Errata (Revision 7)
For
Ecma/TC39/2009/018
ECMAScript 3.1 Candidate Specification
March 2, 2009 Draft

Open Issues

The value of the caller and arguments property of Function instances.
Sections 13.2, 15.3.4.5, and 15.3.5.

ES3.1 specified that function instances have non-writable, non-configurable properties whose values are **null**. This are intended to prevent conforming implementations from continuing to support legacy use of these property names. Should the specified value of these properties be **undefined** rather than **null**? Should this definition of these properties apply to all function instances or just strict mode functions? 13.2 and 15.3.4.5 creates them for all function instances but section 15.3.5 says they only apply to strict mode functions.

Should we clarify how much flexibility have in implementing complex algorithms in Section 15?
Section 15.

Add the following paragraph at the end of section 15:

> Functions in this section are generally defined using algorithms that are intended to describe the result produced by each function in the absence of any implicit error conditions. These algorithms are not intended to imply the use of any specific implementation technique. The each algorithm produces its results by using a specific sequence of calls to internal methods and abstract operations. Implementation may use other algorithms that use a different sequencing of these calls as long as an identical result is obtained in the absence of errors conditions that are not explicitly handled by the specified algorithm. If any internal method call invokes a get or set function of an accessor property that has side-effects or if an implicit error occurs the observable side-effects of a function are implementation dependent but are restricted to those that would be produced by some sequence of the internal method and abstract operation calls that would be made by the specified algorithm.

Normative Changes

Remove name property of function objects and assignment of names to functions.

10.5 (Arguments Object)

In step 2 of *MakeArgGetter* and step 3 of *MakeArgSetter*, replace the phrase "*env* as *Scope*, and the empty string as *Name*" with "and *env* as *Scope*".

11.1.5 (Object Initialiser)

In step 3 of both the algorithm for *PropertyAssignment* **: *get*** *PropertyName*… and *PropertyAssignment* **: *set*** *PropertyName*… delete the phrase ", and *descriptiveName* as the *Name*"

Delete step 2, and renumber of former steps 3-5 as 2-4.

13 (Function Definition Semantics)

In the first step 1, delete the phrase "and the string value of *Identifier* as *Name*". In the second step 1, delete the phrase "and an empty string as *Name*". In step 4 of the third algorithm, delete the phrase "and the string value of *Identifier* as *Name*".

13.2 (Creating Function Objects)

In the descriptive paragraph, replace the phrases "a Boolean flag *Strict*, and a possibly empty string *Name*," with ", and a Boolean flag *Strict*,".

Delete steps 15 and 16. Renumber existing steps 17 and 18 as 15 and 16.

15 (Native ECMAScript Objects)

Delete paragraph 10, which describes the **name** property for built-in functions.

15.3.2.1 (new Function …)

In step 10, delete the phrase "and the empty string as *Name*".

15.3.4 (Properties of Function Prototype)

Delete the third paragraph that begins "The value of the **name** property…"

15.3.4.5 (Function.prototype.bind)

Deletes steps 18-22 of the algorithm. Renumber step 23 as step 18.

15.3.5.4 (name)

Delete this section

Fix method calls to a with bound property use the correct this value

Section 10.2.1 (Environment Records)

Add the following row to the end of the table:

ImplicitThisValue()	Returns the value to use as the **this** value on calls to function objects that are obtained as binding values from this environment record.

Section 10.2.1.1.7 (ImplicitThisValue)

Add the following new section:

10.2.1.1.7 ImplicitThisValue()

Declarative Environment Records always return **null** as their ImplicitThisValue.

1. Return **null**.

Section 10.2.1.2.5 (ImplicitThisValue)

Add the following new section:

10.2.1.2.5 ImplicitThisValue()

Object Environment Records return **null** as their ImplicitThisValue unless their *provideThis* flag is **true**.

1. Let *envRec* be the object environment record for which the method was invoked.
2. If the *provideThis* flag of *envRec* is **true**, return the binding object for *envRec*.
3. Otherwise, return **null**.

Section 10.2.1.2 (Object Environment Records)

Add the following as the second paragraph of this section:

Object environment records can be configured to provide their binding object as an implicit this value for use in function calls. This capability is used to specify the behaviour of With Statement (12.10) induced bindings. The capability is controlled by a *provideThis* Boolean value that is associated with each object environment record. By default, the value of *provideThis* is **false** for any object environment record.

Section 10.2.1.2.5 (ImplicitThisValue)

Add the following new section:

10.2.1.2.5 ImplicitThisValue()

Object Environment Records return **null** as their ImplicitThisValue unless their *provideThis* flag is **true**.

4. Let *envRec* be the object environment record for which the method was invoked.
5. If the *provideThis* flag of *envRec* is **true**, return the binding object for *envRec*.
6. Otherwise, return **null**.

Section 11.2.3

Replace steps 6 and 7 with the following:

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*).
7. Else, Type(*ref*) is not Reference.
   a. Let *thisValue* be **null**.
8. Return the result of calling the [[Call]] internal method on *func*, providing *thisValue* as the **this** value and providing the list *argList* as the argument values.

Section 12.10

Add a new step 5 and renumber the previous steps 5-8 increasing them by 1:

1. Set the *provideThis* flag of *newEnv* to **true**.

Clarify meaning of "early" or "scan time" error reporting.

