[MS-ES6]:

Microsoft Edge / Internet Explorer ECMA-262 ECMAScript Language Specification (Sixth 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.

Copyright © 2017 Microsoft Corporation

Revision Summary

Date	Revision History	Revision Class	Comments
12/7/2015	1.0	New	Released new document.
3/22/2016	1.0	None	No changes to the meaning, language, or formatting of the technical content.
7/19/2016	1.1	Minor	Clarified the meaning of the technical content.
11/2/2016	1.1	None	No changes to the meaning, language, or formatting of the technical content.
3/14/2017	1.1	None	No changes to the meaning, language, or formatting of the technical content.
10/3/2017	1.1	None	No changes to the meaning, language, or formatting of the technical content.

Copyright © 2017 Microsoft Corporation

Table of Contents

1 Introduction 7
1.1 Glossary 7
1.2 References 7
1.2.1 Normative References 7
1.2.2 Informative References 7
1.3 Microsoft Implementations 7
1.4 Standards Support Requirements 8
1.5 Notation 8

2 Standards Support Statements 9
2.1 Normative Variations 9
2.1.1 [ECMA-262/6] Section 19.4.3 Properties of the Symbol Prototype Object 9
2.1.2 [ECMA-262/6] Section 19.4.4 Properties of Symbol Instances 10
2.1.3 [ECMA-262/6] Section 19.5.3 Properties of the Error Prototype Object 10
2.1.4 [ECMA-262/6] Section 20.3.4 Properties of the Date Prototype Object 11
2.1.5 [ECMA-262/6] Section 21.1.3.24 String.prototype.toUpperCase ( ) 11
2.1.6 [ECMA-262/6] Section 7.1.1 ToPrimitive (input [, PreferredType]) 11
2.1.7 [ECMA-262/6] Section 7.1.15 ToLength (argument) 12
2.1.8 [ECMA-262/6] Section 7.3.18 Invoke(O,P, [argumentsList]) 13
2.1.9 [ECMA-262/6] Section 7.4.6 IteratorClose(iterator, completion) 13
2.1.10 [ECMA-262/6] Section 8.1.1.2.1 HasBinding(N) 14
2.1.11 [ECMA-262/6] Section 9.2.4 FunctionInitialize (F, kind, ParameterList, Body, Scope) 15
2.1.12 [ECMA-262/6] Section 9.2.7 AddRestrictedFunctionProperties (F, realm) 15
2.1.13 [ECMA-262/6] Section 9.4.1 Bound Function Exotic Objects 16
2.1.14 [ECMA-262/6] Section 9.4.4.5 [[Delete]] (P) 16
2.1.15 [ECMA-262/6] Section 11.8.6 Template Literal Lexical Components 17
2.1.16 [ECMA-262/6] Section 11.9.1 Rules of Automatic Semicolon Insertion 18
2.1.17 [ECMA-262/6] Section 12.2.6 Object Initializer 19
2.1.18 [ECMA-262/6] Section 12.2.9 Template Literals 19
2.1.19 [ECMA-262/6] Section 12.3.5 The super Keyword 19
2.1.20 [ECMA-262/6] Section 12.3.7 Tagged Templates 20
2.1.21 [ECMA-262/6] Section 12.4.1 Static Semantics: Early Errors 20
2.1.22 [ECMA-262/6] Section 12.4.4.1 Runtime Semantics: Evaluation 21
2.1.23 [ECMA-262/6] Section 12.4.5.1 Runtime Semantics: Evaluation 21
2.1.24 [ECMA-262/6] Section 12.5.1 Static Semantics: Early Errors 22
2.1.25 [ECMA-262/6] Section 12.5.7.1 Runtime Semantics: Evaluation 22
2.1.26 [ECMA-262/6] Section 12.5.8.1 Runtime Semantics: Evaluation 23
2.1.27 [ECMA-262/6] Section 12.9.4 Runtime Semantics: InstanceofOperator(O, C) 23
2.1.28 [ECMA-262/6] Section 12.14 Assignment Operators 24
2.1.29 [ECMA-262/6] Section 12.14.1 Static Semantics: Early Errors 24
2.1.30 [ECMA-262/6] Section 12.14.4 Runtime Semantics: Evaluation 25
2.1.31 [ECMA-262/6] Section 12.14.4 Runtime Semantics: Evaluation 25
2.1.32 [ECMA-262/6] Section 12.14.5 Destructuring Assignment 27
2.1.33 [ECMA-262/6] Section 13 ECMAScript Language: Statements and Declarations 27
2.1.34 [ECMA-262/6] Section 13.2.1 Static Semantics: Early Errors 28
2.1.35 [ECMA-262/6] Section 13.3.3 Destructuring Binding Patterns 29
2.1.36 [ECMA-262/6] Section 13.7 Iteration Statements 29
2.1.37 [ECMA-262/6] Section 13.7.4.1 Static Semantics: Early Errors 30
2.1.38 [ECMA-262/6] Section 13.7.4.7 Runtime Semantics: LabelledEvaluation 30
2.1.39 [ECMA-262/6] Section 13.7.5.1 Static Semantics: Early Errors 30
2.1.40 [ECMA-262/6] Section 13.7.5.12 Runtime Semantics: ForIn/OfHeadEvaluation (TDZnames, expr, iterationKind) 31

Copyright © 2017 Microsoft Corporation

2.1.41 [ECMA-262/6] Section 13.7.5.13 Runtime Semantics: ForIn/OfBodyEvaluation (lhs, stmt, iterator, lhsKind, labelSet) 32
2.1.42 [ECMA-262/6] Section 13.13 Labelled Statements 33
2.1.43 [ECMA-262/6] Section 14.1.2 Static Semantics: Early Errors 34
2.1.44 [ECMA-262/6] Section 14.2 Arrow Function Definitions 34
2.1.45 [ECMA-262/6] Section 14.3.5 Static Semantics: HasDirectSuper 34
2.1.46 [ECMA-262/6] Section 14.3.8 Runtime Semantics: DefineMethod 35
2.1.47 [ECMA-262/6] Section 14.4 Generator Function Definitions 36
2.1.48 [ECMA-262/6] Section 14.4.6 Static Semantics: HasDirectSuper 36
2.1.49 [ECMA-262/6] Section 14.5 Class Definitions 37
2.1.50 [ECMA-262/6] Section 14.5.1 Static Semantics: Early Errors 37
2.1.51 [ECMA-262/6] Section 14.5.2 Static Semantics: BoundNames 38
2.1.52 [ECMA-262/6] Section 14.5.3 Static Semantics: ConstructorMethod 38
2.1.53 [ECMA-262/6] Section 14.5.4 Static Semantics: Contains 39
2.1.54 [ECMA-262/6] Section 14.5.5 Static Semantics: ComputedPropertyContains 39
2.1.55 [ECMA-262/6] Section 14.5.6 Static Semantics: HasName 40
2.1.56 [ECMA-262/6] Section 14.5.7 Static Semantics: IsConstantDeclaration 40
2.1.57 [ECMA-262/6] Section 14.5.8 Static Semantics: IsFunctionDefinition 41
2.1.58 [ECMA-262/6] Section 14.5.9 Static Semantics: IsStatic 41
2.1.59 [ECMA-262/6] Section 14.5.10 Static Semantics: NonConstructorMethodDefinitions 42
2.1.60 [ECMA-262/6] Section 14.5.11 Static Semantics: PrototypePropertyNameList 42
2.1.61 [ECMA-262/6] Section 14.5.12 Static Semantics: PropName 42
2.1.62 [ECMA-262/6] Section 14.5.13 Static Semantics: StaticPropertyNameList 43
2.1.63 [ECMA-262/6] Section 14.5.14 Runtime Semantics: ClassDefinitionEvaluation ... 43
2.1.64 [ECMA-262/6] Section 14.5.15 Runtime Semantics: BindingClassDeclarationEvaluation 44
2.1.65 [ECMA-262/6] Section 14.5.16 Runtime Semantics: Evaluation 44
2.1.66 [ECMA-262/6] Section 15.1.1 Static Semantics: Early Errors 45
2.1.67 [ECMA-262/6] Section 15.2.1 Module Semantics 45
2.1.68 [ECMA-262/6] Section 16.1 Forbidden Extensions 46
2.1.69 [ECMA-262/6] Section 18.3.33 WeakSet (. . .) 46
2.1.70 [ECMA-262/6] Section 19.1.2 Properties of the Object Constructor 46
2.1.71 [ECMA-262/6] Section 19.1.3.2 Object.prototype.hasOwnProperty (V) 47
2.1.72 [ECMA-262/6] Section 19.1.3.4 Object.prototype.propertyIsEnumerable (V) 47
2.1.73 [ECMA-262/6] Section 19.1.3.5 Object.prototype.toLocaleString ([reserved1 [, reserved2]]) 48
2.1.74 [ECMA-262/6] Section 19.1.3.6 Object.prototype.toString ( ) 48
2.1.75 [ECMA-262/6] Section 19.2.3.2 Function.prototype.bind (thisArg , ...args) 49
2.1.76 [ECMA-262/6] Section 19.2.3.6 Function.prototype[@@hasInstance] (V) 49
2.1.77 [ECMA-262/6] Section 19.2.4 Function Instances 50
2.1.78 [ECMA-262/6] Section 19.4 Symbol Objects 50
2.1.79 [ECMA-262/6] Section 19.4.1 The Symbol Constructor 50
2.1.80 [ECMA-262/6] Section 19.4.2 Properties of the Symbol Constructor 51
2.1.81 [ECMA-262/6] Section 20.3.1.15 TimeClip (time) 51
2.1.82 [ECMA-262/6] Section 20.3.1.16 Date Time String Format 52
2.1.83 [ECMA-262/6] Section 21.1.3 Properties of the String Prototype Object 52
2.1.84 [ECMA-262/6] Section 21.1.3.11 String.prototype.match (regexp) 53
2.1.85 [ECMA-262/6] Section 21.1.3.14 String.prototype.replace (searchValue, replaceValue) 53
2.1.86 [ECMA-262/6] Section 21.1.3.15 String.prototype.search (regexp) 53
2.1.87 [ECMA-262/6] Section 21.1.3.17 String.prototype.split (separator, limit) 54
2.1.88 [ECMA-262/6] Section 21.1.3.22 String.prototype.toLowerCase ( ) 55
2.1.89 [ECMA-262/6] Section 21.1.3.27 String.prototype [@@iterator]( ) 55
2.1.90 [ECMA-262/6] Section 21.2 RegExp (Regular Expression) Objects 56
2.1.91 [ECMA-262/6] Section 21.2.1 Patterns 56
2.1.92 [ECMA-262/6] Section 21.2.1.1 Static Semantics: Early Errors 57

Copyright © 2017 Microsoft Corporation

2.1.93 [ECMA-262/6] Section 21.2.2 Pattern Semantics 57
2.1.94 [ECMA-262/6] Section 21.2.2.8.2 Runtime Semantics: Canonicalize (ch) 58
2.1.95 [ECMA-262/6] Section 21.2.2.10 CharacterEscape 58
2.1.96 [ECMA-262/6] Section 21.2.5 Properties of the RegExp Prototype Object 59
2.1.97 [ECMA-262/6] Section 21.2.5 Properties of the RegExp Prototype Object 59
2.1.98 [ECMA-262/6] Section 21.2.5.2.3 AdvanceStringIndex (S, index, unicode) 59
2.1.99 [ECMA-262/6] Section 21.2.5.3 get RegExp.prototype.flags 60
2.1.100 [ECMA-262/6] Section 21.2.5.4 get RegExp.prototype.global 60
2.1.101 [ECMA-262/6] Section 21.2.5.5 get RegExp.prototype.ignoreCase 61
2.1.102 [ECMA-262/6] Section 21.2.5.6 RegExp.prototype [@@match] (string) 61
2.1.103 [ECMA-262/6] Section 21.2.5.7 get RegExp.prototype.multiline 61
2.1.104 [ECMA-262/6] Section 21.2.5.8 RegExp.prototype [@@replace] (string, replaceValue) 62
2.1.105 [ECMA-262/6] Section 21.2.5.9 RegExp.prototype [@@search] (string) 62
2.1.106 [ECMA-262/6] Section 21.2.5.10 get RegExp.prototype.source 62
2.1.107 [ECMA-262/6] Section 21.2.5.11 RegExp.prototype [@@split] (string, limit) .. 63
2.1.108 [ECMA-262/6] Section 21.2.5.12 get RegExp.prototype.sticky 63
2.1.109 [ECMA-262/6] Section 21.2.5.15 get RegExp.prototype.unicode 63
2.1.110 [ECMA-262/6] Section 21.2.6 Properties of RegExp Instances 64
2.1.111 [ECMA-262/6] Section 22.1.3.1.1 Runtime Semantics: IsConcatSpreadable (O) 64
2.1.112 [ECMA-262/6] Section 22.1.3.3 Array.prototype.copyWithin (target, start [, end]) 64
2.1.113 [ECMA-262/6] Section 22.1.3.4 Array.prototype.entries ( ) 65
2.1.114 [ECMA-262/6] Section 22.1.3.13 Array.prototype.keys ( ) 65
2.1.115 [ECMA-262/6] Section 22.1.3.17 Array.prototype.push (...items) 66
2.1.116 [ECMA-262/6] Section 22.1.3.24 Array.prototype.sort (comparefn) 66
2.1.117 [ECMA-262/6] Section 22.1.3.26 Array.prototype.toLocaleString ([reserved1 [, reserved2]]) 67
2.1.118 [ECMA-262/6] Section 22.1.3.29 Array.prototype.values ( ) 68
2.1.119 [ECMA-262/6] Section 22.1.3.30 Array.prototype [@@iterator] ( ) 68
2.1.120 [ECMA-262/6] Section 22.1.3.31 Array.prototype [@@unscopables] 68
2.1.121 [ECMA-262/6] Section 22.2.1.1 %TypedArray% ( ) 69
2.1.122 [ECMA-262/6] Section 22.2.2.1 %TypedArray%.from (source [, mapfn [, thisArg]]) 70
2.1.123 [ECMA-262/6] Section 22.2.2.1.1 Runtime Semantics: TypedArrayFrom(constructor, items, mapfn, thisArg) 70
2.1.124 [ECMA-262/6] Section 22.2.2.2 %TypedArray%.of (...items) 71
2.1.125 [ECMA-262/6] Section 22.2.3 Properties of the %TypedArrayPrototype% Object 71
2.1.126 [ECMA-262/6] Section 22.2.5 Properties of the TypedArray Constructors 73
2.1.127 [ECMA-262/6] Section 22.2.6 Properties of TypedArray Prototype Objects 74
2.1.128 [ECMA-262/6] Section 22.2.7 Properties of TypedArray Instances 74
2.1.129 [ECMA-262/6] Section 23.1.1.1 Map ([iterable]) 75
2.1.130 [ECMA-262/6] Section 23.1.3 Properties of the Map Prototype Object 75
2.1.131 [ECMA-262/6] Section 23.2 Set Objects 76
2.1.132 [ECMA-262/6] Section 23.2.3 Properties of the Set Prototype Object 76
2.1.133 [ECMA-262/6] Section 23.3.1.1 WeakMap ([iterable]) 77
2.1.134 [ECMA-262/6] Section 24.1.3.3 get ArrayBuffer [@@species] 77
2.1.135 [ECMA-262/6] Section 25.2 GeneratorFunction Objects 78
2.1.136 [ECMA-262/6] Section 25.3 Generator Objects 78
2.1.137 [ECMA-262/6] Section 25.3.3 Generator Abstract Operations 78
2.1.138 [ECMA-262/6] Section 25.4 Promise Objects 79
2.1.139 [ECMA-262/6] Section 25.4.2 Promise Jobs 79
2.1.140 [ECMA-262/6] Section 25.4.3 The Promise Constructor 79
2.1.141 [ECMA-262/6] Section 25.4.4 Properties of the Promise Constructor 80
2.1.142 [ECMA-262/6] Section 25.4.4.1 Promise.all (iterable) 80
2.1.143 [ECMA-262/6] Section 25.4.5 Properties of the Promise Prototype Object 81
2.1.144 [ECMA-262/6] Section 25.4.5.4 Promise.prototype [@@toStringTag] 81

Copyright © 2017 Microsoft Corporation

2.1.145 [ECMA-262/6] Section 25.4.6 Properties of Promise Instances 81
2.1.146 [ECMA-262/6] Section B.1.4 Regular Expressions Patterns 82
2.1.147 [ECMA-262/6] Section B.2.2 Additional Properties of the Object.prototype Object 89
2.1.148 [ECMA-262/6] Section B.3.1 __proto__ Property Names in Object Initializers 90
2.2 Clarifications 91
2.2.1 [ECMA-262/6] Section 14.1.7 Static Semantics: HasInitializer 91
2.2.2 [ECMA-262/6] Section 14.2.6 Static Semantics: HasInitializer 91
2.3 Extensions 92
2.4 Error Handling 92
2.5 Security 92
3 Change Tracking 93
4 Index 94

Copyright © 2017 Microsoft Corporation

1 Introduction

The ECMAScript of the Microsoft web browsers is a dialect of the language defined in the ECMAScript Language Specification (Standard ECMA-262) Sixth Edition [ECMA-262/6], published June 2015. This document describes the level of support provided by the browsers for that specification.

