[MS-ES2017]:

Microsoft Edge ECMAScript 2017 Language Specification (ECMA-262, 8th edition, June 2017) Standards Support Document

Revision Summary

Date	Revision History	Revision Class	Comments
12/5/2017	1.0	New	Released new document.
1/23/2018	1.0	None	No changes to the meaning, language, or formatting of the technical content.
2/22/2018	1.0	None	No changes to the meaning, language, or formatting of the technical content.
3/23/2018	1.0	None	No changes to the meaning, language, or formatting of the technical content.
8/28/2018	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® 2017 Language Specification*, [ECMA-262/8], published June 2017. The [ECMA-262/8] specification is the eighth 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-262/8] Ecma International, "ECMAScript® 2017 Language Specification", Standard ECMA-262 8th Edition / June 2017, http://www.ecma-international.org/ecma-262/8.0/index.html

[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-262/8] 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:
Quirks Mode, IE7 Mode, and IE8 Mode (All Versions)

1.4 Standards Support Requirements

To conform to [ECMA-262/8], 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-262/8] and whether they are considered normative or informative.

Sections	Normative/Informative
1-6	Informative
7-27	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-262/8].

- 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-262/8].

2.1.1   [ECMA-ES2017] 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 ...:

    ...
    ... Type(input) is Object ...
        ...
        ... Let exoticToPrim be ? GetMethod(input, @@toPrimitive).
```

EdgeHTML Mode

@@toPrimitive is not implemented.

2.1.2   [ECMA-ES2017] 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:

        ... Assert: Type(iterator) is Object.
```

```
... Assert: completion is a Completion Record.
... ...
... Let return be ? GetMethod(iterator, "return").
... If return is undefined, return Completion(completion).
... Let innerResult be Call(return, iterator, « »).
... If completion.[[type]] is throw, return Completion(completion).
... If innerResult.[[type]] is throw, return Completion(innerResult).
... If Type(innerResult.[[value]]) is not Object, throw a TypeError exception.
... 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-ES2017] 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-ES2017] 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-ES2017] 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:
```

```
                  ...

              3.  When ... 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-ES2017] 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-ES2017] 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-ES2017] 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-ES2017] 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-ES2017] 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-ES2017] Section 12.15.4 Runtime Semantics: Evaluation

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 : LeftHandSidelrefession 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.

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.

2.1.12 [ECMA-ES2017] 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-ES2017] Section 13.2.1 Static Semantics: Early Errors

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.

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

}
```

2.1.14 [ECMA-ES2017] 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-ES2017] Section 13.7.5.1 Static Semantics: 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-ES2017] 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 undefined or null, 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-ES2017] 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-ES2017] 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-ES2017] Section 14.3.7 Runtime Semantics: DefineMethod

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

The specification states:

```
14.3.7 Runtime Semantics: DefineMethod

    With parameters object and optional parameter functionPrototype.

    MethodDefinition : PropertyName ( UniqueFormalParameters ) { 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, then
            a. Let kind be Normal.
            b. Let prototype be functionPrototype.
        6.  Else,
            1.  Let kind be Method.
            2.  Let prototype be the intrinsic object %FunctionPrototype%.
        7.  Let closure be FunctionCreate(kind, UniqueFormalParameters, FunctionBody,
            scope, strict, prototype).
        8.  Perform MakeMethod(closure, object).
        9.  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'));

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

The specification states:

```
14.3.7 Runtime Semantics: DefineMethod
```

```
        With parameters object and optional parameter functionPrototype.

        MethodDefinition : PropertyName ( UniqueFormalParameters ) { 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, then
               a. Let kind be Normal.
               b. Let prototype be functionPrototype.
            6. Else,
               1. Let kind be Method.
               2. Let prototype be the intrinsic object %FunctionPrototype%.
            7. Let closure be FunctionCreate(kind, UniqueFormalParameters, FunctionBody,
               scope, strict, prototype).
            8. Perform MakeMethod(closure, object).
            9. 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();

2.1.20 [ECMA-ES2017] Section 14.5.13 Runtime Semantics: ClassDefinitionEvaluation

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

The specification states:

```
   ... 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-ES2017] 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-ES2017] Section 16.2 Forbidden Extensions

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

The specification states:

```
16.2 Forbidden Extensions

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

      • ECMAScript function objects defined using syntactic constructors in strict mode
      code must not be created with own properties named "caller" or "arguments". Such
      own properties also must not be created for function objects defined using an
      ArrowFunction, MethodDefinition, GeneratorDeclaration, GeneratorExpression,
      ClassDeclaration, ClassExpression, AsyncFunctionDeclaration,
      AsyncFunctionExpression, or AsyncArrowFunction regardless of whether the
      definition is contained in strict mode code. Built-in functions, strict functions
      created using the Function constructor, generator functions created using the
      Generator constructor, async functions created using the AsyncFunction
      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-ES2017] Section 19.1.2.20 Object.setPrototypeOf (O, proto)