Section 7.8.5 (Regular Expression Literals)

Replace the last sentence of the last paragraph with:

> If the call to **`new RegExp`** would generate an error specified in 15.10.4.1, the error must be treated as is an early error (Section 16).

Section 11.1.5 (Object Initialiser)

Replace "If the above steps would throw a SyntaxError then an implementation must report the error immediately when scanning the program." With "If the above steps would throw a SyntaxError then an implementation must treat the error as an early error (Section 16)."

Section 16 (Errors)

Replace the first and second paragraphs with:

> An implementation must report most errors at the time the relevant ECMAScript language construct is evaluated. An *early error* is an error that can be detected and reported prior to the evaluation of any construct in the *Program* containing the error. An implementation must report early errors in a *Program* prior to the first evaluation of that *Program*. Early errors in **eval** code are reported at the time **`eval`** is called but prior to evaluation of any construct within the **eval** code. All errors that are not early errors are runtime errors.
>
> An implementation must treat any instance of the following kinds of errors as an early error:

Replace the paragraph immediately preceding the second bulleted list with:

> An implementation may treat any instance of the following kinds of errors as an early error, or the implementation may, at its option, treat them as runtime errors that are reported when the relevant construct is evaluated:

To the second bulleted list, add the following as the first item:

- Any syntax error.

In the first sentence of the paragraph immediately following the second bulleted list, replace the phrase: "An implementation shall not treat other kinds of errors as early errors even" with "An implementation shall not treat other kinds of errors as early errors even".

Single Section Changes

Section 8.6.2 (Internal Properties)

Add the paragraph as the last paragraph immediately before Table 5:

> If a host object provides its own internal [[GetOwnProperty]] method, the mutability implies by the property descriptors (8.10) it returns must be an upper bound on the possible mutations of the described property. For example, if a property is described as a data property and it may return different values over time, then the [[Writable]] attribute must be **true** even if no mechanism to change the value is exposed via the other internal methods. If the attributes may change over time or if the property might disappear, then the [[Configurable]] attribute must be **true**. If [[Writable]] and [[Configurable]] are both **false** and the property is described as a data property, then the host is effectively promising that the [[Value]] will be stable and may validly be cached.

Section 8.7 (Reference Specification Type)

Replace the bulleted item for IsPropertyReference with:

1. IsPropertyReference(V). Returns **true** if either the base value is an object or HasPrimitiveBase(V) is **true**; otherwise **false** is returned.

Section 8.10.4 (FromPropertyDescriptor)

Every call to [[Put]] needs to be replaced with call to [[DefineOwnProperty]]. The new version of the algorithm is:

1. If *Desc* is **undefined**, then return **undefined**.
2. Let *obj* be the result of creating a new object as if by the expression **new Object()** where **Object** is the standard built-in constructor with that name.
3. If IsDataDescriptor(*Desc*) is **true**, then
    a. Call the [[DefineOwnProperty]] internal method of *obj* with arguments "**`value`**", Property Descriptor {[[Value]]: *Desc*.[[Value]], [[Writable]]: **true**, [[Enumerable]]: **true**, [[Configurable]]: **true**}, and **false**.
    b. Call the [[DefineOwnProperty]] internal method of *obj* with arguments "**`writable`**", Property Descriptor {[[Value]]: *Desc*.[[Writable]], [[Writable]]: **true**, [[Enumerable]]: **true**, [[Configurable]]: **true**}, and **false**.
4. Else, IsAccessorDescriptor(*Desc*) must be **true**, so
    a. Call the [[DefineOwnProperty]] internal method of *obj* with arguments "**`get`**", Property Descriptor {[[Value]]: *Desc*.[[Get]], [[Writable]]: **true**, [[Enumerable]]: **true**, [[Configurable]]: **true**}, and **false**.
    b. Call the [[DefineOwnProperty]] internal method of *obj* with arguments "**`set`**", Property Descriptor {[[Value]]: *Desc*.[[Set]], [[Writable]]: **true**, [[Enumerable]]: **true**, [[Configurable]]: **true**}, and **false**.
5. Call the [[DefineOwnProperty]] internal method of *obj* with arguments "**`enumerable`**", Property Descriptor {[[Value]]: *Desc*.[[Enumerable]], [[Writable]]: **true**, [[Enumerable]]: **true**, [[Configurable]]: **true**}, and **false**.
6. Call the [[DefineOwnProperty]] internal method of *obj* with arguments "**`configurable`**", Property Descriptor {[[Value]]: *Desc*.[[Configurable]], [[Writable]]: **true**, [[Enumerable]]: **true**, [[Configurable]]: **true**}, and **false**.
7. Return *obj*.

Section 8.12.9 ([[DefaultValue]])

Steps 3,4, and 7 of the first algorithm are not correctly indented causing renumbering of them and other steps. The revised algorithms is:

1. Let *toString* be the result of calling the [[Get]] internal method of object *O* with argument "**`toString`**".
2. If IsCallable(*toString)* is **true** then,
    a. Let *str* be the result of calling the [[Call]] internal method of *toString*, with *O* as the **this** value and an empty argument list.
    b. If *str* is a primitive value, return *str*.
3. Let *valueOf* be the result of calling the [[Get]] internal method of object *O* with argument "**`valueOf`**".
4. If IsCallable(*valueOf)* is **true** then,
    a. Let *val* be the result of calling the [[Call]] internal method of *valueOf*, with *O* as the this value and an empty argument list.
    b. If *val* is a primitive value, return *val*.
5. Throw a **TypeError** exception.

