[MS-ES2016]:

Microsoft Edge ECMA-262 ECMAScript Language Specification (7th 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.
- **License Programs**. To see all of the protocols in scope under a specific license program and the associated patents, visit the Patent Map.
- **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.

Support. For questions and support, please contact dochelp@microsoft.com.

Revision Summary

Date	Revision History	Revision Class	Comments
5/17/2017	1.0	New	Released new document.
10/3/2017	1.0	None	No changes to the meaning, language, or formatting of the technical content.

Table of Contents

1 Introduction

This document describes the level of support provided by Microsoft Edge for the ECMAScript® 2016 Language Specification, [ECMA-262/7], published June 2016.

This specification is the seventh edition of the ECMAScript Language Specification. Since publication of the first edition in 1997, ECMAScript has grown to be one of the most widely used general purpose programming languages. It is best known as the language embedded in web browsers but has also been widely adopted for server and embedded applications.

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-262/7] Ecma International, "ECMAScript® 2016 Language Specification", Standard ECMA-262 7th Edition / June 2016, https://go.microsoft.com/fwlink/p/?linkid=846935

[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

None.

1.3 Microsoft Implementations

The following Microsoft web browsers implement some portion of the [ECMA-262/7] specification:

- 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 supported by each browser version.

Browser Version	Document Modes Supported
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 three document modes in all browser versions that support these modes:

1.4 Standards Support Requirements

To conform to [ECMA-262/7], a user agent must implement all required portions of the specification. 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-262/7] and whether they are considered normative or informative.

Sections	Normative/Informative
1-6	Informative
7-26	Normative
Annex A	Informative
Annex B	Normative
Annex C, Annex D, Annex E, Annex F, Annex G	Informative

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, clarifications, and extensions for the Microsoft implementation of [ECMA-262/7].

- 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 describes extensions to the requirements.
- Section 2.4 considers error handling aspects of the implementation.
- Section 2.5 considers security aspects of the implementation.

2.1 Normative Variations

The following subsections describe normative variations from the MUST requirements of [ECMA-262/7].

2.1.1 [ECMA-262/7] Section 7.1.1 ToPrimitive (input [, PreferredType])

V0164: @@toPrimitive is not implemented

The specification states:

```
7.1.1 ToPrimitive ( input [, PreferredType] )

    The abstract operation ToPrimitive takes an input argument and an optional argument
    PreferredType. The abstract operation ToPrimitive converts its input argument to a
    non-Object type. If an object is capable of converting to more than one primitive
    type, it may use the optional hint PreferredType to favour that type. Conversion
    occurs according to Table 9:
    ...
    When Type(input) is Object, the following steps are taken:
        ...
        4.  Let exoticToPrim be ? GetMethod(input, @@toPrimitive).
```

EdgeHTML Mode

@@toPrimitive is not implemented.

2.1.2 [ECMA-262/7] Section 7.4.6 IteratorClose (iterator, completion)

V0187: IteratorClose is not correctly implemented

The specification states:

```
7.4.6 IteratorClose( iterator, completion )

    The abstract operation IteratorClose with arguments iterator and completion is used
    to notify an iterator that it should perform any actions it would normally perform
    when it has reached its completed state:

        1.  Assert: Type(iterator) is Object.
        2.  Assert: completion is a Completion Record.
        3.  Let return be ? GetMethod(iterator, "return").
```

```
4.  If return is undefined, return Completion(completion).
5.  Let innerResult be Call(return, iterator, « »).
6.  If completion.[[type]] is throw, return Completion(completion).
7.  If innerResult.[[type]] is throw, return Completion(innerResult).
8.  If Type(innerResult.[[value]]) is not Object, throw a TypeError exception.
9.  Return Completion(completion).
```

EdgeHTML Mode

IteratorClose is not correctly implemented. It behaves as follows:

7.4.6 IteratorClose(iterator, completion)

1. Assert: Type(iterator) is Object.
2. Assert: completion is a Completion Record.
3. Return Completion(completion).

2.1.3 [ECMA-262/7] Section 9.2.7 AddRestrictedFunctionProperties (F, realm)

V0188: The caller and arguments properties are set incorrectly

The specification states:

```
9.2.7 AddRestrictedFunctionProperties ( F, realm )

    The abstract operation AddRestrictedFunctionProperties is called with a function
    object F and Realm Record realm as its argument. It performs the following steps:
    ...
    3.  Perform ! DefinePropertyOrThrow(F, "caller", PropertyDescriptor {[[Get]]:
        thrower, [[Set]]: thrower, [[Enumerable]]: false, [[Configurable]]: true}).
    4.  Return ! DefinePropertyOrThrow(F, "arguments", PropertyDescriptor {[[Get]]:
        thrower, [[Set]]: thrower, [[Enumerable]]: false, [[Configurable]]: true}).
```

EdgeHTML Mode

The `caller` and `arguments` properties are set incorrectly:

3. Perform ! DefinePropertyOrThrow(F, "caller", PropertyDescriptor {[[Get]]:

   thrower, [[Set]]: undefined, [[Enumerable]]: false, [[Configurable]]: false}).

4. Return ! DefinePropertyOrThrow(F, "arguments", PropertyDescriptor {[[Get]]: \

   thrower, [[Set]]: undefined, [[Enumerable]]: false, [[Configurable]]: false}).

2.1.4 [ECMA-262/7] Section 11.8.6 Template Literal Lexical Components

V0040: The escape sequence \0 is treated as a legacy octal escape sequence and a SyntaxError is thrown

The specification states:

```
11.8.6 Template Literal Lexical Components
```

```
Syntax

    Template ::
        NoSubstitutionTemplate
        TemplateHead

    NoSubstitutionTemplate ::
        ` TemplateCharactersopt `

    TemplateHead ::
        ` TemplateCharactersopt ${

    TemplateSubstitutionTail ::
        TemplateMiddle
        TemplateTail

    TemplateMiddle ::
        } TemplateCharactersopt ${

    TemplateTail ::
        } TemplateCharactersopt `

    TemplateCharacters ::
        TemplateCharacter TemplateCharactersopt

    TemplateCharacter ::
        $ [lookahead ≠ { ]
        \ EscapeSequence
        LineContinuation
        LineTerminatorSequence
        SourceCharacter but not one of ` or \ or $ or LineTerminator

A conforming implementation must not use the extended definition of EscapeSequence
described in B.1.2 when parsing a TemplateCharacter.

NOTE  TemplateSubstitutionTail is used by the InputElementTemplateTail alternative
lexical goal.
```

EdgeHTML Mode

The escape sequence \0 is treated as a legacy octal escape sequence and a **SyntaxError** is thrown; instead it should be translated into a null character.

