Annex A
(informative)

Grammar Summary

A.1 Lexical Grammar

SourceCharacter **::** *See clause 6*
any Unicode code unit

InputElementDiv **::** *See clause 7*
WhiteSpace
LineTerminator
Comment
Token
DivPunctuator

InputElementRegExp **::** *See clause 7*
WhiteSpace
LineTerminator
Comment
Token
RegularExpressionLiteral

WhiteSpace **::** *See 7.2*
<TAB>
<VT>
<FF>
<SP>
<NBSP>
<BOM>
<USP>

LineTerminator **::** *See 7.3*
<LF>
<CR>
<LS>
<PS>

LineTerminatorSequence **::** *See 7.3*
<LF>
<CR> [lookahead ∉ <LF>]
<LS>
<PS>
<CR> <LF>

Comment **::** *See 7.4*
MultiLineComment
SingleLineComment

MultiLineComment **::** *See 7.4*
/* *MultiLineCommentChars*$_{opt}$ ***/**

MultiLineCommentChars **::** *See 7.4*
MultiLineNotAsteriskChar MultiLineCommentChars$_{opt}$
***** *PostAsteriskCommentChars*$_{opt}$

PostAsteriskCommentChars **::** *See 7.4*
MultiLineNotForwardSlashOrAsteriskChar MultiLineCommentChars$_{opt}$
***** *PostAsteriskCommentChars*$_{opt}$

MultiLineNotAsteriskChar **::** *See 7.4*
SourceCharacter **but not ***

MultiLineNotForwardSlashOrAsteriskChar **::** *See 7.4*
SourceCharacter **but not one of / or ***

SingleLineComment **::** *See 7.4*
// *SingleLineCommentChars*$_{opt}$

SingleLineCommentChars **::** *See 7.4*
SingleLineCommentChar SingleLineCommentChars$_{opt}$

SingleLineCommentChar **::** *See 7.4*
SourceCharacter **but not** *LineTerminator*

Token **::** *See 7.5*
IdentifierName
Punctuator
NumericLiteral
StringLiteral

Identifier **::** *See 7.6*
IdentifierName **but not** *ReservedWord*

IdentifierName **::** *See 7.6*
IdentifierStart
IdentifierName IdentifierPart

IdentifierStart **::** *See 7.6*
UnicodeLetter
$
_
**** *UnicodeEscapeSequence*

IdentifierPart **::** *See 7.6*
IdentifierStart
UnicodeCombiningMark
UnicodeDigit
UnicodeConnectorPunctuation
<ZWNJ>
<ZWJ>

UnicodeLetter **::** *See 7.6*
any character in the Unicode categories “Uppercase letter (Lu)”, “Lowercase letter (Ll)”, “Titlecase letter (Lt)”, “Modifier letter (Lm)”, “Other letter (Lo)”, or “Letter number (Nl)”.

UnicodeCombiningMark **::** *See 7.6*
any character in the Unicode categories “Non-spacing mark (Mn)” or “Combining spacing mark (Mc)”

UnicodeDigit **::** *See 7.6*
any character in the Unicode category “Decimal number (Nd)”

UnicodeConnectorPunctuation **::** *See 7.6*
any character in the Unicode category “Connector punctuation (Pc)”

ReservedWord **::** *See 7.6.1*
Keyword
FutureReservedWord
NullLiteral
BooleanLiteral

Keyword **:: one of** *See 7.6.1.1*

break	**do**	**instanceof**	**typeof**
case	**else**	**new**	**var**
catch	**finally**	**return**	**void**
continue	**for**	**switch**	**while**
debugger	**function**	**this**	**with**
default	**if**	**throw**	
delete	**in**	**try**	

FutureReservedWord **:: one of** *See 7.6.1.2*

class	**enum**	**extends**	**super**
const	**export**	**import**	

The following tokens are also considered to be *FutureReservedWords* when parsing strict mode code (see 10.1.1).

implements	**let**	**private**	**public**
interface	**package**	**protected**	**static**
yield			

Punctuator **:: one of** *See 7.7*

`{`	`}`	`(`	`)`	`[`	`]`
`.`	`;`	`,`	`<`	`>`	`<=`
`>=`	`==`	`!=`	`===`	`!==`	
`+`	`-`	`*`	`%`	`++`	`--`
`<<`	`>>`	`>>>`	`&`	`|`	`^`
`!`	`~`	`&&`	`||`	`?`	`:`
`=`	`+=`	`-=`	`*=`	`%=`	`<<=`
`>>=`	`>>>=`	`&=`	`|=`	`^=`	

DivPunctuator **:: one of** *See 7.7*

`/` `/=`

Literal **::** *See 7.8*

NullLiteral
BooleanLiteral
NumericLiteral
StringLiteral
RegularExpressionLiteral

NullLiteral **::** *See 7.8.1*

`null`

BooleanLiteral **::** *See 7.8.2*

`true`
`false`

NumericLiteral **::** *See 7.8.3*

DecimalLiteral
HexIntegerLiteral

DecimalLiteral **::** *See 7.8.3*

DecimalIntegerLiteral `.` $DecimalDigits_{opt}$ $ExponentPart_{opt}$
`.` *DecimalDigits* $ExponentPart_{opt}$
DecimalIntegerLiteral $ExponentPart_{opt}$

DecimalIntegerLiteral **::** *See 7.8.3*

`0`
NonZeroDigit $DecimalDigits_{opt}$

DecimalDigits **::** *See 7.8.3*

DecimalDigit
DecimalDigits DecimalDigit

DecimalDigit **:: one of** *See 7.8.3*

`0` `1` `2` `3` `4` `5` `6` `7` `8` `9`

NonZeroDigit :: **one of** *See 7.8.3*
`1 2 3 4 5 6 7 8 9`

ExponentPart :: *See 7.8.3*
ExponentIndicator SignedInteger

ExponentIndicator :: **one of** *See 7.8.3*
`e E`

SignedInteger :: *See 7.8.3*
DecimalDigits
+ *DecimalDigits*
– *DecimalDigits*

HexIntegerLiteral :: *See 7.8.3*
0x *HexDigit*
0X *HexDigit*
HexIntegerLiteral HexDigit

HexDigit :: **one of** *See 7.8.3*
`0 1 2 3 4 5 6 7 8 9 a b c d e f A B C D E F`

StringLiteral :: *See 7.8.4*
" *DoubleStringCharacters*$_{opt}$ "
' *SingleStringCharacters*$_{opt}$ '

DoubleStringCharacters :: *See 7.8.4*
DoubleStringCharacter DoubleStringCharacters$_{opt}$

SingleStringCharacters :: *See 7.8.4*
SingleStringCharacter SingleStringCharacters$_{opt}$

DoubleStringCharacter :: *See 7.8.4*
SourceCharacter **but not one of** `"` **or** `\` **or** *LineTerminator*
`\` *EscapeSequence*
LineContinuation

SingleStringCharacter :: *See 7.8.4*
SourceCharacter **but not one of** `'` **or** `\` **or** *LineTerminator*
`\` *EscapeSequence*
LineContinuation

LineContinuation :: *See 7.8.4*
`\` *LineTerminatorSequence*

EscapeSequence :: *See 7.8.4*
CharacterEscapeSequence
0 [lookahead ∉ *DecimalDigit*]
HexEscapeSequence
UnicodeEscapeSequence