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-262/6] Ecma International, "ECMAScript® 2015 Language Specification", Standard ECMA-262 6th Edition / June 2015, http://www.ecma-international.org/ecma-262/6.0/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 browser versions implement some portion of the [ECMA-262/6] specification:

specification:

- Internet Explorer 11
- 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
Internet Explorer 11	Quirks Mode IE7 Mode IE8 Mode

Copyright © 2017 Microsoft Corporation

Browser Version	Document Modes Supported
	IE9 Mode IE10 Mode IE11 Mode
Microsoft Edge	EdgeHTML Mode

For each variation presented in this document there is a list of the document modes and browser versions that exhibit the behavior described by the variation. All combinations of modes and versions that are not listed conform to the specification. For example, the following list for a variation indicates that the variation exists in 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-262/6], 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-262/6] 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	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.

Copyright © 2017 Microsoft Corporation

2 Standards Support Statements

This section contains all variations and clarifications for the Microsoft implementation of [ECMA-262/6].

- 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 2.4 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-262/6].

2.1.1 [ECMA-262/6] Section 19.4.3 Properties of the Symbol Prototype Object

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 }.
```

All document modes (All versions)

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

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

The specification states:

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


Copyright © 2017 Microsoft Corporation

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

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

All document modes (All versions)

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

V0180: Symbol is not implemented

The specification states:

```
    19.4.3 Properties of the Symbol Prototype Object

        The Symbol prototype object is the intrinsic object %SymbolPrototype%. The Symbol
        prototype object is an ordinary object. It is not a Symbol instance and does not have
        a [[SymbolData]] internal slot.
```

IE11 Mode (All versions)

`Symbol` is not implemented.

2.1.2 [ECMA-262/6] Section 19.4.4 Properties of Symbol Instances

V0181: Symbol is not implemented

The specification states:

```
    19.4.4 Properties of Symbol Instances

        Symbol instances are ordinary objects that inherit properties from the Symbol
        prototype object. ...
```

IE11 Mode (All versions)

`Symbol` is not implemented.

2.1.3 [ECMA-262/6] 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.
```


Copyright © 2017 Microsoft Corporation

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

All document modes (All versions)

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.

2.1.4 [ECMA-262/6] 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.
```

All document modes (All versions)

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

2.1.5 [ECMA-262/6] 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.
```

All document modes (All versions)

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

2.1.6 [ECMA-262/6] Section 7.1.1 ToPrimitive (input [, PreferredType])

V0164: @@toPrimitive is not implemented

Copyright © 2017 Microsoft Corporation

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).
```

All document modes (All versions)

@@toPrimitive is not implemented.

2.1.7 [ECMA-262/6] Section 7.1.15 ToLength (argument)

V0153: "lastIndex" doesn't go through "ToLength"

The specification states:

```
7.1.15 ToLength ( argument )

    The abstract operation ToLength converts argument to an integer suitable for use as
    the length of an array-like object. It performs the following steps:

         1. ReturnIfAbrupt(argument).
         2. Let len be ToInteger(argument).
         3. ReturnIfAbrupt(len).
         4. If len ≤ +0, return +0.
         5. If len is +∞, return 253-1.
         6. Return min(len, 253-1).
```

IE11 Mode, IE10 Mode, IE9 Mode, IE8 Mode, IE7 Mode, and IE5 (Quirks) Mode (All versions)

A negative `lastIndex` is not treated the same as a `lastIndox` of 0, but should be. As a result, there are strings that `RegExp` functions should match, but don't.

For example,

```
var re = /./;

re.lastIndex = 0;

var result = re.test('abc');

// result == true

re.lastIndex = -1;

result = re.test('abc');

// result == false
```


Copyright © 2017 Microsoft Corporation

2.1.8 [ECMA-262/6] Section 7.3.18 Invoke(O,P, [argumentsList])

V0034: Add InvokeBuiltinMethod(O,P,[argumentsList])

The specification states:

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

IE11 Mode and EdgeHTML Mode (All versions)

Add the following section:

7.3.18.1 InvokeBuiltinMethod(O,P, [argumentsList])

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

2.1.9 [ECMA-262/6] Section 7.4.6 IteratorClose(iterator, completion)

V0035: 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.  ReturnIfAbrupt(return).
        5.  If return is undefined, return Completion(completion).
        6.  Let innerResult be Call(return, iterator, « »).
        7.  If completion.[[type]] is throw, return Completion(completion).
        8.  If innerResult.[[type]] is throw, return Completion(innerResult).
        9.  If Type(innerResult.[[value]]) is not Object, throw a TypeError exception.
        10. Return Completion(completion).
```

IE11 Mode and EdgeHTML Mode (All versions)

`IteratorClose` is not correctly implemented. It behaves as follows:

7.4.6 IteratorClose(iterator, completion)

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

Copyright © 2017 Microsoft Corporation

2.1.10 [ECMA-262/6] Section 8.1.1.2.1 HasBinding(N)

V0168: Incorrect implementation of Let foundBinding be HasProperty(bindings, N)

The specification states:

```
8.1.1.2.1 HasBinding(N)

    The concrete Environment Record method HasBinding for object Environment Records
    determines if its associated binding object has a property whose name is the value of
    the argument N:
        ...
         1. Let envRec be the object Environment Record for which the method was invoked.
         2. Let bindings be the binding object for envRec.
         3. Let foundBinding be HasProperty(bindings, N)
```

IE11 Mode, IE10 Mode, IE9 Mode, IE8 Mode, IE7 Mode, and IE5 (Quirks) Mode (All versions)

Incorrect implementation of `Let foundBinding be HasProperty(bindings, N)`:

```
var x = 0;

var env = {};

var callCount = 0;

Object.defineProperty(env, Symbol.unscopables, {

   get: function() {

   callCount += 1;

}

});

with (env) {

   void x;

}

print(callCount); // should be 0 because x does not exist on the environment
```

V0169: Incorrect implementation of Let blocked be ToBoolean(Get(unscopables, N))

The specification states:

```
8.1.1.2.1 HasBinding(N)

    The concrete Environment Record method HasBinding for object Environment Records
    determines if its associated binding object has a property whose name is the value of
    the argument N:
        ...
         9. If Type(unscopables) is Object, then
            a. Let blocked be ToBoolean(Get(unscopables, N)).
            b. ReturnIfAbrupt(blocked).
```


Copyright © 2017 Microsoft Corporation

IE11 Mode, IE10 Mode, IE9 Mode, IE8 Mode, IE7 Mode, and IE5 (Quirks) Mode (All versions)

Incorrect implementation of `Let blocked be ToBoolean(Get(unscopables, N))`:

```
var x = 0;

var env = { x: 1 };

env[ Symbol.unscopables ] = {};

env[ Symbol.unscopables ].x = false;

with (env) {

   print(x) // should print 1 b\ToBoolean of x is false and thus x is scoped

}
```

2.1.11 [ECMA-262/6] Section 9.2.4 FunctionInitialize (F, kind, ParameterList, Body, Scope)

V0001: Function length property is non-configurable

The specification states:

```
9.2.4 FunctionInitialize (F, kind, ParameterList, Body, Scope)
    ...
     3. Let status be DefinePropertyOrThrow(F, "length", PropertyDescriptor{[[Value]]:
        len, [[Writable]]: false, [[Enumerable]]: false, [[Configurable]]: true}).
```

IE11 Mode (All versions)

The length property is made non-configurable in step 3:

3. Let status be DefinePropertyOrThrow(F, "length", PropertyDescriptor{[[Value]]: len, [[Writable]]: false, [[Enumerable]]: false, [[Configurable]]: false}).

2.1.12 [ECMA-262/6] Section 9.2.7 AddRestrictedFunctionProperties (F, realm)

V0002: The caller and arguments properties are set incorrectly

The specification states:

```
9.2.7 AddRestrictedFunctionProperties ( F, realm )
    ...
    3.  Let status be DefinePropertyOrThrow(F, "caller", PropertyDescriptor {[[Get]]:
        thrower, [[Set]]: thrower, [[Enumerable]]: false, [[Configurable]]: true}).
    4.  Assert: status is not an abrupt completion.
    5.  Return DefinePropertyOrThrow(F , "arguments", PropertyDescriptor {[[Get]]:
        thrower, [[Set]]: thrower, [[Enumerable]]: false, [[Configurable]]: true}).
```

IE11 Mode and EdgeHTML Mode (All versions)

Copyright © 2017 Microsoft Corporation

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

3. Let status be DefinePropertyOrThrow(F, "caller", PropertyDescriptor {[[Get]]: thrower, [[Set]]: undefined, [[Enumerable]]: false, [[Configurable]]: false}).

4. Assert: status is not an abrupt completion.

5. Return DefinePropertyOrThrow(F , "arguments", PropertyDescriptor {[[Get]]: thrower, [[Set]]: undefined, [[Enumerable]]: false, [[Configurable]]: false}).

2.1.13 [ECMA-262/6] Section 9.4.1 Bound Function Exotic Objects

V0037: The arguments and caller own properties for bound functions are not inherited from Function.prototype

The specification states:

```
9.4.1 Bound Function Exotic Objects

    A bound function is an exotic object that wraps another function object. A bound
    function is callable (it has a [[Call]] internal method and may have a [[Construct]]
    internal method). Calling a bound function generally results in a call of its wrapped
    function.

    Bound function objects do not have the internal slots of ECMAScript function objects
    defined in Table 27. Instead they have the internal slots defined in Table 28.

                Table 28 — Internal Slots of Exotic Bound Function Objects

    Internal Slot           | Type             | Description
    ------------------------|------------------|-----------------------------------------
    [[BoundTargetFunction]] | Callable Object  | The wrapped function object.
    [[BoundThis]]           | Any              | The value that is always passed as the
                            |                  | this value when calling the wrapped
                            |                  | function.
    [[BoundArguments]]      | List of Any      | A list of values whose elements are used
                            |                  | as the first arguments to any call to the
                            |                  | wrapped function.

    Unlike ECMAScript function objects, bound function objects do not use an alternative
    definition of the [[GetOwnProperty]] internal methods. Bound function objects provide
    all of the essential internal methods as specified in 9.1. However, they use the
    following definitions for the essential internal methods of function objects.
```

IE11 Mode, IE10 Mode, IE9 Mode, IE8 Mode, IE7 Mode, and IE5 (Quirks) Mode (All versions)

The `arguments` and `caller` own properties for bound functions are inherited from own properties instead of from`Function.prototype`. Bound functions are treated the same as non-strict ordinary functions.

2.1.14 [ECMA-262/6] Section 9.4.4.5 [[Delete]] (P)

V0038: [[Delete]] removes the property even if the configurable property is false

The specification states:

Copyright © 2017 Microsoft Corporation

```
9.4.4.5 [[Delete]] (P)

    The [[Delete]] internal method of an arguments exotic object when called with a
    property key P performs the following steps:

         1. Let map be the value of the [[ParameterMap]] internal slot of the arguments
            object.
         2. Let isMapped be HasOwnProperty(map, P).
         3. Assert: isMapped is not an abrupt completion.
         4. Let result be the result of calling the default [[Delete]] internal method
            for ordinary objects (9.1.10) on the arguments object passing P as the
            argument.
         5.  ReturnIfAbrupt(result).
         6.  If result is true and the value of isMapped is true, then
             a. Call map.[[Delete]](P).
         7. Return result.
```

IE11 Mode (All versions)

In step 4, the `[[Delete]]` internal method removes the property even if the `configurable` property is false.

2.1.15 [ECMA-262/6] 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
```


Copyright © 2017 Microsoft Corporation

```
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 (All versions)

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.

2.1.16 [ECMA-262/6] 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.
```

All document modes (All versions)

Rule 3 is not applied to yield* productions.

```
var obj = {
  *g() {
  yield
```


Copyright © 2017 Microsoft Corporation

```
    * 1
    }
};
```

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

```
yield;*1
```

This would throw a **SyntaxError**.

2.1.17 [ECMA-262/6] Section 12.2.6 Object Initializer

V0042: Object literal enhancements are not supported

The specification states:

```
12.2.6 Object Initializer
    ...
    PropertyDefinition[Yield] :
        IdentifierReference[?Yield]
        CoverInitializedName[?Yield]
        PropertyName[?Yield] : AssignmentExpression[In, ?Yield]
        MethodDefinition[?Yield]
    ...
    PropertyName[Yield] :
        LiteralPropertyName
        ComputedPropertyName[?Yield]
```

IE11 Mode (All versions)

None of the object literal enhancements are supported, so `IdentifierReference`, `MethodDefintion` and `ComputedProperty` don't work.

2.1.18 [ECMA-262/6] Section 12.2.9 Template Literals

V0043: String Templates are not implemented

The specification states:

```
12.2.9 Template Literals
```

IE11 Mode (All versions)

String Templates are not implemented.

2.1.19 [ECMA-262/6] Section 12.3.5 The super Keyword

V0003: The super keyword is not implemented

The specification states:

Copyright © 2017 Microsoft Corporation

```
12.3.5 The super Keyword
    ...
    SuperProperty : super [ Expression ]
    ...
    SuperProperty : super . IdentifierName
    ...
    SuperCall : super Arguments
    ...
```

IE11 Mode (All versions)

The `super` keyword is not implemented.

2.1.20 [ECMA-262/6] Section 12.3.7 Tagged Templates

V0044: String Templates are not implemented

The specification states:

```
12.3.7 Tagged Templates

    NOTE A tagged template is a function call where the arguments of the call are derived
    from a TemplateLiteral (12.2.9). The actual arguments include a template object
    (12.2.9.3) and the values produced by evaluating the expressions embedded within the
    TemplateLiteral.
```

IE11 Mode (All versions)

String templates are not implemented.

2.1.21 [ECMA-262/6] Section 12.4.1 Static Semantics: Early Errors

V0045: Postfix increment or decrement gives an early Syntax Error if applied to an immutable binding

The specification states:

```
12.4.1 Static Semantics: Early Errors

        PostfixExpression :
            LeftHandSideExpression ++
            LeftHandSideExpression --

    • It is an early Reference Error if IsValidSimpleAssignmentTarget of
    LeftHandSideExpression is false.
```

IE11 Mode (All versions)

It is an early SyntaxError if `LeftHandSideExpression` is an immutable binding reference (i.e. a `const` declared variable).

Copyright © 2017 Microsoft Corporation

2.1.22 [ECMA-262/6] Section 12.4.4.1 Runtime Semantics: Evaluation

V0046: The reference is retrieved twice

The specification states:

```
12.4.4.1 Runtime Semantics: Evaluation

    PostfixExpression : LeftHandSideExpression ++

        1.  Let lhs be the result of evaluating LeftHandSideExpression.
        2.  Let oldValue be ToNumber(GetValue(lhs)).
        3.  ReturnIfAbrupt(oldValue).
        4.  Let newValue be the result of adding the value 1 to oldValue, using the same
            rules as for the + operator (see 12.7.5).
        5.  Let status be PutValue(lhs, newValue).
        6.  ReturnIfAbrupt(status).
        7.  Return oldValue.
```

IE11 Mode and EdgeHTML Mode (All versions)

Between steps 3 and 4, 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);

This retrieves the reference twice.

2.1.23 [ECMA-262/6] Section 12.4.5.1 Runtime Semantics: Evaluation

V0047: The reference is retrieved twice

The specification states:

```
12.4.5.1 Runtime Semantics: Evaluation

    PostfixExpression : LeftHandSideExpression --

        1.  Let lhs be the result of evaluating LeftHandSideExpression.
        2.  Let oldValue be ToNumber(GetValue(lhs)).
        3.  ReturnIfAbrupt(oldValue).
        4.  Let newValue be the result of subtracting the value 1 from oldValue, using
            the same rules as for the - operator (12.7.5).
        5.  Let status be PutValue(lhs, newValue).
        6.  ReturnIfAbrupt(status).
        7.  Return oldValue.
```

IE11 Mode and EdgeHTML Mode (All versions)

Between steps 3 and 4 the following steps are added:

Copyright © 2017 Microsoft Corporation

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);

This retrieves the reference twice.

2.1.24 [ECMA-262/6] Section 12.5.1 Static Semantics: Early Errors

V0048: The increment or decrement of the prefix is an early Syntax Error if LeftHandSideExpression is an immutable binding reference

The specification states:

```
12.5.1 Static Semantics: Early Errors

    UnaryExpression :
        ++ UnaryExpression
        -- UnaryExpression

    •It is an early Reference Error if IsValidSimpleAssignmentTarget of UnaryExpression
    is false.
```

IE11 Mode (All versions)

The increment or decrement of the prefix is an early Syntax Error if `LeftHandSideExpression` is an immutable binding reference (i.e. a `const` declared variable).

2.1.25 [ECMA-262/6] Section 12.5.7.1 Runtime Semantics: Evaluation

V0049: The reference is retrieved twice

The specification states:

```
12.5.7.1 Runtime Semantics: Evaluation

    UnaryExpression : ++ UnaryExpression

        1.  Let expr be the result of evaluating UnaryExpression.
        2.  Let oldValue be ToNumber(GetValue(expr)).
        3.  ReturnIfAbrupt(oldValue).
        4.  Let newValue be the result of adding the value 1 to oldValue, using the same
            rules as for the + operator (see 12.7.5).
        5.  Let status be PutValue(expr, newValue).
        6.  ReturnIfAbrupt(status).
        7.  Return newValue.
```

IE11 Mode and EdgeHTML Mode (All versions)

Between steps 3 and 4, the following steps are added:

Copyright © 2017 Microsoft Corporation

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 retrieved twice.

2.1.26 [ECMA-262/6] Section 12.5.8.1 Runtime Semantics: Evaluation

V0050: The reference is retrieved twice

The specification states:

```
12.5.8.1 Runtime Semantics: Evaluation

    UnaryExpression : -- UnaryExpression

        1.  Let expr be the result of evaluating UnaryExpression.
        2.  Let oldValue be ToNumber(GetValue(expr)).
        3.  ReturnIfAbrupt(oldValue).
        4.  Let newValue be the result of subtracting the value 1 from oldValue, using
            the same rules as for the - operator (see 12.7.5).
        5.  Let status be PutValue(expr, newValue).
        6.  ReturnIfAbrupt(status).
        7.  Return newValue.
```

IE11 Mode and EdgeHTML Mode (All versions)

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.

2.1.27 [ECMA-262/6] Section 12.9.4 Runtime Semantics: InstanceofOperator(O, C)

V0051: InstanceofOperator(O, C) is not implemented

The specification states:

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

    The abstract operation InstanceofOperator(O, C) implements the generic algorithm for
```


Copyright © 2017 Microsoft Corporation

```
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.  ReturnIfAbrupt(instOfHandler).
    4.  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).
```

IE11 Mode and EdgeHTML Mode (All versions)

`InstanceofOperator(O, C)` is not implemented.

2.1.28 [ECMA-262/6] Section 12.14 Assignment Operators

V0052: ArrowFunction[?In, ?Yield] is not recognized

The specification states:

```
12.14 Assignment Operators

    Syntax

    AssignmentExpression[In, Yield] :
        ConditionalExpression[?In, ?Yield]
        [+Yield] YieldExpression[?In]
        ArrowFunction[?In, ?Yield]
        LeftHandSideExpression[?Yield] = AssignmentExpression[?In, ?Yield]
        LeftHandSideExpression[?Yield] AssignmentOperator AssignmentExpression[?In,
        ?Yield]
```

IE11 Mode (All versions)

ArrowFunction[?In, ?Yield] is not recognized.

2.1.29 [ECMA-262/6] Section 12.14.1 Static Semantics: Early Errors

V0053: It is an early Syntax Error if LeftHandSideExpression is an immutable binding reference

The specification states:

```
12.14.1 Static Semantics: Early Errors

    AssignmentExpression : LeftHandSideExpression = AssignmentExpression

        • It is a Syntax Error if LeftHandSideExpression is either an ObjectLiteral or an
        ArrayLiteral and the lexical token sequence matched by LeftHandSideExpression
        cannot be parsed with no tokens left over using AssignmentPattern as the goal
        symbol.

        • It is an early Reference Error if LeftHandSideExpression is neither an
        ObjectLiteral nor an ArrayLiteral and IsValidSimpleAssignmentTarget of
        LeftHandSideExpression is false.
```


Copyright © 2017 Microsoft Corporation

```
AssignmentExpression : LeftHandSideExpression AssignmentOperator AssignmentExpression

    • It is an early Reference Error if IsValidSimpleAssignmentTarget of
    LeftHandSideExpression is false.
```

IE11 Mode (All versions)

It is an early Syntax Error if `LeftHandSideExpression` is an immutable binding reference.

2.1.30 [ECMA-262/6] Section 12.14.4 Runtime Semantics: Evaluation

V0170: After an assignment, the name of the function is the empty string

The specification states:

```
12.14.4 Runtime Semantics: Evaluation

    AssignmentExpression[In, Yield] :LeftHandSideExpression[?Yield] =
    AssignmentExpression[?In, ?Yield]

        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. ReturnIfAbrupt(hasNameProperty).
                 iii. If hasNameProperty is false, perform SetFunctionName(rval,
                 GetReferencedName(lref)).
```

All document modes (All versions)

After the following assignment:

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

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

2.1.31 [ECMA-262/6] Section 12.14.4 Runtime Semantics: Evaluation

V0054: The reference is retrieved twice

The specification states:

```
12.14.4 Runtime Semantics: Evaluation

    AssignmentExpression[In, Yield] : LeftHandSideExpression[?Yield] =
    AssignmentExpression[?In, ?Yield]

        1.  If LeftHandSideExpression is neither an ObjectLiteral nor an ArrayLiteral,
            then
            a.  Let lref be the result of evaluating LeftHandSideExpression.
            b.  ReturnIfAbrupt(lref).
```


Copyright © 2017 Microsoft Corporation

```
            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. ReturnIfAbrupt(hasNameProperty).
                iii. If hasNameProperty is false, perform SetFunctionName(rval,
                GetReferencedName(lref)).
            f.  Let status be PutValue(lref, rval).
            g.  ReturnIfAbrupt(status).
            h.  Return rval.
        ...

    AssignmentExpression : LeftHandSideExpression AssignmentOperator AssignmentExpression

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

IE11 Mode and EdgeHTML Mode (All versions)

In the algorithm for

AssignmentExpression[In, Yield] : LeftHandSideExpression[?Yield] = AssignmentExpression[?In, ?Yield]

the following steps are added before step1f:

- Type(lref) is Reference and if IsUnresolvableReference(_lref_) is false and IsPropertyReference(_lref_) is false then
    - Assert: lref is a reference to an Environment Record.
    - Let lref be the result of evaluating an Identifier _id_ whose StringValue is GetReferencedName(lref) as if _id_ were a LeftHandSidelrefession.
    - ReturnIfAbrupt(lref);

This retrieves the reference twice.

In the algorithm for

Assignmentlrefession : LeftHandSidelrefession AssignmentOperator Assignmentlrefession

the following steps are added between steps 6 and 7:

- Type(lref) is Reference and if IsUnresolvableReference(_lref_) is false and IsPropertyReference(_lref_) is false then
    - Assert: lref is a reference to an Environment Record.
    - Let lref be the result of evaluating an Identifier _id_ whose StringValue is GetReferencedName(lref) as if _id_ were a LeftHandSidelrefession.

Copyright © 2017 Microsoft Corporation

- ReturnIfAbrupt(lref);

This retrieves the reference twice.

2.1.32 [ECMA-262/6] Section 12.14.5 Destructuring Assignment

V0055: The destructuring pattern is not implemented

The specification states:

```
12.14.5 Destructuring Assignment

    Supplemental Syntax

    In certain circumstances when processing the production AssignmentExpression :
    LeftHandSideExpression = AssignmentExpression the following grammar is used to refine
    the interpretation of LeftHandSideExpression.

    ...
```

IE11 Mode (All versions)

The destructuring pattern is not implemented.

2.1.33 [ECMA-262/6] 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]
```

IE11 Mode and EdgeHTML Mode (All versions)

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

Statement[Yield, Return] :

BlockStatement[?Yield, ?Return]

...

Copyright © 2017 Microsoft Corporation

DebuggerStatement

HoistableDeclaration[?Yield]

Declaration[Yield] :

ClassDeclaration[?Yield]

LexicalDeclaration[In, ?Yield]

HoistableDeclaration[Yield, Default] :

FunctionDeclaration[?Yield,?Default]

GeneratorDeclaration[?Yield, ?Default]

2.1.34 [ECMA-262/6] 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.
```

IE11 Mode and EdgeHTML Mode (All versions)

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.
```


Copyright © 2017 Microsoft Corporation

IE11 Mode and EdgeHTML Mode (All versions)

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

2.1.35 [ECMA-262/6] Section 13.3.3 Destructuring Binding Patterns

V0059: Destructuring binding patterns are not implemented

The specification states:

```
13.3.3 Destructuring Binding Patterns

    Syntax

        BindingPattern[Yield] :
            ObjectBindingPattern[?Yield]
            ArrayBindingPattern[?Yield
        ...
```

IE11 Mode (All versions)

Destructuring binding patterns are not implemented.

2.1.36 [ECMA-262/6] Section 13.7 Iteration Statements

V0060: Iteration statements of the form for (... of ...) are not implemented; they are ignored

The specification states:

```
13.7 Iteration Statements

    Syntax

        IterationStatement[Yield, Return] :
            do Statement[?Yield, ?Return] while ( Expression[In, ?Yield] ) ;
            while ( Expression[In, ?Yield] ) Statement[?Yield, ?Return]
            for ( [lookahead ∉ {let [}] Expression[?Yield]opt ; Expression[In, ?Yield]opt
            ; Expression[In, ?Yield]opt ) Statement[?Yield, ?Return]
            for ( var VariableDeclarationList[?Yield] ; Expression[In, ?Yield]opt ;
            Expression[In, ?Yield]opt ) Statement[?Yield, ?Return]
            for ( LexicalDeclaration[?Yield] Expression[In, ?Yield]opt ; Expression[In,
            ?Yield]opt ) Statement[?Yield, ?Return]
            for ( [lookahead ∉ {let [}] LeftHandSideExpression[?Yield] in Expression[In,
            ?Yield] ) Statement[?Yield, ?Return]
            for ( var ForBinding[?Yield] in Expression[In, ?Yield] ) Statement[?Yield,
            ?Return]
            for ( ForDeclaration[?Yield] in Expression[In, ?Yield] ) Statement[?Yield,
            ?Return]
            for ( [lookahead ≠ let ] LeftHandSideExpression[?Yield] of
            AssignmentExpression[In, ?Yield] ) Statement[?Yield, ?Return]
            for ( var ForBinding[?Yield] of AssignmentExpression[In, ?Yield] )
            Statement[?Yield, ?Return]
            for ( ForDeclaration[?Yield] of AssignmentExpression[In, ?Yield] )
            Statement[?Yield, ?Return]
```

IE11 Mode (All versions)

Copyright © 2017 Microsoft Corporation

Iteration statements of the form `for ( ... of ... )` are not implemented; they are ignored.

2.1.37 [ECMA-262/6] 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.
```

IE11 Mode and EdgeHTML Mode (All versions)

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

2.1.38 [ECMA-262/6] Section 13.7.4.7 Runtime Semantics: LabelledEvaluation

V0127: For loops don't get per-iteration environments for let variable declarations

The specification states:

```
13.7.4.7 Runtime Semantics: LabelledEvaluation

    With argument labelSet.

    See also: 13.1.7, 13.7.2.6, 13.7.3.6, 13.7.5.11, 13.13.14.
    ...
    IterationStatement : for ( LexicalDeclaration Expressionopt ; Expressionopt )
    Statement
        ...
         9. If isConst is false, let perIterationLets be boundNames otherwise let
            perIterationLets be « ».
        ...
```

IE11 Mode (All versions)

In step 9, `perIterationLets` is in effect « » regardless of the value of `isConst`. As a result, `for` loops don't get per-iteration environments for let variable declarations.

2.1.39 [ECMA-262/6] Section 13.7.5.1 Static Semantics: Early Errors

V0128: A const declaration in a for-in or for-of loop throws a Syntax Error

The specification states:

```
13.7.5.1 Static Semantics: Early Errors
    ...
            IterationStatement :
```


Copyright © 2017 Microsoft Corporation

```
            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.
```

IE11 Mode (All versions)

A `const` declaration in a `for-in` or `for-of` loop throws a Syntax Error, as if there were a fourth bullet as follows:

- It is a Syntax Error if IsConstantDeclaration of ForDeclaration is true

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.
```

IE11 Mode and EdgeHTML Mode (All versions)

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

2.1.40 [ECMA-262/6] Section 13.7.5.12 Runtime Semantics: ForIn/OfHeadEvaluation (TDZnames, expr, iterationKind)

V0130: 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.
        ...
        7.  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).
```


Copyright © 2017 Microsoft Corporation

```
        c.  Return obj.[[Enumerate]]().
    8.  Else,
        a.  Assert: iterationKind is iterate.
        b.  Return GetIterator(exprValue).
```

IE11 Mode and EdgeHTML Mode (All versions)

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

8. 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 *AbruptCompletion* 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) {}
```

2.1.41 [ECMA-262/6] Section 13.7.5.13 Runtime Semantics: ForIn/OfBodyEvaluation (lhs, stmt, iterator, lhsKind, labelSet)

V0171: Lexical declarations in for-of/for-in loops do not get new environments for each iteration of the loop

The specification states:

```
13.7.5.13 Runtime Semantics: ForIn/OfBodyEvaluation ( lhs, stmt, iterator, lhsKind,
labelSet )

    The abstract operation ForIn/OfBodyEvaluation is called with arguments lhs, stmt,
    iterator, lhsKind, and labelSet. The value of lhsKind is either assignment,
    varBinding or lexicalBinding.

        1. Let oldEnv be the running execution context's LexicalEnvironment.
        2. Let V = undefined .
        3. Let destructuring be IsDestructuring of lhs.
        4. If destructuring is true and if lhsKind is assignment, then
           a. Assert: lhs is a LeftHandSideExpression.
           b. Let assignmentPattern be the parse of the source text corresponding to lhs
           using AssignmentPattern as the goal symbol.
        5. Repeat
           a. Let nextResult be IteratorStep(iterator).
           b. ReturnIfAbrupt(nextResult).
           c. If nextResult is false, return NormalCompletion(V).
           d. Let nextValue be IteratorValue(nextResult).
           e. ReturnIfAbrupt(nextValue).
           f. If lhsKind is either assignment or varBinding, then
                 i. If destructuring is false, then
                     1. Let lhsRef be the result of evaluating lhs ( it may be evaluated
                        repeatedly).
```


Copyright © 2017 Microsoft Corporation

```
g. Else
      i. Assert: lhsKind is lexicalBinding.
     ii. Assert: lhs is a ForDeclaration.
    iii. Let iterationEnv be NewDeclarativeEnvironment(oldEnv).
     iv. Perform BindingInstantiation for lhs passing iterationEnv as the
     argument.
      v. Set the running execution context's LexicalEnvironment to
      iterationEnv.
     vi. If destructuring is false, then
          1. Assert: lhs binds a single name.
          2. Let lhsName be the sole element of BoundNames of lhs.
          3. Let lhsRef be ResolveBinding(lhsName).
          4. Assert: lhsRef is not an abrupt completion.
...
```

IE11 Mode (All versions)

Steps 5f and 5g are replaced by the substeps of 5f only:

f. (nothing)

i. If destructuring is false, then

1. Let lhsRef be the result of evaluating lhs (it may be evaluated repeatedly).

g. (nothing)

h. ...