2.1.5 [ECMA-262/7] Section 11.9.1 Rules of Automatic Semicolon Insertion

V0041: Automatic semicolon insertion is not applied to yield* productions

The specification states:

```
11.9.1 Rules of Automatic Semicolon Insertion

    In the following rules, "token" means the actual recognized lexical token determined
    using the current lexical goal symbol as described in clause 11.

    There are three basic rules of semicolon insertion:

        1.  When, as a Script or Module is parsed from left to right, a token (called the
            offending token) is encountered that is not allowed by any production of the
            grammar, then a semicolon is automatically inserted before the offending
            token if one or more of the following conditions is true:

            • The offending token is separated from the previous token by at least one
```

```
        LineTerminator.
        • The offending token is }.
        • The previous token is ) and the inserted semicolon would then be parsed as
        the terminating semicolon of a do-while statement (13.7.2).

    2.  When, as the Script or Module is parsed from left to right, the end of the
        input stream of tokens is encountered and the parser is unable to parse the
        input token stream as a single complete ECMAScriptScript or Module, then a
        semicolon is automatically inserted at the end of the input stream.

    3.  When, as the Script or Module is parsed from left to right, a token is
        encountered that is allowed by some production of the grammar, but the
        production is a restricted production and the token would be the first token
        for a terminal or nonterminal immediately following the annotation "[no
        LineTerminator here]" within the restricted production (and therefore such a
        token is called a restricted token), and the restricted token is separated
        from the previous token by at least one LineTerminator, then a semicolon is
        automatically inserted before the restricted token.
```

EdgeHTML Mode

Rule 3 is not applied to yield* productions.

```
var obj = {
  *g() {
  yield
  * 1
  }
};
```

A semicolon should be inserted in the yield* production as follows:

```
yield;*1
```

This would throw a **SyntaxError**.

2.1.6 [ECMA-262/7] Section 12.4.4.1 Runtime Semantics: Evaluation

V0189: The reference is retrieved twice

The specification states:

```
12.4.4.1 Runtime Semantics: Evaluation

    UpdateExpression : LeftHandSideExpression ++

        1.  Let lhs be the result of evaluating LeftHandSideExpression.
        2.  Let oldValue be ? ToNumber(? GetValue(lhs)).
        3.  Let newValue be the result of adding the value 1 to oldValue, using the same
            rules as for the + operator (see 12.8.5).
        4.  Perform ? PutValue(lhs, newValue).
        5.  Return oldValue.
```

EdgeHTML Mode

Between steps 2 and 3, the following steps are added:

a. If Type(lhs) is a Reference and if IsUnresolvableReference(_lhs_) is false and IsPropertyReference(_lhs_) is false:

1. Assert: lhs is a reference to an Environment Record.

2. Let hs be the result of evaluating an Identifier _id_ whose StringValue is GetReferencedName(lhs) as if _id_ were a LeftHandSideExpression.

3. ReturnIfAbrupt(lhs);

As a result, the reference is retrieved twice.

2.1.7 [ECMA-262/7] Section 12.4.5.1 Runtime Semantics: Evaluation

V0190: The reference is retrieved twice

The specification states:

```
12.4.5.1 Runtime Semantics: Evaluation

    UpdateExpression : LeftHandSideExpression --

        1.  Let lhs be the result of evaluating LeftHandSideExpression.
        2.  Let oldValue be ? ToNumber(GetValue(lhs)).
        3.  Let newValue be the result of subtracting the value 1 from oldValue, using
            the same rules as for the - operator (see 12.8.5).
        4.  Perform ? PutValue(lhs, newValue).
        5.  Return oldValue.
```

EdgeHTML Mode

Between steps 2 and 3 the following steps are added:

a. If Type(lhs) is a Reference and if IsUnresolvableReference(_lhs_) is false and IsPropertyReference(_lhs_) is false:

1. Assert: lhs is a reference to an Environment Record.

2. Let hs be the result of evaluating an Identifier _id_ whose StringValue is GetReferencedName(lhs) as if _id_ were a LeftHandSideExpression.

3. ReturnIfAbrupt(lhs);

As a result, the reference is retrieved twice.

2.1.8 [ECMA-262/7] Section 12.4.6.1 Runtime Semantics: Evaluation

V0191: The reference is returned twice

The specification states:

```
12.4.6.1 Runtime Semantics: Evaluation

    UpdateExpression : ++ UnaryExpression
```

```
1.  Let expr be the result of evaluating UnaryExpression.
2.  Let oldValue be ? ToNumber(? GetValue(expr)).
3.  Let newValue be the result of adding the value 1 to oldValue, using the same
    rules as for the + operator (see 12.8.5).
5.  Perform ? PutValue(expr, newValue).
5.  Return newValue.
```

EdgeHTML Mode

Between steps 2 and 3 the following steps are added:

a. If Type(expr) is a Reference and if IsUnresolvableReference(_expr_) is false:

1. Assert: expr is a reference to an Environment Record.

2. Let hs be the result of evaluating an Identifier _id_ whose StringValue is GetReferencedName(expr) as if _id_ were a LeftHandSideExpression.

3. ReturnIfAbrupt(expr);

As a result, the reference is returned twice.

2.1.9 [ECMA-262/7] Section 12.4.7.1 Runtime Semantics: Evaluation

V0192: The reference is retrieved twice

The specification states:

```
12.4.7.1 Runtime Semantics: Evaluation

    UpdateExpression : -- UnaryExpression

        1.  Let expr be the result of evaluating UnaryExpression.
        2.  Let oldValue be ? ToNumber(? GetValue(expr)).
        3.  Let newValue be the result of subtracting the value 1 from oldValue, using
            the same rules as for the - operator (see 12.8.5).
        4.  Perform ? PutValue(expr, newValue).
        5.  Return newValue.
```

EdgeHTML Mode

Between steps 3 and 4 the following steps are added:

a. If Type(expr) is Reference and if IsUnresolvableReference(_expr_) is false and IsPropertyReference(_expr_) is false then

1. Assert: expr is a reference to an Environment Record.

2. Let expr be the result of evaluating an Identifier _id_ whose StringValue is GetReferencedName(expr) as if _id_ were a LeftHandSideExpression.

3. ReturnIfAbrupt(expr);

As a result, the reference is retrieved twice.

2.1.10 [ECMA-262/7] Section 12.10.4 Runtime Semantics: InstanceofOperator(O, C)

V0193: The abstract operation InstanceofOperator(O, C) is not implemented

The specification states:

```
12.10.4 Runtime Semantics: InstanceofOperator(O, C)

    The abstract operation InstanceofOperator(O, C) implements the generic algorithm for
    determining if an object O inherits from the inheritance path defined by constructor
    C. This abstract operation performs the following steps:

        1.  If Type(C) is not Object, throw a TypeError exception.
        2.  Let instOfHandler be ? GetMethod(C,@@hasInstance).
        3.  If instOfHandler is not undefined, then
            a.  Return ToBoolean(? Call(instOfHandler, C, «O»)).
        5.  If IsCallable(C) is false, throw a TypeError exception.
        6.  Return OrdinaryHasInstance(C, O).
```

EdgeHTML Mode

The abstract operation InstanceofOperator(O, C) is not implemented.

2.1.11 [ECMA-262/7] Section 12.15.4 Runtime Semantics: Evaluation

V0194: After an assignment, the name of the function is the empty string

The specification states:

```
12.15.4 Runtime Semantics: Evaluation

    AssignmentExpression : LeftHandSideExpression = AssignmentExpression

        1.  If LeftHandSideExpression is neither an ObjectLiteral nor an ArrayLiteral,
            then
            a.  Let lref be the result of evaluating LeftHandSideExpression.
            b.  ReturnIfAbrupt(lref).
            c.  Let rref be the result of evaluating AssignmentExpression.
            d.  Let rval be ? GetValue(rref).
            e.  If IsAnonymousFunctionDefinition(AssignmentExpression) and
            IsIdentifierRef of LeftHandSideExpression are both true, then
                i.  Let hasNameProperty be ? HasOwnProperty(rval, "name").
                ii. If hasNameProperty is false, perform SetFunctionName(rval,
                GetReferencedName(lref)).
```

EdgeHTML Mode

After the following assignment:

```
var f = function () {}
```

the name of the function held in f is the empty string.