CharacterEscapeSequence :: *See 7.8.4*
SingleEscapeCharacter
NonEscapeCharacter

SingleEscapeCharacter :: **one of** *See 7.8.4*
`' " \ b f n r t v`

NonEscapeCharacter **::** *See 7.8.4*
SourceCharacter **but not one of** *EscapeCharacter* **or** *LineTerminator*

EscapeCharacter **::** *See 7.8.4*
SingleEscapeCharacter
DecimalDigit
x
u

HexEscapeSequence **::** *See 7.8.4*
x *HexDigit HexDigit*

UnicodeEscapeSequence **::** *See 7.8.4*
u *HexDigit HexDigit HexDigit HexDigit*

RegularExpressionLiteral **::** *See 7.8.5*
/ *RegularExpressionBody* **/** *RegularExpressionFlags*

RegularExpressionBody **::** *See 7.8.5*
RegularExpressionFirstChar RegularExpressionChars

RegularExpressionChars **::** *See 7.8.5*
[empty]
RegularExpressionChars RegularExpressionChar

RegularExpressionFirstChar **::** *See 7.8.5*
RegularExpressionNonTerminator **but not one of * or \ or / or [**
RegularExpressionBackslashSequence
RegularExpressionClass

RegularExpressionChar **::** *See 7.8.5*
RegularExpressionNonTerminator **but not \ or / or [**
RegularExpressionBackslashSequence
RegularExpressionClass

RegularExpressionBackslashSequence **::** *See 7.8.5*
**** *RegularExpressionNonTerminator*

RegularExpressionNonTerminator **::** *See 7.8.5*
SourceCharacter **but not** *LineTerminator*

RegularExpressionClass **::** *See 7.8.5*
[*RegularExpressionClassChars* **]**

RegularExpressionClassChars **::** *See 7.8.5*
[empty]
RegularExpressionClassChars RegularExpressionClassChar

RegularExpressionClassChar **::** *See 7.8.5*
RegularExpressionNonTerminator **but not] or **
RegularExpressionBackslashSequence

RegularExpressionFlags **::** *See 7.8.5*
[empty]
RegularExpressionFlags IdentifierPart

A.2 Number Conversions

StringNumericLiteral **:::** *See 9.3.1*
StrWhiteSpace$_{opt}$
StrWhiteSpace$_{opt}$ *StrNumericLiteral StrWhiteSpace*$_{opt}$

StrWhiteSpace **:::** *See 9.3.1*
StrWhiteSpaceChar StrWhiteSpace$_{opt}$

StrWhiteSpaceChar **:::** *See 9.3.1*
WhiteSpace
LineTerminator

StrNumericLiteral **:::** *See 9.3.1*
StrDecimalLiteral
HexIntegerLiteral

StrDecimalLiteral **:::** *See 9.3.1*
StrUnsignedDecimalLiteral
+ *StrUnsignedDecimalLiteral*
- *StrUnsignedDecimalLiteral*

StrUnsignedDecimalLiteral **:::** *See 9.3.1*
`Infinity`
DecimalDigits **`.`** *DecimalDigits*$_{opt}$ *ExponentPart*$_{opt}$
`.` *DecimalDigits ExponentPart*$_{opt}$
DecimalDigits ExponentPart$_{opt}$

DecimalDigits **:::** *See 9.3.1*
DecimalDigit
DecimalDigits DecimalDigit

DecimalDigit **::: one of** *See 9.3.1*
`0 1 2 3 4 5 6 7 8 9`

ExponentPart **:::** *See 9.3.1*
ExponentIndicator SignedInteger

ExponentIndicator **::: one of** *See 9.3.1*
`e E`

SignedInteger **:::** *See 9.3.1*
DecimalDigits
+ *DecimalDigits*
- *DecimalDigits*

HexIntegerLiteral **:::** *See 9.3.1*
0x *HexDigit*
0X *HexDigit*
HexIntegerLiteral HexDigit

HexDigit **::: one of** *See 9.3.1*
0 1 2 3 4 5 6 7 8 9 a b c d e f A B C D E F

A.3 Expressions

PrimaryExpression **:** *See 11.1*
this
Identifier
Literal
ArrayLiteral
ObjectLiteral
(*Expression* **)**

ArrayLiteral **:** *See 11.1.4*
[$Elision_{opt}$ **]**
[*ElementList* **]**
[*ElementList* **,** $Elision_{opt}$ **]**

ElementList **:** *See 11.1.4*
$Elision_{opt}$ *AssignmentExpression*
ElementList **,** $Elision_{opt}$ *AssignmentExpression*

Elision **:** *See 11.1.4*
,
Elision **,**

ObjectLiteral **:** *See 11.1.5*
{ }
{ *PropertyNameAndValueList* **}**
{ *PropertyNameAndValueList* **, }**

PropertyNameAndValueList **:** *See 11.1.5*
PropertyAssignment
PropertyNameAndValueList **,** *PropertyAssignment*

PropertyAssignment **:** *See 11.1.5*
PropertyName **:** *AssignmentExpression*
get *PropertyName* **( ) {** *FunctionBody* **}**
set *PropertyName* **(** *PropertySetParameterList* **) {** *FunctionBody* **}**

PropertyName **:** *See 11.1.5*
IdentifierName
StringLiteral
NumericLiteral

PropertySetParameterList **:** *See 11.1.5*
Identifier

MemberExpression **:** *See 11.2*
PrimaryExpression
FunctionExpression
MemberExpression **[** *Expression* **]**
MemberExpression **.** *IdentifierName*
new *MemberExpression Arguments*

NewExpression **:** *See 11.2*
MemberExpression
new *NewExpression*

CallExpression **:** *See 11.2*
MemberExpression Arguments
CallExpression Arguments
CallExpression **[** *Expression* **]**
CallExpression **.** *IdentifierName*

Arguments **:** *See 11.2*
( )
(*ArgumentList* **)**

ArgumentList **:** *See 11.2*
AssignmentExpression
ArgumentList **,** *AssignmentExpression*

LeftHandSideExpression **:** *See 11.2*
NewExpression
CallExpression

PostfixExpression **:** *See 11.3*
LeftHandSideExpression
LeftHandSideExpression [no *LineTerminator* here] **++**
LeftHandSideExpression [no *LineTerminator* here] **--**

UnaryExpression **:** *See 11.4*
PostfixExpression
delete *UnaryExpression*
void *UnaryExpression*
typeof *UnaryExpression*
++ *UnaryExpression*
-- *UnaryExpression*
+ *UnaryExpression*
- *UnaryExpression*
~ *UnaryExpression*
! *UnaryExpression*

MultiplicativeExpression **:** *See 11.5*
UnaryExpression
MultiplicativeExpression ***** *UnaryExpression*
MultiplicativeExpression **/** *UnaryExpression*
MultiplicativeExpression **%** *UnaryExpression*

AdditiveExpression **:** *See 11.6*

MultiplicativeExpression
AdditiveExpression **+** *MultiplicativeExpression*
AdditiveExpression **–** *MultiplicativeExpression*

ShiftExpression **:** *See 11.7**

AdditiveExpression
ShiftExpression **<<** *AdditiveExpression*
ShiftExpression **>>** *AdditiveExpression*
ShiftExpression **>>>** *AdditiveExpression*

RelationalExpression **:** *See 11.8*

ShiftExpression
RelationalExpression **<** *ShiftExpression*
RelationalExpression **>** *ShiftExpression*
RelationalExpression **<=** *ShiftExpression*
RelationalExpression **>=** *ShiftExpression*
RelationalExpression **`instanceof`** *ShiftExpression*
RelationalExpression **`in`** *ShiftExpression*

