Erratum for ECMAScript, 5th Edition Specification (ECMA-262-5)

(Last Updated July 19, 2010)

Items with Technical Significance

7.1 Unicode Format-Control Characters

(Table 1)

Code Unit Value	*Name*	*Formal Name*	*Usage*
\u200C	Zero width non-joiner	<ZWNJ>	*IdentifierPart*
\u200D~~C~~	Zero width joiner	<ZWJ>	*IdentifierPart*
\uFEFF	Byte Order Mark	<BOM>	*Whitespace*

10.2.1.1.3 SetMutableBinding (N,V,S)

(Algorithm should only throw an exception when S is true)

The concrete Environment Record method SetMutableBinding for declarative environment records attempts to change the bound value of the current binding of the identifier whose name is the value of the argument *N* to the value of argument *V*. A binding for *N* must already exist. If the binding is an immutable binding, a **TypeError** is ~~always~~ thrown if *S* is **true**. ~~The *S* argument is ignored because strict mode does not change the meaning of setting bindings in declarative environment records.~~

4. Else this must be an attempt to change the value of an immutable binding so if *S* is **true** throw a **TypeError** exception.

10.2.1.2.2 CreateMutableBinding (N, D)

(Incorrect algorithm parameter, property should be created with throw parameter true to deal with case where global object is not extensible)

5. Call the [[DefineOwnProperty]] internal method of *bindings*, passing *N*, Property Descriptor {[[Value]]:**undefined**, [[Writable]]: **true**, [[Enumerable]]: **true** , [[Configurable]]: *configValue*}, and **true** ~~**false**~~ as arguments.

10.5 Declaration Binding Instantiation

(Step 5 of original algorithm handled redefining existing global function declarations in a manner that was incompatible with ES3 and which in some cases would unintentionally invoke accessor functions)

5. For each *FunctionDeclaration f* in *code*, in source text order do
    a. Let *fn* be the *Identifier* in *FunctionDeclaration f*.
    b. Let *fo* be the result of instantiating *FunctionDeclaration f* as described in Clause 13.
    c. Let *funcAlreadyDeclared* be the result of calling *env's* HasBinding concrete method passing *fn* as the argument.
    d. If *funcAlreadyDeclared* is **false**, call *env's* CreateMutableBinding concrete method passing *fn* and *configurableBindings* as the arguments.
    e. Else if *env* is the environment record component of the global environment then
        i. Let *go* be the global object.
        ii. Let *existingProp* be the resulting of calling the [[GetProperty]] internal method of *go* with argument *fn*.
        iii. If *existingProp* .[[Configurable]] is **true**, then
            1. Call the [[DefineOwnProperty]] internal method of *go*, passing *fn*, Property Descriptor {[[Value]]: **undefined**, [[Writable]]: **true**, [[Enumerable]]: **true** , [[Configurable]]: *configurableBindings* }, and **true** as arguments.
        iv. Else if IsAccessorDescrptor(*existingProp*) or *existingProp* does not have attribute values {[[Writable]]: **true**, [[Enumerable]]: **true**}, then
            1. Throw a TypeError exception.
    f. Call *env's* SetMutableBinding concrete method passing *fn*, *fo*, and *strict* as the arguments.

12.6.4 The `for-in` Statement

(Implementers have found the spec. to be unclear regarding whether shadowed inherited properties are included in an enumeration)

The mechanics and order of enumerating the properties (step 6.a in the first algorithm, step 7.a in the second) is not specified. Properties of the object being enumerated may be deleted during enumeration. If a property that has not yet been visited during enumeration is deleted, then it will not be visited. If new properties are added to the object being enumerated during enumeration, the newly added properties are not guaranteed to be visited in the active enumeration. A property name must not be visited more than once in any enumeration.

Enumerating the properties of an object includes enumerating properties of its prototype, and the prototype of the prototype, and so on, recursively; but a property of a prototype is not enumerated if it is "shadowed" because some previous object in the prototype chain has a property with the same name. The values of [[Enumerable]] attributes are not considered when determining if a property of a prototype object is shadowed by a previous object on the prototype chain.