V0195: The reference is retrieved twice

The specification states:

```
12.15.4 Runtime Semantics: Evaluation

    AssignmentExpression : LeftHandSideExpression = AssignmentExpression

        1.  If LeftHandSideExpression is neither an ObjectLiteral nor an ArrayLiteral,
            then
            a.  Let lref be the result of evaluating LeftHandSideExpression.
            b.  ReturnIfAbrupt(lref).
            c.  Let rref be the result of evaluating AssignmentExpression.
            d.  Let rval be ? GetValue(rref).
            e.  If IsAnonymousFunctionDefinition(AssignmentExpression) and
            IsIdentifierRef of LeftHandSideExpression are both true, then
                i.  Let hasNameProperty be ? HasOwnProperty(rval, "name").
                ii. If hasNameProperty is false, perform SetFunctionName(rval,
                GetReferencedName(lref)).
            f.  Perform ? PutValue(lref, rval).
            g.  Return rval.
        ...

    AssignmentExpression : LeftHandSideExpression AssignmentOperator AssignmentExpression

        1.  Let lref be the result of evaluating LeftHandSideExpression.
        2.  Let lval be ? GetValue(lref).
        3.  Let rref be the result of evaluating AssignmentExpression.
        4.  Let rval be ? GetValue(rref).
        5.  Let op be the @ where AssignmentOperator is @=
        6.  Let r be the result of applying op to lval and rval as if evaluating the
            expression lval op rval.
        7.  Perform ? PutValue(lref, r).
        8.  Return r.
```

EdgeHTML Mode

In the algorithm for

AssignmentExpression : LeftHandSideExpression = AssignmentExpression

the following steps are added before step1f:

i. Type(lref) is Reference and if IsUnresolvableReference(_lref_) is false and IsPropertyReference(_lref_) is false then

1. Assert: lref is a reference to an Environment Record.

2. Let lref be the result of evaluating an Identifier _id_ whose StringValue is GetReferencedName(lref) as if _id_ were a LeftHandSidelrefession.

3. ReturnIfAbrupt(lref);

As a result, the reference is retrieved twice.

In the algorithm for

AssignmentIrefession : LeftHandSideIrefession AssignmentOperator AssignmentIrefession

the following steps are added between steps 4 and 5:

a. Type(lref) is Reference and if IsUnresolvableReference(_lref_) is false and IsPropertyReference(_lref_) is false then

i. Assert: lref is a reference to an Environment Record.

ii. Let lref be the result of evaluating an Identifier _id_ whose StringValue is GetReferencedName(lref) as if _id_ were a LeftHandSidelrefession.

iii. ReturnIfAbrupt(lref);

As a result, the reference is retrieved twice.

2.1.12 [ECMA-262/7] Section 13 ECMAScript Language: Statements and Declarations

V0056: HoistableDeclaration is treated as a production of Statement, not Declaration

The specification states:

```
13  ECMAScript Language: Statements and Declarations

    Statement[Yield, Return]  :
        BlockStatement[?Yield, ?Return]
        ...
        DebuggerStatement

    Declaration[Yield]  :
        HoistableDeclaration[?Yield]
        ClassDeclaration[?Yield]
        LexicalDeclaration[In, ?Yield]

    HoistableDeclaration[Yield, Default]  :
        FunctionDeclaration[?Yield,?Default]
        GeneratorDeclaration[?Yield, ?Default]
```

EdgeHTML Mode

HoistableDeclaration is treated as a production of *Statement*, not *Declaration*.

Statement[Yield, Return] :

BlockStatement[?Yield, ?Return]

...

DebuggerStatement

HoistableDeclaration[?Yield]

Declaration[Yield] :

ClassDeclaration[?Yield]

LexicalDeclaration[In, ?Yield]

HoistableDeclaration[Yield, Default] :

FunctionDeclaration[?Yield,?Default]

GeneratorDeclaration[?Yield, ?Default]

2.1.13 [ECMA-262/7] Section 13.2.1 Static Semantics: Early Errors

V0057: No error is issued if an element of LexicallyDeclaredNames also occurs in VarDeclaredNames

The specification states:

```
13.2.1 Static Semantics: Early Errors

    Block : { StatementList }

        • It is a Syntax Error if the LexicallyDeclaredNames of StatementList contains
        any duplicate entries.
        • It is a Syntax Error if any element of the LexicallyDeclaredNames of
        StatementList also occurs in the VarDeclaredNames of StatementList.
```

EdgeHTML Mode

No error is issued if an element of LexicallyDeclaredNames also occurs in VarDeclaredNames. For example:

```
{
  let x;
  var x;  // should be a syntax error but is not
}
```

V0058: Functions and generator functions are allowed to have duplicates in LexicallyDeclaredNames

The specification states:

```
13.2.1 Static Semantics: Early Errors

    Block : { StatementList }

        • It is a Syntax Error if the LexicallyDeclaredNames of StatementList contains
        any duplicate entries.
        • It is a Syntax Error if any element of the LexicallyDeclaredNames of
        StatementList also occurs in the VarDeclaredNames of StatementList.
```

EdgeHTML Mode

Functions and generator functions are allowed to have duplicates in LexicallyDeclaredNames.

2.1.14 [ECMA-262/7] Section 13.7.4.1 Static Semantics: Early Errors

V0061: It is not a Syntax Error for BoundNames of LexicalDeclaration to contain let or const

The specification states:

```
13.7.4.1 Static Semantics: Early Errors

    IterationStatement : for ( LexicalDeclaration Expression; Expression ) Statement

        • It is a Syntax Error if any element of the BoundNames of LexicalDeclaration
        also occurs in the VarDeclaredNames of Statement.
```

EdgeHTML Mode

It is not a Syntax Error for BoundNames of *LexicalDeclaration* to contain **let** or **const**.

2.1.15 [ECMA-262/7] Section 13.7.5.1 Static Symantics: Early Errors

V0129: It is not a Syntax Error if an element of the BoundNames of ForDeclaration also occurs in the VarDeclaredNames of Statement

The specification states:

```
13.7.5.1 Static Semantics: Early Errors
    ...
            IterationStatement :
                for ( ForDeclaration in Expression ) Statement
                for ( ForDeclaration of AssignmentExpression ) Statement

        • It is a Syntax Error if the BoundNames of ForDeclaration contains "let".
        • It is a Syntax Error if any element of the BoundNames of ForDeclaration also
        occurs in the VarDeclaredNames of Statement.
        • It is a Syntax Error if the BoundNames of ForDeclaration contains any duplicate
        entries.
```

EdgeHTML Mode

It is not a Syntax Error if an element of the BoundNames of *ForDeclaration* also occurs in the VarDeclaredNames of *Statement*.