RelationalExpressionNoIn **:** *See 11.8*

ShiftExpression
RelationalExpressionNoIn **<** *ShiftExpression*
RelationalExpressionNoIn **>** *ShiftExpression*
RelationalExpressionNoIn **<=** *ShiftExpression*
RelationalExpressionNoIn **>=** *ShiftExpression*
RelationalExpressionNoIn **`instanceof`** *ShiftExpression*

EqualityExpression **:** *See 11.9*

RelationalExpression
EqualityExpression **==** *RelationalExpression*
EqualityExpression **!=** *RelationalExpression*
EqualityExpression **===** *RelationalExpression*
EqualityExpression **!==** *RelationalExpression*

EqualityExpressionNoIn **:** *See 11.9*

RelationalExpressionNoIn
EqualityExpressionNoIn **==** *RelationalExpressionNoIn*
EqualityExpressionNoIn **!=** *RelationalExpressionNoIn*
EqualityExpressionNoIn **===** *RelationalExpressionNoIn*
EqualityExpressionNoIn **!==** *RelationalExpressionNoIn*

BitwiseANDExpression **:** *See 11.10*

EqualityExpression
BitwiseANDExpression **&** *EqualityExpression*

BitwiseANDExpressionNoIn **:** *See 11.10*

EqualityExpressionNoIn
BitwiseANDExpressionNoIn **&** *EqualityExpressionNoIn*

BitwiseXORExpression **:** *See 11.10*
BitwiseANDExpression
BitwiseXORExpression **^** *BitwiseANDExpression*

BitwiseXORExpressionNoIn **:** *See 11.10*
BitwiseANDExpressionNoIn
BitwiseXORExpressionNoIn **^** *BitwiseANDExpressionNoIn*

BitwiseORExpression **:** *See 11.10*
BitwiseXORExpression
BitwiseORExpression **|** *BitwiseXORExpression*

BitwiseORExpressionNoIn **:** *See 11.10*
BitwiseXORExpressionNoIn
BitwiseORExpressionNoIn **|** *BitwiseXORExpressionNoIn*

LogicalANDExpression **:** *See 11.11*
BitwiseORExpression
LogicalANDExpression **&&** *BitwiseORExpression*

LogicalANDExpressionNoIn **:** *See 11.11*
BitwiseORExpressionNoIn
LogicalANDExpressionNoIn **&&** *BitwiseORExpressionNoIn*

LogicalORExpression **:** *See 11.11*
LogicalANDExpression
LogicalORExpression **||** *LogicalANDExpression*

LogicalORExpressionNoIn **:** *See 11.11*
LogicalANDExpressionNoIn
LogicalORExpressionNoIn **||** *LogicalANDExpressionNoIn*

ConditionalExpression **:** *See 11.12*
LogicalORExpression
LogicalORExpression **?** *AssignmentExpression* **:** *AssignmentExpression*

ConditionalExpressionNoIn **:** *See 11.12*
LogicalORExpressionNoIn
LogicalORExpressionNoIn **?** *AssignmentExpression* **:** *AssignmentExpressionNoIn*

AssignmentExpression **:** *See 11.13*
ConditionalExpression
LeftHandSideExpression **=** *AssignmentExpression*
LeftHandSideExpression AssignmentOperator AssignmentExpression

AssignmentExpressionNoIn **:** *See 11.13*
ConditionalExpressionNoIn
LeftHandSideExpression **=** *AssignmentExpressionNoIn*
LeftHandSideExpression AssignmentOperator AssignmentExpressionNoIn

AssignmentOperator : **one of** *See 11.13*

`*=` `/=` `%=` `+=` `-=` `<<=` `>>=` `>>>=` `&=` `^=` `|=`

Expression : *See 11.14*

AssignmentExpression
Expression **`,`** *AssignmentExpression*

ExpressionNoIn : *See 11.14*

AssignmentExpressionNoIn
ExpressionNoIn **`,`** *AssignmentExpressionNoIn*

A.4 Statements

Statement : *See clause 12*

Block
VariableStatement
EmptyStatement
ExpressionStatement
IfStatement
IterationStatement
ContinueStatement
BreakStatement
ReturnStatement
WithStatement
LabelledStatement
SwitchStatement
ThrowStatement
TryStatement
DebuggerStatement

Block : *See 12.1*

`{` *StatementList*$_{opt}$ **`}`**

StatementList : *See 12.1*

Statement
StatementList Statement

VariableStatement : *See 12.2*

`var` *VariableDeclarationList* **`;`**

VariableDeclarationList : *See 12.2*

VariableDeclaration
VariableDeclarationList **`,`** *VariableDeclaration*

VariableDeclarationListNoIn : *See 12.2*

VariableDeclarationNoIn
VariableDeclarationListNoIn **`,`** *VariableDeclarationNoIn*

VariableDeclaration : *See 12.2*

Identifier Initialiser$_{opt}$

VariableDeclarationNoIn : *See 12.2*

Identifier InitialiserNoIn$_{opt}$

Initialiser **:** *See 12.2*
= *AssignmentExpression*

InitialiserNoIn **:** *See 12.2*
= *AssignmentExpressionNoIn*

EmptyStatement **:** *See 12.3*
;

ExpressionStatement **:** *See 12.4*
[lookahead ∉ {**{**, **function**}] *Expression* **;**

IfStatement **:** *See 12.5*
if (*Expression* **)** *Statement* **else** *Statement*
if (*Expression* **)** *Statement*

IterationStatement **:** *See 12.6*
do *Statement* **while (** *Expression* **);**
while (*Expression* **)** *Statement*
for (*ExpressionNoIn*$_{opt}$**;** *Expression*$_{opt}$ **;** *Expression*$_{opt}$ **)** *Statement*
for (var *VariableDeclarationListNoIn***;** *Expression*$_{opt}$ **;** *Expression*$_{opt}$ **)** *Statement*
for (*LeftHandSideExpression* **in** *Expression* **)** *Statement*
for (var *VariableDeclarationNoIn* **in** *Expression* **)** *Statement*

ContinueStatement **:** *See 12.7*
continue ;
continue [no *LineTerminator* here] *Identifier* **;**

BreakStatement **:** *See 12.8*
break ;
break [no *LineTerminator* here] *Identifier* **;**

ReturnStatement **:** *See 12.9*
return ;
return [no *LineTerminator* here] *Expression* **;**

WithStatement **:** *See 12.10*
with (*Expression* **)** *Statement*

SwitchStatement **:** *See 12.11*
switch (*Expression* **)** *CaseBlock*

CaseBlock **:** *See 12.11*
{ *CaseClauses*$_{opt}$ **}**
{ *CaseClauses*$_{opt}$ *DefaultClause CaseClauses*$_{opt}$ **}**

CaseClauses **:** *See 12.11*
CaseClause
CaseClauses CaseClause

CaseClause **:** *See 12.11*
case *Expression* **:** *StatementList*$_{opt}$

DefaultClause **:** *See 12.11*
default **:** *StatementList*$_{opt}$

LabelledStatement **:** *See 12.12*
Identifier **:** *Statement*

ThrowStatement **:** *See 12.13*
throw [no *LineTerminator* here] *Expression* **;**

TryStatement **:** *See 12.14*
try *Block Catch*
try *Block Finally*
try *Block Catch Finally*

