JScript Deviations from ES3

-- *Draft* –

Sep 24, 2007

Pratap Lakshman
Microsoft Corporation

This draft provides a description of JScript's deviations and extensions to ECMAScript 3rd Edition Specification. This is work-in-progress, and there are still a few sections to document or correct.

1 Introduction

This document lists the differences between JScript and ECMAScript Edition 3 standard. Please send feedback on this document to pratapL (pratapL@microsoft.com).

2 Deviations from ES3

The following are JScript's (COM Classic) deviations from ECMAScript Edition 3. Each deviation is introduced with a title, a brief note, an example, and the output from running the example on IE7, FF2.0.0.5, Opera 9.02, and Safari 3.0, running on WinXP SP2 32 bit edition. The relevant section within the ES3 specification is quoted as part of the title.

2.1 White Space: §7.2

JScript does not support the \v vertical tab character as a white space character. It treats \v as v.

Example:

```
<script>
alert('\v supported ' + (String.fromCharCode(11) == '\v'));
</script>
```

Output:
IE: false
FF: true
Opera: true
Safari: true

2.2 Future Reserved Words: §7.5.3

JScript permits certain future reserved words (`abstract, int, short, boolean, interface, static, byte, long, char, final, native, synchronized, float, package, throws, goto, private, transient, implements, protected, volatile, double, public`) to be used as identifiers.

While all the implementations permit future reserved words to be declared as identifiers, there are some differences.

Example:

```
<script>
// test if a word can be declared as var
function testdecl(words) {
    var e;
    for (var i = 0; i < words.length; i++) {
        var word = words[i];
        try {
            eval('var ' + word + ';');
            document.write('test failed: ' + word + ' is
declarable' + '<br>');
        }
        catch(e) {}
    }
}


// keywords
var keywords =
['break',        'else',      'new',      'var',        'case',
'finally',      'return',    'void',     'catch',      'for',
'switch',       'while',     'continue', 'function',   'this',
'with',         'default',   'if',       'throw',      'delete',
'in',           'try',       'do',       'instanceof', 'typeof'];

// future reserved words
var reserved =
['abstract',     'enum',      'int',      'short',      'boolean',
'export',       'interface', 'static',   'byte',       'extends',
'long',         'super',     'char',     'final',      'native',
'synchronized', 'class',     'float',    'package',    'throws',
'const',        'goto',      'private',  'transient',
'debugger',
'implements',   'protected', 'volatile', 'double',     'import',
'public'];

testdecl(keywords);
testdecl(reserved);
</script>
```

Output:
Differences in the set of reserved words that can be declared are as shown below - ‘n’ indicates that a particular word is not declarable and blank cells indicate that a particular word is declarable.

	Declarable?			
reserved words	IE	FF	Opera	Safari
abstract				
enum	n			n
int				
short				
boolean				
export	n	n		n
interface				
static				
byte				
extends	n			n

long				
super	n			n
char				
final				
native				
synchronized				
class	n			n
float				
package				
throws				
const	n	n	n	n
goto			n	
private				
transient				
debugger	n	n		n
implements				
protected				
volatile				
double				
import	n	n		n
public				

2.3 String Literals: §7.8.4

JScript allows C-style escaped newlines in string literals; according to ES3 this would be a syntax error for unterminated string constant

Example:

```
<script>
var s = "this is a \
        multiline string";
</script>
```

Output:
IE: taken as a single string.
FF: same as IE
Opera: same as IE
Safari: same as IE

IE eats away the '\' and the character following it. The `s.length` will evaluate to 34 (this includes the leading blanks in the second line). FF, Opera and Safari emulate IE.

2.4 Arguments Object: §10.1.8

There is no uniform handling of a variable named `arguments` amongst the various script engine implementations.

Example:

```
<script>
function foo() {
    document.write(arguments);
    document.write(arguments[0]);
    eval("arguments = 10;");
    document.write(arguments);
    document.write(arguments[0]);
}

foo("test");
</script>
```

Output:
IE: [object Object]test[object Object]test
FF: [object Object]test10undefined
Opera: "testtest10undefined
Safari: [object Arguments]test10undefined

Note:
Refer §10.1.6 - arguments is added as a {DontDelete} property to the "variable object" before the variable instantiations steps (§10.1.3) take place. Now, consider the following:

```
<script>
function foo() {
    document.write(arguments);
    document.write(arguments[0]);
    arguments = 10;
    document.write(arguments);
    document.write(arguments[0]);
}
foo(42);
</script>
```

And here is the outputs:
IE: [object Object]4210undefined
FF: same as IE
Opera: 424210undefined
Safari: [object Arguments]4210undefined

Note:
In JScript `arguments` isn't included in the variable context that is used by the `eval`, so the eval'ed assignment isn't actually changing the value of the enclosing functions arguments variable.

2.5 Global Object: §10.2.1

In JScript, global functions cannot be iterated using the global `this` object.

Example:

```
<script>
var __global__ = this;

function invisibleToIE() {
  document.write("IE can't see me");
}

__global__.visibleToIE = function() {
  document.write("IE sees me");
}

for (func in __global__) {
  var f = __global__[func];

  if (func.match(/visible/)) {
    f();
  }
}
</script>
```

Output:
IE: IE sees me
FF: IE sees meIE can't see me
Opera: IE can't see meIE sees me
Safari: same as Opera

IE incorrectly implements the `delete` operator when applied to the global object. Attempting to execute `delete this.invisibleToIE` in the example below produces a runtime error (object doesn’t support this action) in IE while in FF delete returns false. The FF behaviour seems correct according to §11.4.1, §10.2.1, and §8.6.2.5

Example:

```
<script>
var __global__ = this;

function invisibleToIE() {
  document.write("IE can't see me");
}

__global__.visibleToIE = function() {
  document.write("IE sees me");
}
```

```
document.write(delete this.invisibleToIE);
</script>
```

Output:
IE: runtime error (object doesn't support this action)
FF: false
Opera: same as FF
Safari: same as FF

In IE the global object does not inherit from `Object.prototype` even though its "type" is object. §15 – paragraph 8 seems to imply that the "Global Object" should inherit from `Object.prototype` (the value of its [[Prototype]] property is implementation dependent, but what ever it is it must be a "built-in prototype" and hence must follow the rules of paragraph 8. Of the standard methods of `Object.prototype`, the only one support by the global object in IE is `toString`

Example:

```
<script>
var __global__ = this;
document.write(typeof(__global__) + '<br>');

var f = ['toString', 'toLocaleString',
         'valueOf', 'hasOwnProperty',
         'isPrototypeOf', 'propertyIsEnumerable'];

for (i = 0; i < f.length; i++) {
  test(f[i]);
}

function test(s) {
    if (__global__[s]) {
        document.write(s + ' supported' + '<br>');
    }
}
</script>
```

Output:
IE:
object
toString supported

FF:
object
toString supported
toLocaleString supported
valueOf supported
hasOwnProperty supported

isPrototypeOf supported
propertyIsEnumerable supported

Opera: same as FF
Safari: same as FF

2.6 *Array Initialiser: §11.1.4*

Trailing commas in array literals add to the length, but they shouldn't. JScript treats the empty element after the trailing comma as `undefined`.

Example:

```
<script>
document.write([1, 2, 3,].length);
</script>
```

Output:
IE: 4
FF: 3
Opera: same as FF
Safari: same as FF

2.7 *Function Expressions: §11.2.5*

In JScript the identifier in a function expression is visible in the enclosing scope because such expressions are treated as function declarations.

Example:

```
<script>
var foo = function bar(b)
{
    if (b == true) {
        bar(); // works because bar is visible inside itself
    }
    else {
        document.write("hello");
    }
}

foo();       // works because foo is a ref to Function object
bar(false);  // should fail because bar is not visible here
</script>
```

Output:
IE: prints "hellohello"
FF: prints "hello" followed by syntax error (bar is not defined)
Opera: prints "hello" followed by ReferenceError (Reference to undefined variable: bar)
Safari: prints "hello"

Note:
IE treats nested function expressions containing a function name as a function declaration that is scoped to the enclosing function (applying the second bullet rule from §10.1.3 instead of case 3 of the semantics in §13. In the following example the forward reference to x gets resolved in IE, whereas we get a syntax error in FF and TypeError in Opera.