15.2.3.7 Object.defineProperties (O, Properties)

(confusing use of *P* in steps 5 and 6 of algorithm)

5. For each element *P* of *names* in list order,
    a. Let *descObj* be the result of calling the [[Get]] internal method of *props* with *P* as the argument.
    b. Let *desc* be the result of calling ToPropertyDescriptor with *descObj* as the argument.
    c. Append the pair (a two element List) consisting of *P* and *desc* to the end of *descriptors*.
6. For each element *pair* from ~~*desc of*~~ *descriptors* in list order,
    a. Let *P* be the first element of *pair*.
    b. Let *desc* be the second element of *pair*.
    c.~~a.~~ Call the [[DefineOwnProperty]] internal method of *O* with arguments *P*, *desc*, and **true**.

15.2.4.2 Object.prototype.toString ( )

(Original algorithm caused failure of widely used web frameworks.)

1. If the **this** value is **undefined**, return **"[object Undefined]"**.
2. If the **this** value is **null**, return **"[object Null]"**.
~~1~~3. Let *O* be the result of calling ToObject passing the **this** value as the argument.
~~2~~4. Let *class* be the value of the [[Class]] internal property of *O*.
~~3~~5. Return the String value that is the result of concatenating the three Strings **"[object "**, *class*, and **"]"**.

15.3.4.3 Function.prototype.apply (thisArg, argArray)

(Original algorithm performs validation checks in steps 5 and 7 that are inconsistent with other generic array usages in the specification.)

4. Let *len* be the result of calling the [[Get]] internal method of *argArray* with argument **"length"**.
~~5. If *len* is **null** or **undefined**, then throw a **TypeError** exception.~~
5. Let *n* be ToUint32(*len*).
~~7. If *n* is not equal to ToNumber(*len*), then throw a **TypeError** exception.~~
6. Let *argList* be an empty List.
7. Let *index* be 0.
8. Repeat while *index* < *n*
    a. Let *indexName* be ToString(*index*).
    b. Let *nextArg* be the result of calling the [[Get]] internal method of *argArray* with *indexName* as the argument.
    c. Append *nextArg* as the last element of *argList*.
    d. Set *index* to *index* + 1.
9. Return the result of calling the [[Call]] internal method of *func*, providing *thisArg* as the **this** value and *argList* as the list of arguments.

15.4.4.18 Array.prototype.forEach (callbackfn [, thisArg])

(Return value not specified in step 8)

8. Return **undefined**.

15.4.4.21 Array.prototype.reduce (callbackfn [, initialValue])

(Fourth paragraph)

The range of elements processed by **reduce** is set before the first call to *callbackfn*. Elements that are appended to the array after the call to **reduce** begins will not be visited by *callbackfn*. If existing elements of the array are changed, their value as passed to *callbackfn* will be the value at the time **reduce** visits them; elements that are deleted after the call to **reduce** ~~**filter**~~ begins and before being visited are not visited.

15.4.4.22 Array.prototype.reduceRight (callbackfn [, initialValue])

(Fourth paragraph)

The range of elements processed by **reduceRight** is set before the first call to *callbackfn*. Elements that are appended to the array after the call to **reduceRight** begins will not be visited by *callbackfn*. If existing elements of the array are changed by *callbackfn*, their value as passed to *callbackfn* will be the value at the time **reduceRight** visits them; elements that are deleted after the call to **reduceRight** ~~**filter**~~ begins and before being visited are not visited.

(Algorithm step 9.c.ii)

9. Repeat, while $k \geq 0$
    a. Let *Pk* be ToString(*k*).
    b. Let *kPresent* be the result of calling the [[HasProperty]] internal method of *O* with argument *Pk*.
    c. If *kPresent* is **true**, then
        i. Let *kValue* be the result of calling the [[Get]] internal method of *O* with argument *Pk*.
        ii. Let *accumulator* be the result of calling the [[Call]] internal method of *callbackfn* with ~~**null**~~ **undefined** as the **this** value and argument list containing *accumulator*, *kValue*, *k*, and *O*.
    d. Decrease *k* by 1.

15.5.5.2 [[GetOwnProperty]] (P)

(First paragraph, individual character properties should not have “array index” restrictions)

String objects use a variation of the [[GetOwnProperty]] internal method used for other native ECMAScript objects (8.12.1). This special internal method is used to add access for ~~specify the array index~~ named properties corresponding to individual characters of String objects.

(Algorithm corrections)

3. If ToString(abs(ToInteger(*P*))) is not the same value as *P* ~~an array index (15.4)~~, return **undefined**.
4. Let *str* be the String value of the [[PrimitiveValue]] internal property of *S*.
5. Let *index* be ~~ToUint32~~ ToInteger(*P*).