Catch **:** *See 12.14*
catch (*Identifier* **)** *Block*

Finally **:** *See 12.14*
finally *Block*

DebuggerStatement **:** *See 12.15*
debugger **;**

A.5 Functions and Programs

FunctionDeclaration **:** *See clause 13*
function *Identifier* **(** *FormalParameterList*$_{opt}$ **) {** *FunctionBody* **}**

FunctionExpression **:** *See clause 13*
function *Identifier*$_{opt}$ **(** *FormalParameterList*$_{opt}$ **) {** *FunctionBody* **}**

FormalParameterList **:** *See clause 13*
Identifier
FormalParameterList **,** *Identifier*

FunctionBody **:** *See clause 13*
SourceElements$_{opt}$

Program **:** *See clause 14*
SourceElements$_{opt}$

SourceElements **:** *See clause 14*
SourceElement
SourceElements SourceElement

SourceElement **:** *See clause 14*
Statement
FunctionDeclaration

A.6 Universal Resource Identifier Character Classes

uri **:::** *See 15.1.3*
uriCharacters$_{opt}$

uriCharacters **:::** *See 15.1.3*
uriCharacter uriCharacters$_{opt}$

uriCharacter **:::** *See 15.1.3*
uriReserved
uriUnescaped
uriEscaped

uriReserved **::: one of** *See 15.1.3*
`;` `/` `?` `:` `@` `&` `=` `+` `$` `,`

uriUnescaped **:::** *See 15.1.3*
uriAlpha
DecimalDigit
uriMark

uriEscaped **:::** *See 15.1.3*
`%` *HexDigit HexDigit*

uriAlpha **::: one of** *See 15.1.3*
`a` `b` `c` `d` `e` `f` `g` `h` `i` `j` `k` `l` `m` `n` `o` `p` `q` `r` `s` `t` `u` `v` `w` `x` `y` `z`
`A` `B` `C` `D` `E` `F` `G` `H` `I` `J` `K` `L` `M` `N` `O` `P` `Q` `R` `S` `T` `U` `V` `W` `X` `Y` `Z`

uriMark **::: one of** *See 15.1.3*
`-` `_` `.` `!` `~` `*` `'` `(` `)`

A.7 Regular Expressions

Pattern **::** *See 15.10.1*
Disjunction

Disjunction **::** *See 15.10.1*
Alternative
Alternative **|** *Disjunction*

Alternative **::** *See 15.10.1*
[empty]
Alternative Term

Term **::** *See 15.10.1*

Assertion
Atom
Atom Quantifier

Assertion **::** *See 15.10.1**

^
$
\ b
\ B
(? = *Disjunction* **)**
(? ! *Disjunction* **)**

Quantifier **::** *See 15.10.1*

QuantifierPrefix
QuantifierPrefix **?**

QuantifierPrefix **::** *See 15.10.1*

*****
+
?
{ *DecimalDigits* **}**
{ *DecimalDigits* **, }**
{ *DecimalDigits* **,** *DecimalDigits* **}**

Atom **::** *See 15.10.1*

PatternCharacter
.
**** *AtomEscape*
CharacterClass
(*Disjunction* **)**
(? : *Disjunction* **)**

PatternCharacter :: *See 15.10.1*

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

AtomEscape **::** *See 15.10.1*

DecimalEscape
CharacterEscape
CharacterClassEscape

CharacterEscape **::** *See 15.10.1*

ControlEscape
c *ControlLetter*
HexEscapeSequence
UnicodeEscapeSequence
IdentityEscape

ControlEscape **:: one of** *See 15.10.1*

f n r t v

ControlLetter **:: one of** *See 15.10.1*

a b c d e f g h i j k l m n o p q r s t u v w x y z
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

IdentityEscape **::** *See 15.10.1*

SourceCharacter **but not** *IdentifierPart*
<ZWJ>
<ZWNJ>

DecimalEscape **::** *See 15.10.1*

DecimalIntegerLiteral [lookahead ∉ *DecimalDigit*]

CharacterClassEscape **:: one of** *See 15.10.1*

d D s S w W

CharacterClass **::** *See 15.10.1*

[[lookahead ∉ {**^**}] *ClassRanges* **]**
[^ *ClassRanges* **]**

ClassRanges **::** *See 15.10.1*

[empty]
NonemptyClassRanges

NonemptyClassRanges **::** *See 15.10.1*

ClassAtom
ClassAtom NonemptyClassRangesNoDash
ClassAtom **–** *ClassAtom ClassRanges*

NonemptyClassRangesNoDash **::** *See 15.10.1*

ClassAtom
ClassAtomNoDash NonemptyClassRangesNoDash
ClassAtomNoDash **–** *ClassAtom ClassRanges*

ClassAtom **::** *See 15.10.1*

-
ClassAtomNoDash

ClassAtomNoDash **::** *See 15.10.1*

SourceCharacter **but not one of \ or] or -**
**** *ClassEscape*

ClassEscape **::** *See 15.10.1*

DecimalEscape
b
CharacterEscape
CharacterClassEscape

A.8 JSON

A.8.1 JSON Lexical Grammar

JSONWhiteSpace **::** *See 15.12.1.1*
<TAB>
<CR>
<LF>
<SP>

JSONString **::** *See 15.12.1.1*
" *JSONStringCharacters*$_{opt}$ **"**

JSONStringCharacters **::** *See 15.12.1.1*
JSONStringCharacter JSONStringCharacters$_{opt}$

JSONStringCharacter **::** *See 15.12.1.1*
SourceCharacter **but not one of " or \ or** *U+0000* **through** *U+001F*
**** *JSONEscapeSequence*

JSONEscapeSequence **::** *See 15.12.1.1*
JSONEscapeCharacter
UnicodeEscapeSequence

JSONEscapeCharacter **:: one of** *See 15.12.1.1*
" / \ b f n r t

JSONNumber **::** *See 15.12.1.1*
-$_{opt}$ *DecimalIntegerLiteral JSONFraction*$_{opt}$ *ExponentPart*$_{opt}$

JSONFraction **::** *See 15.12.1.1*
. *DecimalDigits*

JSONNullLiteral **::** *See 15.12.1.1*
NullLiteral

JSONBooleanLiteral **::** *See 15.12.1.1*
BooleanLiteral

A.8.2 JSON Syntactic Grammar

JSONText **:** *See 15.12.1.2*
JSONValue

JSONValue **:** *See 15.12.1.2*
JSONNullLiteral
JSONBooleanLiteral
JSONObject
JSONArray
JSONString
JSONNumber

JSONObject **:** *See 15.12.1.2*
{ }
{ *JSONMemberList* **}**

JSONMember **:** *See 15.12.1.2*
JSONString **:** *JSONValue*

JSONMemberList **:** *See 15.12.1.2*

JSONMember

JSONMemberList **,** *JSONMember*

JSONArray **:** *See 15.12.1.2*

[ ]

[*JSONElementList* **]**

JSONElementList **:** *See 15.12.1.2*

JSONValue

JSONElementList **,** *JSONValue*

Annex B
(informative)

Compatibility

B.1 Additional Syntax

Past editions of ECMAScript have included additional syntax and semantics for specifying octal literals and octal escape sequences. These have been removed from this edition of ECMAScript. This non-normative annex presents uniform syntax and semantics for octal literals and octal escape sequences for compatibility with some older ECMAScript programs.