Example:

```
<script>
function f(x) {
    x();
    y = function x() {document.write("inner called ")};
    document.write(x);
    document.write(arguments[0]);
}
document.write("test 4 ");
f("param");
</script>
```

Output:
IE: test 4 inner called function x() {document.write("inner called ")}function x() {document.write("inner called ")}
FF: test 4
Opera: test 4
Safari: test 4
Here is another example of the same issue:

```
<script>
function foo() {
    function bar() {}
    var x = function baz(z) { document.write("baz" + z); if (z)
baz(false); };
    x(true);   // legal;
    bar();     // legal
    baz(true); // should be illegal, is legal
}
foo();
</script>
```

The function statement bar should add "bar" to foo's variable object. The function expression baz should NOT add "baz" to foo's variable object, but it should be in scope when baz is called, so that it can do the recursive call.

Output:
IE: baztruebazfalsebaztruebazfalse
FF: baztruebazfalse *(followed by an error – baz not defined)*
Opera: same as FF

Safari: same as FF

2.8 The Abstract Relational Comparison Algorithm: §11.8.5

Step2 of this algorithm in ES3 states that *"If Type(Result(1)) is String and Type(Result(2)) is String, go to step 16."*

JScript uses "or" instead of "and" to decide to do string comparison.

Example:

```
<script>
document.write('1 < 10 == ' + (1 < 10) + '<br>');
document.write('NaN < 1 == ' + (NaN < 1) + '<br>');
document.write('1 < Infinity == ' + (1 < Infinity) + '<br>');
document.write('"10" < 1 == ' + ("10" < 1) + '<br>');
document.write('1 < "a" == ' + (1 < "a") + '<br>');
document.write('"a" < "b" == ' + ("a" < "b") + '<br>');
</script>
```

Output:
IE:
```
1 < 10 == true
NaN < 1 == false
1 < Infinity == true
"10" < 1 == false
1 < "a" == false
"a" < "b" == true
```

FF: same as IE
Opera: same as IE
Safari: same as IE

2.9 Function Declarations used as Statements within function bodies: §12

When a function is defined using a function declaration, JScript binds the function to the global scope regardless of any scope changes introduced by `with` clauses.

Example:

```
<script>
var v = 'value 1';
var o = {v: 'value 2'};

function f1() { alert('v == ' + v); };

with (o) {
    function f2() { alert('v == ' + v); };
}

// call with the initial values
f1();
f2();

// now modify the values
v = 'modified value 1';
o.v = 'modified value 2';
f1();
f2();
</script>
```

Output:
IE:
v == value 1
v == value 1
v == modified value 1
v == modified value 1

FF:
v == value 1
v == value 2
v == modified value 1
v == modified value 2

Opera: same as IE
Safari: same as FF

Contrast the above example with the case where the function is defined as a function expression:

Example:

```
<script>
var v = 'value 1';
var o = {v: 'value 2'};

var f1 = function () { alert('v == ' + v); };

with (o) {
    var f2 = function () { alert('v == ' + v); };
}

// call with the initial values
f1();
f2();

// now modify the values
```

```
v = 'modified value 1';
o.v = 'modified value 2';
f1();
f2();
</script>
```

In this case, there is conformance amongst the 4 implementations.

Output:
IE, FF, Opera, Safari:
v == value 1
v == value 2
v == modified value 1
v == modified value 2

Furthermore, function declarations are not legal in statements (although one might consider a function declaration as syntactically identical to a function expression that includes the optional function name). However, all browser hosted implementations allow function declarations in statements, but do it differently. For e.g. JScript compile time initialises the function in surrounding scope (likewise compile time initialises function expressions binding the identifier in the surrounding scope, instead of the contained scope). FF runtime initialises when control flow reaches the function declaration.

Example:

```
<script>
// case 1
var o = false;
with ({o : true}) {
    function f() { return o; }
}
document.write("FunDecl in with-smt is run time instantiated? "
+f() +" (should be: true)");
document.write("<br>");


// case 2
var usebeforedeclare = (typeof fn === 'function');
if (true) {
    function fn() { return true; }
}
else {
    function fn() { return false; }
}
document.write("FunDecl in if-smt is parse time instantiated? "
+usebeforedeclare +" (should be: false)\nThe correct path is
taken: " +fn() +" (should be: true)");
```

```
document.write("<br>");


// case 3:
BRANCH : {
    break BRANCH;
    function fnc(){}
}
document.write("FunDecl in labelled statement is parse time
instantiated? " +(typeof fnc === 'function') +" (should be:
false)");
document.write("<br>");


// case 4
while (false)
    function func(){}
document.write("FunDecl in while-smt is parse time instantiated?
" +(typeof func === 'function')+ " (should be: false)");
document.write("<br>");


// case 5
for ( ; false; )
    function funct(){}
document.write("FunDecl in for-smt is parse time instantiated? "
+(typeof funct === 'function') +" (should be: false)");
document.write("<br>");


// case 6
for(var t in {})
    function functi(){}
document.write("FunDecl in for..in-smt is parse time
instantiated? " +(typeof functi === 'function') +" (should be:
false)");
document.write("<br>");


// case 7
try {
}
catch(e) {
    function functio(){}
}
document.write("FunDecl in try..catch-smt is parse time
instantiated? " +(typeof functio === 'function') +" (should be:
false)");
document.write("<br>");
</script>
```

Output:
IE:
FunDecl in with-smt is run time instantiated? false (should be: true)
FunDecl in if-smt is parse time instantiated? true (should be: false) The correct path is taken: false (should be: true)
FunDecl in labelled statement is parse time instantiated? true (should be: false)
FunDecl in while-smt is parse time instantiated? true (should be: false)
FunDecl in for-smt is parse time instantiated? true (should be: false)
FunDecl in for..in-smt is parse time instantiated? true (should be: false)
FunDecl in try..catch-smt is parse time instantiated? true (should be: false)

FF:
FunDecl in with-smt is run time instantiated? true (should be: true)
FunDecl in if-smt is parse time instantiated? false (should be: false) The correct path is taken: true (should be: true)
FunDecl in labelled statement is parse time instantiated? false (should be: false)
FunDecl in while-smt is parse time instantiated? false (should be: false)
FunDecl in for-smt is parse time instantiated? false (should be: false)
FunDecl in for..in-smt is parse time instantiated? false (should be: false)
FunDecl in try..catch-smt is parse time instantiated? false (should be: false)

Opera: Same as IE
Safari: No output generated!

Note: Refer §2.7 in this document for a discussion on function expressions.

JScript uses the definition of the last occurrence of the function.

```
<script>
if (true) {
    function bar() { document.write("bar1"); }
}
else {
    function bar() { document.write("bar2"); }
}

bar();

function foo() {
    if (true) {
        function baz() { document.write("baz1"); }
    }
    else {
        function baz() { document.write("baz2"); }
    }

    baz();
}
```

```
foo();
</script>
```

Output:
IE: bar2baz2
FF: bar1baz1
Opera: same as IE
Safari: same as FF

2.10 Enumerating shadowed [[DontEnum]] properties: §15.2.4

Custom properties that shadow [[DontEnum]] properties on **`Object.prototype`** are not enumerated using `for-in`. In the following example `toString` is a property available on `Object.prototype` and is shadowed on `cowboy`. Since such properties are not enumerated through `for-in`, it is not possible to transfer them from a one object to another using `for-in`.

Example:

```
<script>
function cowboy() {
    this.toString = function () { return "cowboy"; }
    this.shoot = function () { return "bang!"; }
}

var p = new cowboy();

document.write("Enumerable properties:");
for (var i in p) {
    document.write(" ", i);
}

document.write("<br/>cowboy propertyIsEnumerable(\"toString\"):
", p.propertyIsEnumerable("toString"));
document.write("<br/>cowboy hasOwnProperty(\"toString\"): ",
p.hasOwnProperty("toString"));
</script>
```

Output:
IE:

```
Enumerable properties: shoot
cowboy propertyIsEnumerable("toString"): false
cowboy hasOwnProperty("toString"): true
```

FF:

```
Enumerable properties: toString shoot
cowboy propertyIsEnumerable("toString"): true
cowboy hasOwnProperty("toString"): true
```

Opera: same as FF
Safari: same as FF

Note:
Refer §15.2.4.7 and §11.13.1, and §8.6.2.2 which say that creating a property on an object gives it an empty set of attributes.