Therefore, lexical declarations in `for-of`/`for-in` loops do not get new environments per iteration of the loop.

2.1.42 [ECMA-262/6] 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]
```

IE11 Mode and EdgeHTML Mode (All versions)

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

LabelledItem[Yield, Return] :

Statement[?Yield, ?Return]

Declaration[?Yield]

Copyright © 2017 Microsoft Corporation

2.1.43 [ECMA-262/6] 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.
```

IE11 Mode and EdgeHTML Mode (All versions)

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

2.1.44 [ECMA-262/6] Section 14.2 Arrow Function Definitions

V0064: Arrow functions are not implemented

The specification states:

```
14.2 Arrow Function Definitions

    ...
```

IE11 Mode (All versions)

Arrow functions are not implemented; the syntax is not recognized.

2.1.45 [ECMA-262/6] Section 14.3.5 Static Semantics: HasDirectSuper

V0004: Nothing in the section is implemented

The specification states:

```
14.3.5 Static Semantics: HasDirectSuper
```

IE11 Mode (All versions)

Copyright © 2017 Microsoft Corporation

Nothing in the section is implemented.

2.1.46 [ECMA-262/6] 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}.
```

All document modes (All versions)

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).
```


Copyright © 2017 Microsoft Corporation

```
        8.  Return the Record{[[key]]: propKey, [[closure]]: closure}.
```

All document modes (All versions)

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'));
```

2.1.47 [ECMA-262/6] Section 14.4 Generator Function Definitions

V0005: A YieldExpression is not treated as restricted

The specification states:

```
14.4 Generator Function Definitions

    Syntax
        ...
        YieldExpression[In] :
            yield
            yield [no LineTerminator here] AssignmentExpression[?In, Yield]
            yield [no LineTerminator here] * AssignmentExpression[?In, Yield]
```

IE11 Mode (All versions)

A `YieldExpression` production is not treated as restricted; that is, it may have `LineTerminator` characters between `yield` and the remainder of the production. The following production is implemented:

```
YieldExpression[In] :

  yield

  yield AssignmentExpression[?In, Yield]

  yield * AssignmentExpression[?In, Yield]
```

2.1.48 [ECMA-262/6] Section 14.4.6 Static Semantics: HasDirectSuper

V0006: Nothing in this section is implemented

The specification states:

```
14.4.6 Static Semantics: HasDirectSuper

    See also: 14.3.5.

    GeneratorMethod : * PropertyName ( StrictFormalParameters ) { GeneratorBody }
         1. If StrictFormalParameters Contains SuperCall is true, return true.
         2. Return GeneratorBody Contains SuperCall.
```


Copyright © 2017 Microsoft Corporation

```
GeneratorDeclaration : function * BindingIdentifier ( FormalParameters ) {
GeneratorBody } GeneratorDeclaration : function * ( FormalParameters ) {
GeneratorBody }

GeneratorExpression : function * ( FormalParameters ) { GeneratorBody }

GeneratorExpression : function * BindingIdentifier ( FormalParameters ) {
GeneratorBody }
     1. If FormalParameters Contains SuperCall is true, return true.
     2. Return GeneratorBody Contains SuperCall.
```

IE11 Mode (All versions)

Nothing in this section is implemented.

2.1.49 [ECMA-262/6] Section 14.5 Class Definitions

V0007: Nothing in this section is implemented

The specification states:

```
14.5 Class Definitions

    Syntax

        ClassDeclaration[Yield, Default] :
            class BindingIdentifier[?Yield] ClassTail[?Yield]
            [+Default] class ClassTail[?Yield]

        ClassExpression[Yield] :
            class BindingIdentifier[?Yield]opt ClassTail[?Yield]

        ClassTail[Yield] :
            ClassHeritage[?Yield]opt { ClassBody[?Yield]opt }

        ClassHeritage[Yield] :
            extends LeftHandSideExpression[?Yield]

        ClassBody[Yield] :
            ClassElementList[?Yield]

        ClassElementList[Yield] :
            ClassElement[?Yield]
            ClassElementList[?Yield] ClassElement[?Yield]

        ClassElement[Yield] :
            MethodDefinition[?Yield]
            static MethodDefinition[?Yield]
            ;
```

IE11 Mode (Internet Explorer 11)

Nothing in this section is implemented.

2.1.50 [ECMA-262/6] Section 14.5.1 Static Semantics: Early Errors

V0008: Nothing in this section is implemented

Copyright © 2017 Microsoft Corporation

The specification states:

```
14.5.1 Static Semantics: Early Errors

    ClassTail : ClassHeritageopt { ClassBody }
        • It is a Syntax Error if ClassHeritage is not present and the following
        algorithm evaluates to true:
             1. Let constructor be ConstructorMethod of ClassBody.
             2. If constructor is empty, return false.
             3. Return HasDirectSuper of constructor.

    ClassBody : ClassElementList
        • It is a Syntax Error if PrototypePropertyNameList of ClassElementList contains
        more than one occurrence of "constructor".

    ClassElement : MethodDefinition
        • It is a Syntax Error if PropName of MethodDefinition is not "constructor" and
        HasDirectSuper of MethodDefinition is true.
        • It is a Syntax Error if PropName of MethodDefinition is "constructor" and
        SpecialMethod of MethodDefinition is true.

    ClassElement : static MethodDefinition
        • It is a Syntax Error if HasDirectSuper of MethodDefinition is true.
        • It is a Syntax Error if PropName of MethodDefinition is "prototype".
```

IE11 Mode (All versions)

Nothing in this section is implemented.

2.1.51 [ECMA-262/6] Section 14.5.2 Static Semantics: BoundNames

V0009: Nothing in this section is implemented

The specification states:

```
14.5.2 Static Semantics: BoundNames

    See also: 12.1.2, 13.3.1.2, 13.3.2.1, 13.3.3.1, 13.7.5.2, 14.1.3, 14.2.2, 14.4.2,
    15.2.2.2, 15.2.3.2.

    ClassDeclaration : class BindingIdentifier ClassTail
         1. Return the BoundNames of BindingIdentifier.

    ClassDeclaration : class ClassTail
         1. Return «"*default*"».
```

IE11 Mode (Internet Explorer 11)

Nothing in this section is implemented.

2.1.52 [ECMA-262/6] Section 14.5.3 Static Semantics: ConstructorMethod

V0010: Nothing in this section is implemented

The specification states:

Copyright © 2017 Microsoft Corporation

```
14.5.3 Static Semantics: ConstructorMethod

    ClassElementList : ClassElement
         1. If ClassElement is the production ClassElement : ; , return empty.
         2. If IsStatic of ClassElement is true, return empty.
         3. If PropName of ClassElement is not "constructor", return empty.
         4. Return ClassElement.

    ClassElementList : ClassElementList ClassElement
         1. Let head be ConstructorMethod of ClassElementList.
         2. If head is not empty, return head.
         3. If ClassElement is the production ClassElement : ; , return empty.
         4. If IsStatic of ClassElement is true, return empty.
         5. If PropName of ClassElement is not "constructor", return empty.
         6. Return ClassElement.
```

IE11 Mode (All versions)

Nothing in this section is implemented.

2.1.53 [ECMA-262/6] Section 14.5.4 Static Semantics: Contains

V0011: Nothing in this section is implemented

The specification states:

```
14.5.4 Static Semantics: Contains

    With parameter symbol.

    See also: 5.3, 12.2.6.3, 12.3.1.1, 14.1.4, 14.2.3, 14.4.4

    ClassTail : ClassHeritageopt { ClassBody }
        1. If symbol is ClassBody, return true.
        2. If symbol is ClassHeritage, then
            a. If ClassHeritage is present, return true otherwise return false.
        3. Let inHeritage be ClassHeritage Contains symbol.
        4. If inHeritage is true, return true.
        5. Return the result of ComputedPropertyContains for ClassBody with argument
           symbol.
```

IE11 Mode (All versions)

Nothing in this section is implemented.

2.1.54 [ECMA-262/6] Section 14.5.5 Static Semantics: ComputedPropertyContains

V0012: Nothing in this section is implemented

The specification states:

```
14.5.5 Static Semantics: ComputedPropertyContains

    With parameter symbol.

    See also: 12.2.6.2, 14.3.2, 14.4.3.
```


Copyright © 2017 Microsoft Corporation

```
ClassElementList : ClassElementList ClassElement
    1. Let inList be the result of ComputedPropertyContains for ClassElementList with
       argument symbol.
    2. If inList is true, return true.
    3. Return the result of ComputedPropertyContains for ClassElement with argument
       symbol.

ClassElement : MethodDefinition
    1. Return the result of ComputedPropertyContains for MethodDefinition with
       argument symbol.

ClassElement : static MethodDefinition
    1. Return the result of ComputedPropertyContains for MethodDefinition with
       argument symbol.

ClassElement : ;
    1. Return false.
```

IE11 Mode (All versions)

Nothing in this section is implemented.

2.1.55 [ECMA-262/6] Section 14.5.6 Static Semantics: HasName

V0013: Nothing in this section is implemented

The specification states:

```
14.5.6 Static Semantics: HasName

    See also: 12.2.1.2, 14.1.8, 14.2.7, 14.4.7.

    ClassExpression : class ClassTail
         1. Return false.

    ClassExpression : class BindingIdentifier ClassTail
         1. Return true.
```

IE11 Mode (All versions)

Nothing in this section is implemented.

2.1.56 [ECMA-262/6] Section 14.5.7 Static Semantics: IsConstantDeclaration

V0014: Nothing in this section is implemented

The specification states:

```
14.5.7 Static Semantics: IsConstantDeclaration

    See also: 13.3.1.3, 14.1.10, 14.4.8, 15.2.3.7.

    ClassDeclaration : class BindingIdentifier ClassTail

    ClassDeclaration : class ClassTail
```


Copyright © 2017 Microsoft Corporation

```
        1. Return false.
```

IE11 Mode (All versions)

Nothing in this section is implemented.

2.1.57 [ECMA-262/6] Section 14.5.8 Static Semantics: IsFunctionDefinition

V0015: Nothing in this section is implemented

The specification states:

```
14.5.8 Static Semantics: IsFunctionDefinition

    See also: 12.2.1.3, 12.2.10.2, 12.3.1.2, 12.4.2, 12.5.2, 12.6.1, 12.7.1, 12.8.1,
    12.9.1, 12.10.1, 12.11.1, 12.12.1, 12.13.1, 12.14.2, 12.15.1, 14.1.11, 14.4.9.

    ClassExpression : class ClassTail
         1. Return true.

    ClassExpression : class BindingIdentifier ClassTail
         1. Return true.
```

IE11 Mode (All versions)

Nothing in this section is implemented.

2.1.58 [ECMA-262/6] Section 14.5.9 Static Semantics: IsStatic

V0016: Nothing in this section is implemented

The specification states:

```
14.5.9 Static Semantics: IsStatic

ClassElement : MethodDefinition
     1. Return false.

ClassElement : static MethodDefinition
     1. Return true.

ClassElement : ;
     1. Return false.
```

IE11 Mode (All versions)

Nothing in this section is implemented.

Copyright © 2017 Microsoft Corporation

2.1.59 [ECMA-262/6] Section 14.5.10 Static Semantics: NonConstructorMethodDefinitions

V0017: Section not implemented

The specification states:

```
14.5.10 Static Semantics: NonConstructorMethodDefinitions

    ClassElementList : ClassElement
         1. If ClassElement is the production ClassElement : ; , return a new empty List.
         2. If IsStatic of ClassElement is false and PropName of ClassElement is
            "constructor", return a new empty List.
         3. Return a List containing ClassElement.

    ClassElementList : ClassElementList ClassElement
         1. Let list be NonConstructorMethodDefinitions of ClassElementList.
         2. If ClassElement is the production ClassElement : ; , return list.
         3. If IsStatic of ClassElement is false and PropName of ClassElement is
            "constructor", return list.
         4. Append ClassElement to the end of list.
         5. Return list.
```

IE11 Mode (All versions)

Nothing in this section is implemented.

2.1.60 [ECMA-262/6] Section 14.5.11 Static Semantics: PrototypePropertyNameList

V0018: Nothing in this section is implemented

The specification states:

```
14.5.11 Static Semantics: PrototypePropertyNameList

    ClassElementList : ClassElement
        1.  If PropName of ClassElement is empty, return a new empty List.
        2.  If IsStatic of ClassElement is true, return a new empty List.
        3.  Return a List containing PropName of ClassElement.

    ClassElementList : ClassElementList ClassElement
        1.  Let list be PrototypePropertyNameList of ClassElementList.
        2.  If PropName of ClassElement is empty, return list.
        3.  If IsStatic of ClassElement is true, return list.
        4.  Append PropName of ClassElement to the end of list.
        5.  Return list.
```

IE11 Mode (All versions)

Nothing in this section is implemented.

2.1.61 [ECMA-262/6] Section 14.5.12 Static Semantics: PropName

V0019: Nothing in this section is implemented

The specification states:

Copyright © 2017 Microsoft Corporation

```
14.5.12 Static Semantics: PropName

    See also: 12.2.6.6, 14.3.6, 14.4.10

    ClassElement : ;
         1. Return empty.
```

IE11 Mode (All versions)

Nothing in this section is implemented.

2.1.62 [ECMA-262/6] Section 14.5.13 Static Semantics: StaticPropertyNameList

V0020: Nothing in this section is implemented

The specification states:

```
14.5.13 Static Semantics: StaticPropertyNameList

    ClassElementList : ClassElement
         1. If PropName of ClassElement is empty, return a new empty List.
         2. If IsStatic of ClassElement is false, return a new empty List.
         3. Return a List containing PropName of ClassElement.

    ClassElementList : ClassElementList ClassElement
         1. Let list be StaticPropertyNameList of ClassElementList.
         2. If PropName of ClassElement is empty, return list.
         3. If IsStatic of ClassElement is false, return list.
         4. Append PropName of ClassElement to the end of list.
         5. Return list.
```

IE11 Mode (All versions)

Nothing in this section is implemented.

2.1.63 [ECMA-262/6] 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).
```


Copyright © 2017 Microsoft Corporation

IE11 Mode and EdgeHTML Mode (All versions)

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

2.1.64 [ECMA-262/6] Section 14.5.15 Runtime Semantics: BindingClassDeclarationEvaluation

V0022: Nothing in this section is implemented

The specification states:

```
14.5.15 Runtime Semantics: BindingClassDeclarationEvaluation

    ClassDeclaration : class BindingIdentifier ClassTail
        1.  Let className be StringValue of BindingIdentifier.
        2.  Let value be the result of ClassDefinitionEvaluation of ClassTail with
            argument className.
        3.  ReturnIfAbrupt(value).
        4.  Let hasNameProperty be HasOwnProperty(value, "name").
        5.  ReturnIfAbrupt(hasNameProperty).
        6.  If hasNameProperty is false, then perform SetFunctionName(value, className).
        7.  Let env be the running execution context's LexicalEnvironment.
        8.  Let status be InitializeBoundName(className, value, env).
        9.  ReturnIfAbrupt(status).
        10. Return value.

    ClassDeclaration : class ClassTail
        1.  Return the result of ClassDefinitionEvaluation of ClassTail with argument
            undefined.
```

IE11 Mode (All versions)

Nothing in this section is implemented.