B.1.1 Numeric Literals

The syntax and semantics of 7.8.3 can be extended as follows except that this extension is not allowed for strict mode code:

Syntax

NumericLiteral **::**
DecimalLiteral
HexIntegerLiteral
OctalIntegerLiteral

OctalIntegerLiteral **::**
0 *OctalDigit*
OctalIntegerLiteral OctalDigit

OctalDigit **:: one of**
0 1 2 3 4 5 6 7

Semantics

- The MV of *NumericLiteral* **::** *OctalIntegerLiteral* is the MV of *OctalIntegerLiteral*.
- The MV of *OctalDigit* **::** **0** is 0.
- The MV of *OctalDigit* **::** **1** is 1.
- The MV of *OctalDigit* **::** **2** is 2.
- The MV of *OctalDigit* **::** **3** is 3.
- The MV of *OctalDigit* **::** **4** is 4.
- The MV of *OctalDigit* **::** **5** is 5.
- The MV of *OctalDigit* **::** **6** is 6.
- The MV of *OctalDigit* **::** **7** is 7.
- The MV of *OctalIntegerLiteral* **::** **0** *OctalDigit* is the MV of *OctalDigit*.
- The MV of *OctalIntegerLiteral* **::** *OctalIntegerLiteral OctalDigit* is (the MV of *OctalIntegerLiteral* times 8) plus the MV of *OctalDigit*.

B.1.2 String Literals

The syntax and semantics of 7.8.4 can be extended as follows except that this extension is not allowed for strict mode code:

Syntax

EscapeSequence **::**
CharacterEscapeSequence
OctalEscapeSequence
HexEscapeSequence
UnicodeEscapeSequence

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

ZeroToThree **:: one of**
`0 1 2 3`

FourToSeven **:: one of**
`4 5 6 7`

Semantics

- The CV of *EscapeSequence* **::** *OctalEscapeSequence* is the CV of the *OctalEscapeSequence*.
- The CV of *OctalEscapeSequence* **::** *OctalDigit* [lookahead ∉ *DecimalDigit*] is the character whose code unit value is the MV of the *OctalDigit*.
- The CV of *OctalEscapeSequence* **::** *ZeroToThree OctalDigit* [lookahead ∉ *DecimalDigit*] is the character whose code unit value is (8 times the MV of the *ZeroToThree*) plus the MV of the *OctalDigit*.
- The CV of *OctalEscapeSequence* **::** *FourToSeven OctalDigit* is the character whose code unit value is (8 times the MV of the *FourToSeven*) plus the MV of the *OctalDigit*.
- The CV of *OctalEscapeSequence* **::** *ZeroToThree OctalDigit OctalDigit* is the character whose code unit value is (64 (that is, 8^2) times the MV of the *ZeroToThree*) plus (8 times the MV of the first *OctalDigit*) plus the MV of the second *OctalDigit*.
- The MV of *ZeroToThree* **::** `0` is 0.
- The MV of *ZeroToThree* **::** `1` is 1.
- The MV of *ZeroToThree* **::** `2` is 2.
- The MV of *ZeroToThree* **::** `3` is 3.
- The MV of *FourToSeven* **::** `4` is 4.
- The MV of *FourToSeven* **::** `5` is 5.
- The MV of *FourToSeven* **::** `6` is 6.
- The MV of *FourToSeven* **::** `7` is 7.

B.2 Additional Properties

Some implementations of ECMAScript have included additional properties for some of the standard native objects. This non-normative annex suggests uniform semantics for such properties without making the properties or their semantics part of this standard.

B.2.1 escape (string)

The **`escape`** function is a property of the global object. It computes a new version of a String value in which certain characters have been replaced by a hexadecimal escape sequence.

For those characters being replaced whose code unit value is **`0xFF`** or less, a two-digit escape sequence of the form `%`*xx* is used. For those characters being replaced whose code unit value is greater than **`0xFF`**, a four-digit escape sequence of the form `%u`*xxxx* is used.

When the **escape** function is called with one argument *string*, the following steps are taken:

1. Call ToString(*string*).
2. Compute the number of characters in Result(1).
3. Let *R* be the empty string.
4. Let *k* be 0.
5. If *k* equals Result(2), return *R*.
6. Get the character (represented as a 16-bit unsigned integer) at position *k* within Result(1).
7. If Result(6) is one of the 69 nonblank characters **"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789@*_+-./"** then go to step 13.
8. If Result(6), is less than 256, go to step 11.
9. Let *S* be a String containing six characters **"%u***wxyz***"** where *wxyz* are four hexadecimal digits encoding the value of Result(6).
10. Go to step 14.
11. Let *S* be a String containing three characters **"%***xy***"** where *xy* are two hexadecimal digits encoding the value of Result(6).
12. Go to step 14.
13. Let *S* be a String containing the single character Result(6).
14. Let *R* be a new String value computed by concatenating the previous value of *R* and *S*.
15. Increase *k* by 1.
16. Go to step 5.

NOTE The encoding is partly based on the encoding described in RFC 1738, but the entire encoding specified in this standard is described above without regard to the contents of RFC 1738. This encoding does not reflect changes to RFC 1738 made by RFC 3986.

B.2.2 unescape (string)

The **unescape** function is a property of the global object. It computes a new version of a String value in which each escape sequence of the sort that might be introduced by the **escape** function is replaced with the character that it represents.

When the **unescape** function is called with one argument *string*, the following steps are taken:

1. Call ToString(*string*).
2. Compute the number of characters in Result(1).
3. Let *R* be the empty String.
4. Let *k* be 0.
5. If *k* equals Result(2), return *R*.
6. Let *c* be the character at position *k* within Result(1).
7. If *c* is not **%**, go to step 18.
8. If *k* is greater than Result(2)−6, go to step 14.
9. If the character at position *k*+1 within Result(1) is not **u**, go to step 14.
10. If the four characters at positions *k*+2, *k*+3, *k*+4, and *k*+5 within Result(1) are not all hexadecimal digits, go to step 14.
11. Let *c* be the character whose code unit value is the integer represented by the four hexadecimal digits at positions *k*+2, *k*+3, *k*+4, and *k*+5 within Result(1).
12. Increase k by 5.
13. Go to step 18.
14. If *k* is greater than Result(2)−3, go to step 18.
15. If the two characters at positions *k*+1 and *k*+2 within Result(1) are not both hexadecimal digits, go to step 18.
16. Let *c* be the character whose code unit value is the integer represented by two zeroes plus the two hexadecimal digits at positions *k*+1 and *k*+2 within Result(1).
17. Increase *k* by 2.
18. Let *R* be a new String value computed by concatenating the previous value of *R* and *c*.
19. Increase *k* by 1.
20. Go to step 5.

B.2.3 String.prototype.substr (start, length)

The **substr** method takes two arguments, *start* and *length*, and returns a substring of the result of converting the this object to a String, starting from character position *start* and running for *length* characters (or through the end of the String if *length* is **undefined**). If *start* is negative, it is treated as (*sourceLength*+*start*) where *sourceLength* is the length of the String. The result is a String value, not a String object. The following steps are taken:

1. Call ToString, giving it the **this** value as its argument.
2. Call ToInteger(*start*).
3. If *length* is **undefined**, use +∞; otherwise call ToInteger(*length*).
4. Compute the number of characters in Result(1).
5. If Result(2) is positive or zero, use Result(2); else use max(Result(4)+Result(2),0).
6. Compute min(max(Result(3),0), Result(4)–Result(5)).
7. If Result(6) ≤ 0, return the empty String "".
8. Return a String containing Result(6) consecutive characters from Result(1) beginning with the character at position Result(5).

The **length** property of the **substr** method is **2**.