2.1.16 [ECMA-262/7] Section 13.7.5.12 Runtime Semantics: ForIn/OfHeadEvaluation (TDZnames, expr, iterationKind)

V0208: ForIn/OfHeadEvaluation does not return an AbruptCompletion when exprValue.[[value]] is null or undefined

The specification states:

```
13.7.5.12 Runtime Semantics: ForIn/OfHeadEvaluation ( TDZnames, expr, iterationKind)

    The abstract operation ForIn/OfHeadEvaluation is called with arguments TDZnames,
    expr, and iterationKind. The value of iterationKind is either enumerate or iterate.
        ...
        6.  If iterationKind is enumerate, then
            a.  If exprValue.[[value]] is null or undefined, then
                i.  Return Completion{[[type]]: break, [[value]]: empty, [[target]]:
                empty}.
            b.  Let obj be ToObject(exprValue).
            c.  Return ? EnumerateObjectProperties(obj).
        7.  Else,
            a.  Assert: iterationKind is iterate.
            b.  Return ? GetIterator(exprValue).
```

EdgeHTML Mode

Logic in the If branch is also executed in the Else branch:

7. Else,

   -. If exprValue.[[value]] is null or undefined, then

      i. Return Completion{[[type]]: break, [[value]]: empty, [[target]]: empty}.

a. Assert: iterationKind is iterate.

b. Return ? GetIterator(exprValue).

Therefore ForIn/OfHeadEvaluation does not return an abrupt completion for *iterationKind* is iterate when *exprValue*.`[[value]]` is null or undefined. For example, the following statements do not throw errors:

```
for (let x of null) {}

for (let x of undefined) {}
```

2.1.17 [ECMA-262/7] Section 13.13 Labelled Statements

V0062: The LabelledItem production replaces FunctionDeclaration with Declaration

The specification states:

```
13.13 Labelled Statements

    Syntax

        LabelledStatement[Yield, Return] :
            LabelIdentifier[?Yield] : LabelledItem[?Yield, ?Return]

        LabelledItem[Yield, Return] :
            Statement[?Yield, ?Return]
            FunctionDeclaration[?Yield]
```

EdgeHTML Mode

The *LabelledItem* production replaces *FunctionDeclaration* with *Declaration*.

```
LabelledItem[Yield, Return] :

  Statement[?Yield, ?Return]

  Declaration[?Yield]
```

2.1.18 [ECMA-262/7] Section 14.1.2 Static Semantics: Early Errors

V0063: The LexicallyDeclaredNames of FunctionStatementList may have duplicate function and generator function entries

The specification states:

```
14.1.2 Static Semantics: Early Errors
    ...
    FunctionBody : FunctionStatementList

        • It is a Syntax Error if the LexicallyDeclaredNames of FunctionStatementList
        contains any duplicate entries.
        • It is a Syntax Error if any element of the LexicallyDeclaredNames of
        FunctionStatementList also occurs in the VarDeclaredNames of
        FunctionStatementList.
        • It is a Syntax Error if ContainsDuplicateLabels of FunctionStatementList with
        argument « » is true.
```

```
• It is a Syntax Error if ContainsUndefinedBreakTarget of FunctionStatementList
with argument « » is true.
• It is a Syntax Error if ContainsUndefinedContinueTarget of
FunctionStatementList with arguments « » and « » is true.
```

EdgeHTML Mode

The LexicallyDeclaredNames of *FunctionStatementList* may have duplicate function and generator function entries.

2.1.19 [ECMA-262/7] Section 14.3.8 Runtime Semantics: DefineMethod

V0066: Object literal methods are created with a [[Construct]] slot

The specification states:

```
14.3.8 Runtime Semantics: DefineMethod

    With parameters object and optional parameter functionPrototype.

    MethodDefinition : PropertyName ( StrictFormalParameters ) { FunctionBody }

        1. Let propKey be the result of evaluating PropertyName.
        2. ReturnIfAbrupt(propKey).
        3. If the function code for this MethodDefinition is strict mode code, let
           strict be true. Otherwise let strict be false.
        4. Let scope be the running execution context's LexicalEnvironment.
        5. If functionPrototype was passed as a parameter, let kind be Normal; otherwise
           let kind be Method.
        6. Let closure be FunctionCreate(kind, StrictFormalParameters, FunctionBody,
           scope, strict). If functionPrototype was passed as a parameter then pass its
           value as the functionPrototype optional argument of FunctionCreate.
        7. Perform MakeMethod(closure, object).
        8. Return the Record{[[key]]: propKey, [[closure]]: closure}.
```

EdgeHTML Mode

Object literal methods are created with a `[[Construct]]` slot, contrary to DefineMethod. Therefore the methods can successfully be used as the target of **new** expressions. In the following example, the **new** expression should throw a **TypeError**, but doesn't.

```
var obj = { meth() { } };

new obj.meth();
```

V0067: Methods defined in object literals are created with their own property named prototype

The specification states:

```
14.3.8 Runtime Semantics: DefineMethod

    With parameters object and optional parameter functionPrototype.

    MethodDefinition : PropertyName ( StrictFormalParameters ) { FunctionBody }

        1.  Let propKey be the result of evaluating PropertyName.
        2.  ReturnIfAbrupt(propKey).
```

```
3.  If the function code for this MethodDefinition is strict mode code, let
    strict be true. Otherwise let strict be false.
4.  Let scope be the running execution context's LexicalEnvironment.
5.  If functionPrototype was passed as a parameter, let kind be Normal; otherwise
    let kind be Method.
6.  Let closure be FunctionCreate(kind, StrictFormalParameters, FunctionBody,
    scope, strict). If functionPrototype was passed as a parameter then pass its
    value as the functionPrototype optional argument of FunctionCreate.
7.  Perform MakeMethod(closure, object).
8.  Return the Record{[[key]]: propKey, [[closure]]: closure}.
```

EdgeHTML Mode

Methods defined in object literals are created with their own property named `prototype`, contrary to DefineMethod. In the following example, **false** should be logged, but instead **true** is.

```
var obj = { method() { } };

console.log(Object.hasOwnProperty(obj.method, 'property'));
```

2.1.20 [ECMA-262/7] Section 14.5.14 Runtime Semantics: ClassDefinitionEvaluation

V0021: ClassDefinitionEvaluation uses the lexical environment of the running execution context

The specification states:

```
14.5.14 Runtime Semantics: ClassDefinitionEvaluation

    With parameter className.

    ClassTail : ClassHeritage { ClassBody }

        1.  Let lex be the LexicalEnvironment of the running execution context.
        2.  Let classScope be NewDeclarativeEnvironment(lex).
        3.  Let classScopeEnvRec be classScope's EnvironmentRecord.
        4.  If className is not undefined, then
            a.  Perform classScopeEnvRec.CreateImmutableBinding(className, true).
        ...
        23. If className is not undefined, then
            a.  Perform classScopeEnvRec.InitializeBinding(className, F).
```

EdgeHTML Mode

Step 2 is omitted; therefore ClassDefinitionEvaluation uses the lexical environment of the running execution context.

2.1.21 [ECMA-262/7] Section 15.1.1 Static Semantics: Early Errors

V0069: Duplicate function and generator function entries are allowed in LexicallyDeclaredNames of ScriptBody

The specification states:

```
15.1.1 Static Semantics: Early Errors

    Script : ScriptBody
```

```
    • It is a Syntax Error if the LexicallyDeclaredNames of ScriptBody contains any
    duplicate entries.
    • It is a Syntax Error if any element of the LexicallyDeclaredNames of ScriptBody
    also occurs in the VarDeclaredNames of ScriptBody.
```

EdgeHTML Mode

Duplicate function and generator function entries are allowed in LexicallyDeclaredNames of *ScriptBody*.