2.1.65 [ECMA-262/6] Section 14.5.16 Runtime Semantics: Evaluation

V0023: Nothing in this section is implemented

The specification states:

```
14.5.16 Runtime Semantics: Evaluation

    ClassDeclaration : class BindingIdentifier ClassTail
         1. Let status be the result of BindingClassDeclarationEvaluation of this
            ClassDeclaration.
         2. ReturnIfAbrupt(status).
         3. Return NormalCompletion(empty).

    ...

    ClassExpression : class BindingIdentifieropt ClassTail
         1. If BindingIdentifieropt is not present, let className be undefined.
         2. Else, let className be StringValue of BindingIdentifier.
         3. Let value be the result of ClassDefinitionEvaluation of ClassTail with
            argument className.
         4. ReturnIfAbrupt(value).
         5. If className is not undefined, then
```


Copyright © 2017 Microsoft Corporation

```
      a. Let hasNameProperty be HasOwnProperty(value, "name").
      b. ReturnIfAbrupt(hasNameProperty).
      c. If hasNameProperty is false, then
           i. Perform SetFunctionName(value, className).
   6. Return NormalCompletion(value).
```

IE11 Mode (All versions)

Nothing in this section is implemented.

2.1.66 [ECMA-262/6] 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.
```

IE11 Mode and EdgeHTML Mode (All versions)

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

2.1.67 [ECMA-262/6] Section 15.2.1 Module Semantics

V0024: The check for ModuleItemList containing NewTarget is not implemented

The specification states:

```
15.2.1.1 Static Semantics: Early Errors

    ModuleBody : ModuleItemList
        ...
        • It is a Syntax Error if ModuleItemList Contains NewTarget
        ...
```

IE11 Mode (All versions)

The check for `ModuleItemList` containing `NewTarget` is not implemented.

Copyright © 2017 Microsoft Corporation

2.1.68 [ECMA-262/6] Section 16.1 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.
```

IE11 Mode and EdgeHTML Mode (All versions)

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

2.1.69 [ECMA-262/6] Section 18.3.33 WeakSet (. . .)

V0072: The WeakSet API is not implemented, and there is no WeakSet property on the global object

The specification states:

```
18.3.33 WeakSet ( . . . )

    See 23.4.
```

IE11 Mode (All versions)

The `WeakSet` API is not implemented, and there is no `WeakSet` property on the global object.

2.1.70 [ECMA-262/6] Section 19.1.2 Properties of the Object Constructor

V0073: 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. ReturnIfAbrupt(O).
        3. If Type(proto) is neither Object nor Null, throw a TypeError exception.
```


Copyright © 2017 Microsoft Corporation

```
4. If Type(O) is not Object, return O.
5. Let status be O.[[SetPrototypeOf]](proto).
6. ReturnIfAbrupt(status).
7. If status is false, throw a TypeError exception.
8. Return O.
```

All document modes (All versions)

`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.

2.1.71 [ECMA-262/6] Section 19.1.3.2 Object.prototype.hasOwnProperty (V)

V0132: 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.  ReturnIfAbrupt(P).
        3.  Let O be ToObject(this value).
        4.  ReturnIfAbrupt(O).
        5.  Return HasOwnProperty(O, P).
```

All document modes (All versions)

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

2.1.72 [ECMA-262/6] Section 19.1.3.4 Object.prototype.propertyIsEnumerable (V)

V0133: Object.prototype.propertyIsEnumerable throws a TypeError if its argument is a symbol

The specification states:

```
19.1.3.4 Object.prototype.propertyIsEnumerable ( V )

    When the propertyIsEnumerable method is called with argument V, the following steps
    are taken:
         1. Let P be ToPropertyKey(V).
         2. ReturnIfAbrupt(P).
         3. Let O be ToObject(this value).
         4. ReturnIfAbrupt(O).
         5. Let desc be O.[[GetOwnProperty]](P).
         6. ReturnIfAbrupt(desc).
         7. If desc is undefined, return false.
         8. Return the value of desc.[[Enumerable]].
```

IE11 Mode (All versions)

Copyright © 2017 Microsoft Corporation

In step 1, `ToString(V)` is invoked instead of `ToPropertyKey(V)`. The result is that a TypeError is thrown if `V`] is a symbol or if an error occurs in coercing [InlineCode|V]] to a string.

2.1.73 [ECMA-262/6] Section 19.1.3.5 Object.prototype.toLocaleString ([reserved1 [, reserved2]])

V0134: 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").
```

All document modes (All versions)

`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. ReturnIfAbrupt(obj).
4. Return ToString(obj).

2.1.74 [ECMA-262/6] Section 19.1.3.6 Object.prototype.toString ( )

V0131: @@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.  ...
        ...
        16. Let tag be Get (O, @@toStringTag).
        ...
```

All document modes (All versions)

`@@toStringTag` is not implemented.

Copyright © 2017 Microsoft Corporation

2.1.75 [ECMA-262/6] Section 19.2.3.2 Function.prototype.bind (thisArg , ...args)

V0026: 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.
        ...
        12.  Let targetName be Get(Target, "name").
        13.  ReturnIfAbrupt(targetName).
        14.  If Type(targetName) is not String, let targetName be the empty string.
        15.  Perform SetFunctionName(F, targetName, "bound").
        16.  Return F.
```

All document modes (All versions)

Steps 12 to 15 are replaced by:

12. Let getName(Target) be a new dynamic function that does following:
    a. Let targetName be Get(Target, "name").
    b. ReturnIfAbrupt(targetName).
    c. Return "bound"+targetName
13. Set (F, "name", getName)

Because of this, the bound function name accessor calls the target function's counterpart. Note that steps 14 and 15 are deleted.

2.1.76 [ECMA-262/6] Section 19.2.3.6 Function.prototype[@@hasInstance] (V)

V0126: 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.
```

All document modes (All versions)

Copyright © 2017 Microsoft Corporation

Calling `@@hasInstance` has no effect.

2.1.77 [ECMA-262/6] Section 19.2.4 Function Instances

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 }.
```

All document modes (All versions)

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**.

2.1.78 [ECMA-262/6] Section 19.4 Symbol Objects

V0075: Symbol is not implemented

The specification states:

```
19.4 Symbol Objects
```

IE11 Mode (All versions)

`Symbol` is not implemented.

2.1.79 [ECMA-262/6] Section 19.4.1 The Symbol Constructor

V0160: Symbol is not implemented

The specification states:

```
19.4.1 The Symbol Constructor
```

IE11 Mode (All versions)

`Symbol` is not implemented.

Copyright © 2017 Microsoft Corporation

2.1.80 [ECMA-262/6] 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:
```

All document modes (All versions)

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

hasInstance

isConcatSpreadable

toPrimitive

toStringTag

V0162: Symbol is not implemented

The specification states:

```
19.4.2 Properties of the Symbol Constructor
```

IE11 Mode (All versions)

`Symbol` is not implemented.

2.1.81 [ECMA-262/6] Section 20.3.1.15 TimeClip (time)

V0027: 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 × 1015, return NaN.
        3.  Return ToInteger(time) + (+0). (Adding a positive zero converts −0 to +0.)
```

All document modes (All versions)

Copyright © 2017 Microsoft Corporation

`TimeClip` does not convert negative zero to positive zero in step 3. Instead, the step is implemented as:

3. Return ToInteger(time).

2.1.82 [ECMA-262/6] 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
```

All document modes (All versions)

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)

2.1.83 [ECMA-262/6] Section 21.1.3 Properties of the String Prototype Object

V0076: 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 itself an ordinary object. It is not a String instance and does
    not have a [[StringData]] internal slot.
```

All document modes (All versions)

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

Copyright © 2017 Microsoft Corporation

2.1.84 [ECMA-262/6] Section 21.1.3.11 String.prototype.match (regexp)

V0135: @@match is not implemented

The specification states:

```
21.1.3.11 String.prototype.match ( regexp )

    When the match method is called with argument regexp, the following steps are taken:

        ...
         3. If regexp is neither undefined nor null, then
            a. Let matcher be GetMethod(regexp, @@match).
            b. ReturnIfAbrupt(matcher).
            c. If matcher is not undefined, then
                  i. Return Call(matcher, regexp, «O»).
```

IE11 Mode (All versions)

`@@match` is not implemented.

2.1.85 [ECMA-262/6] Section 21.1.3.14 String.prototype.replace (searchValue, replaceValue)

V0136: @@replace is not implemented

The specification states:

```
21.1.3.14 String.prototype.replace (searchValue, replaceValue )

    When the replace method is called with arguments searchValue and replaceValue the
    following steps are taken:

        ...
         3. If searchValue is neither undefined nor null, then
            a. Let replacer be GetMethod(searchValue, @@replace).
            b. ReturnIfAbrupt(replacer).
            c. If replacer is not undefined, then
                 i. Return Call(replacer, searchValue, «O, replaceValue»).
```

IE11 Mode (All versions)

`@@replace` is not implemented.

2.1.86 [ECMA-262/6] Section 21.1.3.15 String.prototype.search (regexp)

V0137: @@search is not implemented

The specification states:

```
21.1.3.15 String.prototype.search ( regexp )

    When the search method is called with argument regexp, the following steps are taken:

        ...
```


Copyright © 2017 Microsoft Corporation

```
3. If regexp is neither undefined nor null, then
   a. Let searcher be GetMethod(regexp, @@search).
   b. ReturnIfAbrupt(searcher).
   c. If searcher is not undefined, then
        i. Return Call(searcher, regexp, «O»)
```

IE11 Mode (All versions)

`@@search` is not implemented.

2.1.87 [ECMA-262/6] Section 21.1.3.17 String.prototype.split (separator, limit)

V0138: @@split is not implemented

The specification states:

```
21.1.3.17 String.prototype.split ( separator, limit )

    Returns an Array object into which substrings of the result of converting this object
    to a String have been stored. The substrings are determined by searching from left to
    right for occurrences of separator; these occurrences are not part of any substring
    in the returned array, but serve to divide up the String value. The value of
    separator may be a String of any length or it may be an object, such as an RegExp,
    that has a @@split method.

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

        ...
         3. If separator is neither undefined nor null, then
            a. Let splitter be GetMethod(separator, @@split).
            b. ReturnIfAbrupt(splitter).
            c. If splitter is not undefined , then
                  i. Return Call(splitter, separator, «O, limit»).
```

IE11 Mode (All versions)

`@@split` is not implemented.

V0184: lim should be ToLength(limit) but may not be

The specification states:

```
21.1.3.17 String.prototype.split ( separator, limit )

    Returns an Array object into which substrings of the result of converting this object
    to a String have been stored. ...

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

        ...
        8.  If limit is undefined, let lim = 2^53–1; else let lim = ToLength(limit).
```

All document modes (All versions)

Copyright © 2017 Microsoft Corporation

If `limit` is not undefined, `lim` should be `ToLength(limit)`, but instead is taken to be the value of `limit` converted to a 32-bit unsigned value. For example, if `limit` = -1, `lim` is 0xFFFFFFFF, the largest 32-bit number. If `limit` = 0xA987654321, `lim` is 0x87654321.

2.1.88 [ECMA-262/6] 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).
```

IE11 Mode and EdgeHTML Mode (All versions)

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 (All versions)

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

2.1.89 [ECMA-262/6] Section 21.1.3.27 String.prototype [@@iterator]( )

V0141: The @@iterator method is not present on String.prototype

The specification states:

Copyright © 2017 Microsoft Corporation

```
21.1.3.27 String.prototype [ @@iterator ]( )

    When the @@iterator method is called it returns an Iterator object (25.1.1.2) that
    iterates over the code points of a String value, returning each code point as a
    String value. ...

        ...

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

IE11 Mode (All versions)

The `@@iterator` method is not present on `String.prototype`.

2.1.90 [ECMA-262/6] Section 21.2 RegExp (Regular Expression) Objects

V0077: The implementation of RegExp accords with edition 5.1 of the ECMAScript Standard, not the 6th edition

The specification states:

```
21.2 RegExp (Regular Expression) Objects

    A RegExp object contains a regular expression and the associated flags.

    NOTE The form and functionality of regular expressions is modelled after the regular
    expression facility in the Perl 5 programming language.
```

IE11 Mode (All versions)

The implementation of `RegExp`] accords with edition 5.1 of the ECMAScript Standard, not the 6th edition.

2.1.91 [ECMA-262/6] 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 }
```


Copyright © 2017 Microsoft Corporation

EdgeHTML Mode (All versions)

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}')
```

2.1.92 [ECMA-262/6] 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 (All versions)

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.

2.1.93 [ECMA-262/6] 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 (All versions)

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')
```


Copyright © 2017 Microsoft Corporation

2.1.94 [ECMA-262/6] 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 (All versions)

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');
```

2.1.95 [ECMA-262/6] 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.
```

All document modes (All versions)

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

Copyright © 2017 Microsoft Corporation

2.1.96 [ECMA-262/6] Section 21.2.5 Properties of the RegExp Prototype Object

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.
```

All document modes (All versions)

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.

2.1.97 [ECMA-262/6] 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 (All versions)

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.

2.1.98 [ECMA-262/6] 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 )
```


Copyright © 2017 Microsoft Corporation

```
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 (All versions)

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'
```

2.1.99 [ECMA-262/6] Section 21.2.5.3 get RegExp.prototype.flags

V0152: RegExp.prototype.flags is not implemented

The specification states:

```
21.2.5.3 get RegExp.prototype.flags

    RegExp.prototype.flags is an accessor property whose set accessor function is
    undefined. ...
```

IE11 Mode (All versions)

RegExp.prototype.flags is not implemented.

2.1.100 [ECMA-262/6] Section 21.2.5.4 get RegExp.prototype.global

V0144: RegExp.prototype.global is a data property, not an accessor property

The specification states:

```
21.2.5.4 get RegExp.prototype.global

RegExp.prototype.global is an accessor property whose set accessor function is undefined.
...
```

IE11 Mode (All versions)

Copyright © 2017 Microsoft Corporation

`RegExp.prototype.global` is a data property, not an accessor property. It has the following attributes:

{[[Writable]]: false, [[Enumerable]]: false, [[Configurable]]: false}

2.1.101 [ECMA-262/6] Section 21.2.5.5 get RegExp.prototype.ignoreCase

V0145: RegExp.prototype.ignoreCase is a data property, not an accessor property

The specification states:

```
21.2.5.5 get RegExp.prototype.ignoreCase

RegExp.prototype.ignoreCase is an accessor property whose set accessor function is
undefined. ...
```

IE11 Mode (All versions)

RegExp.prototype.ignoreCase is a data property, not an accessor property. It has the following attributes:

{[[Writable]]: false, [[Enumerable]]: false, [[Configurable]]: false}

2.1.102 [ECMA-262/6] Section 21.2.5.6 RegExp.prototype [@@match] (string)

V0147: The @@match method is not implemented

The specification states:

```
21.2.5.6 RegExp.prototype [ @@match ] ( string )

    When the @@match method is called with argument string, the following steps are taken:
```

IE11 Mode (All versions)

The `@@match` method is not implemented.

2.1.103 [ECMA-262/6] Section 21.2.5.7 get RegExp.prototype.multiline

V0146: RegExp.prototype.multiline is a data property, not an accessor property

The specification states:

```
21.2.5.7 get RegExp.prototype.multiline

RegExp.prototype.multiline is an accessor property whose set accessor function is
undefined. ...
```

IE11 Mode (All versions)

Copyright © 2017 Microsoft Corporation

`RegExp.prototype.multiline` is a data property, not an accessor property. It has the following attributes:

{[[Writable]]: false, [[Enumerable]]: false, [[Configurable]]: false}

2.1.104 [ECMA-262/6] Section 21.2.5.8 RegExp.prototype [@@replace] (string, replaceValue)

V0176: RegExp.prototype[@@replace] is not implemented

The specification states:

```
21.2.5.8 RegExp.prototype [ @@replace ] ( string, replaceValue )

    When the @@replace method is called with arguments string and replaceValue the
    following steps are taken:
```

EdgeHTML Mode (All versions)

`@@replace` is not implemented.

2.1.105 [ECMA-262/6] Section 21.2.5.9 RegExp.prototype [@@search] (string)

V0148: The @@search method is not implemented

The specification states:

```
21.2.5.9 RegExp.prototype [ @@search ] ( string )

    When the @@search method is called with argument string, the following steps are
    taken:
```

IE11 Mode (All versions)

The `@@search` method is not implemented.