15.9.1.15 Date Time String Format

(Time only variations of this string format should not have been included. Ranges were not specified for some fields)

Where the fields are as follows:

YYYY is the decimal digits of the year in the Gregorian calendar.
- “**:**” (~~hyphon~~ hyphen) appears literally twice in the string.
MM is the month of the year from 01 (January) to 12 (December).
DD is the day of the month from 01 to 31.

T "**T**" appears literally in the string, to indicate the beginning of the time element.

HH is the number of complete hours that have passed since midnight as two decimal digits from 00 to 24.

: "**:**" (colon) appears literally twice in the string.

mm is the number of complete minutes since the start of the hour as two decimal digits from 00 to 59.

ss is the number of complete seconds since the start of the minute as two decimal digits from 00 to 59.

. "**.**" (dot) appears literally in the string.

sss is the number of complete milliseconds since the start of the second as three decimal digits.

~~Both the "." and the milliseconds field may be omitted.~~

Z is the time zone offset specified as "**Z**" (for UTC) or either "**+**" or "**-**" followed by a time expression ~~**hh**~~**HH:mm**

This format includes date-only forms:

YYYY
YYYY-MM
YYYY-MM-DD

It also includes "date-time" forms that consist of one of the above date-only forms immediately followed by one of the following time ~~which It also includes time-only~~ forms with an optional time zone offset appended:

THH:mm
THH:mm:ss
THH:mm:ss.sss

~~Also included are "date-times" which may be any combination of the above.~~

All numbers must be base 10. If the **MM** or **DD** fields are absent "**01**" is used as the value. If the **mm** or **ss** fields are absent "**00**" is used as the value and the value of an absent **sss** file is "**000**". The value of an absent time zone offset is "**Z**".

15.11.1.1 Error (message)

(Last paragraph, Algorithm incorrect when message is **undefined**)

If the argument *message* is not **undefined**, the **message** own property of the newly constructed object is set to ToString(*message*). ~~Otherwise, the **message** own property is set to the empty String.~~

15.11.2.1 new Error (message)

(Last paragraph, Algorithm incorrect when message is **undefined**)

If the argument *message* is not **undefined**, the **message** own property of the newly constructed object is set to ToString(*message*). ~~Otherwise, the **message** own property is set to the empty String.~~

15.11.4.4 Error.prototype.toString ( )

(Algorithm incorrect when message is the empty string or undefined)

6. If *msg* is undefined, then let ~~*R*~~ *msg* be ~~*msg*~~ the empty String; else let *msg* be ToString(*msg*).
~~7. Else, let *R* be the result of concatenating *name*, **":"**, a single space character, and ToString(*msg*).~~
~~8. Return *R*.~~
7. If *name* and *msg* are both the empty String, return **"Error"**.
8. If *name* is the empty String, return *msg*.
9. If *msg* is the empty String, return *name*.
10. Return the result of concatenating *name*, ":", a single space character, and *msg*.

15.11.7.4 new *NativeError* (message)

(Last paragraph, Algorithm incorrect when message is **undefined**)

If the argument *message* is not **undefined**, the **message** own property of the newly constructed object is set to ToString(*message*). ~~Otherwise, the **message** own property is set to the empty String.~~

A.1 Lexical Grammar

(Insert between *DecimalDigit* and *ExponentIndicator* production)

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

(incorrect right-hand-side)

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

(missing right-hand-side term)

Literal **::** *See 7.8*
NullLiteral
BooleanLiteral
NumericLiteral
StringLiteral
RegularExpressionLiteral

A.8.1 JSON Lexical Grammar

(incorrect right-hand-side)

JSONStringCharacter **::** *See 15.12.1.1*
~~*JSON*~~*SourceCharacter* **but not** *double-quote* **"** **or** *backslash* **** **or** *U+0000* ***thru*** *U+001F*
**** *JSONEscapeSequence*

ANNEX C

(missing item, add as first bullet item)

- The identifiers "**`implements`**", "**`interface`**", "**`let`**", "**`package`**", "**`private`**", "**`protected`**", "**`public`**", "**`static`**", and "**`yield`**" are classified as *FutureReservedWord* tokens within strict mode code. (7.6.12).

(next to last bullet item, confusing wording)

- An implementation may not extend, beyond that defined in this specification, the ~~associate special~~ meanings within strict mode functions of ~~to~~ 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).