2.11 The try statement: §12.14

In JScript, the variable used to hold a caught exception is visible in the enclosing scope. The variable exists after the `catch` clause has finished executing but ceases to exist after the function where the `catch` clause was located exits.

Example:

```
<script>
function foo() {
    try {
        throw "hello";
    }
    catch(x) {
        document.write(x);
    }

    document.write(x); // x should not be visible here
}
foo();
</script>
```

Output:
IE: hellohello
FF: hello (followed by error ‘x is not defined’)
Opera: same as FF
Safari: same as FF

Catch is defined as creating a new object with the caught object as a property and putting the new object at the head of the scope chain.

If the caught object is a function, calling it within the catch supplies the head of the scope chain as the `this` value. The called function can add properties to this object. This implies that for code of this shape:

```
    var x;
    try {
        // ...
    }
    catch (E) {
        E();
        return x;
    }
```

The reference to 'x' within the catch is not necessarily to the local declaration of 'x'; this gives Catch the same performance problems as with. If the call to E above were specified to supply the global object (rather than the head of the scope chain) as the `this` value, the performance issue evaporates and the catch variable can be treated as a local scoped to the catch clause.

Example:

```
<script>

function foo() {
  this.x = 11;
}

x = "global.x";

try {
  throw foo;
} catch(e) {
  document.write(x) // Should print "global.x"
  e();
  document.write(x) // Should add x to e
                    // (Both IE and Firefox modify the global x)
}

document.write(x);  // Should print "global.x".
</script>
```

Output:
IE: prints "global.x1111
FF: same as IE
Opera: prints "global.x11global.x"
Safari: same as Opera

2.12 Array.prototype.join: §15.4.4.5

Step 3 in §15.4.4.5 says "*If separator is* `undefined`*, let separator be the single-character string ",".*"

JScript does not default the separator to "`,`" if the separator value is `undefined`.

Example:

```
<script>
var array = [1, 2];

alert(array.join());
alert(array.join(undefined));
alert(array.join('-'));
</script>
```

Output:
IE:
1,2
1undefined2
1-2

FF:
1,2
1,2
1-2

Opera: same as FF
Safari: same as FF

2.13 Array.prototype.unshift: §15.4.4.13

`Array.unshift` prepends its arguments to the start of the array and is supposed to return the length of the resultant array.

The returned value of this function call in JScript is “undefined”.

Example:

```
<script>
var a = new Array(1, 2, 3);
var l = a.unshift();
document.write(l, " ");
document.write(a.length, " ");
document.write(a, " ");

l = a.unshift(2);
document.write(l, " ");
document.write(a.length, " ");
document.write(a, " ");
</script>
```

Output:
IE: undefined 3 1,2,3 undefined 4 2,1,2,3
FF: 3 3 1,2,3 4 4 2,1,2,3
Opera: same as FF
Safari: same as FF

2.14 Function length properties: §15.2.3, §15.5.3.2, §15.5.4.7, §15.5.4.8, §15.5.4.13

In JScript the properties `Object.length`, `String.fromCharCode.length`, `String.indexOf.length`, `String.lastIndexOf.length` do not evaluate to the values as called out in ES3.

Refer the table below for the values returned by IE as well as FF, Opera and Safari. Note that the other implementations too deviate from ES3 in certain cases.

Property	**ES3**	**IE**	**FF**	**Opera**	**Safari**
Object.length	1	0	1	1	1
Function.length	1	1	1	1	1
Function.prototype.apply.length	2	2	2	2	2
Function.prototype.call.length	1	1	1	1	1
Array.length	1	1	1	1	1
Array.prototype.length	0	0	0	0	0
Array.prototype.concat.length	1	1	1	1	1
Array.prototype.join.length	1	1	1	1	1
Array.prototype.push.length	1	1	1	1	1
Array.prototype.slice.length	2	2	2	1	2
Array.prototype.splice.length	2	2	2	2	2
Array.prototype.unshift.length	1	1	1	1	1
String.length	1	1	1	1	1
String.fromCharCode.length	1	0	1	1	1
String.prototype.concat.length	1	1	0	1	1
String.prototype.indexOf.length	1	2	1	1	1
String.prototype.lastIndexOf.length	1	2	1	1	1
String.prototype.slice.length	2	0	0	2	2
String.prototype.split.length	2	2	2	2	2
String.prototype.substring.length	2	2	2	2	2
String.prototype.substr.length	2	2	2	2	2
Boolean.length	1	1	1	1	1
Number.length	1	1	1	1	1
Number.prototype.toFixed.length	1	1	1	1	1
Number.prototype.toExponential.length	1	1	1	1	1
Number.prototype.toPrecision.length	1	1	1	1	1
Math.max.length	2	2	2	2	2
Math.min.length	2	2	2	2	2
Date.length	7	7	7	7	7
Date.UTC.length	7	7	7	7	7
Date.prototype.setSeconds.length	2	2	2	2	2
Date.prototype.setUTCSeconds.length	2	2	2	2	2
Date.prototype.setMinutes.length	3	3	3	3	3
Date.prototype.setUTCMinutes.length	3	3	3	3	3

Date.prototype.setHours.length	4	4	4	4	4
Date.prototype.setUTCHours.length	4	4	4	4	4
Date.prototype.setMonth.length	2	2	2	2	2
Date.prototype.setUTCMonth.length	2	2	2	2	2
Date.prototype.setFullYear.length	3	3	3	3	3
Date.prototype.setUTCFullYear.length	3	3	3	3	3
RegExp.length	2	2	1	2	2
Error.length	1	1	3	1	1
EvalError.length	1	1	3	1	1
RangeError.length	1	1	3	1	1
ReferenceError.length	1	1	3	1	1
SyntaxError.length	1	1	3	1	1
TypeError.length	1	1	3	1	1

Here is an example that shows a list of 'length' properties that ES3 explicitly calls out.

```
<script>
function test(p, v) {
    // p: property to test
    // v: expected value as per ES3;
    var r = (eval(p) == v);
    if (r == false) {
        document.write(p + ' == ' + v + ' : ' + r + '. Actual
value: ' + eval(p) + '<br>');
    }
}

test('Object.length', 1);
test('Function.length', 1);
test('Function.prototype.apply.length', 2);
test('Function.prototype.call.length', 1);

test('Array.length', 1);
test('Array.prototype.length', 0);
test('Array.prototype.concat.length', 1);
test('Array.prototype.join.length', 1);
test('Array.prototype.push.length', 1);
test('Array.prototype.slice.length', 2);
test('Array.prototype.splice.length', 2);
test('Array.prototype.unshift.length', 1);

test('String.length', 1);
test('String.fromCharCode.length', 1);
test('String.prototype.concat.length', 1);
test('String.prototype.indexOf.length', 1);
test('String.prototype.lastIndexOf.length', 1);
test('String.prototype.slice.length', 2);
test('String.prototype.split.length', 2);
test('String.prototype.substring.length', 2);
test('String.prototype.substr.length', 2);

test('Boolean.length', 1);

test('Number.length', 1);
test('Number.prototype.toFixed.length', 1);
test('Number.prototype.toExponential.length', 1);
test('Number.prototype.toPrecision.length', 1);

test('Math.max.length', 2);
test('Math.min.length', 2);

test('Date.length', 7);
test('Date.UTC.length', 7);
test('Date.prototype.setSeconds.length', 2);
test('Date.prototype.setUTCSeconds.length', 2);
test('Date.prototype.setMinutes.length', 3);
test('Date.prototype.setUTCMinutes.length', 3);
test('Date.prototype.setHours.length', 4);
test('Date.prototype.setUTCHours.length', 4);
test('Date.prototype.setMonth.length', 2);
test('Date.prototype.setUTCMonth.length', 2);
test('Date.prototype.setFullYear.length', 3);
test('Date.prototype.setUTCFullYear.length', 3);

test('RegExp.length', 2);

test('Error.length', 1);
test('EvalError.length', 1);
test('RangeError.length', 1);
test('ReferenceError.length', 1);
test('SyntaxError.length', 1);
test('TypeError.length', 1);
</script>
```

2.15 String.prototype.split: §15.4.4.14

ES3 states that *"If separator is a regular expression that contains capturing parentheses, then each time separator is matched the results (including any undefined results) of the capturing parentheses are spliced into the output array."*

JScript ignores the capturing parentheses. FF outputs empty strings instead of `undefined`.