NOTE The **substr** function is intentionally generic; it does not require that its **this** value be a String object. Therefore it can be transferred to other kinds of objects for use as a method.

B.2.4 Date.prototype.getYear ( )

NOTE The **getFullYear** method is preferred for nearly all purposes, because it avoids the "year 2000 problem."

When the **getYear** method is called with no arguments, the following steps are taken:

1. Let *t* be this time value.
2. If *t* is **NaN**, return **NaN**.
3. Return YearFromTime(LocalTime(*t*)) – 1900.

B.2.5 Date.prototype.setYear (year)

NOTE The **setFullYear** method is preferred for nearly all purposes, because it avoids the "year 2000 problem."

When the **setYear** method is called with one argument *year*, the following steps are taken:

1. Let *t* be the result of LocalTime(this time value); but if this time value is **NaN**, let *t* be **+0**.
2. Call ToNumber(*year*).
3. If Result(2) is **NaN**, set the [[PrimitiveValue]] internal property of the **this** value to **NaN** and return **NaN**.
4. If Result(2) is not **NaN** and 0 ≤ ToInteger(Result(2)) ≤ 99 then Result(4) is ToInteger(Result(2)) + 1900. Otherwise, Result(4) is Result(2).
5. Compute MakeDay(Result(4), MonthFromTime(*t*), DateFromTime(*t*)).
6. Compute UTC(MakeDate(Result(5), TimeWithinDay(*t*))).
7. Set the [[PrimitiveValue]] internal property of the **this** value to TimeClip(Result(6)).
8. Return the value of the [[PrimitiveValue]] internal property of the **this** value.

B.2.6 Date.prototype.toGMTString ( )

NOTE The property **toUTCString** is preferred. The **toGMTString** property is provided principally for compatibility with old code. It is recommended that the **toUTCString** property be used in new ECMAScript code.

The Function object that is the initial value of **Date.prototype.toGMTString** is the same Function object that is the initial value of **Date.prototype.toUTCString**.

Annex C
(informative)

The Strict Mode of ECMAScript

The strict mode restriction and exceptions

- The identifiers "**`implements`**", "**`interface`**", "**`let`**", "**`package`**", "**`private`**", "**`protected`**", "**`public`**", "**`static`**", and "**`yield`**" are classified as *FutureReservedWord* tokens within strict mode code. (7.6.12).
- A conforming implementation, when processing strict mode code, may not extend the syntax of *NumericLiteral* (7.8.3) to include *OctalIntegerLiteral* as described in B.1.1.
- A conforming implementation, when processing strict mode code (see 10.1.1), may not extend the syntax of *EscapeSequence* to include *OctalEscapeSequence* as described in B.1.2.
- Assignment to an undeclared identifier or otherwise unresolvable reference does not create a property in the global object. When a simple assignment occurs within strict mode code, its *LeftHandSide* must not evaluate to an unresolvable Reference. If it does a **ReferenceError** exception is thrown (8.7.2). The *LeftHandSide* also may not be a reference to a data property with the attribute value {[[Writable]]:**false**}, to an accessor property with the attribute value {[[Set]]:**undefined**}, nor to a non-existent property of an object whose [[Extensible]] internal property has the value **false**. In these cases a **`TypeError`** exception is thrown (11.13.1).
- The identifier **`eval`** or **`arguments`** may not appear as the *LeftHandSideExpression* of an Assignment operator (11.13) or of a *PostfixExpression* (11.3) or as the *UnaryExpression* operated upon by a Prefix Increment (11.4.4) or a Prefix Decrement (11.4.5) operator.
- Arguments objects for strict mode functions define non-configurable accessor properties named "**`caller`**" and "**`callee`**" which throw a **TypeError** exception on access (10.6).
- Arguments objects for strict mode functions do not dynamically share their array indexed property values with the corresponding formal parameter bindings of their functions. (10.6).
- For strict mode functions, if an arguments object is created the binding of the local identifier **`arguments`** to the arguments object is immutable and hence may not be the target of an assignment expression. (10.5).
- It is a **SyntaxError** if strict mode code contains an *ObjectLiteral* with more than one definition of any data property (11.1.5).
- It is a **SyntaxError** if the *Identifier* **`"eval"`** or the *Identifier* **`"arguments"`** occurs as the *Identifier* in a *PropertySetParameterList* of a *PropertyAssignment* that is contained in strict code or if its *FunctionBody* is strict code (11.1.5).
- Strict mode eval code cannot instantiate variables or functions in the variable environment of the caller to eval. Instead, a new variable environment is created and that environment is used for declaration binding instantiation for the eval code (10.4.2).
- If **this** is evaluated within strict mode code, then the **this** value is not coerced to an object. A **this** value of **null** or **undefined** is not converted to the global object and primitive values are not converted to wrapper objects. The **this** value passed via a function call (including calls made using **`Function.prototype.apply`** and **`Function.prototype.call`**) do not coerce the passed this value to an object (10.4.3, 11.1.1, 15.3.4.3, 15.3.4.4).
- When a **`delete`** operator occurs within strict mode code, a **SyntaxError** is thrown if its *UnaryExpression* is a direct reference to a variable, function argument, or function name(11.4.1).

- When a **`delete`** operator occurs within strict mode code, a **TypeError** is thrown if the property to be deleted has the attribute { [[Configurable]]:**false** } (11.4.1).
- It is a **SyntaxError** if a *VariableDeclaration* or *VariableDeclarationNoIn* occurs within strict code and its *Identifier* is **`eval`** or **`arguments`** (12.2.1).
- Strict mode code may not include a *WithStatement*. The occurrence of a *WithStatement* in such a context is an **SyntaxError** (12.10).
- It is a **SyntaxError** if a *TryStatement* with a *Catch* occurs within strict code and the *Identifier* of the *Catch* production is **`eval`** or **`arguments`** (12.14.1)
- It is a **SyntaxError** if the identifier **`eval`** or **`arguments`** appears within a *FormalParameterList* of a strict mode *FunctionDeclaration* or *FunctionExpression* (13.1)
- A strict mode function may not have two or more formal parameters that have the same name. An attempt to create such a function using a *FunctionDeclaration*, *FunctionExpression*, or **`Function`** constructor is a **SyntaxError** (13.1, 15.3.2).
- An implementation may not extend, beyond that defined in this specification, the meanings within strict mode functions of properties named **`caller`** or **`arguments`** of function instances. ECMAScript code may not create or modify properties with these names on function objects that correspond to strict mode functions (10.6, 13.2, 15.3.4.5.3).
- It is a **SyntaxError** to use within strict mode code the identifiers **`eval`** or **`arguments`** as the *Identifier* of a *FunctionDeclaration* or *FunctionExpression* or as a formal parameter name (13.1). Attempting to dynamically define such a strict mode function using the **`Function`** constructor (15.3.2) will throw a **SyntaxError** exception.

Annex D
(informative)

Corrections and Clarifications in the 5th Edition with Possible 3rd Edition Compatibility Impact

Throughout: In the Edition 3 specification the meaning of phrases such as "as if by the expression **`new Array()`**" are subject to misinterpretation. In the Edition 5 specification text for all internal references and invocations of standard built-in objects and methods has been clarified by making it explicit that the intent is that the actual built-in object is to be used rather than the current dynamic value of the correspondingly named property.

11.8.2, 11.8.3, 11.8.5: ECMAScript generally uses a left to right evaluation order, however the Edition 3 specification language for the > and <= operators resulted in a partial right to left order. The specification has been corrected for these operators such that it now specifies a full left to right evaluation order. However, this change of order is potentially observable if side-effects occur during the evaluation process.