Editorial Items with no Technical Significance

6 Source Text

(First paragraph)

ECMAScript source text is represented as a sequence of characters in the Unicode character encoding, version 3.0 or later. The text is expected to have been normalised to Unicode Normalised Form C (canonical composition), as described in Unicode Technical Report #15. Conforming ECMAScript implementations are not required to perform any normalisation of text, or behave as though they were performing normalisation of text, themselves. ECMAScript source text is assumed to be a sequence of 16-bit code units for the purposes of this specification. Such a source text may include sequences of 16-bit code units that are not valid UTF-16 character encodings. If an actual source text is encoded in a form other than 16-bit code units it must be processed as if it was first converted to UTF-16.

7.6 Identifier Names and Identifiers

(Missing :: in several grammar productions)

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

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

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

UnicodeEscapeSequence **::**
see 7.8.4.

7.8.4 String Literals

(Incorrect section reference)

The definition~~s~~ of the nonterminal *HexDigit* is given in ~~7.6~~ 7.8.3.

7.9.1 Rules of Automatic Semicolon Insertion

(Wrong font emphasis in *ThrowStatement* production)

ThrowStatement **:**
throw ~~***throw***~~ [no *LineTerminator* here] *Expression* **;**

("A" should be "An" in last sentence)

An *Identifier* in a **`break`** or **`continue`** statement should be on the same line as the **`break`** or **`continue`** token.

9.8.1 ToString Applied to the Number Type

(Incorrect font emphasis for variables in algorithm step 10)

10. Return the String consisting of the most significant digit of the decimal representation of *s*, followed by a decimal point '.', followed by the remaining *k*–1 digits of the decimal representation of *s*, followed by the lowercase character 'e', followed by a plus sign '+' or minus sign '–' according to whether *n*–1 is positive or negative, followed by the decimal representation of the integer abs(*n*–1) (with no leading zeros).

10.2.1.1.1 HasBinding (N)

(Period missing at end of step 3)

3. If it does not have such a binding, return **false**.

10.5 Declaration Binding Instantiation

(Period missing at end of step 6)

6. Let *argumentsAlreadyDeclared* be the result of calling *env's* HasBinding concrete method passing **`"arguments"`** as the argument.

11.2.3 Function Calls

(Incorrect font emphasis for variable in algorithm step 6.b.1)

6. If Type(*ref*) is Reference, then
    a. If IsPropertyReference(*ref*) is **true**, then
        i. Let *thisValue* be GetBase(*ref*).
    b. Else, the base of *ref* is an Environment Record
        i. Let *thisValue* be the result of calling the ImplicitThisValue concrete method of GetBase(*ref*).

12.10 The `with` Statement

(Period missing at end of step 4)

4. Let *newEnv* be the result of calling NewObjectEnvironment passing *obj* and *oldEnv* as the arguments.

12.11 The `switch` Statement

(Incorrect font emphasis for grammar productions in second *CaseBlock* algorithm steps 3, 5.b.i, 9.b, and 9.b.i)

3. Let *B* be the list of *CaseClause* items in the second *CaseClauses*, in source text order.
4. Let *found* be **false**.
5. Repeat letting *C* be in order each *CaseClause* in *A*
    a. If *found* is **false**, then
        i. Let *clauseSelector* be the result of evaluating *C*.
        ii. If *input* is equal to *clauseSelector* as defined by the === operator, then set *found* to **true**.
    b. If *found* is **true**, then
        i. If *C* has a *StatementList*, then
            1. Evaluate *C*'s *StatementList* and let *R* be the result.
            2. If *R*.value is not empty, then let *V* = *R*.value.
            3. *R* is an abrupt completion, then return *(R*.type, *V*, *R*.target).

9. Repeat (Note that if step 7.a.i has been performed this loop does not start at the beginning of *B*)
    a. Let *C* be the next *CaseClause* in *B*. If there is no such *CaseClause*, return (normal, *V*, empty).
    b. If *C* has a *StatementList*, then
        i. Evaluate *C*'s *StatementList* and let *R* be the result.
        ii. If *R*.value is not empty, then let *V* = *R*.value.
        iii. If *R* is an abrupt completion, then return (*R*.type, *V*, *R*.target).

15.1.2.1 eval (x)

(Incorrect font emphasis for variables in algorithm step 5)

If *radix* is 16, the number may also optionally begin with the character pairs **`0x`** or **`0X`**.