Example:

```
<script>
alert("A<B>bold</B>and<CODE>coded</CODE>".split(/<(\/)?([^<>]+)>/
));
</script>
```

Output:
IE: A,bold,and,coded
FF: A,,B,bold,/,B,and,,CODE,coded,/,CODE,
Opera: same as FF
Safari: same as IE

2.16 Number.prototype.toPrecision: §15.7.4.7

ES3 states that *"If precision is undefined, call ToString (9.8.1) instead."*

JScript throws a RangeError if precision is `undefined`.

Example:

```
<script>
var number = 123.456;
document.write('number.toString() == ' + number.toString() +
'<br>');
document.write('number == ' + number + '<br>');

try {
    document.write('number.toPrecision(undefined) == ' +
number.toPrecision(undefined) + '<br>');
}
catch (e) {
    document.write('Exception thrown. ' + e.name + ': ' +
e.message + '<br>');
}
</script>
```

Output:
IE:
number.toString() == 123.456
number == 123.456
Exception thrown. RangeError: The precision is out of range

FF:
number.toString() == 123.456
number == 123.456
number.toPrecision(undefined) == 123.456

Opera: same as FF
Safari: same as FF

2.17 Date.prototype.valueOf: §15.9.5.8

ES3 states that *"The valueOf function returns a number, which is this time value."*

JScript returns 0 when the `valueOf` method is invoked directly.

Example:

```
<script>
document.write('Date.prototype.valueOf() == ' +
Date.prototype.valueOf());
</script>
```

Output:
IE: 0
FF: NaN
Opera: same as FF
Safari: same as FF

Note:
ES3 §15.9.5 says that `Date.prototype` is itself a `Date` object whose value is NaN

2.18 Disjunction: §15.10.2.3

ES3 states that *"Any capturing parentheses inside a portion of the pattern skipped by | produce undefined values instead of strings"*.

JScript produces empty strings instead of `undefined` values.

Example:

```
<script>
Array.prototype.toString = function() {
    var s = '';

    for (var i = 0; i < this.length; i++) {
        if (s) {
            s += ',';
        }

        switch (typeof this[i]) {
            case 'string':
                s += "'" + this[i] + "'";
                break;
            case 'undefined':
                s += 'undefined';
                break;
            default:
                s += this[i];
                break;
        }
    }

    return '[' + s + ']';
}
```

```
var a = /((a)|(ab))((c)|(bc))/.exec('abc');
document.write(a);
</script>
```

Output:
IE: ['abc','a','a','','bc','','bc']
FF: ['abc','a','a',undefined,'bc',undefined,'bc']
Opera: same as FF
Safari: same as FF

2.19 Term: §15.10.2.5

ES3 states that *"Step 4 of the RepeatMatcher clears Atom's captures each time Atom is repeated. We can see its behaviour in the regular expression"*.

JScript does not clear the Atom's matches each time the Atom is repeated.

Example:

```
<script>
// override Array.prototype.toString to wrap strings in quotes
// and to show undefined values
Array.prototype.toString = function() {
    var s = '';

    for (var i = 0; i < this.length; i++) {
        if (s) {
            s += ',';
        }

        switch (typeof this[i]) {
            case 'string':
                s += "'" + this[i] + "'";
                break;
            case 'undefined':
                s += 'undefined';
                break;
            default:
                s += this[i];
                break;
        }
    }

    return '[' + s + ']';
}

var a1 = /(z)((a+)?(b+)?(c))*/.exec('zaacbbbcac');
var a2 = /(a*)*/.exec('b');
document.write(a1);
document.write('<br>');
document.write(a2);
</script>
```

Output:
IE:
['zaacbbbcac','z','ac','a','bbb','c']
['','']

FF:
['zaacbbbcac','z','ac','a',undefined,'c']
['',undefined]

Opera: same as FF
Safari: same as IE

2.20 Atom: §15.10.2.8

ES3 states that *"Backreferences to these capturing parentheses from elsewhere in the pattern always return `undefined` because the negative lookahead must fail for the pattern to succeed."*

JScript produces the empty string instead of `undefined`.

Example:

```
<script>
Array.prototype.toString = function() {
    var s = '';

    for (var i = 0; i < this.length; i++) {
        if (s) {
            s += ',';
        }

        switch (typeof this[i]) {
            case 'string':
                s += "'" + this[i] + "'";
                break;
            case 'undefined':
                s += 'undefined';
                break;
            default:
                s += this[i];
                break;
        }
    }

    return '[' + s + ']';
}
```

```
var a = /(.*?)a(?!(a+)b\2c)\2(.*)/.exec('baaabaac');
document.write(a);
<script>
```

Output:
IE: ['baaabaac','ba','','abaac']
FF: ['baaabaac','ba',undefined,'abaac']
Opera: same as FF
Safari: null

2.21 new RegExp(pattern, flags): §15.10.4.1

ES3 states that "*If pattern is an object R whose [[Class]] property is "RegExp" and flags is undefined, then let P be the pattern used to construct R and let F be the flags used to construct R. If pattern is an object R whose [[Class]] property is "RegExp" and flags is not undefined, then throw a TypeError exception. Otherwise, let P be the empty string if pattern is undefined and ToString(pattern) otherwise, and let F be the empty string if flags is undefined and ToString(flags) otherwise.*
...
If F contains any character other than "g", "i", or "m", or if it contains the same one more than once, then throw a SyntaxError exception."

JScript does not throw either the SyntaxError or the TypeError exceptions; it throws a RegExpError instead (this is a new NativeError). JScript is also lenient in the case where the flags are repeated and does not throw any exception.

Example:

```
<script>
function test(p, f) {
    try {
        var r = new RegExp(p, f);
        document.write(r.toString() + '<br>');
    }
    catch (e) {
        document.write(e.name + ': ' + e.message + '<br>');
    }
}

test(new RegExp('foo'));               // ok
test(new RegExp('foo'), undefined);    // ok
test(new RegExp('foo'), 'gim');        // TypeError
test('foo');                           // ok
test('foo', undefined);                // ok
test(undefined, 'gim');                // ok
test('foo', 'gimgim');                 // SyntaxError
test('foo', 'pvl');                    // SyntaxError
</script>
```

Output:

```
IE:
/foo/
/foo/
RegExpError: Syntax error in regular expression
/foo/
/foo/
//igm
/foo/igm
RegExpError: Syntax error in regular expression

FF:
/foo/
/foo/
TypeError: can't supply flags when constructing one RegExp from another
/foo/
/foo/
/undefined/gim
/foo/gim
SyntaxError: invalid regular expression flag p

Opera:
/foo/
/foo/
TypeError: Statement on line 4: new RegExp cannot be called on both a RegExp and a
defined set of flags Backtrace: Line 4 of inline#1 script in
file://localhost/D:/jsJavaScript/test.html var r = new RegExp(p, f); Line 14 of inline#1
script in file://localhost/D:/jsJavaScript/test.html test(new RegExp("foo"), "gim");
/foo/
/foo/
//gim
SyntaxError: Statement on line 4: RegExp.prototype.compile: syntax error in pattern or
string Backtrace: Line 4 of inline#1 script in file://localhost/D:/jsJavaScript/test.html var
r = new RegExp(p, f); Line 18 of inline#1 script in
file://localhost/D:/jsJavaScript/test.html test("foo", "gimgim");
SyntaxError: Statement on line 4: RegExp.prototype.compile: syntax error in pattern or
string Backtrace: Line 4 of inline#1 script in file://localhost/D:/jsJavaScript/test.html var
r = new RegExp(p, f); Line 19 of inline#1 script in
file://localhost/D:/jsJavaScript/test.html test("foo", "pvl");

Safari:
/foo/
/foo/
TypeError: Type error
/foo/
/foo/
//gim
```

/foo/gim
/foo/

2.22 Date.prototype.getYear: §B.2.4

`Date.prototype.getYear` is treated similar to `Date.prototype.getFullYear` by JScript.

Example:

```
<script>
var d = new Date("10 July 2001");
var y = d.getYear();
document.write(y + '<br>');
</script>
```

Output:
IE: 2001
FF: 101
Opera: same as FF
Safari: same as FF

2.23 Date.prototype.setYear: §B.2.5

`Date.prototype.setYear` is treated similar to `Date.prototype.setFullYear` by JScript.