11.1.4: Edition 5 clarifies the fact that a trailing comma at the end of an *ArrayInitialiser* does not add to the length of the array. This is not a semantic change from Edition 3 but some implementations may have previously misinterpreted this.

11.2.3: Edition 5 reverses the order of steps 2 and 3 of the algorithm. The original order as specified in Editions 1 through 3 was incorrectly specified such that side-effects of evaluating *Arguments* could affect the result of evaluating *MemberExpression*.

12.4: In Edition 3, an object is created, as if by **`new Object()`** to serve as the scope for resolving the name of the exception parameter passed to a **`catch`** clause of a **`try`** statement. If the actual exception object is a function and it is called from within the **`catch`** clause, the scope object will be passed as the **this** value of the call. The body of the function can then define new properties on its **this** value and those property names become visible identifiers bindings within the scope of the **catch** clause after the function returns. In Edition 5, when an exception parameter is called as a function, **undefined** is passed as the **this** value.

13: In Edition 3, the algorithm for the production *FunctionExpression* with an *Identifier* adds an object created as if by **`new Object()`** to the scope chain to serve as a scope for looking up the name of the function. The identifier resolution rules (10.1.4 in Edition 3) when applied to such an object will, if necessary, follow the object's prototype chain when attempting to resolve an identifier. This means all the properties of Object.prototype are visible as identifiers within that scope. In practice most implementations of Edition 3 have not implemented this semantics. Edition 5 changes the specified semantics by using a Declarative Environment Record to bind the name of the function.

14: In Edition 3, the algorithm for the production *SourceElements* **:** *SourceElements SourceElement* did not correctly propagate statement result values in the same manner as *Block*. This could result in the **`eval`** function producing an incorrect result when evaluating a *Program* text. In practice most implementations of Edition 3 have implemented the correct propagation rather than what was specified in Edition 5.

15.10.6: RegExp.prototype is now a RegExp object rather than an instance of Object. The value of its [[Class]] internal property which is observable using Object.prototype.toString is now "RegExp" rather than "Object".

Annex E
(informative)

Additions and Changes in the 5th Edition that Introduce Incompatibilities with the 3rd Edition

7.1: Unicode format control characters are no longer stripped from ECMAScript source text before processing. In Edition 5, if such a character appears in a *StringLiteral* or *RegularExpressionLiteral* the character will be incorporated into the literal where in Edition 3 the character would not be incorporated into the literal.

7.2: Unicode character <BOM> is now treated as whitespace and its presence in the middle of what appears to be an identifier could result in a syntax error which would not have occurred in Edition 3

7.3: Line terminator characters that are preceded by an escape sequence are now allowed within a string literal token. In Edition 3 a syntax error would have been produced.

7.8.5: Regular expression literals now return a unique object each time the literal is evaluated. This change is detectable by any programs that test the object identity of such literal values or that are sensitive to the shared side effects.

7.8.5: Edition 5 requires early reporting of any possible RegExp constructor errors that would be produced when converting a *RegularExpressionLiteral* to a RegExp object. Prior to Edition 5 implementations were permitted to defer the reporting of such errors until the actual execution time creation of the object.

7.8.5: In Edition 5 unescaped "/" characters may appear as a *CharacterClass* in a regular expression literal. In Edition 3 such a character would have been interpreted as the final character of the literal.

10.4.2: In Edition 5, indirect calls to the **`eval`** function use the global environment as both the variable environment and lexical environment for the eval code. In Edition 3, the variable and lexical environments of the caller of an indirect **`eval`** was used as the environments for the eval code.

15.4.4: In Edition 5 all methods of **`Array.prototype`** are intentionally generic. In Edition 3 **`toString`** and **`toLocaleString`** were not generic and would throw a **`TypeError`** exception if applied to objects that were not instances of Array.

10.6: In Edition 5 the array indexed properties of argument objects that correspond to actual formal parameters are enumerable. In Edition 3, such properties were not enumerable.

10.6: In Edition 5 the value of the [[Class]] internal property of an arguments object is **`"Arguments"`**. In Edition 3, it was **`"Object"`**. This is observable if **`toString`** is called as a method of an arguments object.

12.6.4: for-in statements no longer throw a **TypeError** if the **`in`** expression evaluates to **null** or **undefined**. Instead, the statement behaves as if the value of the expression was an object with no enumerable properties.

15: In Edition 5, the following new properties are defined on built-in objects that exist in Edition 3: **`Object.getPrototypeOf`**, **`Object.getOwnPropertyDescriptor`**, **`Object.getOwnPropertyNames`**, **`Object.create`**, **`Object.defineProperty`**, **`Object.defineProperties`**, **`Object.seal`**, **`Object.freeze`**, **`Object.preventExtensions`**, **`Object.isSealed`**, **`Object.isFrozen`**, **`Object.isExtensible`**, **`Object.keys`**, **`Function.prototype.bind`**, **`Array.prototype.indexOf`**, **`Array.prototype.lastIndexOf`**, **`Array.prototype.every`**, **`Array.prototype.some`**, **`Array.prototype.forEach`**, **`Array.prototype.map`**, **`Array.prototype.filter`**, **`Array.prototype.reduce`**, **`Array.prototype.reduceRight`**, **`String.prototype.trim`**, **`Date.now`**, **`Date.prototype.toISOString`**, **`Date.prototype.toJSON`**.

15: Implementations are now required to ignore extra arguments to standard built-in methods unless otherwise explicitly specified. In Edition 3 the handling of extra arguments was unspecified and implementations were explicitly allowed to throw a **TypeError** exception.

15.1.1: The value properties **NaN**, **Infinity**, and **undefined** of the Global Object have been changed to be read-only properties.

15.1.2.1. Implementations are no longer permitted to restrict the use of eval in ways that are not a direct call. In addition, any invocation of eval that is not a direct call uses the global environment as its variable environment rather than the caller's variable environment.

15.1.2.2: The specification of the function **`parseInt`** no longer allows implementations to treat Strings beginning with a **`0`** character as octal values.

15.3.4.3: In Edition 3, a **TypeError** is thrown if the second argument passed to **`Function.prototype.apply`** is neither an array object nor an arguments object. In Edition 5, the second argument may be any kind of generic array-like object that has a valid **`length`** property.

15.3.4.3, 15.3.4.4: In Edition 3 passing **undefined** or **null** as the first argument to either **`Function.prototype.apply`** or **`Function.prototype.call`** causes the global object to be passed to the indirectly invoked target function as the **this** value. If the first argument is a primitive value the result of calling ToObject on the primitive value is passed as the **this** value. In Edition 5, these transformations are not performed and the actual first argument value is passed as the **this** value. This difference will normally be unobservable to existing ECMAScript Edition 3 code because a corresponding transformation takes place upon activation of the target function. However, depending upon the implementation, this difference may be observable by host object functions called using **`apply`** or **`call`**. In addition, invoking a standard built-in function in this manner with **null** or **undefined** passed as the this value will in many cases cause behaviour in Edition 5 implementations that differ from Edition 3 behaviour. In particular, in Edition 5 built-in functions that are specified to actually use the passed **this** value as an object typically throw a **TypeError** exception if passed **null** or **undefined** as the **this** value.

15.3.5.2: In Edition 5, the **`prototype`** property of Function instances is not enumerable. In Edition 3, this property was enumerable.