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

The specification states:

```
... 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-ES2017] 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-ES2017] 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-ES2017] 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-ES2017] 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-ES2017] 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-ES2017] 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-ES2017] 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-ES2017] 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 s.[[SymbolData]].

    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-ES2017] 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-ES2017] 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-ES2017] 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, set clippedTime to +0.
5.  Return clippedTime.
```

EdgeHTML Mode

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

2.1.35 [ECMA-ES2017] 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-ES2017] 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-ES2017] Section 21.1.3.24 String.prototype.toLowerCase ( )

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

The specification states:

```
... 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.

2.1.38 [ECMA-ES2017] Section 21.1.3.26 String.prototype.toUpperCase ( )

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

The specification states:

```
... 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.

2.1.39 [ECMA-ES2017] 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}')
```

2.1.40 [ECMA-ES2017] Section 21.2.1.1 Static Semantics: Early Errors

V0142: When the mathematical value of HexDigits is above 0x10FFFF, 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 > 0x10FFFF.
```

EdgeHTML Mode

When the mathematical value (MV) of *HexDigits* is above 0x10FFFF (decimal 1114111), the \u{...} is treated as a regular string, not as a Unicode code point, and no Syntax Error exception is thrown.

2.1.41 [ECMA-ES2017] 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.

2.1.42 [ECMA-ES2017] 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.

2.1.43 [ECMA-ES2017] 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'

2.1.44 [ECMA-ES2017] 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).

2.1.45 [ECMA-ES2017] 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.

2.1.46 [ECMA-ES2017] 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.

2.1.47 [ECMA-ES2017] 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.

2.1.48 [ECMA-ES2017] 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")).

2.1.49 [ECMA-ES2017] 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:
       ...
       6.  Repeat, while k < len
       ...
           c. If nextElement is not undefined or null, then
              i.   Let S be ? ToString(? Invoke(nextElement, "toLocaleString")).
```

EdgeHTML Mode

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

...

6. Repeat, while k < len

...

c. If nextElement is not undefined or null, then

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

2.1.50 [ECMA-ES2017] 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.

2.1.51 [ECMA-ES2017] 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-262/8].

2.3   Extensions

The following subsections describe extensions to the requirements of [ECMA-262/8].

2.3.1   [ECMA-ES2017] Section 7.3.18 Invoke (V, P [, argumentsList])

E0009: Add InvokeBuiltinMethod(V,P [, argumentsList])

The specification states:

```
7.3.18 Invoke(V,P [, argumentsList])
```

EdgeHTML Mode

Add the following section:

7.3.18.1 InvokeBuiltinMethod(V,P [, argumentsList])

The abstract operation `Invoke` is used to call a built-in method property of an ECMAScript language value. This operation behaves the same way as `Invoke(V,P [, argumentsList])` except that it always invokes the initial property `P` of `V` regardless of subsequent changes to the property.

2.3.2 [ECMA-ES2017] Section 19.2.3.2 Function.prototype.bind (thisArg, …args)

E0010: Function.prototype.bind() creates functions with additional caller and arguments properties

The specification states:

```
4.3.34 own property

    property that is directly contained by its object

8.2.2 CreateIntrinsics ( realmRec )
    ...
        ...
        9.  Let funcProto be CreateBuiltinFunction(realmRec, noSteps, objProto).
        10. Set intrinsics.[[%FunctionPrototype%]] to funcProto.
        11. Call thrower.[[SetPrototypeOf]](funcProto).
        12. Perform AddRestrictedFunctionProperties(funcProto, realmRec).

9.2.5 FunctionCreate (kind, ParameterList, Body, Scope, Strict [, prototype])
    ...
    1.  If the prototype argument was not passed, then
        a. Let prototype be the intrinsic object %FunctionPrototype%.

9.2.7 AddRestrictedFunctionProperties ( F, realm )
    ...
        ...
        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}).

9.3 Built-in Function Objects
    ...
    Unless otherwise specified every built-in function object has the %FunctionPrototype%
    object (19.2.3) as the initial value of its [[Prototype]] internal slot.

9.4.1.3 BoundFunctionCreate (targetFunction, boundThis, boundArgs)
    ...
        ...
        2.  Let proto ? be targetFunction.[[GetPrototypeOf]]().
        ...
        7.  Set the [[Prototype]] internal slot of obj to proto.