2.1.106 [ECMA-262/6] Section 21.2.5.10 get RegExp.prototype.source

V0143: RegExp.prototype.source is a data property, not an accessor property

The specification states:

```
21.2.5.10 get RegExp.prototype.source

RegExp.prototype.source is an accessor property whose set accessor function is undefined.
...
```

IE11 Mode (All versions)

RegExp.prototype.source is a data property, not an accessor property. It has the following attributes:

Copyright © 2017 Microsoft Corporation

{[[Writable]]: false, [[Enumerable]]: false, [[Configurable]]: false}

2.1.107 [ECMA-262/6] Section 21.2.5.11 RegExp.prototype [@@split] (string, limit)

V0149: The @@split method is not implemented

The specification states:

```
21.2.5.11 RegExp.prototype [ @@split ] ( string, limit )
    ...
    When the @@split method is called, the following steps are taken:
```

IE11 Mode (All versions)

The `@@split` method is not implemented.

2.1.108 [ECMA-262/6] Section 21.2.5.12 get RegExp.prototype.sticky

V0150: RegExp.prototype.sticky is a data property, not an accessor property

The specification states:

```
21.2.5.12 get RegExp.prototype.sticky

    RegExp.prototype.sticky is an accessor property whose set accessor function is
    undefined. ...
```

IE11 Mode (All versions)

`RegExp.prototype.sticky` is a data property, not an accessor property. It has the following attributes:

{[[Writable]]: false, [[Enumerable]]: false, [[Configurable]]: false}

2.1.109 [ECMA-262/6] Section 21.2.5.15 get RegExp.prototype.unicode

V0151: RegExp.prototype.unicode is a data property, not an accessor property

The specification states:

```
21.2.5.15 get RegExp.prototype.unicode

RegExp.prototype.unicode is an accessor property whose set accessor function is
undefined. ...
```

IE11 Mode (All versions)

RegExp.prototype.unicode is a data property, not an accessor property. It has the following attributes:

Copyright © 2017 Microsoft Corporation

{[[Writable]]: false, [[Enumerable]]: false, [[Configurable]]: false}

2.1.110 [ECMA-262/6] Section 21.2.6 Properties of RegExp Instances

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 }.
```

All document modes (All versions)

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).

2.1.111 [ECMA-262/6] Section 22.1.3.1.1 Runtime Semantics: IsConcatSpreadable (O)

V0155: @@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.  ReturnIfAbrupt(spreadable).
        4.  If spreadable is not undefined, return ToBoolean(spreadable).
        5.  Return IsArray(O).
```

All document modes (All versions)

`@@isConcatSpreadable` is not implemented.

2.1.112 [ECMA-262/6] Section 22.1.3.3 Array.prototype.copyWithin (target, start [, end])

V0028: 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 ] )
```


Copyright © 2017 Microsoft Corporation

```
...
The following steps are taken:
    ...
    17. Repeat, while count > 0
        a.  Let fromKey be ToString(from).
        b.  Let toKey be ToString(to).
        c.  Let fromPresent be HasProperty(O, fromKey).
        d.  ReturnIfAbrupt(fromPresent).
        e.  If fromPresent is true, then
            i.  Let fromVal be Get(O, fromKey).
            ii. ReturnIfAbrupt(fromVal).
            iii. Let setStatus be Set(O, toKey, fromVal, true).
            iv. ReturnIfAbrupt(setStatus).
        f.  Else fromPresent is false,
            i.  Let deleteStatus be DeletePropertyOrThrow(O, toKey).
            ii. ReturnIfAbrupt(deleteStatus).
        g.  Let from be from + direction.
        h.  Let to be to + direction.
        i.  Let count be count – 1.
    18. Return O.
```

All document modes (All versions)

The following steps are not executed:

17.

f. Else fromPresent is false,

i. Let deleteStatus be DeletePropertyOrThrow(O, toKey).

ii. ReturnIfAbrupt(deleteStatus).

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

2.1.113 [ECMA-262/6] Section 22.1.3.4 Array.prototype.entries ( )

V0083: The Array.prototype object does not have an entries property

The specification states:

```
22.1.3.4 Array.prototype.entries ( )

    The following steps are taken:
         1. Let O be ToObject(this value).
         2. ReturnIfAbrupt(O).
         3. Return CreateArrayIterator(O, "key+value").
```

IE11 Mode (All versions)

The `Array.prototype` object does not have an `entries` property.

2.1.114 [ECMA-262/6] Section 22.1.3.13 Array.prototype.keys ( )

V0156: The Array.prototype object does not have a keys property

Copyright © 2017 Microsoft Corporation

The specification states:

```
22.1.3.13 Array.prototype.keys ( )

    The following steps are taken:

         1. Let O be ToObject(this value).
         2. ReturnIfAbrupt(O).
         3. Return CreateArrayIterator(O, "key").
```

IE11 Mode (All versions)

The `Array.prototype` object does not have a `keys` property.

2.1.115 [ECMA-262/6] Section 22.1.3.17 Array.prototype.push (...items)

V0029: Array.prototype.push does not throw TypeError on length overflow

The specification states:

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

All document modes (All versions)

The following step is not executed:

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

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

2.1.116 [ECMA-262/6] Section 22.1.3.24 Array.prototype.sort (comparefn)

V0030: Array.prototype.sort uses ToUint32 for length conversion

The specification states:

```
22.1.3.24 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.
```


Copyright © 2017 Microsoft Corporation

```
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")).
    3.  ReturnIfAbrupt(len).
```

All document modes (All versions)

`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")).
3. ReturnIfAbrupt(len).

2.1.117 [ECMA-262/6] Section 22.1.3.26 Array.prototype.toLocaleString ([reserved1 [, reserved2]])

V0031: Array.prototype.toLocaleString uses InvokeBuiltinMethod instead of Invoke

The specification states:

```
22.1.3.26 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:
        ...
       10. Else
            a.  Let R be ToString(Invoke(firstElement, "toLocaleString")).
            b.  ReturnIfAbrupt(R).
        ...
        12. Repeat, while k < len
            ...
            e.  Else
                i.  Let R be ToString(Invoke(nextElement, "toLocaleString")).
                ii. ReturnIfAbrupt(R).
```

All document modes (All versions)

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

...

10. Else

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

    b. ReturnIfAbrupt(R).

...

Copyright © 2017 Microsoft Corporation

12. Repeat, while k < len

    ...

    e. Else

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

        ii. ReturnIfAbrupt(R).

2.1.118 [ECMA-262/6] Section 22.1.3.29 Array.prototype.values ( )

V0157: The Array.prototype object does not have a values property

The specification states:

```
22.1.3.29 Array.prototype.values ( )

    The following steps are taken:

         1. Let O be ToObject(this value).
         2. ReturnIfAbrupt(O).
         3. Return CreateArrayIterator(O, "value").

This function is the %ArrayProto_values% intrinsic object.
```

IE11 Mode (All versions)

The `Array.prototype` object does not have a `values` property.

2.1.119 [ECMA-262/6] Section 22.1.3.30 Array.prototype [@@iterator] ( )

V0158: The Array.prototype object does not have an @@iterator property

The specification states:

```
22.1.3.30 Array.prototype [ @@iterator ] ( )

    The initial value of the @@iterator property is the same function object as the
    initial value of the Array.prototype.values property.
```

IE11 Mode (All versions)

The `Array.prototype` object does not have an `@@iterator` property.

2.1.120 [ECMA-262/6] Section 22.1.3.31 Array.prototype [@@unscopables]

V0166: Object.getOwnPropertyDescriptor(Array.prototype, Symbol.unscopables).writable is true but should be false

The specification states:

Copyright © 2017 Microsoft Corporation

```
22.1.3.31 Array.prototype [ @@unscopables ]

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

IE11 Mode (All versions)

`Object.getOwnPropertyDescriptor(Array.prototype, Symbol.unscopables).writable` is true but should be false.

V0167: Object.getPrototypeOf(Array.prototype[Symbol.unscopables]) is not null

The specification states:

```
22.1.3.31 Array.prototype [ @@unscopables ]

    The initial value of the @@unscopables data property is an object created by the
    following steps:

        1. Let blackList be ObjectCreate(null).
```

IE11 Mode (All versions)

`Object.getPrototypeOf(Array.prototype[ Symbol.unscopables ])` is not `null`, but should be.

2.1.121 [ECMA-262/6] Section 22.2.1.1 %TypedArray% ( )

V0084: TypedArray constructors can be called without the new keyword

The specification states:

```
22.2.1.1 %TypedArray% ( )

    This description applies only if the %TypedArray% function is called with no
    arguments.

        1. If NewTarget is undefined, throw a TypeError exception.
        2. Return AllocateTypedArray(NewTarget, 0).
```

IE11 Mode (All versions)

`%TypedArray%` does not check that `NewTarget` is not undefined. Since `TypedArray` constructors use `%TypedArray%` to construct their return object, the consequence is that `TypedArray` constructors can be called without the `new` keyword.

This affects the following constructors:

Int8Array

Uint8Array

Uint8ClampedArray

Int16Array

Copyright © 2017 Microsoft Corporation

Uint16Array

Int32Array

Uint32Array

Float32Array

Float64Array

2.1.122 [ECMA-262/6] Section 22.2.2.1 %TypedArray%.from (source [, mapfn [, thisArg]])

V0085: %TypedArray%.from and %TypedArray%.of are not implemented in IE11

The specification states:

```
22.2.2.1 %TypedArray%.from ( source [ , mapfn [ , thisArg ] ] )

    When the from method is called with argument source, and optional arguments mapfn and
    thisArg, the following steps are taken:
        ...
         6. Return TypedArrayFrom(C, source, f, t).

    The length property of the from method is 1.
```

IE11 Mode (All versions)

`%TypedArray%.from` is not implemented.

2.1.123 [ECMA-262/6] Section 22.2.2.1.1 Runtime Semantics: TypedArrayFrom(constructor, items, mapfn, thisArg)

V0087: @@iterator is not supported, so the TypedArray constructors do not accept iterable arguments

The specification states:

```
22.2.2.1.1 Runtime Semantics: TypedArrayFrom( constructor, items, mapfn, thisArg )

    When the TypedArrayFrom abstract operation is called with arguments constructor,
    items, mapfn, and thisArg, the following steps are taken:

         1. Let C be constructor.
        ...
         6. Let usingIterator be GetMethod(items, @@iterator).
         7. ReturnIfAbrupt(usingIterator).
         8. If usingIterator is not undefined, then
            a. Let iterator be GetIterator(items, usingIterator).
            b. ReturnIfAbrupt(iterator).
            ...
         9. Assert: items is not an Iterable so assume it is an array-like object.
        10. Let arrayLike be ToObject(items).
        ...
        18. Return targetObj.
```


Copyright © 2017 Microsoft Corporation

IE11 Mode (All versions)

`@@iterator` is not supported, so the `TypedArray` constructors do not accept iterable arguments. Step 10 is reached for every argument.

This affects the following constructors:

Int8Array

Uint8Array

Uint8ClampedArray

Int16Array

Uint16Array

Int32Array

Uint32Array

Float32Array

Float64Array

2.1.124 [ECMA-262/6] Section 22.2.2.2 %TypedArray%.of (...items)

V0159: %TypedArray%.of is not implemented

The specification states:

```
22.2.2.2 %TypedArray%.of ( ...items )

    When the of method is called with any number of arguments, the following steps are
    taken:
        ...
         9. Return newObj.

    The length property of the of method is 0.
```

IE11 Mode (All versions)

`%TypedArray%.of` is not implemented.

2.1.125 [ECMA-262/6] Section 22.2.3 Properties of the %TypedArrayPrototype% Object

V0086: %TypedArrayPrototype% has incorrect properties

The specification states:

```
22.2.3 Properties of the %TypedArrayPrototype% Object

    The value of the [[Prototype]] internal slot of the %TypedArrayPrototype% object is
    the intrinsic object %ObjectPrototype% (19.1.3). The %TypedArrayPrototype% object is
```


Copyright © 2017 Microsoft Corporation

```
an ordinary object. It does not have a [[ViewedArrayBuffer]] or any other of the
internal slots that are specific to TypedArray instance objects.
```

IE11 Mode (All versions)

The following properties of [InlineCode|%TypedArrayPrototype% are not implemented:

```
%TypedArrayPrototype%.join

%TypedArrayPrototype%.indexOf

%TypedArrayPrototype%.lastIndexOf

%TypedArrayPrototype%.slice

%TypedArrayPrototype%.every

%TypedArrayPrototype%.filter

%TypedArrayPrototype%.forEach

%TypedArrayPrototype%.map

%TypedArrayPrototype%.reduce

%TypedArrayPrototype%.reduceRight

%TypedArrayPrototype%.reverse

%TypedArrayPrototype%.some

%TypedArrayPrototype%.sort

%TypedArrayPrototype%.copyWithin

%TypedArrayPrototype%.find

%TypedArrayPrototype%.findIndex

%TypedArrayPrototype%.fill

%TypedArrayPrototype%.keys

%TypedArrayPrototype%.values

%TypedArrayPrototype%.entries

%TypedArrayPrototype%[@@iterator]
```

The following properties are located on `TypedArray` constructor prototypes instead of `%TypedArrayPrototype%`, and they are data properties rather than accessor functions:

```
%TypedArrayPrototype%.buffer

%TypedArrayPrototype%.byteLength

%TypedArrayPrototype%.byteOffset

%TypedArrayPrototype%.set

%TypedArrayPrototype%.subarray
```


Copyright © 2017 Microsoft Corporation

2.1.126 [ECMA-262/6] Section 22.2.5 Properties of the TypedArray Constructors

V0088: TypedArray constructors have their internal prototype slot set to Function.prototype instead of %TypedArray%

The specification states:

```
22.2.5 Properties of the TypedArray Constructors

    The value of the [[Prototype]] internal slot of each TypedArray constructor is the
    %TypedArray% intrinsic object (22.2.1).

    Each TypedArray constructor has a [[TypedArrayConstructorName]] internal slot
    property whose value is the String value of the constructor name specified for it in
    Table 49.

    Each TypedArray constructor has a name property whose value is the String value of
    the constructor name specified for it in Table 49.

    Besides a length property (whose value is 3), each TypedArray constructor has the
    following properties:
```

IE11 Mode (All versions)

`TypedArray` constructors have their internal prototype slot set to `Function.prototype` instead of `%TypedArray%`.

V0089: TypedArray constructors should have a name property corresponding to the TypedArray constructor global name

The specification states:

```
22.2.5 Properties of the TypedArray Constructors

    The value of the [[Prototype]] internal slot of each TypedArray constructor is the
    %TypedArray% intrinsic object (22.2.1).

    Each TypedArray constructor has a [[TypedArrayConstructorName]] internal slot
    property whose value is the String value of the constructor name specified for it in
    Table 49.

    Each TypedArray constructor has a name property whose value is the String value of
    the constructor name specified for it in Table 49.

    Besides a length property (whose value is 3), each TypedArray constructor has the
    following properties:
```

IE11 Mode (All versions)

`TypedArray` constructors should have a name property corresponding to the `TypedArray` constructor global name.