15.5.5.2: In Edition 5, the individual characters of a String object's [[PrimitiveValue] may be accessed as array indexed properties of the String object. These properties are non-writable and non-configurable and shadow any inherited properties with the same names. In Edition 3, these properties did not exist and ECMAScript code could dynamically add and remove writable properties with such names and could access inherited properties with such names.

15.9.4.2: **`Date.parse`** is now required to first attempt to parse its argument as an ISO format string. Programs that use this format but depended upon implementation specific behaviour (including failure) may behave differently.

15.10.2.12: In Edition 5, **`\s`** now additionally matches <BOM>.

15.10.4.1: In Edition 3, the exact form of the String value of the **`source`** property of an object created by the **`RegExp`** constructor is implementation defined. In Edition 5, the String must conform to certain specified requirements and hence may be different from that produced by an Edition 3 implementation.

15.10.6.4: In Edition 3, the result of **`RegExp.prototype.toString`** need not be derived from the value of the RegExp object's **`source`** property. In Edition 5 the result must be derived from the **`source`** property in a specified manner and hence may be different from the result produced by an Edition 3 implementation.

15.11.2.1, 15.11.4.3: In Edition 5, if an initial value for the **`message`** property of an Error object is not specified via the **`Error`** constructor the initial value of the property is the empty String. In Edition 3, such an initial value is implementation defined.

15.11.4.4: In Edition 3, the result of **`Error.prototype.toString`** is implementation defined. In Edition 5, the result is fully specified and hence may differ from some Edition 3 implementations.

15.12: In Edition 5, the name **`JSON`** is defined in the global environment. In Edition 3, testing for the presence of that name will show it to be undefined unless it is defined by the program or implementation.

Annex F
(informative)

Technically Significant Corrections and Clarifications in the 5.1 Edition

7.8.4: CV definitions added for *DoubleStringCharacter* **::** *LineContinuation* and *SingleStringCharacter* **::** *LineContinuation*.

10.2.1.1.3: The argument S is not ignored. It controls whether an exception is thrown when attempting to set an immutable binding.

10.2.1.2.2: In algorithm step 5, **true** is passed as the last argument to [[DefineOwnProperty]].

10.5: Former algorithm step 5.e is now 5.f and a new step 5.e was added to restore compatibility with 3rd Edition when redefining global functions.

11.5.3: In the final bullet item, use of IEEE 754 round-to-nearest mode is specified.

12.6.3: Missing ToBoolean restored in step 3.a.ii of both algorithms.

12.6.4: Additional final sentences in each of the last two paragraphs clarify certain property enumeration requirements.

12.7, 12.8, 12.9: BNF modified to clarify that a **`continue`** or **`break`** statement without an *Identifier* or a **`return`** statement without an *Expression* may have a *LineTerminator* before the semi-colon.

12.14: Step 3 of algorithm 1 and step 2.a of algorithm 3 are corrected such that the value field of *B* is passed as a parameter rather than *B* itself.

15.1.2.2: In step 2 of algorithm, clarify that *S* may be the empty string.

15.1.2.3: In step 2 of algorithm clarify that *trimmedString* may be the empty string.

15.1.3: Added notes clarifying that ECMAScript's URI syntax is based upon RFC 2396 and not the newer RFC 3986. In the algorithm for Decode, a step was removed that immediately preceded the current step 4.d.vii.10.a because it tested for a condition that cannot occur.

15.2.3.7: Corrected use of variable *P* in steps 5 and 6 of algorithm.

15.2.4.2: Edition 5 handling of **undefined** and **null** as **this** value caused existing code to fail. Specification modified to maintain compatibility with such code. New steps 1 and 2 added to the algorithm.

15.3.4.3: Steps 5 and 7 of Edition 5 algorithm have been deleted because they imposed requirements upon the *argArray* argument that are inconsistent with other uses of generic array-like objects.

15.4.4.12: In step 9.a, incorrect reference to *relativeStart* was replaced with a reference to *actualStart*.

15.4.4.15: Clarified that the default value for *fromIndex* is the length minus 1 of the array.

15.4.4.18: In step 9 of the algorithm, **undefined** is now the specified return value.

15.4.4.22: In step 9.c.ii the first argument to the [[Call]] internal method has been changed to **undefined** for consistency with the definition of `Array.prototype.reduce`.

15.4.5.1: In Algorithm steps 3.l.ii and 3.l.iii the variable name was inverted resulting in an incorrectly inverted test.

15.5.4.9: Normative requirement concerning canonically equivalent strings deleted from paragraph following algorithm because it is listed as a recommendation in NOTE 2.

15.5.4.14: In `split` algorithm step 11.a and 13.a, the positional order of the arguments to *SplitMatch* was corrected to match the actual parameter signature of *SplitMatch*. In step 13.a.iii.7.d, *lengthA* replaces *A*.`length`.

15.5.5.2: In first paragraph, removed the implication that the individual character property access had "array index" semantics. Modified algorithm steps 3 and 5 such that they do not enforce "array index" requirement.

15.9.1.15: Specified legal value ranges for fields that lacked them. Eliminated "time-only" formats. Specified default values for all optional fields.

15.10.2.2: The step numbers of the algorithm for the internal closure produced by step 2 were incorrectly numbered in a manner that implied that they were steps of the outer algorithm.

15.10.2.6: In the abstract operation *IsWordChar* the first character in the list in step 3 is "**a**" rather than "**A**".

15.10.2.8: In the algorithm for the closure returned by the abstract operation *CharacterSetMatcher*, the variable defined by step 3 and passed as an argument in step 4 was renamed to *ch* in order to avoid a name conflict with a formal parameter of the closure.

15.10.6.2: Step 9.e was deleted because It performed an extra increment of *i*.

15.11.1.1: Removed requirement that the `message` own property is set to the empty String when the *message* argument is **undefined**.

15.11.1.2: Removed requirement that the `message` own property is set to the empty String when the *message* argument is **undefined**.

15.11.4.4: Steps 6-10 modified/added to correctly deal with missing or empty `message` property value.

15.11.1.2: Removed requirement that the `message` own property is set to the empty String when the *message* argument is **undefined**.

15.12.3: In step 10.b.iii of the *JA* internal operation, the last element of the concatenation is "**]**".

B.2.1: Added to NOTE that the encoding is based upon RFC 1738 rather than the newer RFC 3986.

Annex C: An item was added corresponding to 7.6.12 regarding *FutureReservedWords* in strict mode.

Bibliography

[1] IEEE Std 754-2008: IEEE Standard for Floating-Point Arithmetic. Institute of Electrical and Electronic Engineers, New York (2008)

[2] The Unicode Consortium. The Unicode Standard, Version 3.0, defined by: The Unicode Standard, Version 3.0 (Reading, MA, Addison-Wesley, 2000. ISBN 0-201-61633-5)

[3] Unicode Inc. (2010), Unicode Technical Report #15: Unicode Normalization Forms

[4] ISO 8601:2004(E) *Data elements and interchange formats – Information interchange -- Representation of dates and times*

[5] RFC 1738 "Uniform Resource Locators (URL)", available at <http://tools.ietf.org/html/rfc1738>

[6] RFC 2396 "Uniform Resource Identifiers (URI): Generic Syntax", available at <http://tools.ietf.org/html/rfc2396>

[7] RFC 3629 "UTF-8, a transformation format of ISO 10646", available at <http://tools.ietf.org/html/rfc3629>

[8] RFC 4627 "The application/json Media Type for JavaScript Object Notation (JSON)" , available at <http://tools.ietf.org/html/rfc4627>