2.1.22 [ECMA-262/7] Section 16.2 Forbidden Extensions

V0025: Functions created using the bind method are given caller and arguments restricted own properties

The specification states:

```
... Forbidden Extensions

    An implementation must not extend this specification in the following ways:

        • Other than as defined in this specification, ECMAScript Function objects
        defined using syntactic constructors in strict mode code must not be created with
        own properties named "caller" or "arguments" other than those that are created by
        applying the AddRestrictedFunctionProperties abstract operation to the function.
        Such own properties also must not be created for function objects defined using
        an ArrowFunction, MethodDefinition, GeneratorDeclaration, GeneratorExpression,
        ClassDeclaration, or ClassExpression regardless of whether the definition is
        contained in strict mode code. Built-in functions, strict mode functions created
        using the Function constructor, generator functions created using the Generator
        constructor, and functions created using the bind method also must not be created
        with such own properties.
```

EdgeHTML Mode

Functions created using the `bind` method are given `caller` and `arguments` restricted own properties.

2.1.23 [ECMA-262/7] Section 19.1.2.18 Object.setPrototypeOf (O, proto)

V0196: Object.setPrototypeOf throws an error immediately if parameter O is not an object

The specification states:

```
19.1.2.18 Object.setPrototypeOf ( O, proto )

    When the setPrototypeOf function is called with arguments O and proto, the following
    steps are taken:

        1. Let O be ? RequireObjectCoercible(O).
        2. If Type(proto) is neither Object nor Null, throw a TypeError exception.
        3. If Type(O) is not Object, return O.
        4. Let status be ? O.[[SetPrototypeOf]](proto).
        5. If status is false, throw a TypeError exception.
        6. Return O.
```

EdgeHTML Mode

ToObject(*O*) is done instead of RequireObjectCoercible(*O*) in step 1. As a result, `Object.setPrototypeOf` throws an error immediately if parameter *O* is not an object.

2.1.24 [ECMA-262/7] Section 19.1.3.2 Object.prototype.hasOwnProperty (V)

V0197: An error is thrown if the argument is a symbol

The specification states:

```
19.1.3.2 Object.prototype.hasOwnProperty ( V )

    When the hasOwnProperty method is called with argument V, the following steps are
    taken:

        1.  Let P be ? ToPropertyKey(V).
        2.  Let O ? be ToObject(this value).
        3.  Return ? HasOwnProperty(O, P).
```

EdgeHTML Mode

In step 1, ToString is invoked instead of ToPropertyKey. Because of this, an error is thrown if *V* is a symbol.

2.1.25 [ECMA-262/7] Section 19.1.3.5 Object.prototype.toLocaleString ([reserved1 [, reserved2]])

V0198: Object.prototype.toLocaleString passes ToObject(this) to the toString method instead of this

The specification states:

```
19.1.3.5 Object.prototype.toLocaleString ( [ reserved1 [ , reserved2 ] ] )

    When the toLocaleString method is called, the following steps are taken:

        1.  Let O be the this value.
        2.  Return ? Invoke(O, "toString").
```

EdgeHTML Mode

`Object.prototype.toLocaleString` passes ToObject(*this*) to the toString method instead of *this*. These are the steps:

1. Let O be the this value.
2. Let obj be ? ToObject(O).
3. Return ToString(obj).

2.1.26 [ECMA-262/7] Section 19.1.3.6 Object.prototype.toString ( )

V0199: @@toStringTag is not implemented

The specification states:

```
19.1.3.6 Object.prototype.toString ( )

    When the toString method is called, the following steps are taken:

        1.  ...
        ...
        15. Let tag be ? Get (O, @@toStringTag).
```

EdgeHTML Mode

@@toStringTag is not implemented.

2.1.27 [ECMA-262/7] Section 19.2.3.2 Function.prototype.bind (thisArg, ...args)

V0200: The bound function name accessor calls the target function's counterpart

The specification states:

```
19.2.3.2 Function.prototype.bind ( thisArg , ...args)

    When the bind method is called with argument thisArg and zero or more args, it
    performs the following steps:

        1.  Let Target be the this value.
        ...
        9.  Let targetName be ? Get(Target, "name").
        10. If Type(targetName) is not String, let targetName be the empty string.
        11. Perform SetFunctionName(F, targetName, "bound").
        12. Return F.
```

EdgeHTML Mode

Steps 9 to 11 are replaced by:

9. Let getName(Target) be a new dynamic function that does following:
    a. Let targetName be ? Get(Target, "name").
    b. Return "bound"+targetName
10. Set (F, "name", getName)

Because of this, the bound function name accessor calls the target function's counterpart. Note that steps 10 and 11 are deleted.

2.1.28 [ECMA-262/7] Section 19.2.3.6 Function.prototype [@@hasInstance] (V)

V0209: Calling @@hasInstance has no effect

The specification states:

```
19.2.3.6 Function.prototype[@@hasInstance] ( V )

    When the @@hasInstance method of an object F is called with value V, the following
    steps are taken:
        1.  Let F be the this value.
        2.  Return ? OrdinaryHasInstance(F, V).

    The value of the name property of this function is "[Symbol.hasInstance]".

    This property has the attributes { [[Writable]]: false, [[Enumerable]]: false,
    [[Configurable]]: false }.
    ...
    This property is non-writable and non-configurable to prevent tampering that could be
    used to globally expose the target function of a bound function.
```

EdgeHTML Mode

Calling @@hasInstance has no effect.

2.1.29 [ECMA-262/7] Section 19.2.4.1 length

V0074: The [[writable]] attribute of the length property cannot be set to true, regardless of the setting of [[configurable]]

The specification states:

```
19.2.4.1 length

    The value of the length property is an integer that indicates the typical number of
    arguments expected by the function. However, the language permits the function to be
    invoked with some other number of arguments. The behaviour of a function when invoked
    on a number of arguments other than the number specified by its length property
    depends on the function. This property has the attributes { [[Writable]]: false,
    [[Enumerable]]: false, [[Configurable]]: true }.
```

EdgeHTML Mode

The `[[writable]]` attribute of the `length` property cannot be set to **true**, regardless of the setting of `[[configurable]]`. No error is thrown on an attempt to set it **true**.

2.1.30 [ECMA-262/7] Section 19.4.2 Properties of the Symbol Constructor

V0161: Some properties of the Symbol constructor are not implemented

The specification states:

```
19.4.2 Properties of the Symbol Constructor

    The value of the [[Prototype]] internal slot of the Symbol constructor is the
    intrinsic object %FunctionPrototype% ... .

    ... The Symbol constructor has the following properties:
```

EdgeHTML Mode

These properties of the `Symbol` constructor are not implemented:

hasInstance

isConcatSpreadable

toPrimitive

toStringTag

2.1.31 [ECMA-262/7] Section 19.4.3.4 Symbol.prototype [@@toPrimitive] (hint)

V0178: Symbol.prototype[@@toPrimitive] is not implemented because @@toPrimitive is not implemented

The specification states:

```
19.4.3.4 Symbol.prototype [ @@toPrimitive ] ( hint )

    This function is called by ECMAScript language operators to convert a Symbol object
    to a primitive value. The allowed values for hint are "default", "number",
    and "string".

    When the @@toPrimitive method is called with argument hint, the following steps are
    taken:

        1.  Let s be the this value.
        2.  If Type(s) is Symbol, return s.
        3.  If Type(s) is not Object, throw a TypeError exception.
        4.  If s does not have a [[SymbolData]] internal slot, throw
            a TypeError exception.
        5.  Return the value of s's [[SymbolData]] internal slot.

    The value of the name property of this function is "[Symbol.toPrimitive]".

    This property has the attributes { [[Writable]]: false, [[Enumerable]]: false,
    [[Configurable]]: true }.
```

EdgeHTML Mode

`Symbol.prototype`[@@toPrimitive] is not implemented because @@toPrimitive is not implemented.