Section 9.8 (ToString)

Replace the content of the Result cell for item Number with "See 9.8.1 below."

Section 10.5 (Arguments Object)

In step 3 of the CreateArgumentsObject algorithm, replace "Object" with "Arguments".

Immediately following step 6 add the following three steps and renumber the following steps accordingly:

7. Let *toString* be the standard built-in Object.prototype.toString method (15.2.4.2)
8. Call the [[DefineOwnProperty]] internal method on *obj* passing **"toString"**, the Property Descriptor {[[Value]]: *toString*, [[Writable]]: **true**, [[Enumerable]]: **false**, [[Configurable]]: **true**}, and **false** as arguments.
9. Call the [[DefineOwnProperty]] internal method on *obj* passing **"toLocaleString"**, the Property Descriptor {[[Value]]: *toString*, [[Writable]]: **true**, [[Enumerable]]: **false**, [[Configurable]]: **true**}, and **false** as arguments.

In the [[DefineOwnProperty]] algorithm steps 6 and 7 (and their substeps) should be substeps of step 5. Starting with step 5 the code of the algorithm should be:

5. If the value of *isMapped* is not **undefined**, then
    a. If IsAccessorDescriptor(*Desc*) is **true**, then
        i. Call the [[Delete]] internal method of *map* passing *P*, and **false** as the arguments.
    b. Else
        i. If *Desc*.[[Value]] is present, then
            1. Call the [[ThrowingPut]] internal method of *map* passing *P*, *Desc*.[[Value]], and *Throw* as the arguments.
        ii. If *Desc*.[[Writable]] is present and its value is **false**, then
            1. Call the [[Delete]] internal method of *map* passing *P* and **false** as arguments.
6. Return **true**.

Section 11.4.3 (typeof Operator)

Remove the word "not" in step 2:

2. If Type(*val*) is ~~not~~ Reference, then

Section 11.5.1 (Applying *)

In the second paragraph insert the word "binary" immediately before "double-precision".

Section 11.5.2 (Applying /)

In the 4th sentence of the first paragraph insert the word "binary" immediately after "double-precision".

Section 11.6.3 (Applying the Additive Operators)

In the third paragraph insert the word "binary" immediately before "double-precision".

Section 11.13.2 (Compund Assignment)

In step 6, replace "Return" with "Call". Add the following as step 7:

7. Return *r*.

Section 13.2 (Creating Function Objects)

Use [[DefineOwnProperty]] in algorithm to initially set own property values.

Assuming the above changes to 13.2 have already been made, then starting with step 10, replace the steps of the algorithm with the following:

10. Set the [[Extensible]] internal property of *F* to **true**.
11. Let *len* be the number of formal parameters specified in *FormalParameterList*. If no parameters are specified, let *len* be 0.
12. Call the [[DefineOwnProperty]] internal method of *F* with arguments **`"length"`**, Property Descriptor {[[Value]]: *len*, [[Writable]]: **false**, [[Enumerable]]: **false**, [[Configurable]]: **false**}, and **false**.
13. Let *proto* be the result of creating a new object as would be constructed by the expression **`new Object()`** where **`Object`** is the standard built-in constructor with that name.
14. Call the [[DefineOwnProperty]] internal method of *proto* with arguments **`"constructor"`**, Property Descriptor {[[Value]]: *F*, { [[Writable]]: **true**, [[Enumerable]]: **false**, [[Configurable]]: **true**}, and **false**.
15. Call the [[DefineOwnProperty]] internal method of *F* with arguments **`"prototype"`**, Property Descriptor {[[Value]]: *proto*, { [[Writable]]: **true**, [[Enumerable]]: **false**, [[Configurable]]: **false**}, and **false**.
16. Call the [[DefineOwnProperty]] internal method of *F* with arguments **`"caller"`**, PropertyDescriptor {[[Value: **null**, [[Writable]]: **false**, [[Enumerable]]: **false**, [[Configurable]]: **false**}, and **false**.
17. Call the [[DefineOwnProperty]] internal method of *F* with arguments **`"arguments"`**, PropertyDescriptor {[[Value: **null**, [[Writable]]: **false**, [[Enumerable]]: **false**, [[Configurable]]: **false**}, and **false**.
18. Return *F*.

Section 13.2.1 ([[Call]])

Add the following sentence at the end of the text for step 2: "If *F* does not have a [[Code]] internal property or if its value is an empty *FunctionBody*, then *result* is (**normal**, **undefined**, **empty**). "

Section 13.2.2 ([[Construct]])

Replace step 4 with:

4. Let *proto* be the value of calling the [[Get]] internal property of *F* with argument **`"prototype"`**.

Section 15.2.3.4 (Object.getOwnPropertyNames)

Starting with step 3 the algorithm should be :

3. Let *n* be 0.
4. For each named own property *P* of *O*
    a. Let *name* be the string value that is the name of *P*.
    b. Call the [[DefineOwnProperty]] internal method of *array* with arguments ToString(*n*), the PropertyDescriptor {[[Value]]: *name*, [[Writable]]: **true**, [[Enumerable]]: **true**, [[Configurable]]: **true**}, and **false**.
    c. Increment *n* by 1.
5. Return *array*.