Copyright © 2017 Microsoft Corporation

2.1.127 [ECMA-262/6] Section 22.2.6 Properties of TypedArray Prototype Objects

V0090: TypedArray prototype objects have own properties that should instead be on %TypedArrayPrototype%

The specification states:

```
22.2.6Properties of TypedArray Prototype Objects
The value of the [[Prototype]] internal slot of a TypedArray prototype object is the
intrinsic object %TypedArrayPrototype% (22.2.3). A TypedArray prototype object is an
ordinary object. It does not have a [[ViewedArrayBuffer]] or any other of the internal
slots that are specific to TypedArray instance objects.

22.2.6.1TypedArray.prototype.BYTES_PER_ELEMENT
The value of TypedArray.prototype.BYTES_PER_ELEMENT is the Number value of the Element
Size value specified in Table 49 for TypedArray.
This property has the attributes { [[Writable]]: false, [[Enumerable]]: false,
[[Configurable]]: false }.

22.2.6.2TypedArray.prototype.constructor
The initial value of a TypedArray.prototype.constructor is the corresponding %TypedArray%
intrinsic object.
```

IE11 Mode (All versions)

`TypedArray` prototype objects have these own properties that should instead be on `%TypedArrayPrototype%`:

`buffer`

`byteOffset`

`byteLength`

`set`

`subarray`

2.1.128 [ECMA-262/6] Section 22.2.7 Properties of TypedArray Instances

V0091: TypedArray instances have own properties that should instead be on %TypedArrayPrototype% or TypedArray.prototype

The specification states:

```
22.2.7 Properties of TypedArray Instances

    TypedArray instances are Integer Indexed exotic objects. Each TypedArray instance
    inherits properties from the corresponding TypedArray prototype object. Each
    TypedArray instance has the following internal slots: [[TypedArrayName]],
    [[ViewedArrayBuffer]], [[ByteLength]], [[ByteOffset]], and [[ArrayLength]].
```

IE11 Mode (All versions)

`TypedArray` instances have these own properties that should instead be on `%TypedArrayPrototype%` or `TypedArray.prototype`:

Copyright © 2017 Microsoft Corporation

BYTES_PER_ELEMENT

buffer

byteLength

byteOffset

2.1.129 [ECMA-262/6] Section 23.1.1.1 Map ([iterable])

V0092: The Map constructor ignores the iterable parameter

The specification states:

```
23.1.1.1 Map ( [ iterable ] )

    When the Map function is called with optional argument the following steps are taken:

         1. If NewTarget is undefined, throw a TypeError exception.
         2. Let map be OrdinaryCreateFromConstructor(NewTarget, "%MapPrototype%",
            «[[MapData]]» ).
         3. ReturnIfAbrupt(map).
         4. Set map's [[MapData]] internal slot to a new empty List.
         5. If iterable is not present, let iterable be undefined.
         6. If iterable is either undefined or null, let iter be undefined.
        ...
         9. Repeat
            a. Let next be IteratorStep(iter).
            ...
            l. If status is an abrupt completion, return IteratorClose(iter, status).
```

IE11 Mode (All versions)

The `Map` constructor ignores the `iterable` parameter; it replaces steps 5 to 9 with:

5. return map

2.1.130 [ECMA-262/6] Section 23.1.3 Properties of the Map Prototype Object

V0093: The entries, keys, values, and @@iterator properties are missing from the Map.prototype object

The specification states:

```
23.1.3 Properties of the Map Prototype Object

    The Map prototype object is the intrinsic object %MapPrototype%. The value of the
    [[Prototype]] internal slot of the Map prototype object is the intrinsic object
    %ObjectPrototype% (19.1.3). The Map prototype object is an ordinary object. It does
    not have a [[MapData]] internal slot.
```

IE11 Mode (All versions)

These properties are missing from the `Map.prototype` object:

Copyright © 2017 Microsoft Corporation

entries (see 23.1.3.4)

keys (see 23.1.3.8)

values (see 23.1.3.11)

@@iterator (see 23.1.3.12)

2.1.131 [ECMA-262/6] Section 23.2 Set Objects

V0094: The Set constructor ignores its iterable parameter

The specification states:

```
23.2.1.1 Set ( [ iterable ] )

    When the Set function is called with optional argument iterable the following steps
    are taken:
         1. If NewTarget is undefined, throw a TypeError exception.
         2. Let set be OrdinaryCreateFromConstructor(NewTarget, "%SetPrototype%",
            «[[SetData]]» ).
         3. ReturnIfAbrupt(set).
         4. Set set's [[SetData]] internal slot to a new empty List.
         5. If iterable is not present, let iterable be undefined.
         6. If iterable is either undefined or null, let iter be undefined.
        ...
         9. Repeat
            a. Let next be IteratorStep(iter).
            ...
            g. If status is an abrupt completion, return IteratorClose(iter, status).
```

IE11 Mode (All versions)

The `Set` constructor ignores its `iterable` parameter; steps 5 and later are replaced with:

5. return set

2.1.132 [ECMA-262/6] Section 23.2.3 Properties of the Set Prototype Object

V0095: The entries, keys, values, and [@@iterator] properties are missing from the Set.prototype object

The specification states:

```
23.2.3 Properties of the Set Prototype Object

    The Set prototype object is the intrinsic object %SetPrototype%. The value of the
    [[Prototype]] internal slot of the Set prototype object is the intrinsic object
    %ObjectPrototype% (19.1.3). The Set prototype object is an ordinary object. It does
    not have a [[SetData]] internal slot.
```

IE11 Mode (All versions)

These properties are missing from the `Set.prototype` object:

Copyright © 2017 Microsoft Corporation

entries (see 23.2.3.5)

keys (see 23.2.3.8)

values (see 23.2.3.10)

@@iterator (see 23.2.3.11)

2.1.133 [ECMA-262/6] Section 23.3.1.1 WeakMap ([iterable])

V0096: The WeakMap constructor ignores its iterable parameter

The specification states:

```
23.3.1.1 WeakMap ( [ iterable ] )

    When the WeakMap function is called with optional argument iterable the following
    steps are taken:

        1. If NewTarget is undefined, throw a TypeError exception.
        2. Let map be OrdinaryCreateFromConstructor(NewTarget, "%WeakMapPrototype%",
           «[[WeakMapData]]» ).
        3. ReturnIfAbrupt(map).
        4. Set map's [[WeakMapData]] internal slot to a new empty List.
        5. If iterable is not present, let iterable be undefined.
        6. If iterable is either undefined or null, let iter be undefined.
        9. Repeat
           a. Let next be IteratorStep(iter).
           ...
           l. If status is an abrupt completion, return IteratorClose(iter, status).
```

IE11 Mode (All versions)

The `WeakMap` constructor ignores its iterable parameter; Steps 5 and below are replaced by:

5. return map

2.1.134 [ECMA-262/6] Section 24.1.3.3 get ArrayBuffer [@@species]

V0097: ArrayBuffer[@@species] is not implemented

The specification states:

```
24.1.3.3 get ArrayBuffer [ @@species ]

    ArrayBuffer[@@species] is an accessor property whose set accessor function is
    undefined. Its get accessor function performs the following steps:

        1. Return the this value.

    The value of the name property of this function is "get [Symbol.species]".
```

IE11 Mode (All versions)

`ArrayBuffer[@@species]` is not implemented.

Copyright © 2017 Microsoft Corporation

2.1.135 [ECMA-262/6] Section 25.2 GeneratorFunction Objects

V0174: GeneratorFunction Objects are not implemented

The specification states:

```
25.2 GeneratorFunction Objects

    Generator Function objects are constructor functions that are usually created by
    evaluating GeneratorDeclaration, GeneratorExpression, and GeneratorMethod syntactic
    productions. They may also be created by calling the %GeneratorFunction% intrinsic.
```

IE11 Mode (All versions)

`GeneratorFunction` Objects are not implemented.

2.1.136 [ECMA-262/6] Section 25.3 Generator Objects

V0098: Generator Objects are not implemented

The specification states:

```
25.3 Generator Objects

    A Generator object is an instance of a generator function and conforms to both the
    Iterator and Iterable interfaces.

    Generator instances directly inherit properties from the object that is the value of
    the prototype property of the Generator function that created the instance. Generator
    instances indirectly inherit properties from the Generator Prototype intrinsic,
    %GeneratorPrototype%.
```

IE11 Mode (All versions)

`Generator` Objects are not implemented.

2.1.137 [ECMA-262/6] Section 25.3.3 Generator Abstract Operations

V0101: Generator abstract operations are not implemented

The specification states:

```
25.3.3 Generator Abstract Operations
```

IE11 Mode (All versions)

`Generator` abstract operations are not implemented.

Copyright © 2017 Microsoft Corporation

2.1.138 [ECMA-262/6] Section 25.4 Promise Objects

V0102: Promise objects are not implemented

The specification states:

```
25.4 Promise Objects

    A Promise is an object that is used as a placeholder for the eventual results of a
    deferred (and possibly asynchronous) computation.

    Any Promise object is in one of three mutually exclusive states: fulfilled, rejected,
    and pending:

        • A promise p is fulfilled if p.then(f, r) will immediately enqueue a Job to call
        the function f.
        • A promise p is rejected if p.then(f, r) will immediately enqueue a Job to call
        the function r.
        • A promise is pending if it is neither fulfilled nor rejected.

    A promise is said to be settled if it is not pending, i.e. if it is either fulfilled
    or rejected.

    A promise is resolved if it is settled or if it has been "locked in" to match the
    state of another promise. Attempting to resolve or reject a resolved promise has no
    effect. A promise is unresolved if it is not resolved. An unresolved promise is
    always in the pending state. A resolved promise may be pending, fulfilled or rejected.
```

IE11 Mode (All versions)

`Promise` objects are not implemented.

2.1.139 [ECMA-262/6] Section 25.4.2 Promise Jobs

V0103: Promise jobs are not implemented

The specification states:

```
25.4.2 Promise Jobs
```

IE11 Mode (All versions)

`Promise` jobs are not implemented.

2.1.140 [ECMA-262/6] Section 25.4.3 The Promise Constructor

V0104: The Promise constructor is not implemented

The specification states:

```
25.4.3 The Promise Constructor

    The Promise constructor is the %Promise% intrinsic object and the initial value of
    the Promise property of the global object. When called as a constructor it creates
    and initializes a new Promise object. Promise is not intended to be called as a
```


Copyright © 2017 Microsoft Corporation

```
function and will throw an exception when called in that manner.

The Promise constructor is designed to be subclassable. It may be used as the value
in an extends clause of a class definition. Subclass constructors that intend to
inherit the specified Promise behaviour must include a super call to the Promise
constructor to create and initialize the subclass instance with the internal state
necessary to support the Promise and Promise.prototype built-in methods.
```

IE11 Mode (All versions)

The `Promise` constructor is not implemented.

2.1.141 [ECMA-262/6] 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:
```

All document modes (All versions)

The `Promise.length` property is not configurable.

2.1.142 [ECMA-262/6] Section 25.4.4.1 Promise.all (iterable)

V0105: 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 S be Get(C, @@species).
        4.  ReturnIfAbrupt(S).
        5.  If S is neither undefined nor null, let C be S.
        6.  Let promiseCapability be NewPromiseCapability(C).
        7.  ReturnIfAbrupt(promiseCapability).
        8.  Let iterator be GetIterator(iterable).
        9.  IfAbruptRejectPromise(iterator, promiseCapability).
        10. Let iteratorRecord be Record {[[iterator]]: iterator, [[done]]: false}.
        11. Let result be PerformPromiseAll(iteratorRecord, C, promiseCapability).
        12. If result is an abrupt completion,
            a.  If iteratorRecord.[[done]] is false, let result be
            IteratorClose(iterator, result).
```


Copyright © 2017 Microsoft Corporation

```
        b.  IfAbruptRejectPromise(result, promiseCapability).
    13. Return Completion(result).
```

All document modes (All versions)

Step 12a is not done; the `IteratorClose` abstract operation is not implemented.

2.1.143 [ECMA-262/6] Section 25.4.5 Properties of the Promise Prototype Object

V0109: Promise is not implemented

The specification states:

```
25.4.5 Properties of the Promise Prototype Object

    The Promise prototype object is the intrinsic object %PromisePrototype%. The value of
    the [[Prototype]] internal slot of the Promise prototype object is the intrinsic
    object %ObjectPrototype% (19.1.3). The Promise prototype object is an ordinary
    object. It does not have a [[PromiseState]] internal slot or any of the other
    internal slots of Promise instances.
```

IE11 Mode (All versions)

`Promise` is not implemented.

2.1.144 [ECMA-262/6] Section 25.4.5.4 Promise.prototype [@@toStringTag]

V0108: Promise.prototype[@@toStringTag] is not implemented

The specification states:

```
25.4.5.4 Promise.prototype [ @@toStringTag ]

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

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

IE11 Mode (All versions)

`Promise.prototype[ @@toStringTag ]` is not implemented.

2.1.145 [ECMA-262/6] Section 25.4.6 Properties of Promise Instances

V0110: Promise is not implemented

The specification states:

```
25.4.6 Properties of Promise Instances
```


Copyright © 2017 Microsoft Corporation

```
    Promise instances are ordinary objects that inherit properties from the Promise
    prototype object (the intrinsic, %PromisePrototype%). Promise instances are initially
    created with the internal slots described in Table 59.
```

IE11 Mode (All versions)

`Promise` is not implemented.

2.1.146 [ECMA-262/6] Section B.1.4 Regular Expressions Patterns

V0111: The regular expression pattern modifications and extensions are applied to Unicode patterns

The specification states:

```
B.1.4 Regular Expressions Patterns

    The syntax of 21.2.1 is modified and extended as follows. These changes introduce
    ambiguities that are broken by the ordering of grammar productions and by contextual
    information. When parsing using the following grammar, each alternative is considered
    only if previous production alternatives do not match.

    This alternative pattern grammar and semantics only changes the syntax and semantics
    of BMP patterns. The following grammar extensions include productions parameterized
    with the [U] parameter. However, none of these extensions change the syntax of
    Unicode patterns recognized when parsing with the [U] parameter present on the goal
    symbol.
```

EdgeHTML Mode (All versions)

The modifications and extensions described in this section are applied to Unicode patterns (i.e., when the `RegExp` "u" flag is specified), but shouldn't be.

For example, /[\d-a]/u should fail whereas /[\d-a]/ is allowed. The only difference between these two regular expressions is the "u" flag.

V0112: RegExp allows additional \c patterns

The specification states:

```
B.1.4 Regular Expressions Patterns

    The syntax of 21.2.1 is modified and extended as follows. These changes introduce
    ambiguities that are broken by the ordering of grammar productions and by contextual
    information. When

parsing using the following grammar, each alternative is considered only if previous
production alternatives do not match.

    This alternative pattern grammar and semantics only changes the syntax and semantics
    of BMP patterns. The following grammar extensions include productions parameterized
    with the [U]

parameter. However, none of these extensions change the syntax of Unicode patterns
recognized when parsing with the [U] parameter present on the goal symbol.