Example:

```
<script>
var d = new Date(+0);

d.setYear(95);
y = d.getYear();
document.write("setYear: " + y + "  ");

d.setFullYear(95);
y = d.getYear();
document.write("setFullYear: " + y);
</script>
```

Output:
IE: setYear: 95 setFullYear: 95
FF: setYear: 95 setFullYear: -1805
Opera: same as FF
Safari: same as FF

3 Implementation dependent, or underspecified, in ES3

The following are the implement are JScript's (COM Classic) deviations from ECMAScript Edition 3. Each deviation is introduced with a title, a brief note, an example, and the output from running the example on IE7, FF2.0.0.5, Opera 9.02, and Safari 3.0, running on WinXP SP2 32 bit edition. The relevant section within the ES3 specification is quoted as part of the title.

3.1 Identifiers: §7.6

Identifier characters from later Unicode versions

3.2 The Number Type: §8.5

ES3 allows for implementations to be able to detect various NaN values. JScript does not provide any way to detect various NaN values.

3.3 ToString Applied to the Number Type: §9.8.1

The ES3 spec leaves the precision of converting floating point numbers to their string representation as implementation dependent. JScript supports a precision of 16 digits when representing floating point numbers as a string.

The relevant section from the spec is as follows:
The operator ToString converts a number m to string format as follows:
…
5. Otherwise, let n, k, and s be integers such that k ³ 1, 10k-1 £ s < 10k, the number value for s ´ 10n-k is m, and k is as small as possible. Note that k is the number of digits in the decimal representation of s, that s is not divisible by 10, and that the least significant digit of s is not necessarily uniquely determined by these criteria.

NOTE The following observations may be useful as guidelines for implementations, but are not part of the normative requirements of this standard.

- *…*
- *The least significant digit of s is not always uniquely determined by the requirements listed in step 5.*

For implementations that provide more accurate conversions than required by the rules above, it is recommended that the following alternative version of step 5 be used as a guideline:
Otherwise, let n, k, and s be integers such that $k \geq 1$, $10^{k-1} \leq s < 10^{k}$, the number value for $s \times 10^{n-k}$ is m, and k is as small as possible. If there are multiple possibilities for s, choose the value of s for which $s \times 10^{n-k}$ is closest in value to m. If there are two such possible values of s, choose the one that is even. Note that k is the number of digits in the decimal representation of s and that s is not divisible by 10.

Example:

```
<script>
```

```
var x = 0x8000000000000800;
document.write(x, " ");

var y = 9223372036854777856;    // same as above
document.write(y);
</script>
```

Output:
IE: `9223372036854777000 9223372036854777000`
FF: `9223372036854778000 9223372036854778000`
Opera: same as FF
Safari: same as FF

3.4 Function Calls: §11.2.3

Host object returning a value of type Reference

3.5 The typeof operator: §11.4.3

The result of the `typeof` operator when applied to host objects is implementation dependent.

Example:

```
<script>
alert(typeof(window.alert));
</script>
```

Output:
IE: object
FF: function
Opera: same as FF
Safari: same as FF

The above example as extended below does not work in IE.

Example:

```
<script>
alert(typeof(window.alert));

var a = window.alert;
a.apply(null, [45]);
</script>
```

Output:
IE:
Object
Line: 4 Error: object doesn’t support this property or method

FF:
function
45

Opera: same as FF
Safari: same as FF

3.6 The for-in statement: §12.6.4

The ES3 spec leaves the mechanics of enumerating the properties as implementation dependent. In IE, we enumerate the properties as follows:

- Enumerate properties on immediate prototype in reverse order (w.r.t the order in which they were added)
- Enumerate properties in-order on rest of the objects on the prototype chain
- Enumerate properties in-order on present object.

Example:

```
<script>
function dough() {
    this.da = "da";
    this.db = "db";
    this.dc = "dc";
}

bread.prototype = new dough();

function bread() {
    this.ba = "ba";
    this.bb = "bb";
    this.bc = "bc";
}

pizza.prototype = new bread();

function pizza() {
    this.pa = "pa";
    this.pb = "pb";
    this.pc = "pc";
}

function getProperties(obj, objName) {
    var res = "";

    for (var i in obj) {
        res += objName + "." + i + " = " + obj[i] + "<br>";
    }

    return res;
}

var p = new pizza();
document.write(getProperties(p, "pizza"));
</script>
```

Output:
IE:

```
pizza.bc = bc
pizza.bb = bb
pizza.ba = ba
pizza.da = da
pizza.db = db
pizza.dc = dc
pizza.pa = pa
pizza.pb = pb
pizza.pc = pc
```

FF:

```
pizza.pa = pa
pizza.pb = pb
pizza.pc = pc
pizza.ba = ba
pizza.bb = bb
pizza.bc = bc
pizza.da = da
pizza.db = db
pizza.dc = dc
```

Opera: Same as FF
Safari: Same as FF

3.7 Joined Objects: §13.1.2

Consider the following example:

```
<script>
function foo() {
    function bar() {}
    return bar;
}

var x = foo();
var y = foo();

x.blah = 1;
y.blah = 2;

document.write(x.blah + y.blah);
```

```
</script>
```

As per §13.1.1, the two calls to `foo()` produce a function from the same FunctionBody, so they are equated and the resulting function objects can be joined. By §13.2 items 1, 13, and 14, the two uses of `bar` create two objects that are joined: they have different internal properties, but act as "one object"; side effects on one are visible on the other.

IE, FF, Opera and Safari currently show "3", but an implementation could show "4" and still be compliant.

3.8 Creating Function Objects: §13.2

JScript uses the definition of the last occurrence of the function.

```
<script>
if (true) {
    function bar() { document.write("bar1"); }
}
else {
    function bar() { document.write("bar2"); }
}

bar();

function foo() {
    if (true) {
        function baz() { document.write("baz1"); }
    }
    else {
        function baz() { document.write("baz2"); }
    }

    baz();
}

foo();
</script>
```

Output:
IE: bar2baz2
FF: bar1baz1
Opera: same as IE
Safari: same as FF

3.9 eval(x): §15.1.2.1

ES3 states that an instance of the `EvalError` class may be thrown when the global function `eval()` is invoked under any other name.

However, JScript (and FF and Safari) supports indirect `eval`.

Example:

```
<script>
var sum = eval("1 + 2");    // Returns 3
alert(sum);

var myeval = eval;
try {
    sum = myeval("1 + 2");
    alert(sum);
} catch (e) {
    alert("indirect eval not supported!");
}
</script>
```

Ouput:
IE: alerts 3 twice
FF: same as IE
Opera: alerts 3 first, and then alerts “indirect eval not supported”
Safari: same as IE

3.10 parseInt(string, radix): §15.1.2.2

When radix is 0 or undefined and the string's number begins with a 0 digit not followed by an x or X, then the implementation may, at its discretion, interpret the number either as being octal or as being decimal. Implementations are encouraged to interpret numbers in this case as being decimal.

JScript interprets it as octal.

Example;

```
<script>
alert(parseInt("08", undefined));
alert(parseInt("08", 0));

alert(parseInt("011", undefined));
alert(parseInt("011", 0));
</script>
```

Output:
IE: alerts 0, 0, 9, 9
FF: same as IE
Opera: alerts 8, 8, 11, 11
Safari: same as FF

3.11 Function.prototype.toString: §15.3.4.2

ES3 states that the `Function.prototype.toString` returns an implementation-dependent representation of the function, and that in particular the use and placement of white space, line terminators, and semicolons within the representation string is implementation-dependent. This implies that it may not be possible to `eval` a function's `toString` to get a suitably working function.

JScript, ignores newlines and leading whitespace. FF, Opera and Safari too ignores newlines, but there is a subtle difference when it comes to handling other forms of whitespace, as well as nested functions.

Example:

```
<script>
function foo() {
    var e = { name: "value", id : 11};
    return e;
}

alert(foo.toString().indexOf(','));
</script>
```

Output:
IE: 45
FF: 42
Opera: 43
Safari: 45

The following example of a deliberately poorly formatted script further illustrates the differences.