2.1.32 [ECMA-262/7] Section 19.4.3.5 Symbol.prototype [@@toStringTag]

V0179: Symbol.prototype[@@toStringTag] is not implemented because the @@toStringTag feature is not implemented

The specification states:

```
19.4.3.5 Symbol.prototype [ @@toStringTag ]

    The initial value of the @@toStringTag property is the String value "Symbol".

    This property has the attributes { [[Writable]]: false, [[Enumerable]]: false,
    [[Configurable]]: true }.
```

EdgeHTML Mode

`Symbol.prototype`[@@toStringTag] is not implemented because the @@toStringTag feature is not implemented.

2.1.33 [ECMA-262/7] Section 19.5.3 Properties of the Error Prototype Object

V0182: The error prototype object is the intrinsic object %Error%

The specification states:

```
19.5.3 Properties of the Error Prototype Object

    The Error prototype object is the intrinsic object %ErrorPrototype%. The Error
    prototype object is an ordinary object. It is not an Error instance and does not have
    an [[ErrorData]] internal slot.

    The value of the [[Prototype]] internal slot of the Error prototype object is the
    intrinsic object %ObjectPrototype%.
```

EdgeHTML Mode

The `Error` prototype object is the intrinsic object `%Error%`. It is an `Error` object. It is not an `Error` instance and does have an `[[ErrorData]]` internal slot.

2.1.34 [ECMA-262/7] Section 20.3.1.15 TimeClip (time)

V0201: TimeClip does not convert negative zero to positive zero

The specification states:

```
20.3.1.15 TimeClip (time)

    The abstract operation 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 × 10^15, return NaN.
        3.  Let clippedTime be ToInteger(time).
        4.  If clippedTime is -0, let clippedTime be +0.
        5.  Return clippedTime.
```

EdgeHTML Mode

TimeClip does not convert negative zero to positive zero (step 4).

2.1.35 [ECMA-262/7] Section 20.3.1.16 Date Time String Format

V0125: A date-time without a time zone offset is interpreted incorrectly

The specification states:

```
20.3.1.16 Date Time String Format

    ECMAScript defines a string interchange format for date-times based upon a
    simplification of the ISO 8601 Extended Format. The format is as follows:
    YYYY-MM-DDTHH:mm:ss.sssZ

    Where the fields are as follows:

        YYYY is the decimal digits of the year 0000 to 9999 in the Gregorian calendar.
        ...
        Z is the time zone offset specified as "Z" (for UTC) or either "+" or "-"
        followed by a time expression HH:mm
```

EdgeHTML Mode

When the date-time string does not include a time zone offset, the time is taken, incorrectly, to be UTC, not local time. For example, if the date-time string is "2015-10-01", it is taken to mean:

Wed Sep 30 2015 17:00:00 GMT-0700 (Pacific Daylight Time)

According to the specification, it should be taken as:

Thu Oct 01 2015 00:00:00 GMT-0700 (Pacific Daylight Time)

2.1.36 [ECMA-262/7] Section 20.3.4 Properties of the Date Prototype Object

V0183: The Date prototype object is a Date instance and has a [[DateValue]] internal slot

The specification states:

```
20.3.4 Properties of the Date Prototype Object

    The Date prototype object is the intrinsic object %DatePrototype%. The Date prototype
    object is itself an ordinary object. It is not a Date instance and does not have a
    [[DateValue]] internal slot.
```

EdgeHTML Mode

The `Date` prototype object is a `Date` instance and has a `[[DateValue]]` internal slot.

2.1.37 [ECMA-262/7] Section 21.1.3 Properties of the String Prototype Object

V0210: String.prototype is a string instance object, not an ordinary object

The specification states:

```
21.1.3 Properties of the String Prototype Object

    The String prototype object is the intrinsic object %StringPrototype%. The String
    prototype object is an ordinary object. The String prototype is itself a String
    object; it has a [[StringData]] internal slot with the value "".
```

EdgeHTML Mode

The `String` prototype object is a `String` instance, not an ordinary object.

2.1.38 [ECMA-262/7] Section 21.1.3.22 String.prototype.toLowerCase ( )

V0139: Results are derived according to the mappings in UnicodeData.txt, but not those in SpecialCasings.txt.

The specification states:

```
21.1.3.22 String.prototype.toLowerCase ( )

    This function interprets a String value as a sequence of UTF-16 encoded code points,
    as described in 6.1.4. The following steps are taken:
        ...
        The result must be derived according to the locale-insensitive case mappings in
        the Unicode Character Database (this explicitly includes not only the
        UnicodeData.txt file, but also all locale-insensitive mappings in the
        SpecialCasings.txt file that accompanies it).
```

EdgeHTML Mode

Results are derived according to the mappings in UnicodeData.txt, but not those in SpecialCasings.txt.

V0140: Only characters in the Basic Multilingual Plane (values no greater than 0xFFFF) are converted to lowercase

The specification states:

```
21.1.3.22 String.prototype.toLowerCase ( )

    This function interprets a String value as a sequence of UTF-16 encoded code points,
    as described in 6.1.4. The following steps are taken:

        ...
        ... Let cpList be a List containing in order the code points as defined in 6.1.4
        of S, starting at the first element of S.
        ... For each code point c in cpList, if the Unicode Character Database provides a
        language insensitive lower case equivalent of c then replace c in cpList with
        that equivalent code point(s).
```

EdgeHTML Mode

Only those characters in the Basic Multilingual Plane (values no greater than 0xFFFF) are converted to lower case. Others are left unchanged.

2.1.39 [ECMA-262/7] Section 21.1.3.24 String.prototype.toUpperCase ( )

V0185: Only characters in the Basic Multilingual Plane (values no greater than 0xFFFF) are converted to uppercase

The specification states:

```
21.1.3.24 String.prototype.toUpperCase ( )
```

```
This function interprets a String value as a sequence of UTF-16 encoded code points,
as described in 6.1.4.

This function behaves in exactly the same way as String.prototype.toLowerCase, except
that code points are mapped to their uppercase equivalents as specified in the
Unicode Character Database.
```

EdgeHTML Mode

Only those characters in the Basic Multilingual Plane (values no greater than 0xFFFF) are converted to uppercase. Others are left unchanged.