    Syntax
```


Copyright © 2017 Microsoft Corporation

```
Term[U] ::
    [~U] ExtendedTerm
    [+U] Assertion[U]
    [+U] Atom[U]
    [+U] Atom[U] Quantifier

ExtendedTerm ::
    Assertion
    AtomNoBrace Quantifier
    Atom
    QuantifiableAssertion Quantifier

AtomNoBrace ::
    PatternCharacterNoBrace
    .
    \ AtomEscape
    CharacterClass
    ( Disjunction )
    ( ? : Disjunction )

Atom[U] ::
    PatternCharacter
    .
    \ AtomEscape[?U]
    CharacterClass[?U]
    ( Disjunction[?U] )
    ( ? : Disjunction[?U] )

PatternCharacterNoBrace ::
    SourceCharacter but not one of
    ^ $ \ . * + ? ( ) [ ] { } |

PatternCharacter ::
    SourceCharacter but not one of
    ^ $ \ . * + ? ( ) [ ] |

QuantifiableAssertion ::
    ( ? = Disjunction )
    ( ? ! Disjunction )

Assertion[U] ::
    ^
    $
    \ b
    \ B
    [+U] ( ? = Disjunction[U] )
    [+U] ( ? ! Disjunction[U] )
    [~U] QuantifiableAssertion

AtomEscape[U] ::
    [+U] DecimalEscape
    [+U] CharacterEscape[U]
    [+U] CharacterClassEscape
    [~U] DecimalEscape but only if the integer value of DecimalEscape is <=
    NCapturingParens
    [~U] CharacterClassEscape
    [~U] CharacterEscape

CharacterEscape[U] ::
    ControlEscape
    c ControlLetter
    HexEscapeSequence
    RegExpUnicodeEscapeSequence[?U]
    [~U] LegacyOctalEscapeSequence
    IdentityEscape[?U]

IdentityEscape[U] ::
```


Copyright © 2017 Microsoft Corporation

```
        [+U] SyntaxCharacter
        [+U] /
        [~U] SourceCharacter but not c

    NonemptyClassRanges[U] ::
        ClassAtom[?U]
        ClassAtom[?U] NonemptyClassRangesNoDash[?U]
        [+U] ClassAtom[U] - ClassAtom[U] ClassRanges[U]
        [~U] ClassAtomInRange - ClassAtomInRange ClassRanges

    NonemptyClassRangesNoDash[U] ::
        ClassAtom[?U]
        ClassAtomNoDash[?U] NonemptyClassRangesNoDash[?U]
        [+U] ClassAtomNoDash[U] - ClassAtom[U] ClassRanges[U]
        [~U] ClassAtomNoDashInRange - ClassAtomInRange ClassRanges

    ClassAtom[U] ::
        -
        ClassAtomNoDash[?U]

    ClassAtomNoDash[U] ::
        SourceCharacter but not one of \ or ] or -
        \ ClassEscape[?U]

    ClassAtomInRange ::
        -
        ClassAtomNoDashInRange

    ClassAtomNoDashInRange ::
        SourceCharacter but not one of \ or ] or -
        \ ClassEscape but only if ClassEscape evaluates to a CharSet with exactly one
        character
        \ IdentityEscape

    ClassEscape[U] ::
        [+U] DecimalEscape
        [+U] CharacterEscape[U]
        [+U] CharacterClassEscape
        [~U] DecimalEscape
        b
        [~U] CharacterClassEscape
        [~U] CharacterEscape

NOTE When the same left hand sides occurs with both [+U] and [~U] guards it is to control
the disambiguation priority.
```

All document modes (All versions)

There are divergences in how the escaped 'c' character in `RegExp` (the ones that start with "\c") are treated:

1. The spec disallows \c without a following character outside the character class (i.e. outside the "CharacterClass" rule which starts with '[' and ends with ']'). However, the implementation treats this as two characters, '\' followed by 'c'. In other words, /\c/ is treated the same as /\\c/.

2. The spec disallows \c without a following character inside the character class. However, the implementation treats this as the single character 'c'. In other words, /[\c]/ is treated the same as `/[c]/`. This is different than 1 since the '\' character is ignored.

3. The spec disallows \c with a following character other than a letter (a..z or A..Z) inside the character class. According to the spec, when \c is followed by a letter, first the following character's numerical value is taken and divided by 32, and the remainder is converted to a character. For example, \cI (\c followed by uppercase i) is treated as the TAB character because the numerical value

Copyright © 2017 Microsoft Corporation

of 'I' is 73 and the remainder of dividing this by 32 is 8, which is the numerical value of the TAB character.

V0113: RegExp allows] on its own without being escaped, but shouldn't

The specification states:

```
B.1.4 Regular Expressions Patterns

    The syntax of 21.2.1 is modified and extended as follows. These changes introduce
    ambiguities that are broken by the ordering of grammar productions and by contextual
    information. When

parsing using the following grammar, each alternative is considered only if previous
production alternatives do not match.

    This alternative pattern grammar and semantics only changes the syntax and semantics
    of BMP patterns. The following grammar extensions include productions parameterized
    with the [U]

parameter. However, none of these extensions change the syntax of Unicode patterns
recognized when parsing with the [U] parameter present on the goal symbol.

    Syntax

        Term[U] ::
            [~U] ExtendedTerm
            [+U] Assertion[U]
            [+U] Atom[U]
            [+U] Atom[U] Quantifier

        ExtendedTerm ::
            Assertion
            AtomNoBrace Quantifier
            Atom
            QuantifiableAssertion Quantifier

        AtomNoBrace ::
            PatternCharacterNoBrace
            .
            \ AtomEscape
            CharacterClass
            ( Disjunction )
            ( ? : Disjunction )

        Atom[U] ::
            PatternCharacter
            .
            \ AtomEscape[?U]
            CharacterClass[?U]
            ( Disjunction[?U] )
            ( ? : Disjunction[?U] )

        PatternCharacterNoBrace ::
            SourceCharacter but not one of
            ^ $ \ . * + ? ( ) [ ] { } |

        PatternCharacter ::
            SourceCharacter but not one of
            ^ $ \ . * + ? ( ) [ ] |

        QuantifiableAssertion ::
            ( ? = Disjunction )
            ( ? ! Disjunction )
```


Copyright © 2017 Microsoft Corporation

```
Assertion[U] ::
    ^
    $
    \ b
    \ B
    [+U] ( ? = Disjunction[U] )
    [+U] ( ? ! Disjunction[U] )
    [~U] QuantifiableAssertion

AtomEscape[U] ::
    [+U] DecimalEscape
    [+U] CharacterEscape[U]
    [+U] CharacterClassEscape
    [~U] DecimalEscape but only if the integer value of DecimalEscape is <=
    NCapturingParens
    [~U] CharacterClassEscape
    [~U] CharacterEscape

CharacterEscape[U] ::
    ControlEscape
    c ControlLetter
    HexEscapeSequence
    RegExpUnicodeEscapeSequence[?U]
    [~U] LegacyOctalEscapeSequence
    IdentityEscape[?U]

IdentityEscape[U] ::
    [+U] SyntaxCharacter
    [+U] /
    [~U] SourceCharacter but not c

NonemptyClassRanges[U] ::
    ClassAtom[?U]
    ClassAtom[?U] NonemptyClassRangesNoDash[?U]
    [+U] ClassAtom[U] - ClassAtom[U] ClassRanges[U]
    [~U] ClassAtomInRange - ClassAtomInRange ClassRanges

NonemptyClassRangesNoDash[U] ::
    ClassAtom[?U]
    ClassAtomNoDash[?U] NonemptyClassRangesNoDash[?U]
    [+U] ClassAtomNoDash[U] - ClassAtom[U] ClassRanges[U]
    [~U] ClassAtomNoDashInRange - ClassAtomInRange ClassRanges

ClassAtom[U] ::
    -
    ClassAtomNoDash[?U]

ClassAtomNoDash[U] ::
    SourceCharacter but not one of \ or ] or -
    \ ClassEscape[?U]

ClassAtomInRange ::
    -
    ClassAtomNoDashInRange

ClassAtomNoDashInRange ::
    SourceCharacter but not one of \ or ] or -
    \ ClassEscape but only if ClassEscape evaluates to a CharSet with exactly one
    character
    \ IdentityEscape

ClassEscape[U] ::
    [+U] DecimalEscape
    [+U] CharacterEscape[U]
    [+U] CharacterClassEscape
    [~U] DecimalEscape
    b
    [~U] CharacterClassEscape
```


Copyright © 2017 Microsoft Corporation

```
            [~U] CharacterEscape

NOTE When the same left hand sides occurs with both [+U] and [~U] guards it is to control
the disambiguation priority.
```

All document modes (All versions)

`RegExp` allows] on its own without being escaped, but shouldn't. In other words, /]/ is allowed.

V0114: RegExp allows characters], {, and } to be quantified

The specification states:

```
B.1.4 Regular Expressions Patterns

    The syntax of 21.2.1 is modified and extended as follows. These changes introduce
    ambiguities that are broken by the ordering of grammar productions and by contextual
    information. When

parsing using the following grammar, each alternative is considered only if previous
production alternatives do not match.

    This alternative pattern grammar and semantics only changes the syntax and semantics
    of BMP patterns. The following grammar extensions include productions parameterized
    with the [U]

parameter. However, none of these extensions change the syntax of Unicode patterns
recognized when parsing with the [U] parameter present on the goal symbol.

    Syntax

        Term[U] ::
            [~U] ExtendedTerm
            [+U] Assertion[U]
            [+U] Atom[U]
            [+U] Atom[U] Quantifier

        ExtendedTerm ::
            Assertion
            AtomNoBrace Quantifier
            Atom
            QuantifiableAssertion Quantifier

        AtomNoBrace ::
            PatternCharacterNoBrace
            .
            \ AtomEscape
            CharacterClass
            ( Disjunction )
            ( ? : Disjunction )

        Atom[U] ::
            PatternCharacter
            .
            \ AtomEscape[?U]
            CharacterClass[?U]
            ( Disjunction[?U] )
            ( ? : Disjunction[?U] )

        PatternCharacterNoBrace ::
            SourceCharacter but not one of
            ^ $ \ . * + ? ( ) [ ] { } |
```


Copyright © 2017 Microsoft Corporation

```
PatternCharacter ::
    SourceCharacter but not one of
    ^ $ \ . * + ? ( ) [ ] |

QuantifiableAssertion ::
    ( ? = Disjunction )
    ( ? ! Disjunction )

Assertion[U] ::
    ^
    $
    \ b
    \ B
    [+U] ( ? = Disjunction[U] )
    [+U] ( ? ! Disjunction[U] )
    [~U] QuantifiableAssertion

AtomEscape[U] ::
    [+U] DecimalEscape
    [+U] CharacterEscape[U]
    [+U] CharacterClassEscape
    [~U] DecimalEscape but only if the integer value of DecimalEscape is <=
    NCapturingParens
    [~U] CharacterClassEscape
    [~U] CharacterEscape

CharacterEscape[U] ::
    ControlEscape
    c ControlLetter
    HexEscapeSequence
    RegExpUnicodeEscapeSequence[?U]
    [~U] LegacyOctalEscapeSequence
    IdentityEscape[?U]

IdentityEscape[U] ::
    [+U] SyntaxCharacter
    [+U] /
    [~U] SourceCharacter but not c

NonemptyClassRanges[U] ::
    ClassAtom[?U]
    ClassAtom[?U] NonemptyClassRangesNoDash[?U]
    [+U] ClassAtom[U] - ClassAtom[U] ClassRanges[U]
    [~U] ClassAtomInRange - ClassAtomInRange ClassRanges

NonemptyClassRangesNoDash[U] ::
    ClassAtom[?U]
    ClassAtomNoDash[?U] NonemptyClassRangesNoDash[?U]
    [+U] ClassAtomNoDash[U] - ClassAtom[U] ClassRanges[U]
    [~U] ClassAtomNoDashInRange - ClassAtomInRange ClassRanges

ClassAtom[U] ::
    -
    ClassAtomNoDash[?U]

ClassAtomNoDash[U] ::
    SourceCharacter but not one of \ or ] or -
    \ ClassEscape[?U]

ClassAtomInRange ::
    -
    ClassAtomNoDashInRange

ClassAtomNoDashInRange ::
    SourceCharacter but not one of \ or ] or -
    \ ClassEscape but only if ClassEscape evaluates to a CharSet with exactly one
    character
    \ IdentityEscape
```


Copyright © 2017 Microsoft Corporation

```
ClassEscape[U] ::
    [+U] DecimalEscape
    [+U] CharacterEscape[U]
    [+U] CharacterClassEscape
    [~U] DecimalEscape
    b
    [~U] CharacterClassEscape
    [~U] CharacterEscape
```

```
NOTE When the same left hand sides occurs with both [+U] and [~U] guards it is to control
the disambiguation priority.
```

All document modes (All versions)

`RegExp` allows the characters { and } to be quantified, but shouldn't (see 21.2.1). For example, /}+/ is allowed.

Also, the character] is allowed on its own, and to be quantified, but shouldn't be. For example, /]/ and /]+/ are allowed.

2.1.147 [ECMA-262/6] Section B.2.2 Additional Properties of the Object.prototype Object

V0032: Object.prototype.__proto__.get.name is undefined

The specification states:

```
B.2.2.1.1 get Object.prototype.__proto__

    The value of the [[Get]] attribute is a built-in function that requires no arguments.
    It performs the following steps:

        1. Let O be ToObject(this value).
        2. ReturnIfAbrupt(O).
        3. Return O.[[GetPrototypeOf]]().
```

All document modes (All versions)

`Object.prototype.__proto__.get.name` is undefined. The value of the `[[Get]]` attribute is a built-in function `g` that requires no arguments. `HasOwnProperty(g, "name")` returns `false`.

V0033: Object.prototype.__proto__.set.name is undefined

The specification states:

```
B.2.2.1.2 set Object.prototype.__proto__

    The value of the [[Set]] attribute is a built-in function that takes an argument
    proto. It performs the following steps:

        1. Let O be RequireObjectCoercible(this value).
        2. ReturnIfAbrupt(O).
        3. If Type(proto) is neither Object nor Null, return undefined.
        4. If Type(O) is not Object, return undefined.
        5. Let status be O.[[SetPrototypeOf]](proto).
```


Copyright © 2017 Microsoft Corporation

```
6. ReturnIfAbrupt(status).
7. If status is false, throw a TypeError exception.
8. Return undefined.
```

All document modes (All versions)

`Object.prototype.__proto__.set.name` is undefined. The value of the `[[Set]]` attribute is a built-in function `s` that takes an argument `proto`. `HasOwnProperty(s, "name")` returns `false`.

2.1.148 [ECMA-262/6] Section B.3.1 __proto__ Property Names in Object Initializers

V0115: Duplicate definitions of __proto__ in an object literal do not result in a Syntax Error

The specification states:

```
B.3.1 __proto__ Property Names in Object Initializers

    The following Early Error rule is added to those in 12.2.6.1:

        ObjectLiteral : { PropertyDefinitionList }ObjectLiteral : {
        PropertyDefinitionList , }

        • It is a Syntax Error if PropertyNameList of PropertyDefinitionList contains any
        duplicate entries for "__proto__" and at least two of those entries were obtained
        from productions of the form PropertyDefinition : PropertyName :
        AssignmentExpression .
```

IE11 Mode (All versions)

No Syntax Error is thrown if there is more than one definition of __proto__ obtained from productions of the form PropertyDefinition : PropertyName : AssignmentExpression.

V0163: The function name is not assigned based on the property name

The specification states:

```
B.3.1 __proto__ Property Names in Object Initializers
    ...
    In 12.2.6.9 the PropertyDefinitionEvaluation algorithm for the production

        PropertyDefinition : PropertyName : AssignmentExpression

    is replaced with the following definition:

        PropertyDefinition : PropertyName : AssignmentExpression

         1. Let propKey be the result of evaluating PropertyName.
        ...
         6. If propKey is the String value "__proto__" and if
            IsComputedPropertyKey(propKey) is false, then
            a. If Type(propValue) is either Object or Null, then
                  i. Return object.[[SetPrototypeOf]](propValue).
            b. Return NormalCompletion(empty).
```


Copyright © 2017 Microsoft Corporation

All document modes (All versions)

The following code prints `__proto__`, though the `name` property should not be created:

```
var o;
o = {
      __proto__: function() {}
   };
print(o.__proto__.name !== '__proto__');
```

2.2 Clarifications

The following subsections describe clarifications of the MAY and SHOULD requirements of [ECMA-262/6].

2.2.1 [ECMA-262/6] Section 14.1.7 Static Semantics: HasInitializer

C0001: The initializer for FormalParameters is not implemented

The specification states:

```
14.1.7 Static Semantics: HasInitializer
```

IE11 Mode (All versions)

The initializer for `FormalParameters` is not implemented.

2.2.2 [ECMA-262/6] Section 14.2.6 Static Semantics: HasInitializer

C0002: Initializers are not implemented for FormalParameters

The specification states:

```
14.2.6 Static Semantics: HasInitializer
```

IE11 Mode (All versions)

Initializers are not implemented for `FormalParameters`.

Copyright © 2017 Microsoft Corporation

2.3 Extensions

2.4 Error Handling

There are no additional error handling considerations.

2.5 Security

There are no additional security considerations.

Copyright © 2017 Microsoft Corporation

3 Change Tracking

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

Copyright © 2017 Microsoft Corporation

4 Index

.

...args) 49

[

[argumentsList]) 13

C

C) 23
Change tracking 93
completion) 13

E

expr - iterationKind) 31

G

Glossary 7

I

index - unicode) 59
Informative references 7
Introduction 7
items - mapfn - thisArg) 70

K

kind - ParameterList - Body - Scope) 15

L

limit) (section 2.1.87 54, section 2.1.107 63)

M

mapfn [- thisArg]]) 70

N

Normative references 7

P

PreferredType]) 11

R

realm) 15
References
  informative 7
  normative 7
replaceValue) (section 2.1.85 53, section 2.1.104 62)
reserved2]]) (section 2.1.73 48, section 2.1.117 67)

S

start [- end]) 64
stmt - iterator - lhsKind - labelSet) 32

T

Tracking changes 93

Copyright © 2017 Microsoft Corporation