Example:

```
<script>
function foo() {
document.write(    ( // this is a comment
      function (/* this is another comment */
n) { return n;
 })(4)
);
}

document.write(foo);
</script>
```

Output:
IE:

function foo() { document.write((// this is a comment function (/* this is another comment */ n) { return n; })(4)); }

FF:
function foo() { document.write((function (n) {return n;})(4)); }

Opera:
function foo() { document.write((function (n) { return n; })(4)); }

Safari:
function foo() { document.write((function (n) { return n; })(4)); }

Further, consider the following example with nested functions:

Example:

```
<script>
function foo() {
    var i = 0;

    for (var n = 0; n < 10; n++) {
        i++;
    }

    function bar() {
        var b = 0;
    }
}

document.write(foo.toString());
</script>
```

Output:
IE:
function foo() { var i = 0; for (var n = 0; n < 10; n++) { i++; } function bar() { var b = 0; } }

FF: same as IE

Opera:
function foo() { function bar() { var b = 0; } var i = 0; for (var n = 0;n < 10;n++) { i++; } }

Safari:
function foo() { var i = 0; for (n = 0; n < 10; n++) { i++; } function bar() { var b = 0; } }

Finally, note that a function defined by the Function constructor does not have a function name. However, for example below, both IE and FF make it appear that the function is named anonymous.

Example:

```
<script>
document.write(new Function());
</script>
```

Output:
IE: function anonymous() { }
FF: same as IE
Opera: function () { }
Safari: function anonymous() { ; }

3.12 Object constructor and host objects: §15.2.2.1

The ES3 spec leaves the result of invoking the Object constructor with a host object as an argument is implementation dependent.

The relevant section from the spec is as follows:
4. If the value is a host object, then actions are taken and a result is returned in an implementation-dependent manner that may depend on the host object.

Example:

```
<script>
alert(new Object(window.document));
</script>
```

Output:
IE: [object]
FF: [object HTMLDocument]
Opera: same as FF
Safari: same as FF

3.13 valueOf and host objects: §15.2.4.4

The `valueOf` method returns its `this` value. If the object is the result of calling the Object constructor with a host object (§15.2.2.1), it is implementation-defined whether `valueOf` returns its `this` value or another value such as the host object originally passed to the constructor

Example:

```
<script>
var x = new Object(window.document);
alert(x.valueOf());
```

```
</script>
```

Output:
IE: runtime error (object does'nt support this property or method)
FF: [object HTMLDocument]
Opera: same as FF
Safari: same as FF

3.14 Array.prototype.toLocaleString: §15.4.4.3

ES3 states that the elements of the array are converted to strings using their **`toLocaleString`** methods, and these strings are then concatenated, separated by occurrences of a separator string that has been derived in an implementation-defined locale-specific way. The result of calling this function is intended to be analogous to the result of **`toString`**, except that the result of this function is intended to be locale-specific.

The following example using the English (United States) and the IST time zone illustrates this issue:

Example:

```
<script>

var n = Number(123456789.00);
var d = new Date();
var t = new Date().getTime();
var s = "hello world";

var a = new Array();
a.push(n);
a.push(d);
a.push(d);
a.push(s);

document.write(a.toString() + "<br>");
document.write(a.toLocaleString() + "<br>");

</script>
```

Output:
IE:
```
123456789,Fri Jul 13 01:13:45 UTC+0530 2007,Fri Jul 13 01:13:45
UTC+0530 2007,hello world
123,456,789.00, Friday, July 13, 2007 1:13:45 AM, Friday, July
13, 2007 1:13:45 AM, hello world
```

FF:
```
123456789,Fri Jul 13 2007 01:14:50 GMT+0530 (India Standard
Time),Fri Jul 13 2007 01:14:50 GMT+0530 (India Standard
Time),hello world
123,456,789,Friday, July 13, 2007 1:14:50 AM,Friday, July 13,
2007 1:14:50 AM,hello world
```

Opera

```
123456789,Fri, 13 Jul 2007 01:16:07 GMT+0530,Fri, 13 Jul 2007
01:16:07 GMT+0530,hello world
123456789,7/13/2007 1:16:07 AM,7/13/2007 1:16:07 AM,hello world
```

Safari

```
123456789,Fri Jul 13 2007 01:16:24 GMT+0530 (India Standard
Time),Fri Jul 13 2007 01:16:24 GMT+0530 (India Standard
Time),hello world
123456789,Friday, July 13, 2007 01:16:24,Friday, July 13, 2007
01:16:24,hello world
```

3.15 Array.prototype functions: §15.4.4.4 to §15.4.4.13

ES3 states that *"Whether the specific function can be applied successfully to a host object is implementation-dependent".*

In JScript, `Array.prototype` functions cannot be applied on host objects.

3.16 Array.prototype.sort and compare functions: §15.4.4.11

If `comparefn` is not undefined and is not a consistent comparison function for the elements of this array, the behaviour of sort is implementation-defined. If there exist integers i and j and an object P such that all of the conditions below are satisfied then the behaviour of sort is implementation-defined.

3.17 String.prototype.localeCompare: §15.5.4.9

The two strings are compared in an implementation-defined fashion. The actual return values are left implementation-defined to permit implementers to encode additional information in the result value.

3.18 String.prototype.replace: §15.5.4.11

$n: "If n>m, the result is implementation-defined." ; $nn: "If nn>m, the result is implementation-defined"

3.19 Number([value]): §15.7.1.1

JScript converts all strings representing negative hex numbers to NaN.

Example:

```
<script>
alert('Number("0x10") == ' + Number("0x10"));
alert('Number("-0x10") == ' + Number("-0x10"));
</script>
```

Output:
IE: 16, NaN
FF, 16, -16
Opera: same as FF
Safari: same as IE

Note:
This is actually according to spec for IE. FF deviates from the spec: §15.7.1.1 says that the conversion is defined by To Number (§9.3 and §9.3.1). §9.3.1 says that strings that can not be recognized by the grammar given in that section produce NaN as the ToValue result. The grammar does not recognize negative hex strings, so NaN is the correct result. This is an example, of variation among implementations where IE follows the ES3 spec. and some other implementations aren't.

3.20 Number.prototype.toString: §15.7.4.2

ES3 states that *"If radix is the number 10 or undefined, then this number value is given as an argument to the ToString operator; the resulting string value is returned. If radix is an integer from 2 to 36, but not 10, the result is a string, the choice of which is implementation-dependent."*

When an invalid radix is used, JScript will throw a `TypeError` object with message `Invalid procedure call or argument.`

Example:

```
<script>
var val = 42;

document.write('val.toString() == ' + val.toString() + '<br>');
document.write('val.toString(2) == ' + val.toString(2) + '<br>');
document.write('val.toString(8) == ' + val.toString(8) + '<br>');
document.write('val.toString(16) == ' + val.toString(16) +
'<br>');
document.write('val.toString(36) == ' + val.toString(36) +
'<br>');

// try an invalid radix
try {
    document.write('val.toString(100) == ' + val.toString(100) +
'<br>');
}
catch(e) {
    document.write('Invalid Radix Error: ' + e.name + ': ' +
e.message + '<br>');
}

document.write('val.toString() == ' + val.toString() + '<br>');
```

```
// try an invalid radix
try {
    document.write('val.toString(undefined) == ' +
val.toString(undefined) + '<br>');
}
catch(e) {
    document.write('Invalid Radix Error: ' + e.name + ': ' +
e.message + '<br>');
}
</script>
```

Output:
IE:
```
val.toString() == 42
val.toString(2) == 101010
val.toString(8) == 52
val.toString(16) == 2a
val.toString(36) == 16
Invalid Radix Error: TypeError: Invalid procedure call or argument
val.toString() == 42
Invalid Radix Error: TypeError: Invalid procedure call or argument
```

FF:
```
val.toString() == 42
val.toString(2) == 101010
val.toString(8) == 52
val.toString(16) == 2a
val.toString(36) == 16
Invalid Radix Error: Error: illegal radix 100
val.toString() == 42
Invalid Radix Error: Error: illegal radix 0
```

Opera:
```
val.toString() == 42
val.toString(2) == 101010
val.toString(8) == 52
val.toString(16) == 2a
val.toString(36) == 16
val.toString(100) == 42
val.toString() == 42
val.toString(undefined) == 42
```

Safari: same as Opera.