2.1.40 [ECMA-262/7] Section 21.2.1 Patterns

V0078: If the contents of the braces in \u{...} is not a hexadecimal number, \u{...} is treated as a regular string

The specification states:

```
21.2.1 Patterns

    The RegExp constructor applies the following grammar to the input pattern String. An
    error occurs if the grammar cannot interpret the String as an expansion of Pattern.

    Syntax
        ...
        RegExpUnicodeEscapeSequence[U] ::
            [+U] u LeadSurrogate \u TrailSurrogate
            [+U] u LeadSurrogate
            [+U] u TrailSurrogate
            [+U] u NonSurrogate
            [~U] u Hex4Digits
            [+U] u{ HexDigits }
```

EdgeHTML Mode

If the contents of the braces in \u{...} is not a hexadecimal number, \u{...} is treated as a regular string, rather than a Unicode code point. For example, the following returns **true** but should throw a **SyntaxError** exception:

```
/\u{pp}/u.exec('\\u{pp}')
```

2.1.41 [ECMA-262/7] Section 21.2.1.1 Static Semantics: Early Errors

V0142: When the mathematical value of HexDigits is above 1114111, the \u{...} is not treated as a Unicode code point

The specification states:

```
21.2.1.1 Static Semantics: Early Errors

    RegExpUnicodeEscapeSequence :: u { HexDigits }

        • It is a Syntax Error if the MV of HexDigits > 1114111.
```

EdgeHTML Mode

When the mathematical value (MV) of HexDigits is above 1114111, the `\u{...}` is treated as a regular string, not as a Unicode code point, and no Syntax Error exception is thrown.

2.1.42 [ECMA-262/7] Section 21.2.2 Pattern Semantics

V0079: The input string is not treated as Unicode code points even when the associated flags contain a "u"

The specification states:

```
21.2.2 Pattern Semantics
    ...
    A Pattern is either a BMP pattern or a Unicode pattern depending upon whether or not
    its associated flags contain a "u". A BMP pattern matches against a String
    interpreted as consisting of a sequence of 16-bit values that are Unicode code points
    in the range of the Basic Multilingual Plane. A Unicode pattern matches against a
    String interpreted as consisting of Unicode code points encoded using UTF-16. In the
    context of describing the behaviour of a BMP pattern “character” means a single
    16-bit Unicode BMP code point. In the context of describing the behaviour of a
    Unicode pattern “character” means a UTF-16 encoded code point (6.1.4). In either
    context, “character value” means the numeric value of the corresponding non-encoded
    code point.
```

EdgeHTML Mode

The input string is interpreted as consisting of a sequence of 16-bit values that are Unicode code points in the range of the Basic Multilingual Plane, even when the associated flags contain a "u". For example, the following returns **false**, not **true**:

```
/\ud83d/u.test('\ud83d\udca8')
```

2.1.43 [ECMA-262/7] Section 21.2.2.8.2 Runtime Semantics: Canonicalize (ch)

V0172: Case-insensitive matching misses some characters

The specification states:

```
21.2.2.8.2 Runtime Semantics: Canonicalize ( ch )

    The abstract operation Canonicalize takes a character parameter ch and performs the
    following steps:

        1.  If IgnoreCase is false, return ch.
        2.  If Unicode is true,
            a.  If the file CaseFolding.txt of the Unicode Character Database provides a
            simple or common case folding mapping for ch, return the result of applying
            that mapping to ch.
            b.  Else, return ch.
        3.  Else,
            a.  Assert: ch is a UTF-16 code unit.
            b.  Let s be the ECMAScript String value consisting of the single code unit
            ch.
            c.  Let u be the same result produced as if by performing the algorithm for
            String.prototype.toUpperCase using s as the this value.
            d.  Assert: u is a String value.
            e.  If u does not consist of a single code unit, return ch.
```

```
f.  Let cu be u's single code unit element.
g.  If ch's code unit value ≥ 128 and cu's code unit value < 128, return ch.
h.  Return cu.
```

EdgeHTML Mode

Some mappings in the Unicode Character Database are not handled. Therefore, case-insensitive matching misses some characters.

For example, the following should be **true**, but is **false**:

```
/\u0345/i.test('\u0399');
```

2.1.44 [ECMA-262/7] Section 21.2.2.10 CharacterEscape

V0175: Characters other than those matched by ControlLetter (non-alphabetic characters) are allowed

The specification states:

```
21.2.2.10 CharacterEscape
    ...
    The production CharacterEscape :: c ControlLetter evaluates as follows:

        1.  Let ch be the character matched by ControlLetter.
        2.  Let i be ch's character value.
        3.  Let j be the remainder of dividing i by 32.
        4.  Return the character whose character value is j.
```

EdgeHTML Mode

Characters other than those matched by *ControlLetter* (non-alphabetic characters) are allowed.

2.1.45 [ECMA-262/7] Section 21.2.5 Properties of the RegExp Prototype Object

V0081: The RegExp prototype object is a RegExp object

The specification states:

```
21.2.5 Properties of the RegExp Prototype Object

    The RegExp prototype object is the intrinsic object %RegExpPrototype%. The RegExp
    prototype object is an ordinary object. It is not a RegExp instance and does not have
    a [[RegExpMatcher]] internal slot or any of the other internal slots of RegExp
    instance objects.

    The value of the [[Prototype]] internal slot of the RegExp prototype object is the
    intrinsic object %ObjectPrototype%.
```

EdgeHTML Mode

The `RegExp` prototype object is a `RegExp` object, and its `[[Class]]` is `RegExp`. The value of the `[[Prototype]]` internal property is the standard built-in `Object` prototype object.

The initial values of the `RegExp` prototype object's data properties are set as if the object were created by the expression new `RegExp`() where `RegExp` is the standard built-in constructor with that name.

V0165: The RegExp prototype object is the intrinsic object %RegExp% and is not an ordinary object

The specification states:

```
21.2.5 Properties of the RegExp Prototype Object

    The RegExp prototype object is the intrinsic object %RegExpPrototype%. The RegExp
    prototype object is an ordinary object. It is not a RegExp instance and does not have
    a [[RegExpMatcher]] internal slot or any of the other internal slots of RegExp
    instance objects.
```

EdgeHTML Mode

The `RegExp` prototype object is the intrinsic object `%RegExp%` and is not an ordinary object. It is a `RegExp` instance with a `[[RegExpMatcher]]` internal slot and all other internal slots of `RegExp` instance objects.

2.1.46 [ECMA-262/7] Section 21.2.5.2.3 AdvanceStringIndex (S, index, unicode)

V0173: AdvanceStringIndex advances the index by 1, not 2, when the unicode flag is specified

The specification states:

```
21.2.5.2.3 AdvanceStringIndex ( S, index, unicode )

    The abstract operation AdvanceStringIndex with arguments S, index, and unicode
    performs the following steps:

      1. Assert: Type(S) is String.
      2. Assert: index is an integer such that 0≤index≤2^53-1.
      3. Assert: Type(unicode) is Boolean.
      4. If unicode is false, return index+1.
      5. Let length be the number of code units in S.
      6. If index+1 ≥ length, return index+1.
      7. Let first be the code unit value at index index in S.
      8. If first < 0xD800 or first > 0xDBFF, return index+1.
      9. Let second be the code unit value at index index+1 in S.
     10. If second < 0xDC00 or second > 0xDFFF, return index+1.
     11. Return index+2.
```

EdgeHTML Mode

AdvanceStringIndex advances the index by 1, not 2 when the *unicode* flag is specified. For example, the following should hold:

```
/\udf06/u.exec('\ud834\udf06') == null
```

Instead `exec` returns \udf06; that is:

```
/\udf06/u.exec('\ud834\udf06') == '\udf06'
```

2.1.47 [ECMA-262/7] Section 21.2.6.1 lastIndex

V0082: The [[Writable]] attribute of the lastIndex property cannot be changed from true to false

The specification states:

```
21.2.6.1 lastIndex

    The value of the lastIndex property specifies the String index at which to start the
    next match. It is coerced to an integer when used (see 21.2.5.2.2). This property
    shall have the attributes { [[Writable]]: true, [[Enumerable]]: false,
    [[Configurable]]: false }.
```

EdgeHTML Mode

For `lastIndex`, `[[Writable]]` cannot be changed from **true** to **false**. This operation should be allowed, even though `[[Configurable]]` is **false** (see 6.1.7.1).