Section 15.2.3.5 (Object.create)

Step 4 the algorithm should be :

4. If the argument *Properties* is present and not **undefined**, add own properties to *obj* as if by calling the standard built-in function **`Object.defineProperties`** with arguments *obj* and *Properties*.

Section 15.2.3.11 (Object.isSealed)

Section 15.2.3.12 (Object.isFrozen)

Step 2 of the algorithm for each function should be:

2. For each named own property name *P* of *O*,

Section 15.2.3.14 (Object.keys)

Step 5.a of the algorithm should be :

a. Call the [[DefineOwnProperty]] internal method of *array* with arguments ToString(*index*), the PropertyDescriptor {[[Value]]: *P*, [[Writable]]: **true**, [[Enumerable]]: **true**, [[Configurable]]: **true**}, and **false**.

Section 15.3.2.1 (new function)

Step 6 should be:

6. Let *body* be ToString(*body*).

Starting with step 9, the steps of the algorithm should be:

9. If *body* includes a Use Strict Directive then let *strict* be **true**, else let *strict* be **false**.
10. If *strict* is **true**, throw any exceptions specified in 13.1 that apply.
11. Return a new Function object created as specified in 13.2 using *P* as the *FormalParameterList* and *body* as the *FunctionBody*. Pass in the Global Environment as the *Scope* parameter and *strict* as the *Strict* flag.

Section 15.3.4 (Properties of the Function Prototype)

Add the following paragraph at the end of the section:

The **`length`** property of the Function prototype object is **0**.

Section 15.3.4.5 (Function.prototype.bind)

Algorithm is missing the creation of arguments and caller properties as required by 15.3.5. Immediately before the final step of the algorithm insert these steps:

18. Call the [[DefineOwnProperty]] internal method of *F* with arguments **`"caller"`**, PropertyDescriptor {[[Value: **null**, [[Writable]]: **false**, [[Enumerable]]: **false**, [[Configurable]]: **false**}, and **false**.

19. Call the [[DefineOwnProperty]] internal method of *F* with arguments **"arguments"**, PropertyDescriptor {[[Value: **null**, [[Writable]]: **false**, [[Enumerable]]: **false**, [[Configurable]]: **false**}, and **false**.

Section 15.3.5 (Properties of Function Instances)

Replace the first paragraph with the following two paragraphs:

> In addition to the required internal properties, every function instance has a [[Call]] internal property. Depending on how they are created (see 8.6.2 ,13.2, 15, and 15.3.4.5), function instances may have a [[HasInstance]] internal property, a [[Scope]] internal property, a [[Construct]] internal property, a [[FormalParameters]] internal property, a [[Code]] internal property, a [[TargetFunction]] internal property, a [[BoundThis]] internal property, and a [[BoundArgs]] internal property.
>
> The value of the [[Class]] internal property is **"Function"**.

Section 15.1 (Global Object)

Replace the second paragraph with:

> Unless otherwise specified, the standard built-in properties of the global object have attributes {[[Writable]]: **true**, [[Enumerable]]: **false**, [[Configurable]]: **true**}.

Section 15.4.4.4 (Array.prototype.concat)

In step 5.a.iii, steps 4 to 6 inclusive should be substeps of step 3 numbered a-c. Steps 5.a.iii.7-8 should be renumbered 5.a.iii.4-5. (David-Sarah Hopwood)

Section 15.4.4.19 (Array.prototype.map)

Delete the paragraph:

> The final state of *O* is unspecified if in the above algorithm any call to the [[ThrowingPut]] internal method of *O* throws an exception.

Section 15.5.4.21 (String.prototype. toJSON)

In steps 8.f.ii and 8.f.iii replace "*match*" with *result*"

Section 15.6.4.4 (Number.prototype.toJSON)

In step 5, replace "*valueOf*" with "*R*"

Section 15.6.4.4 (Boolean.prototype.toJSON)

In the second to last step (step 11 prior to renumbering) replace "*valueOf*" with "*R*"

Section 15.6.4.4 (Number.prototype.toJSON)

In step 5, replace "*valueOf*" with "*R*"

Section 15.9.4.2 (Date.parse)

After the line consisting of "**Date.parse(*x*.toUTCString())**" add the following line:

```
Date.parse(x.toISOString())
```

In the last sentence, immediately after the phrase “when given any string value that” insert “does not conform to the Date Time String Format (15.9.1.15) and that”

Section 15.11.4.4 (Error.prototype.toString)

Replace the entire single line body of this section with:

> The following steps are taken:
>
> 1. Let *O* be the **this** value.
> 2. If Type(*O*) is not Object, throw a **TypeError** exception.
> 3. Let *name* be the result of calling the [[Get]] internal method of *O* with argument **"name"**.
> 4. If *name* is **undefined**, then let *name* be **"Error"**; else let *name* be ToString(*name*).
> 5. Let *msg* be the result of calling the [[Get]] internal method of *O* with argument **"message"**.
> 6. If msg is undefined, then let *R* be *msg*.
> 7. Else, let *R* be the result of concatenating *name*, **":"**, a single space character, and ToString(*msg*).
> 8. Return *R*.