3.21 Number.prototype.toLocaleString: §15.7.4.3

ES3 states that *"This function is implementation-dependent, and it is permissible, but not encouraged, for it to return the same thing as toString."*

The JScript implementation uses the current locale settings.

Example:

```
<script>

var val1 = 12;
var val2 = 123456789;

document.write('val1.toString() == ' + val1.toString() + '<br>');
document.write('val1.toLocaleString() == ' +
val1.toLocaleString() + '<br>');

document.write('val2.toString() == ' + val2.toString() + '<br>');
document.write('val2.toLocaleString() == ' +
val2.toLocaleString() + '<br>');

</script>
```

Output:
IE:

```
val1.toString() == 12
val1.toLocaleString() == 12.00
val2.toString() == 123456789
val2.toLocaleString() == 123,456,789.00
```

FF:

```
val1.toString() == 12
val1.toLocaleString() == 12
val2.toString() == 123456789
val2.toLocaleString() == 123,456,789
```

Opera:

```
val1.toString() == 12
val1.toLocaleString() == 12
val2.toString() == 123456789
val2.toLocaleString() == 123456789
```

Safari: same as Opera

3.22 Number.prototype.toFixed: §15.7.4.5

ES3 states that *"An implementation is permitted to extend the behaviour of toFixed for values of fractionDigits less than 0 or greater than 20. In this case toFixed would not necessarily throw RangeError for such values."*

JScript throws a `RangeError` in both cases.

Example:

```
<script>
var number = 123.456;
document.write('number.toFixed(2) == ' + number.toFixed(2) +
'<br>');

// check for RangeError for fractionDigits > 20
try {
    document.write('number.toFixed(21) == ' + number.toFixed(21)
+ '<br>');
    document.write('Test failed. Should have thrown a
RangeError.' + '<br>');
}
catch(e) {
    document.write('Expected Exception thrown. ' + e.name + ': '
+ e.message + '<br>');
    try {
        document.write('Exception is a RangeError: ' + (e
instanceof RangeError) + '<br>');
    }
    catch (e2) {
        document.write('Error using instanceof ' + e2.name + ': '
+ e2.message + '<br>');
    }
}

// check for RangeError for fractionDigits < 0
try {
    document.write('number.toFixed(-1) == ' + number.toFixed(-1)
+ '<br>');
    document.write('Test failed. Should have thrown a
RangeError.' + '<br>');
}
catch (e) {
    document.write('Expected Exception thrown. ' + e.name + ': '
+ e.message + '<br>');
    try {
        document.write('Exception is a RangeError: ' + (e
instanceof RangeError) + '<br>');
    }
    catch (e2) {
        document.write('Error using instanceof ' + e2.name + ': '
+ e2.message + '<br>');
    }
}
</script>
```

Output:
IE:
number.toFixed(2) == 123.46

```
Expected Exception thrown. RangeError: The number of fractional digits is out of range
Exception is a RangeError: true
Expected Exception thrown. RangeError: The number of fractional digits is out of range
Exception is a RangeError: true
```

FF:

```
number.toFixed(2) == 123.46
number.toFixed(21) == 123.456000000000003069545
Test failed. Should have thrown a RangeError.
number.toFixed(-1) == 120
Test failed. Should have thrown a RangeError.
```

Opera:

```
number.toFixed(2) == 123.46
Expected Exception thrown. RangeError: Statement on line 7:
Number.prototype.toFixed: too many decimal places requested Backtrace: Line 7 of
inline#1 script in file://localhost/d:/jsjavaScript/test.html
document.write("number.toFixed(21) == " + number.toFixed(21) + "
");
Exception is a RangeError: true
Expected Exception thrown. RangeError: Statement on line 22:
Number.prototype.toFixed: too many decimal places requested Backtrace: Line 22 of
inline#1 script in file://localhost/d:/jsjavaScript/test.html
document.write("number.toFixed(-1) == " + number.toFixed(- 1) + "
");
Exception is a RangeError: true
```

Safari:

```
number.toFixed(2) == 123.46
Expected Exception thrown. RangeError: toFixed() digits argument must be between 0
and 20
Exception is a RangeError: true
Expected Exception thrown. RangeError: toFixed() digits argument must be between 0
and 20
Exception is a RangeError: true
```

3.23 Number.prototype.toExponential: §15.7.4.6

ES3 states that *"An implementation is permitted to extend the behaviour of `toExponential` for values of `fractionDigits` less than 0 or greater than 20. In this case `toExponential` would not necessarily throw `RangeError` for such values."*

Example:

```
<script>
var number = 123.456;
```

```
document.write('number.toExponential(2) == ' +
number.toExponential(2) + '<br>');

// check for RangeError for fractionDigits > 20
try {
    document.write('number.toExponential(21) == ' +
number.toExponential(21) + '<br>');
    document.write('Test failed. Should have thrown a
RangeError.' + '<br>');
}
catch (e) {
    document.write('Expected Exception thrown. ' + e.name + ': '
+ e.message + '<br>');
    try {
        document.write('Exception is a RangeError: ' + (e
instanceof RangeError) + '<br>');
    }
    catch (e2) {
        document.write('Error using instanceof ' + e2.name + ': '
+ e2.message + '<br>');
    }
}

// check for RangeError for fractionDigits < 0
try {
    document.write('number.toExponential(-1) == ' +
number.toExponential(-1) + '<br>');
    document.write('Test failed. Should have thrown a
RangeError.' + '<br>');
}
catch (e) {
    document.write('Expected Exception thrown. ' + e.name + ': '
+ e.message + '<br>');
    try {
        document.write('Exception is a RangeError: ' + (e
instanceof RangeError) + '<br>');
    }
    catch (e2) {
        document.write('Error using instanceof ' + e2.name + ': '
+ e2.message + '<br>');
    }
}

document.write('number.toExponential(undefined) == ' +
number.toExponential(undefined) + '<br>');
document.write('number.toExponential() == ' +
number.toExponential() + '<br>');

</script>
```

Output:

```
IE:
number.toExponential(2) == 1.23e+2
Expected Exception thrown. RangeError: The number of fractional digits is out of range
Exception is a RangeError: true
Expected Exception thrown. RangeError: The number of fractional digits is out of range
Exception is a RangeError: true
number.toExponential(undefined) == 1e+2
number.toExponential() == 1.23456e+2

FF:
number.toExponential(2) == 1.23e+2
number.toExponential(21) == 1.234560000000000030695e+2
Test failed. Should have thrown a RangeError.
Expected Exception thrown. RangeError: precision -1 out of range
Exception is a RangeError: true
number.toExponential(undefined) == 1.23456e+2
number.toExponential() == 1.23456e+2

Opera:
number.toExponential(2) == 1.23e+2
Expected Exception thrown. RangeError: Statement on line 7:
Number.prototype.toExponential: too many decimal places requested Backtrace: Line 7
of inline#1 script in file://localhost/D:/jsJavaScript/test.html
document.write("number.toExponential(21) == " + number.toExponential(21) + "
");
Exception is a RangeError: true
Expected Exception thrown. RangeError: Statement on line 22:
Number.prototype.toExponential: too many decimal places requested Backtrace: Line 22
of inline#1 script in file://localhost/D:/jsJavaScript/test.html
document.write("number.toExponential(-1) == " + number.toExponential(- 1) + "
");
Exception is a RangeError: true
number.toExponential(undefined) == 1.23456e+2
number.toExponential() == 1.23456e+2

Safari:
number.toExponential(2) == 1.23e+2
Expected Exception thrown. RangeError: toExponential() argument must between 0 and
20
Exception is a RangeError: true
Expected Exception thrown. RangeError: toExponential() argument must between 0 and
20
Exception is a RangeError: true
number.toExponential(undefined) == 1.23456e+2
number.toExponential() == 1.23456e+2
```

Note the difference in the error message too.

3.24 Number.prototype.toPrecision: §15.7.4.7

ES3 states that *"An implementation is permitted to extend the behaviour of toPrecision for values of precision less than 1 or greater than 21."*

JScript throws a RangeError in both cases.