2.1.48 [ECMA-262/7] Section 22.1.3.1.1 Runtime Semantics: IsConcatSpreadable (O)

V0202: @@isConcatSpreadable is not implemented

The specification states:

```
22.1.3.1.1 Runtime Semantics: IsConcatSpreadable ( O )

    The abstract operation IsConcatSpreadable with argument O performs the following
    steps:

        1.  If Type(O) is not Object, return false.
        2.  Let spreadable be ? Get(O, @@isConcatSpreadable).
        3.  If spreadable is not undefined, return ToBoolean(spreadable).
        4.  Return ? IsArray(O).
```

EdgeHTML Mode

@@isConcatSpreadable is not implemented.

2.1.49 [ECMA-262/7] Section 22.1.3.3 Array.prototype.copyWithin (target, start [, end])

V0203: Under certain circumstances Array.prototype.copyWithin does not throw a TypeError when it should

The specification states:

```
22.1.3.3 Array.prototype.copyWithin (target, start [ , end ] )
    ...
    The following steps are taken:
        ...
        12. Repeat, while count > 0
            a.  Let fromKey be ! ToString(from).
            b.  Let toKey be ! ToString(to).
            c.  Let fromPresent be ? HasProperty(O, fromKey).
            d.  If fromPresent is true, then
```

```
            i.  Let fromVal be ? Get(O, fromKey).
            ii. Perform ? Set(O, toKey, fromVal, true).
        e.  Else fromPresent is false,
            i.  Perform ? DeletePropertyOrThrow(O, toKey).
        f.  Let from be from + direction.
        g.  Let to be to + direction.
        h.  Let count be count – 1.
    13. Return O.
```

EdgeHTML Mode

The following steps are not executed:

12.

e. Else fromPresent is false,

i. Perform ? DeletePropertyOrThrow(O, toKey).

As a result, under certain circumstances `Array.prototype.copyWithin` does not throw a **TypeError** when it should.

2.1.50 [ECMA-262/7] Section 22.1.3.18 Array.prototype.push (...items)

V0204: Array.prototype.push does not throw TypeError on length overflow

The specification states:

```
22.1.3.18 Array.prototype.push ( ...items )
    ...
    When the push method is called with zero or more arguments the following steps are
    taken:

        1.  Let O be ? ToObject(this value).
        2.  Let len be ? ToLength(? Get(O, "length")).
        3.  Let items be a List whose elements are, in left to right order, the arguments
            that were passed to this function invocation.
        4.  Let argCount be the number of elements in items.
        5.  If len + argCount > 2^53-1, throw a TypeError exception.
```

EdgeHTML Mode

The following step is not executed:

5. If len + argCount > 2^53-1, throw a TypeError exception.

As a result, `Array.prototype.push` does not throw **TypeError** on length overflow.

2.1.51 [ECMA-262/7] Section 22.1.3.25 Array.prototype.sort (comparefn)

V0205: Array.prototype.sort uses ToUint32 for length conversion

The specification states:

```
22.1.3.25 Array.prototype.sort (comparefn)
```

```
The elements of this array are sorted. The sort is not necessarily stable (that is,
elements that compare equal do not necessarily remain in their original order). If
comparefn is not undefined, it should be a function that accepts two arguments x and
y and returns a negative value if x < y, zero if x = y, or a positive value if x > y.

Upon entry, the following steps are performed to initialize evaluation of the sort
function:

    1.  Let obj be ? ToObject(this value).
    2.  Let len be ? ToLength(? Get(obj, "length")).
```

EdgeHTML Mode

`Array.prototype.sort` uses ToUint32 for length conversion (step 2):

1. Let obj be ToObject(this value).
2. Let len be ? ToUint32(? Get(obj, "length")).

2.1.52 [ECMA-262/7] Section 22.1.3.27 Array.prototype.toLocaleString ([reserved1 [, reserved2]])

V0206: Array.prototype.toLocaleString uses InvokeBuiltinMethod instead of Invoke

The specification states:

```
22.1.3.27 Array.prototype.toLocaleString ( [ reserved1 [ , reserved2 ] ] )

    An ECMAScript implementation that includes the ECMA-402 Internationalization API must
    implement the Array.prototype.toLocaleString method as specified in the ECMA-402
    specification. If an ECMAScript implementation does not include the ECMA-402 API the
    following specification of the toLocaleString method is used.
    ...
    The following steps are taken:
        ...
        7.  Else
            a.  Let R be ? ToString(? Invoke(firstElement, "toLocaleString")).
        ...
        9.  Repeat, while k < len
            ...
            d.  Else
                i.  Let R be ? ToString(? Invoke(nextElement, "toLocaleString")).
```

EdgeHTML Mode

`Array.prototype.toLocaleString` uses `InvokeBuiltinMethod` instead of `Invoke`:

...

7. Else

   a. Let R be ? ToString(? InvokeBuiltinMethod(firstElement, "toLocaleString")).

...

9. Repeat, while k < len

   ...

d. Else

i. Let R be ? ToString(? InvokeBuiltinMethod(nextElement, "toLocaleString")).

2.1.53 [ECMA-262/7] Section 25.4.4 Properties of the Promise Constructor

V0106: The Promise.length property is not configurable

The specification states:

```
25.4.4 Properties of the Promise Constructor

    The value of the [[Prototype]] internal slot of the Promise constructor is the
    intrinsic object %FunctionPrototype%.

    ... The Promise constructor has the following properties:
```

EdgeHTML Mode

The `Promise.length` property is not configurable.

2.1.54 [ECMA-262/7] Section 25.4.4.1 Promise.all (iterable)

V0207: Promise.all does not call IteratorClose

The specification states:

```
25.4.4.1 Promise.all ( iterable )

    The all function returns a new promise which is fulfilled with an array of
    fulfillment values for the passed promises, or rejects with the reason of the first
    passed promise that rejects. It resolves all elements of the passed iterable to
    promises as it runs this algorithm.

        1.  Let C be the this value.
        2.  If Type(C) is not Object, throw a TypeError exception.
        3.  Let promiseCapability be ? NewPromiseCapability(C).
        4.  Let iterator be GetIterator(iterable).
        5.  IfAbruptRejectPromise(iterator, promiseCapability).
        6.  Let iteratorRecord be Record {[[Iterator]]: iterator, [[Done]]: false}.
        7.  Let result be PerformPromiseAll(iteratorRecord, C, promiseCapability).
        8.  If result is an abrupt completion, then
            a.  If iteratorRecord.[[Done]] is false, let result be
            IteratorClose(iterator, result).
            b.  IfAbruptRejectPromise(result, promiseCapability).
        9.  Return Completion(result).
```

EdgeHTML Mode

Step 8a is not done; the IteratorClose abstract operation is not implemented.

2.2 Clarifications

There are no clarifications of the MAY and SHOULD requirements of [ECMA-262/7].

2.3 Extensions

The following subsections describe extensions to the requirements of [ECMA-262/7].

2.4 Error Handling

There are no additional error handling considerations.

2.5 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