Section 15.12.1.2 (JSON Syntactic Grammar)

Section A.8.2 (JSON Syntactic Grammar)

In the definition of the production JSONValue, replace “*NullLiteral*” with “*JSONNullLiteral*” and “*BooleanLiteral*” with “*JSONBooleanLiteral*”.

Section 15.12.3 (JSON.stringify)

In Str abstract operation, line 9.a should be:

> a. If *value* is finite then return ToString(*value*).

Editorial Changes

Clarify meaning and use of “Type(x)”

Section 5.2

Delete the last sentence of paragraph 3 which begins: “Type(x) is used as…”

Section 8

Add the following as the fourth paragraph:

> Within this specification, the notation “Type(*x*)” is used as shorthand for “the type of *x*” where “type” refers to the ECMAScript language and specification types defined in this section.

Section 11.8.6 and 11.8.7

In step 5 replace “*rval* is not an object” with “Type(*rval*) is not Object”

Section 13.2.2 ([[Construct]])

In step 5 replace “*proto* is an Object” with “Type(*proto*) is Object”

In step 6 replace “*proto* is not an Object” with “Type(*proto*) is not Object”

Section 15.2.2.1

In Step 1.a, 1.b, 1.c, and 1.d replace "the type of *value*" with "Type(*value*)".

Section 15.7.4

In the last paragraph replace the phrase "if the type of the **this** value is a Number" with the phrase "if Type(**this** value) is Number".

Section 5.1.6 (JSON Grammar)

In the second sentence of the paragraph, replace "token" with "tokens"

Section 5.1.7 (Grammar Notation)

Page 10, the word "*but*" in "IdentifierName but ***not*** ReservedWord" should be bold and not italic

Section 5.2 (Algorithm Conventions)

The lines:

NOTE

floor(x) = x–(x modulo 1).

Should be formatted in italics using the NOTE conventions

Section 6 (Source Tex)

In the first note paragraph replace "UTF-16 encoding" as follows:

Although this document sometimes refers to a "transformation" between a "character" within a "string" and the 16-bit unsigned integer that is the code unit of that character, there is actually no transformation because a "character" within a "string" is actually represented using that 16-bit unsigned value.

Section 7.2 (White Space) and Section 7.3 (Line Terminators)

The column 1 heading in the tables should be "*Code Point Value*" instead of "*Code Unit Value*" because it is talking abstractly about Unicode characters.

In the last paragraph of 7.3 insert the word "Unicode" before the first occurrence of "character" and expand the right margin of that paragraph to match other paragraphs of the section.

Section 7.5.3 (Future Reserved Words)

"*Note*" needs to be all caps.

Section 8 (Types)

The last sentence of the last paragraph should begin "Specification type values may be used…"

Section 8.4 (The String Type)

The Note needs to be in italics.

In the first two sentence of the last paragraph (before the NOTE) add these indicated words:

> When a string contains actual textual data, each element is considered to be a single UTF-16 code unit. Whether or not this is the actual storage format of a String, the characters within a String are numbered by their initial code unit element position as though they were represented using UTF-16.

Section 8.6 (The Object Type)

In the first bulleted item, delete the word "boolean".

In the last sentence, replace "normal" with "named".

Section 8.6.1 (Property Attributes)

In the data property table description of [[Writable]] change "assign the property's value" to "change the property's [[Value]] attribute".

In the data property table description of [[Configurable]] add "(other than [[Value]]" between "attributes" and "will".

Section 8.6.1 (Object Internal Properties)

In table 4 [[PrimitiveValue], the word "primitive" in the middle column should be in italics.

In table 4 [[Property], the word "boolean" in the middle column should have a capital "B".

Combine the first and third paragraphs following table 4 by deleting the third paragraph an replacing the first paragraph with the following:

> All ECMAScript objects have an internal property called [[Prototype]]. The value of this property is either **null** or an object and is used for implementing inheritance. Whether or not a native object can have a host object as its [[Prototype]] depends on the implementation. Every [[Prototype]] chain must have finite length (that is, starting from any object, recursively accessing the [[Prototype]] internal property must eventually lead to a **null** value). Named data properties of the [[Prototype]] object are inherited (are visible as properties of the child object) for the purposes of get access, but not for put access. Named accessor properties are inherited for both get access and put access.

In table 5 the descriptions of [[TargetFunction]] and [[BoundThis]] the phrase "objects that are bound using" should be "objects created using".

In table 5 the description of [[BoundArguments] the phrase "objects bound using" should be "objects created using".

In table 5 the description of [[ParameterMap]] the phrase "arguments object have" should be "arguments objects have".

Section 8.7 (Reference Specification Type)

In the first sentence of the second paragraph replace the phrase "A **Reference** is a reference to a resolved…" with "A **Reference** is a resolved…"

Section 8.7.1 (GetValue)

In step 3.b, "base" should be in italics.

In step 5.a, replace "*(N,S)*" with section reference (10.2.1.2.4)

In the note, replace the phrase “its use in that step” with “its use in the next step”

Section 8.7.3 (PutValue)

In step 4.a, insert section reference (15.1) after “global object”.

Replace stepo 5.b with:

> a. Call the SetMutableBinding (10.2.1.1.3) concrete method of *base*, passing GetReferencedName(*V*), *W*, and IsStrictReference(*V*) as arguments.