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

```
        4.  Let F be ? BoundFunctionCreate(Target, thisArg, args).
```

EdgeHTML Mode

`Function.prototype.bind()` creates functions with additional `caller` and `arguments` properties. These properties should be inherited from the Function prototype (`%FunctionPrototype%`).

The 5.1 Edition of the ECMAScript® Language Specification of the version of the spec said that `bind` should add `caller` and `Arguments` own properties to the created bound function. However, later specifications do not.

These are the relevant lines from the 5.1 Edition:

15.3.4.5 Function.prototype.bind (thisArg [, arg1 [, arg2, …]])

   …

      …

      20. Call the [[DefineOwnProperty]] internal method of F with arguments "caller",

         PropertyDescriptor.

         {[[Get]]: thrower, [[Set]]: thrower, [[Enumerable]]: false, [[Configurable]]: false},

         and false.

      21. Call the [[DefineOwnProperty]] internal method of F with arguments "arguments",

         PropertyDescriptor,

         {[[Get]]: thrower, [[Set]]: thrower, [[Enumerable]]: false, [[Configurable]]: false},

         and false.

2.3.3 [ECMA-ES2017] Section 21.2.4 Properties of the RegExp Constructor

E0003: The RegExp constructor has a property named lastMatch that holds the matched substring for the last successful match

The specification states:

```
   21.2.4 Properties of the RegExp Constructor

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

      ... the RegExp constructor has the following properties:
```

EdgeHTML Mode

The `RegExp` constructor has a property named `lastMatch` that holds the matched substring for the last successful match. Before a successful match it is set to the empty string. For example:

var re = /a|c/

```
// RegExp.lastMatch === ''

re.exec('az')

// RegExp.lastMatch === 'a'

re.exec('bz')

// RegExp.lastMatch === 'a'

re.exec('cz')

// RegExp.lastMatch === 'c'
```

`lastMatch` is a data property and has the following attributes:

{"writable":true,"enumerable":true,"configurable":false}

Even though the `[[Writable]]` attribute is true, lastMatch is read-only and it is not possible to change its value directly.

The `RegExp` constructor has a property named `$&` that behaves the same as `lastMatch` but has the following attributes:

{"writable":true,"enumerable":false,"configurable":false}

E0004: The RegExp constructor has a property named lastParen that represents the last group from the last successful match

The specification states:

```
21.2.4 Properties of the RegExp Constructor

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

    ... the RegExp constructor has the following properties:
```

EdgeHTML Mode

The `RegExp` constructor has a property named `lastParen` that represents the last group from the last successful match. Before a successful match, it is set to the empty string. For example:

```
var re = /(a|b)(c|d)?/

// RegExp.lastParen === ''

re.exec('ac')

// RegExp.lastParen === 'c'

re.exec('z')

// RegExp.lastParen === 'c'

re.exec('bd')

// RegExp.lastParen === 'd'
```

`lastParen` is a data property and has the following attributes:

{"writable":true,"enumerable":true,"configurable":false}

Even though the `[[Writable]]` attribute is true, `lastParen` is read-only and it is not possible to change its value directly.

The `RegExp` constructor has another property called `$+` which behaves the same as `lastParen` but has the following attributes:

{"writable":true,"enumerable":false,"configurable":false}

E0001: The RegExp constructor has additional properties that represent the first nine groups of the last successful match

The specification states:

```
21.2.4 Properties of the RegExp Constructor

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

    ... the RegExp constructor has the following properties:
```

EdgeHTML Mode

The `RegExp` constructor has additional properties, `$1`, `$2`, ..., and `$9`, that represent the first nine groups of the last successful match. Before a successful match, each property is set to the empty string. For each group of the match (up to nine maximum), the corresponding property is set to a value that represents the group. For example:

var re = /(a|b)(c|d)/;

// RegExp.$1 === ''

// RegExp.$2 === ''

// RegExp.$3 === ''

// ...

// RegExp.$9 === ''

re.exec('ac'); // Successful match

// RegExp.$1 === 'a'

// RegExp.$2 === 'c'

// RegExp.$3 === ''

// ...

// RegExp.$9 === ''

re.exec('yz'); // No match

// $1-$9 are same as before

// RegExp.$1 === 'a'

// RegExp.$2 === 'c'

// RegExp.$3 === ''

// ...

// RegExp.$9 === ''

re.exec('bd'); // Successful match

// $1-$2 are now different

// RegExp.$1 === 'b'

// RegExp.$2 === 'd'

// RegExp.$3 === ''

// ...