15.1.2.2 parseInt (string, radix)

(Missing "the" in last sentence of first paragraph)

5. Exit the running execution context *evalCtx*, restoring the previous execution context.

15.1.3 URI Handling Function Properties

(unnecessary "the")

A URI is composed of a sequence of components separated by component separators. The general form is:

Scheme **:** *First* **/** *Second* **;** *Third* **?** *Fourth*

where the italicised names represent components and ~~the~~ "**:**", "**/**", "**;**" and "**?**" are reserved characters used as separators.

15.2.2.1 new Object ([value])

("Assert" misspelled in step 2)

2. Assert: The argument *value* was not supplied or its type was Null or Undefined.

(Period missing at end of step 7)

7. Set the all the internal methods of *obj* as specified in 8.12.

15.3.2.1 new Function (p1, p2, … , pn, body)

("th" should not be superscript in algorithm steps 5.d.i and 5.e)

i. Let *nextArg* be the *k*'th argument.

d. Let *body* be the *k*'th argument.

15.4.4.9 Array.prototype.shift ( )

(Incorrect font emphasis for variables in algorithm step 7.e)

e. Else, *fromPresent* is **false**

15.4.4.22 Array.prototype.reduceRight (callbackfn [, initialValue])

(Incorrect font and emphasis for variables in second paragraph)

callbackfn is called with four arguments: the *previousValue* (or value from the previous call to *callbackfn*), the *currentValue* (value of the current element), the *currentIndex*, and the object being traversed. The first time the function is called, the *previousValue* and *currentValue* can be one of two values. If an *initialValue* was provided in the call to **`reduceRight`**, then *previousValue* will be equal to *initialValue* and *currentValue* will be equal to the last value in the array. If no *initialValue* was provided, then *previousValue* will be equal to the last value in the array and *currentValue* will be equal to the second-to-last value. It is a **TypeError** if the array contains no elements and *initialValue* is not provided.

15.4.5.1 [[DefineOwnProperty]] (P, Desc, Throw)

(Extra period at end of step 3.k)

k. If *succeeded* is **false**, return **false**~~.~~.

15.5.4.12 String.prototype.search (regexp)

(Incorrect font emphasis for variables in algorithm step 5)

5. Search the value *string* from its beginning for an occurrence of the regular expression pattern *rx*. Let *result* be a Number indicating the offset within *string* where the pattern matched, or –1 if there was no match. The **`lastIndex`** and **`global`** properties of *regexp* are ignored when performing the search. The **`lastIndex`** property of *regexp* is left unchanged.

15.7.3 Properties of the Number Constructor

(second paragraph, last word)

Besides the internal properties and the **`length`** property (whose value is **1**), the Number constructor has the following properties~~property~~:

15.7.4.2 Number.prototype.toString ([radix])

(Incorrect font emphasis for exception name in second paragraph)

If ToInteger(*radix*) is not an integer between 2 and 36 inclusive throw a **RangeError** exception. If ToInteger(*radix*) is an integer from 2 to 36, but not 10, the result is a String representation of this Number value using the specified radix. Letters **`a-z`** are used for digits with values 10 through 35. The precise algorithm is implementation-dependent if the radix is not 10, however the algorithm should be a generalization of that specified in 9.8.1.

15.7.4.6 Number.prototype.toExponential (fractionDigits)

(Misspelling, “decimal” in first sentence)

Return a String containing this Number value represented in decimal exponential notation with one digit before the significand's decimal point and *fractionDigits* digits after the significand's decimal point. If *fractionDigits* is **undefined**, include as many significand digits as necessary to uniquely specify the Number (just like in ToString except that in this case the Number is always output in exponential notation). Specifically, perform the following steps:

15.8.2 Function Properties of the Math Object

(First NOTE paragraph)

NOTE The behaviour of the functions **acos**, **asin**, **atan**, **atan2**, **cos**, **exp**, **log**, **pow**, **sin**, ~~and~~ **sqrt**, and **tan** is not precisely specified here except to require specific results for certain argument values that represent boundary cases of interest. For other argument values, these functions are intended to compute approximations to the results of familiar mathematical functions, but some latitude is allowed in the choice of approximation algorithms. The general intent is that an implementer should be able to use the same mathematical library for ECMAScript on a given hardware platform that is available to C programmers on that platform.