In the second sentence of the paragraph between the two algoritms, the “*Base*” should have a lower case “*b*”.

Section 8.10 (Property Descriptor)

A tab is need in the section heading so the section title aligns with the following paragraph.

In the second paragraph replace “{value: 42, writable: false, configurable: true}” with “{[[Value]]: 42, [[Writable]]: **false**, [[Configurable]]: **true**}”.

In the last paragraph replace “named *value*” with “named [[Value]]”.

Section 8.10.5 (ToPropertyDescriptor)

All of the quoted strings should be bold font, Courier New.

Section 8.12.8 ([[Delete]])

In step 3.a, P and O should be italic.

Section 9.3.1 (To Number applied to String)

Restore lost bullets to last two lines of section.

Section 9.12 (Same Value Algorithm)

Made step 7 align with other top level steps.

Section 10.2.1.1 (Declarative Environment Records)

In the first sentence of the second paragraph replace the word “binds” with “bindings”.

In the table, all mentions of “V” and “N” in the Purpose column should be italic.

Section 10.2.1.1.3 (SetMutableBinding)

“TypeError” in the second sentence should be bold font.

Section 10.2.1.1.4 (GetBindingValue)

“ReferenceError” in the third sentence should be bold font.

Section 10.2.1.1.5 (CreateImmutableBinding)

“undefined” in the first sentence should be bold font.

Section 10.4 (Establishing an Execution Context)
In the first sentence of the second pargraph insert the section reference "(10.6)" immediately after the words "declaration binding instantiation".

Section 10.4.3 (Function code)
In step 8, "F" in "F's" should be in italics.

Section 10.5 (Arguments Object)
In step 1 of the makeArgGetter and step 2 of makeArgSetter algorithms, "body" should be italic.
In step 1 of the makeArgSetter algorithm, "param" should be italic.
In step 2 of the [[DefineOwnProperty] algorithm, "map" should be italic.

Section 10.6 (Declaration Binding Instantations)
In steps 4 and 5, delete the phrase "the execution context's" and make "code" italic.
In step 7, delte "the" and make "code" italic.

Section 11.1.1 (The this keyword)
Replace the sole sentence with:

> The **`this`** keyword evaluates to the value of the ThisBind of the current execution context.

Section 11.1.2 (Identifier Reference)
Replace "using the scoping rules" with "by performing Identifier Resolution as"

Section 11.1.4 (Array Initialiser)
In step 6 of the 5th algorithm, fix the 2nd-3rd line indenting.

Section 11.1.5 (Object Initialiser)
In step 4 of the 5th algorithm, "propValue" should be italic.

Section 11.6.1 (Addition operator)
Remove the parenthesized note from step 3 and add the following paragraph as the first paragraph of the section note:

> *Step 7 differs from step 3 of the comparison algorithm for the relational operators(11.8.5), by using or instead of and.*

Section 11.8.5 (Abstract Relational Comparision)
In step 3.a, "ny" should be italic.
Remove the parenthesized note from step 3 and add the following paragraph to the end of the section note:

> *Step 3 differs from step 7 in the algorithm for the addition operator **+** (11.6.1) in using and instead of or.*

Section 11.9.3 (Abstract Equality Comparision)

In step 1, delete the word “from”.

In the NOTE, delete “2.0” following the word “Unicode”

Section 11.12 (Conditional Operator)

In step 2, delete the word “go”

Section 12 (Statments)

Replace the TBD note at the end of the Semantics text with:

NOTE
Several widely used implementations of ECMAScript are known to support the use of FunctionDeclaration *as a* Statement*. However there are significant and irreconcilable variations among the implementations in the semantics applied to such* FunctionDeclarations*. Because of these irreconcilable difference, the use of a* FunctionDeclaration *as a* Statement *results in code that is not reliably portable among implementations. It is recommended that ECMAScript implementations either disallow this usage of* FunctionDeclaration *or issue a warning when such a usage is encountered. Future editions of ECMAScript may define alternative portable means for declaring functions in a* Statement *context.*

Section 12.4 (Expression Statment)

Reformat the paragraph beginning with “Note” as a proper informative NOTE as follows:

> *NOTE*
> *An* ExpressionStatement *cannot start with an opening curly brace because that might make it ambiguous with a* Block*. Also, an* ExpressionStatement *cannot start with the* **`function`** *keyword because that might make it ambiguous with a* FunctionDeclaration.

Section 12.6.2 (do-while Statement)

In step 3, “*true*” should be bold and not italic.

Section 12.10 (with Statment)

Replace the Description paragraph with:

> The **`with`** statement adds an object environment record for a computed object to the lexical environment of the current execution context. It then executes a statement with using this augmented lexical environment. Finaly, it restores the original lexical environment.

In step 2, add an addition “)” immediately before the period.

In step 3, delete “(O,E)”

Section 12.14 (try Statment)

In the algorithm for the production *Catch* **`: catch (`***Identifier* **`)`** *Block* :

In step 3, delete “(E)”

In step 4, delete “(*N*)”

In step 5, delete "(*N,V,S*)"

Section 13 (Function Definition)

The section reference "10.3.3" in the definition of *FunctionDeclaration* production should be "10.6".