// RegExp.$9 === ''

These properties are data properties and have the following attributes:

{"writable":true,"enumerable":true,"configurable":false}

Even though the `[[Writable]]` attribute is true, the properties are read-only and it is not possible to change their values directly.

E0002: The RegExp constructor has a property named input that represents the input string of the last successful match

The specification states:

```
21.2.4 Properties of the RegExp Constructor

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

    ... the RegExp constructor has the following properties:
```

EdgeHTML Mode

The `RegExp` constructor has a property named `input` that represents the input string of the last successful match. Before a successful match, it is set to the empty string. For example:

var re = /a|c/

// RegExp.input === ''

re.exec('az')

// RegExp.input === 'az'

re.exec('bz')

// RegExp.input === 'az'

re.exec('cz')

// RegExp.input === 'cz'

This is a data property and has the following attributes:

{"writable":true,"enumerable":true,"configurable":false}

Even though the `[[Writable]]` attribute is true, the property is read-only and it is not possible to change its value directly.

`RegExp` constructor has a property named `$_` which behaves the same way as the `input` property but has the following attributes:

{"writable":true,"enumerable":false,"configurable":false}

E0006: The RegExp constructor has a property named rightContext that holds the substring of the input string that is to the right of the matched substring

The specification states:

```
21.2.4 Properties of the RegExp Constructor

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

    ... the RegExp constructor has the following properties:
```

EdgeHTML Mode

The `RegExp` constructor has a property named `rightContext` that holds the substring of the input string that is to the right of the matched substring of the last successful match. Before a successful match, `rightContext` is set to the empty string. For example:

var re = /test/g

// RegExp.rightContext === ''

re.exec('test right')

// RegExp.rightContext === ' right'

re.exec('failure')

// RegExp.rightContext === ' right'

re.exec('test right another')

// RegExp.rightContext === ' right another'

`rightContext` is a data property and has the following attributes:

{"writable":true,"enumerable":true,"configurable":false}

Even though the `[[Writable]]` attribute is true, `rightContext` is read-only and cannot be changed directly.

The `RegExp` constructor also has a property named `$'` which behaves the same as `rightContext` but has the following attributes:

{"writable":true,"enumerable":false,"configurable":false}

E0007: The RegExp constructor has a property named index whose value is the starting index of the matched substring of the last successful match

The specification states:

```
21.2.4 Properties of the RegExp Constructor

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

   ... the RegExp constructor has the following properties:
```

EdgeHTML Mode

The `RegExp` constructor has a property named `index` whose value is the starting index of the matched substring of the last successful match. Before a successful match, it is set to -1. For example:

var re = /world/g

// RegExp.index === -1

re.exec('Hello world')

// RegExp.index === 6

re.exec('failure')

// RegExp.index === 6

re.exec('Another hello world')

// RegExp.index === 14

`index` is a data property and has the following attributes:

{"writable":true,"enumerable":false,"configurable":false}

Even though the `[[Writable]]attribute` is true, `index` is read-only and cannot be changed directly.

E0005: The RegExp constructor has a property named leftContext that holds the substring of the input string that is to the left of the matched substring

The specification states:

```
21.2.4 Properties of the RegExp Constructor

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

   ... the RegExp constructor has the following properties:
```

EdgeHTML Mode

The `RegExp` constructor has a property named `leftContext` that holds the substring of the input string that is to the left of the matched substring of the last successful match. Before a successful match, `leftContext` is set to the empty string. For example:

    var re = /world/g

    // RegExp.leftContext === ''

    re.exec('Hello world')

    // RegExp.leftContext === 'Hello '

    re.exec('failure')

    // RegExp.leftContext === 'Hello '

    re.exec('Another hello world')

    // RegExp.leftContext === 'Another hello '

`leftContext` is a data property and has the following attributes:

    {"writable":true,"enumerable":true,"configurable":false}

Even though the `[[Writable]]` attribute is true, `leftContext` is read-only and cannot be changed directly.

The `RegExp` constructor also has a property named `$`` which behaves the same as `leftContext` but has the following attributes:

    {"writable":true,"enumerable":false,"configurable":false}

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

.

...args) (section 2.1.27 23, section 2.3.2 36)

C

C) 12
Change tracking 44
completion) 7

E

expr - iterationKind) 17

G

Glossary 5

I

index - unicode) 31
Informative references 5
Introduction 5

N

Normative references 5

P

P [- argumentsList]) 35
PreferredType]) 7
proto) 21

R

realm) 8
References
   informative 5
   normative 5
reserved2]]) (section 2.1.25 22, section 2.1.49 34)

S

start [- end]) 32

T

Tracking changes 44