15.9.1.12 MakeDay (year, month, date)

(Delete extra right parenthesis in step 7 following "mn")

7. Find a value *t* such that YearFromTime(*t*) **==** *ym* and MonthFromTime(*t*) **==** *mn*~~)~~ and DateFromTime(*t*) **==** 1; but if this is not possible (because some argument is out of range), return **NaN**.

15.10.2.1 Notation

(Incorrect font, emphasis, and capitalization in fifth bullet item of second list)

- A *Matcher* procedure is an internal closure that takes two arguments -- a *State* and a *Continuation* -- and returns a *MatchResult* result. A *Matcher* attempts to match a middle subpattern (specified by the closure's already-bound arguments) of the pattern against the input String, starting at the intermediate state given by its *State* argument. The *Continuation* argument should be a closure that matches the rest of the pattern. After matching the subpattern of a pattern to obtain a new *State*, the *Matcher* then calls *Continuation* on that new *State* to test if the rest of the pattern can match as well. If it can, the ~~m~~*Matcher* returns the *State* returned by *Continuation*; if not, the *Matcher* may try different choices at its choice points, repeatedly calling *Continuation* until it either succeeds or all possibilities have been exhausted.

15.10.2.6 Assertion

(Incorrect capitalization in step 3 of second algorithm)

3. If ~~m~~*Multiline* is **false**, return **false**.

15.10.2.15 NonemptyClassRanges

(Incorrect font and emphasis for grammar productions in second and third paragraph)

The production *NonemptyClassRanges* **::** *ClassAtom NonemptyClassRangesNoDash* evaluates as follows:

The production *NonemptyClassRanges* **::** *ClassAtom* **-** *ClassAtom ClassRanges* evaluates as follows:

15.10.6.2 RegExp.prototype.exec(string)

(Extra period at end of step 4)

4. Let *lastIndex* be the result of calling the [[Get]] internal method of *R* with argument "**`lastIndex`**"~~.~~.

(Wrong font for “null” in step 9.a.ii)

9. Repeat, while *matchSucceeded* is **false**
    a. If *i* < 0 or *i* > *length*, then
        i. Call the [[Put]] internal method of *R* with arguments "**`lastIndex`**", 0, and **true**.
        ii. Return **null**.

15.10.6.3 RegExp.prototype.test(string)

(Section reference in first step of algorithm)

1. Let *match* be the result of evaluating the **`RegExp.prototype.exec`** (15.10.6.2~~3~~) algorithm upon this RegExp object using *string* as the argument.

15.11.6.5 TypeError

(Incorrect reference to 15.7.4.8)

Indicates the actual type of an operand is different than the expected type. See 8.6.2, 8.7.2, 8.10.5, 8.12.5, 8.12.7, 8.12.8, 8.12.9, 9.9, 9.10, 10.2.1, 10.2.1.1.3, 10.6, 11.2.2, 11.2.3, 11.4.1, 11.8.6, 11.8.7, 11.3.1, 13.2, 13.2.3, 15, 15.2.3.2, 15.2.3.3, 15.2.3.4, 15.2.3.5, 15.2.3.6, 15.2.3.7, 15.2.3.8, 15.2.3.9, 15.2.3.10, 15.2.3.11, 15.2.3.12, 15.2.3.13, 15.2.3.14, 15.2.4.3, 15.3.4.2, 15.3.4.3, 15.3.4.4, 15.3.4.5, 15.3.4.5.2, 15.3.4.5.3, 15.3.5, 15.3.5.3, 15.3.5.4, 15.4.4.3, 15.4.4.11, 15.4.4.16, 15.4.4.17, 15.4.4.18, 15.4.4.19, 15.4.4.20, 15.4.4.21, 15.4.4.22, 15.4.5.1, 15.5.4.2, 15.5.4.3, 15.6.4.2, 15.6.4.3, 15.7.4, 15.7.4.2, 15.7.4.4, ~~15.7.4.8,~~ 15.9.5, 15.9.5.44, 15.10.4.1, 15.10.6, 15.11.4.4 and 15.12.3.

A.1 Lexical Grammar

(Missing :: in several grammar productions)

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)”

ANNEX D

(extra period after “:” in items for clauses 13 and 14)

13:~~.~~ In Edition 3, the algorithm for the production *FunctionExpression* with an *Identifier* adds an object created

14:~~.~~ In Edition 3, the algorithm for the production *SourceElements* **:** *SourceElements SourceElement* did not