In the algorithm for the second *FunctionExpression* production:

In step 1, delete "(E)"

In step 3, delete "(*N*)"

In step 5, delete "(*N,V*)"

Section 13.2.1 ([[Call]])

Replace step 2 with:

2. Let *result* be the result of evaluating the *FunctionBody* that is the value of *F*'s [[Code]] internal property.

Section 15 (Native ECMAScript Objects)

Change section name to "Standard Built-in ECMAScript Objects"

Section 15.2.2.1 (new Object)

Change the section reference at the end of the last line in step 2 from "8.6.2" to "8.12"

Section 15.2.3.6 (Object.defineProperty)

In the first sentence insert "an" before "existing".

Section 15.2.3.7 (Object.defineProperties)

In step 3, "P" should be italic. In the second sentence of the paragraph at the end of the section, "O" should be italic.

Section 15.2.3.8 (Object.seal)

In the paragraph at the end of the section, the two occurrences of "O" should be italic.

Section 15.2.3.9 (Object.freeze)

Section 15.2.3.12 (Object.isFrozen)

In step 2.b , insert "is **true**," immediate falling "(*desc*) ".

Section 15.2.3.14 (Object.keys)

In the first sentence "O" should be italic.

15.3.4 (Properties of Function Prototype)

Delete the paragraph that begins "It is a function with an…"

Section 15.3.4.5 (Function.prototype.bind)

In the first sentence, “thisArg”, “arg1” and “arg2” should be italic. In step 3 “arg1” and “arg2” should be italic. In step 6, delete the text “the value of”.

Section 15.3.4.5 (Function.prototype.bind)
Section 15.3.5.2 (prototype)
Add the following note at the end of each section:

> *NOTE*
> *Function objects created using* **`Function.prototype.bind`** *do not have a* **`prototype`** *property*.

Section 15.4.4.4 (Array.prototype.concat)
Step 4, delete the word “the” immediately before “*O*”
Step 5.c.1, delete extra period at end of sentence.

Section 15.4.4.6 (Array.prototype.pop)
Step 5.a, “len” should be italic.

Section 15.4.4.11 (Array.prototype.sort)
In the bulleted list that specifies the requirements for a consistent comparision function, “a” and “b” in the second bullet item should be italic.
In step 1 of the SortCompare algorithm, “j” should be italic.
In step 2 of the SortCompare algorithm, “k” should be italic.

Section 15.4.4.13 (Array.prototype.unshift)
Step 5.f, “k” should be italic.
Step 8.c, “k” should be italic.

Section 15.4.4.18 (Array.prototype.forEach)
Fix the indent of steps 7.c.i and 7.c.ii

Section 15.4.4.21 (Array.prototype.reduce)
Relabel the step labeled “8.b” (the first step indented under step 8) as “8.a”
Relabel the step labeled “8.a” (the second step indented under step 8) as “8.b”

Section 15.4.5.1 ([[DefineOwnProperty]])
In step 1, the phrase “**undefined** or and accessor descriptor” should be “**undefined** or an accessor descriptor”

Section 15.6.4.4 (Boolean.prototype.toJSON)
Renumber the algorithm, starting at step 1 instead of 7.

Section 15.8.2 (Function Properties of math object)

In the last NOTE paragraph, replace "*fdlibm-comment@sunpro.eng.sun.com*" with "*http://www.netlib.org/fdlibm*".

Section 15.9.4.4 (Date.now)

Replace the single sentence description of the function with:

> The **now** function return a Number value that is the time value designating the UTC date and time of the occurrence of the call to **now**.

Section 15.9.5.27 (Date.prototype.setTime)

Delete step 1 of the algorithm as it is redundant (because of section introduction) and other similar algorithms in this section to not have a corresponding step.

Section 15.10.2.8 (Atom)

In step 5 of the algorithm for *CharacterSetMatcher* replace "else" with "then"

Section 15.10.2.10 (CharacterEscape)

The section heading needs to be reformatted as a level 4 heading.

The algorithm:

1. Let *ch* be the character represented by *ControlLetter*.
2. Let *i* be *ch*'s code unit value.
3. Let *j* be the remainder of dividing *i* by 32.
4. Return the Unicode character numbered *j*.

Should be restated as:

1. Let *ch* be the Unicode character represented by *ControlLetter*.
2. Let *i* be *ch*'s code point value.
3. Let *j* be the remainder of dividing *i* by 32.
4. Return the code unit numbered *j*.

Section 15.10.6.2 (RegExp.prototype.exec)

In step 11 replace "set **lastIndex** to *e*." with "then"

Section 15.11.6.1 (EvalError)

The list of referenced sections should be:

> See 8.7.2, 12.2.1 and 13.1.

Section 15.11.6.2 (RangeError)

The list of referenced sections should be:

> See 15.1.2.1, 15.3.2.1, 15.10.2.5, 15.10.2.9, 15.10.2.15, 15.10.2.19, and 15.10.4.1.

Section 15.11.6.3 (ReferenceError)

The list of referenced sections should be:

> See 11.1.5, 13.1, 15.1.2.1, 15.10.2.2, 15.3.2.1, 15.10.2.5, 15.10.2.9, 15.10.2.15, 15.10.2.19, 15.10.4.1, and 15.12.2.

Section 15.11.6.4 (SyntaxError)

The list of referenced sections should be:

> See 11.1.5, 13.1, 15.1.2.1, 15.10.2.2, 15.3.2.1, 15.10.2.5, 15.10.2.9, 15.10.2.15, 15.10.2.19, 15.10.4.1, and 15.12.2.

