanges
NonemptyClassRangesNoDash ::
  ClassAtom
  ClassAtomNoDash NonemptyClassRangesNoDash
  ClassAtomNoDash - ClassAtom ClassRanges
ClassAtom ::
  -
  ClassAtomNoDash
ClassAtomNoDash ::
  SourceCharacter but not one of \ or ] or -
  \ ClassEscape
ClassEscape ::
  DecimalEscape
  b
  CharacterEscape
  CharacterClassEscape
```

IE9 Mode, IE10 Mode, IE11 Mode, and EdgeHTML Mode (All Versions)

The pattern grammar is instead context sensitive for the productions listed below, and ambiguities are introduced that are broken by ordering and contextual information. The following grammar is used, with each alternative considered only if previous production alternatives do not match:

```
Term ::
  Assertion
  AtomNoBrace Quantifier
  Atom
  QuantifiableAssertion Quantifier
AtomNoBrace ::
  PatternCharacterNoBrace
  .
  \ AtomEscape
  CharacterClass
  ( Disjunction )
  ( ? : Disjunction )
Atom ::
  PatternCharacter
  .
  \ AtomEscape
  CharacterClass
  ( Disjunction )
  ( ? : Disjunction )
PatternCharacterNoBrace ::
  SourceCharacter but not one of
    ^ $ \ . * + ? ( ) [ ] { } |
```

```
PatternCharacter ::
  SourceCharacter but not one of
    ^ $ \ . * + ? ( ) [ ] |
QuantifiableAssertion ::
  ( ? = Disjunction )
  ( ? ! Disjunction )
Assertion ::
  ^
  $
  \ b
  \ B
  QuantifiableAssertion
AtomEscape ::
  DecimalEscape but only if the integer value of DecimalEscape is <= NCapturingParens
  CharacterClassEscape
  CharacterEscape
CharacterEscape ::
  ControlEscape
  c ControlLetter
  HexEscapeSequence
  UnicodeEscapeSequence
  OctalEscapeSequence
  IdentityEscape
IdentityEscape ::
  SourceCharacter but not c
  <ZWJ>
  <ZWNJ>
NonemptyClassRanges ::
  ClassAtom
  ClassAtom NonemptyClassRangesNoDash
  ClassAtomInRange - ClassAtomInRange ClassRanges
NonemptyClassRangesNoDash ::
  ClassAtom
  ClassAtomNoDash NonemptyClassRangesNoDash
  ClassAtomNoDashInRange - ClassAtomInRange ClassRanges
ClassAtom ::
  -
  ClassAtomNoDash
ClassAtomNoDash ::
  SourceCharacter but not one of \ or ] or -
  \ ClassEscape
ClassAtomInRange ::
  -
  ClassAtomNoDashInRange
ClassAtomNoDashInRange ::
  SourceCharacter but not one of \ or ] or -
  \ ClassEscape but only if ClassEscape evaluates to a CharSet with exactly one character
  \ IdentityEscape
ClassEscape ::
  DecimalEscape but only if the integer value of DecimalEscape is <= NCapturingParens
  b
  CharacterClassEscape
  CharacterEscape
```

2.1.20 [ECMA-262/5] Section 15.10.2.5, Term

V0084:

The specification defines the productions for Term.

IE9 Mode, IE10 Mode, IE11 Mode, and EdgeHTML Mode (All Versions)

In addition to the existing productions for Term, the production *Term* :: *QuantifiableAssertion Quantifier* evaluates as follows:

```
1.  Return the result of evaluating the term ( ? : QuantifiableAssertion ) Quantifier
```

2.1.21 [ECMA-262/5] Section 15.10.2.8, Atom

V0085:

IE9 Mode, IE10 Mode, IE11 Mode, and EdgeHTML Mode (All Versions)

The specification defines the productions for Atom.

In addition to the existing productions for Atom, include identical productions for AtomNoBrace, except replacing *Atom* :: *PatternCharacter* with:

The production *AtomNoBrace* :: *PatternCharacterNoBrace* evaluates as follows:

```
1. Let ch be the character represented by PatternCharacterNoBrace.
2. Let A be a one-element CharSet containing the character ch.
3. Call CharacterSetMatcher(A, false) and return its Matcher result.
```

2.1.22 [ECMA-262/5] Section 15.10.2.17, ClassAtom

V0086:

The specification states:

```
The production ClassAtom :: - evaluates by returning the CharSet containing the one character
-.
The production ClassAtom :: ClassAtomNoDash evaluates by evaluating ClassAtomNoDash to obtain
a CharSet and returning that CharSet.
```

IE9 Mode, IE10 Mode, IE11 Mode, and EdgeHTML Mode (All Versions)

The production *ClassAtom* **::** **-** evaluates by returning the CharSet containing the one character **-**.

The production *ClassAtom* **::** *ClassAtomNoDash* evaluates by evaluating *ClassAtomNoDash* to obtain a CharSet and returning that CharSet.

The production *ClassAtomInRange* **::** **-** evaluates by returning the CharSet containing the one character **-**.

The production *ClassAtomInRange* **::** *ClassAtomNoDashInRange* evaluates by evaluating *ClassAtomNoDashInRange* to obtain a CharSet and returning that CharSet.

2.1.23 [ECMA-262/5] Section 15.10.2.18, ClassAtomNoDash

V0087:

The specification states:

```
The production ClassAtomNoDash :: SourceCharacter but not one of \ or ] or - evaluates by
returning a one element CharSet containing the character represented by SourceCharacter.
The production ClassAtomNoDash :: \ ClassEscape evaluates by evaluating ClassEscape to obtain
a CharSet and returning that CharSet.
```

IE9 Mode, IE10 Mode, IE11 Mode, and EdgeHTML Mode (All Versions)

The production *ClassAtomNoDash* **::** *SourceCharacter* **but not one of \ or] or -** evaluates by returning a one element CharSet containing the character represented by *SourceCharacter*.

The production *ClassAtomNoDash* **::** **** *ClassEscape* evaluates by evaluating *ClassEscape* to obtain a CharSet and returning that CharSet.

The production *ClassAtomNoDashInRange* **::** *SourceCharacter* **but not one of \ or] or -** evaluates by returning a one element CharSet containing the character represented by *SourceCharacter*.

The production *ClassAtomNoDashInRange* **::** **** *ClassEscape* evaluates by evaluating *ClassEscape* to obtain a CharSet and returning that CharSet.

The production *ClassAtomNoDashInRange* **::** **** *IdentityEscape* evaluates by evaluating *IdentityEscape* to obtain a CharSet and returning that CharSet.

2.1.24 [ECMA-262/5] Section B.1.2, String Literals

V0041:

The specification states:

```
OctalEscapeSequence ::
   OctalDigit [lookahead ∉ DecimalDigit]
   ZeroToThree OctalDigit [lookahead ∉ DecimalDigit]
   FourToSeven OctalDigit
   ZeroToThree OctalDigit OctalDigit
```

IE9 Mode, IE10 Mode, IE11 Mode, and EdgeHTML Mode (All Versions)

The syntax of the **OctalEscapeSequence** literal can be extended only as follows, with the **lookahead** element characterized with respect to the **OctalDigit** set:

```
OctalEscapeSequence ::
   OctalDigit [lookahead ∉ OctalDigit]
   ZeroToThree OctalDigit [lookahead ∉ OctalDigit]
   FourToSeven OctalDigit
   ZeroToThree OctalDigit OctalDigit
```

V0042:

The specification states:

- `The CV of OctalEscapeSequence :: OctalDigit [lookahead ∉ DecimalDigit] is the character whose code unit value is the MV of the OctalDigit.`
- `The CV of OctalEscapeSequence :: ZeroToThree OctalDigit [lookahead ∉ DecimalDigit] is the character whose code unit value is (8 times the MV of the ZeroToThree) plus the MV of the OctalDigit.`

IE9 Mode, IE10 Mode, IE11 Mode, and EdgeHTML Mode (All Versions)

The semantics of the **OctalEscapeSequence** literal can be extended only as follows, with the **lookahead** element characterized with respect to the **OctalDigit** set:

- The character value (CV) of `OctalEscapeSequence :: OctalDigit [lookahead ∉ OctalDigit]` is the character whose code unit value is the mathematical value (MV) of the **OctalDigit** element.

- The CV of `OctalEscapeSequence :: ZeroToThree OctalDigit [lookahead ∉ OctalDigit]` is the character whose code unit value is 8 times the MV of the **ZeroToThree** element plus the MV of the **OctalDigit** element.

2.2 Clarifications

The following subsections describe clarifications of the MAY and SHOULD requirements of [ECMA-262/5].

2.2.1 [ECMA-262/5] Section 8.5, The Number Type

C0008:

The specification states:

```
In some implementations, external code might be able to detect a difference between various
Not-a-Number values, but such behaviour is implementation-dependent; to ECMAScript code, all
NaN values are indistinguishable from each other.
```

IE9 Mode, IE10 Mode, IE11 Mode, and EdgeHTML Mode (All Versions)

`NaN` values are not normalized to the same value.

2.2.2 [ECMA-262/5] Section 12.6, Iteration Statements

C0023:

The specification states:

```
If new properties are added to the object being enumerated during enumeration, the
newly added properties are not guaranteed to be visited in the active enumeration.
```

IE9 Mode, IE10 Mode, IE11 Mode, and EdgeHTML Mode (All Versions)

Newly added properties are not visited in the active enumeration.

C0016:

The specification states:

```
If new properties are added to the object being enumerated during enumeration, the newly
added properties are not guaranteed to be visited in the active enumeration.
```

IE9 Mode, IE10 Mode, IE11 Mode, and EdgeHTML Mode (All Versions)

Newly added properties are not visited in the active enumeration.

2.2.3 [ECMA-262/5] Section 15.7.4.3, Number.prototype.toLocaleString ()

C0001:

The specification states:

```
Produces a string value that represents the value of this Number value formatted according to
the conventions of the host environment's current locale. This function is implementation-
dependent, and it is permissible, but not encouraged, for it to return the same thing as
toString.
```

IE9 Mode, IE10 Mode, IE11 Mode, and EdgeHTML Mode (All Versions)

Internet Explorer ECMAScript determines a string value as follows.

1. If the value of the **Number** object is an integer, return the result of calling the **Function.prototype.toString** method with the **Number** value as the argument.
2. If this **Number** value is **NaN**, return the string value "NaN".
3. If this **Number** value is +**Infinity** or -**Infinity**, return the statically localized string that describes such a value.
4. Create a string value by using the **Number.prototype.toFixed** algorithm in section 15.7.4.5 of [ECMA-262/5]. Use this **Number** value as the **this** value. Use the actual number of significant decimal fraction digits, *fractionDigits*, of this **Number** value as the argument. The *fractionDigits* value is computed according to the **ToString** algorithm in section 9.8.1 of [ECMA-262/5].
5. Call the **GetNumberFormat** Microsoft Windows system function (http://msdn.microsoft.com/en-us/library/dd318110(VS.85).aspx), passing it *Result(4)* and the current locale information. The values zero and **NULL** are passed as the format flags and the *lpFormat* arguments.
6. If the call in step 5 succeeds, return *Result(5)*.
7. If the calls in either step 4 or step 5 fail, return the result of calling the standard built-in **Date.prototype.toString** method with *Result(1)* as the **this** object.
8. Call the **VariantChangeType** Windows OLE Automation function (http://msdn.microsoft.com/en-us/library/aa910747.aspx), passing it *Result(4)* and the current locale information.
9. Return the string value that corresponds to *Result(8)*.

2.2.4 [ECMA-262/5] Section 15.9.5.3, Date.prototype.toDateString ()

C0002:

The specification states:

```
This function returns a String value. The contents of the String are implementation-
dependent, but are intended to represent the "date" portion of the Date in the current time
zone in a convenient, human-readable form.
```

IE9 Mode, IE10 Mode, IE11 Mode, and EdgeHTML Mode (All Versions)

The returned **String** value is determined from the following steps:

1. Let *tv* be the time value.
2. If *tv* is **NaN**, return the string "NaN".
3. Let *t* be **LocalTime**(*tv*).
4. Using *t*, create a string value that has the following format, according to the variables that are defined in the table in section 2.1.18 of this document:

DDDbMMMbddbyyyyy

5. Return *Result(4)*.

2.2.5 [ECMA-262/5] Section 15.9.5.4, Date.prototype.toTimeString ()

C0003:

The specification states:

```
This function returns a String value. The contents of the String are implementation-
dependent, but are intended to represent the "time" portion of the Date in the current time
zone in a convenient, human-readable form.
```

IE9 Mode, IE10 Mode, IE11 Mode, and EdgeHTML Mode (All Versions)

The returned **String** value is determined from the following steps:

1. Let *tv* be the time value.
2. If *tv* is **NaN**, return the string "`NaN`".
3. Let *t* be **LocalTime**(*tv*).
4. Using *t*, create a string value that has the following format, according to the items that are defined in section 2.1.18 of this document:

   hh:mm:ssbzzzzzz

5. Return *Result(4)*.

2.2.6 [ECMA-262/5] Section 15.9.5.5, Date.prototype.toLocaleString ()

C0004:

The specification states:

```
This function returns a String value. The contents of the String are implementation-
dependent, but are intended to represent the Date in the current time zone in a convenient,
human-readable form that corresponds to the conventions of the host environment’s current
locale.
```

IE9 Mode, IE10 Mode, IE11 Mode, and EdgeHTML Mode (All Versions)

The returned **String** value is determined from the following steps:

1. Using the system locale settings, get the local time value that corresponds to the date value. Apply any appropriate civil time adjustments.
2. If the year of *Result(1)* is less than or equal 1600 or is greater than or equal to 10000, return the result of calling the standard built-in **Date.prototype.toString** method with *Result(1)* as its **this** object.
3. Use the **GetDateFormat** Microsoft Windows system function (http://msdn.microsoft.com/en-us/library/dd318086(VS.85).aspx), to format the date and time that correspond to *Result(1)*. Pass the default value of `DATE_LONGDATE` for format flags. However, if the current locale’s language is Arabic or Hebrew, pass the value `DATE_LONGDATE | Date_RTLREADING` for format flags.

4. If the call in step 3 fails and the current locale language is Hebrew, throw a **RangeError** exception.

5. Use the **GetTimeFormat** Windows system function (http://msdn.microsoft.com/en-us/library/dd318130(VS.85).aspx) to format the date and time that correspond to *Result(1)*. Pass the default value of zero for format flags.

6. If the calls in steps 3 or 5 fail, return the result of calling the standard built-in **Date.prototype.toString** method with *Result(1)* as its **this** object.

7. Return the string value that is the result of concatenating *Result(3)*, a space character, and *Result(5)*.

2.2.7 [ECMA-262/5] Section 15.9.5.6, Date.prototype.toLocaleDateString ()

C0005:

The specification states:

```
This function returns a String value. The contents of the String are implementation-
dependent, but are intended to represent the "date" portion of the Date in the current time
zone in a convenient, human-readable form that corresponds to the conventions of the host
environment’s current locale.
```

IE9 Mode, IE10 Mode, IE11 Mode, and EdgeHTML Mode (All Versions)

The returned **String** value is determined from the following steps:

1. Using the system locale settings, get the local time value that corresponds to the date value. Apply any appropriate civil time adjustments.

2. If the year of *Result(1)* is less than or equal to 1600 or is greater than or equal to 10000, return the result of calling the standard built-in **Date.prototype.toString** method with *Result(1)* as its **this** object.

3. Use the **GetDateFormat** Microsoft Windows system function (http://msdn.microsoft.com/en-us/library/dd318086(VS.85).aspx) to format the date and time that correspond to *Result(1)*. Pass the default value of `DATE_LONGDATE` for format flags. However, if the current locale’s language is Arabic or Hebrew, pass the value `DATE_LONGDATE | Date_RTLREADING` for format flags.

4. If the call in step 3 fails and the current locale language is Hebrew, throw a **RangeError** exception. Go to step 6.

5. If the call in step 3 fails, return the result of calling the standard built-in **Date.prototype.toString** method with *Result(1)* as its **this** object.

6. Return the string value that is *Result(3)*.

2.2.8 [ECMA-262/5] Section 15.9.5.7, Date.prototype.toLocaleTimeString ()

C0006:

The specification states:

```
This function returns a String value. The contents of the String are implementation-
dependent, but are intended to represent the "time" portion of the Date in the current time
zone in a convenient, human-readable form that corresponds to the conventions of the host
environment’s current locale.
```

IE9 Mode, IE10 Mode, IE11 Mode, and EdgeHTML Mode (All Versions)

The returned **String** value is determined from the following steps:

1. Using the system locale settings, get the local time value that corresponds to the date value. Apply any appropriate civil time adjustments.
2. If the year of *Result(1)* is less than or equal to 1600 or is greater than or equal to 10000, return the result of calling the standard built-in **Date.prototype.toString** method with *Result(1)* as its **this** object.
3. Use the **GetTimeFormat** Microsoft Windows system function (http://msdn.microsoft.com/en-us/library/dd318130(VS.85).aspx) to format the date and time that correspond to *Result(1)*. Pass the default value of zero for format flags.
4. If the call in step 3 fails, return the result of calling the standard built-in **Date.prototype.toString** method with *Result(1)* as its **this** object.
5. Return the string value that is *Result(3)*.

2.2.9 [ECMA-262/5] Section 15.9.5.42, Date.prototype.toUTCString ()

C0007:

The specification states:

```
This function returns a String value. The contents of the String are implementation-
dependent, but are intended to represent the Date in a convenient, human-readable form in
UTC.
```

IE9 Mode, IE10 Mode, IE11 Mode, and EdgeHTML Mode (All Versions)

The returned **String** value is determined from the following steps:

1. Let *tv* be the time value.
2. If *tv* is **NaN**, return the string "`NaN`".
3. Using *tv*, create a string value that has the following format, according to the items that are defined in the table in section 2.1.18 of this document:

   DDD,bddbMMMbyyyyybhh:mm:ssbUTC
4. Return *Result(3)*.

2.3 Error Handling

There are no additional error handling considerations.

2.4 Security

There are no additional security considerations.

3 Change Tracking

No table of changes is available. The document is either new or has had no changes since its last release.

4 Index