Section 15.11.6.5 (TypeError)

The list of referenced sections should be:

> See 8.12.5, 8.12.8, 8.12.9, 8.12.10, 9.9, 9.10, 10.2.1, 10.2.1.1.3, 10.5, 11.2.2, 11.2.3, 11.4.1, 11.8.6, 11.8.7, 11.3.1, 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, 15.3.5.3, 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, 15.5.4.21, 15.6.4.2, 15.6.4.3, 15.6.4.4,15.7.4, 15.7.4.2, 15.7.4.4, 15.7.4.8, 15.9.5, 15.9.5.9, 15.9.5.44, 15.10.4.1, 15.10.6, 15.11.4.4, and 15.12.3.

Section 16

Replace paragraph 2 (“An implementation must…”) with:

> An implementation must report early as a syntax error any instance of the following kinds of runtime errors:

(David-Sarah Hopwood)

Section A.8.1

In the production *JSONEscapeCharacter* move **one of** onto the same line as the rule name, immediately following the **::**

Annex C

Delete the last bulleted item, which is redundant.

Revised Versions of Annex D and E follow with highlight changes.

Note that additional items for these Annexes are likely to be identified during the candidate specification evaluation process.

Annex D

(Informative)

Correction and Clarifications in Edition 3.1 with Possible 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. For Edition 3.1 the specification text for all internal references and invocations of standard built-in objects and methods has been clarified by making it implicit 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 user-defined valueOf or toString methods with side-effects are invoked during the evaluation process.

11.1.4 Edition 3.1 clarifies the fact that a trailing comma at the end of an *Array Initialiser* does not add to the length of the array. This is not a semantic change from Edition 3 but some implementation may have previously misinterpreted it.

11. 2.3 Edition 3.1 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*.

12.2 In Edition 3 the algorithm for evaluating the production *VariableDeclaration* **:** *Identifier Initialiser* was specified in a manner that is incorrect for situations where a *VariableDeclaration* is nested within a *WithStatement* for an object that has a property name that is identical to the *Identifier* in the *VariableDeclaration*. In this situation, the Edition 3 specification causes the value of the *Initialiser* to be assigned to the object's property rather than the actual variable introduced by the declaration. For Edition 3.1 the algorithm has been revised such that the value of the *Initialiser* will be assigned to the associated variable regardless of any such nesting. Existing ECMAScript code that depends up faithful implementation of this Edition 3 semantics will not operated as expected using an implementation that conforms to the Edition 3.1 specification.

12.4 In Edition 3, an object is created, as if by **`new Object()`**, to used 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 3.1, when an exception parameter is called as a function, **undefined** as 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 Objct.prototype are visible as identifiers within that scope. In practice most implementations of Edition 3 have not implemented this semantics. In Edition 3.1 changes the specified semantics by using a Declarative Environment Record bind the name of the function.

15.1.2.1. Implementations are no longer permitted restrict to 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.

15.10.6 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 Edition 3.1 which Introduce Incompatibilities with Edition 3.

Section 7.1: Unicode format control characters are no longer stripped from ECMAScript source text before processing.

Section 7.2: Unicode characters <NEL>, <ZWSP>, and <BOM> are now treated as whitespace.

Section 7.3: Line terminator characters that are preceded by an escape sequence are now allowed within a string literal token.

Section 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.

Section 7.8.5: In Edition 3.1 requires scan time reporting of any possible RegExp constructor errors that would be produced when converting a *RegularExpressionLiteral* to a RegExp object. Prior to Edition 3.1 implementations were permitted to defer the reporting of such errors until the actual execution time creation of the object.

Section 10.4.2: In Edition 3.1, indirect calls to the **`eval`** function use the global environment as the variable environment and lexical environment for the eval code. In Edition 3.1, the variable and lexical environments of the caller of an indirect **`eval`** was used and the environments for the eval code.

Section 7.8.5: In Edition 3.1 unescaped "/" characters may appear as a *CharacterClass* in a regular expression literal. In Edition 3 such a character would have been interpreter as the final character of the literal.

Section 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.

Section 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.

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

Section 15.1.22: The specification of the function **`parseInt`** no longer allows implementations to treat strings beginning with a **`0`** character as octal values.

Sections 15.3.4.3, 15.3.4.4: In Edition 3 passing **undefined** or **null** as the first argument to **`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 3.1, 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 that are called in this manner with **null** or **undefined** passed as the this value will in many cases cause behaviour in Edition 3.1 implementations that differ from Edition 3 behaviour. In particular, 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.

Section 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.

Section 15.10.2.12: In Edition 3.1, \s now matches <NEL>, <ZWSP>, and <BOM>.

Section 15.10.4.1: In Edition 3, the exact form of the string value of the **`source`** property of an objected created by the **`RegExp`** constructor is implementation defined. In Edition 3.1, the string must conform to certain specified requirements and hence may be different from that produced by an Edition 3 implementation.

Section 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 3.1 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.

Sections 15.11.2.1, 15.11.4.3: In Edition 3.1, 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.

Section 15.11.4.4: In Edition 3, the result of **`Error.prototype.toString`** is implementation defined. In Edition 3.1, the result is fully specified and hence may differ from some Edition 3 implementations