Example:

```
<script>
var number = 123.456;
alert('number.toPrecision(2) == ' + number.toPrecision(2));

// check for RangeError for fractionDigits > 21
try {
    alert('number.toPrecision(22) == ' + number.toPrecision(22));
    alert('Test failed. Should have thrown a RangeError.');
}
catch(e) {
    alert('Expected Exception thrown. ' +  e.name + ': ' +
e.message);
    try {
        alert('Exception is a RangeError: ' + (e instanceof
RangeError));
    }
    catch(e2) {
        alert('Error using instanceof ' +  e2.name + ': ' +
e2.message);
    }
}

// check for RangeError for fractionDigits < 1
try {
    alert('number.toPrecision(0) == ' + number.toPrecision(0));
    alert('Test failed. Should have thrown a RangeError.');
}
catch(e) {
    alert('Expected Exception thrown. ' +  e.name + ': ' +
e.message);
    try {
        alert('Exception is a RangeError: ' + (e instanceof
RangeError));
    }
    catch(e2) {
        alert('Error using instanceof ' +  e2.name + ': ' +
e2.message);
    }
}
</script>
```

Output:
IE: Throws a RangeError if fractionDigits is greater than 21, as well as if it less than 1.
FF: Does not throw a RangeError if fractionDigits is greater than 21, but does if it is less than 1.
Opera: same as IE
Safari: same as IE

3.25 Date.parse (string): §15.9.4.2

ES3 does not define syntax for date literals. Consequently there is no specification for the syntax accepted by `Date.parse()`.

The following rules govern what strings the `parse` method can successfully parse:

1. Short dates can use either a "/" or "-" date separator, but must follow the month/day/year format, for example "7/10/95".
2. Long dates of the form "July 10 1995" can be given with the year, month, and day in any order, and the year in 2-digit or 4-digit form. In the 2-digit form, the year must be greater than or equal to 70.
3. Both commas and spaces are treated as delimiters. Multiple delimiters are permitted.
4. Any text inside parentheses is treated as a comment. These parentheses may be nested.
5. Month and day names must have two or more characters. Two character names that are not unique are resolved as the last match. For example, "Ju" is resolved as July, not June.
6. The stated day of the week is ignored if it is incorrect given the remainder of the supplied date. For example, "Tuesday July 10 1995" is accepted and parsed even though that date actually falls on a Monday. The resulting Date object represents "Monday July 10 1995".
7. JScript handles all standard time zones, as well as UTC and GMT.
8. Hours, minutes, and seconds are separated by colons, although all need not be specified. "10:", "10:11", and "10:11:12" are all valid.
9. If the 24-hour clock is used, it is an error to specify "PM" for times later than 12 noon. For e.g., "23:15 PM" is an error.
10. A string containing an invalid date is an error. For e.g., a string containing two years or two months is an error.

JScript will report NaN for all erroneous dates.

Example:

```
<script>

// create Date using date string
function test(datestring) {
    document.write('new Date("' + datestring + '") == ' + new
Date(datestring) + '<br>');
```

```
    document.write('Date.parse("' + datestring + '") == ' +
Date.parse(datestring) + '<br>');
    document.write('<br>');
}

// Short dates can use either a "/" or "-" date separator, but
// must follow the month/day/year format, for example "7/10/95".
test("7/10/95");
test("7-10-95");

// Long dates of the form "July 10 1995" can be given with the
// year, month, and day in any order, and the year in 2-digit or
// 4-digit form. In the 2-digit form, the year must be greater
// than or equal to 70.
test("July 10 1995");
test("10 July 1995");
test("10 1995 July");
test("10 95 July");

// Both commas and spaces are treated as delimiters. Multiple
// delimiters are permitted.
test("July,10 1995");
test("July,,10  ,,1995");

// Any text inside parentheses is treated as a comment. These
// parentheses may be nested.
test("July (monthOfYear) 10 (dayOfMonth) 1995 (year)");

// Month and day names must have two or more characters. Two
// character names that are not unique are resolved as the
// last match. For example, "Ju" is resolved as July, not June.
test("Ju (monthOfYear) 10 (dayOfMonth) 1995 (year)");

// The stated day of the week is ignored if it is incorrect
// given the remainder of the supplied date. For example,
// "Tuesday July 10 1995" is accepted and parsed even though
// that date actually falls on a Monday. The resulting Date
// object represents "Monday July 10 1995".
test("Tuesday July 10 1995");

// JScript handles all standard time zones, as well as UTC
// and GMT.
test("10, July, 1995, 11:22, PM, PST");
test("10, July, 1995, 11:22, PM, UTC");
test("10, July, 1995, 11:22, PM, GMT");

// Hours, minutes, and seconds are separated by colons,
// although all need not be specified. "10:", "10:11",
// and "10:11:12" are all valid.
test("10, July, 1995, 11:, PM, PST");
test("10, July, 1995, 11:22, PM, UTC");
test("10, July, 1995, 11:22:00, PM, GMT");
```

```
// If the 24-hour clock is used, it is an error to specify
// "PM" for times later than 12 noon. For e.g., "23:15 PM"
// is an error.
test("10, July, 1995, 23:15, PM, PST");

// A string containing an invalid date is an error.
// For e.g., a string containing two years or two months is
// an error.
test("10, July, July 1995");
test("10, July, 1995 1995");
test("10, July, 1995, 11:22:00, PM, GMT GMT");

</script>
```

Output:
IE:

```
new Date("7/10/95") == Mon Jul 10 00:00:00 UTC+0530 1995
Date.parse("7/10/95") == 805314600000

new Date("7-10-95") == Mon Jul 10 00:00:00 UTC+0530 1995
Date.parse("7-10-95") == 805314600000

new Date("July 10 1995") == Mon Jul 10 00:00:00 UTC+0530 1995
Date.parse("July 10 1995") == 805314600000

new Date("10 July 1995") == Mon Jul 10 00:00:00 UTC+0530 1995
Date.parse("10 July 1995") == 805314600000

new Date("10 1995 July") == Mon Jul 10 00:00:00 UTC+0530 1995
Date.parse("10 1995 July") == 805314600000

new Date("10 95 July") == Mon Jul 10 00:00:00 UTC+0530 1995
Date.parse("10 95 July") == 805314600000

new Date("July,10 1995") == Mon Jul 10 00:00:00 UTC+0530 1995
Date.parse("July,10 1995") == 805314600000

new Date("July,,10 ,,1995") == Mon Jul 10 00:00:00 UTC+0530 1995
Date.parse("July,,10 ,,1995") == 805314600000

new Date("July (monthOfYear) 10 (dayOfMonth) 1995 (year)") == Mon Jul 10 00:00:00
UTC+0530 1995
Date.parse("July (monthOfYear) 10 (dayOfMonth) 1995 (year)") == 805314600000

new Date("Ju (monthOfYear) 10 (dayOfMonth) 1995 (year)") == Mon Jul 10 00:00:00
UTC+0530 1995
Date.parse("Ju (monthOfYear) 10 (dayOfMonth) 1995 (year)") == 805314600000
```

new Date("Tuesday July 10 1995") == Mon Jul 10 00:00:00 UTC+0530 1995
Date.parse("Tuesday July 10 1995") == 805314600000

new Date("10, July, 1995, 11:22, PM, PST") == Tue Jul 11 12:52:00 UTC+0530 1995
Date.parse("10, July, 1995, 11:22, PM, PST") == 805447320000

new Date("10, July, 1995, 11:22, PM, UTC") == Tue Jul 11 04:52:00 UTC+0530 1995
Date.parse("10, July, 1995, 11:22, PM, UTC") == 805418520000

new Date("10, July, 1995, 11:22, PM, GMT") == Tue Jul 11 04:52:00 UTC+0530 1995
Date.parse("10, July, 1995, 11:22, PM, GMT") == 805418520000

new Date("10, July, 1995, 11:, PM, PST") == Tue Jul 11 12:30:00 UTC+0530 1995
Date.parse("10, July, 1995, 11:, PM, PST") == 805446000000

new Date("10, July, 1995, 11:22, PM, UTC") == Tue Jul 11 04:52:00 UTC+0530 1995
Date.parse("10, July, 1995, 11:22, PM, UTC") == 805418520000

new Date("10, July, 1995, 11:22:00, PM, GMT") == Tue Jul 11 04:52:00 UTC+0530 1995
Date.parse("10, July, 1995, 11:22:00, PM, GMT") == 805418520000

new Date("10, July, 1995, 23:15, PM, PST") == NaN
Date.parse("10, July, 1995, 23:15, PM, PST") == NaN

new Date("10, July, July 1995") == NaN
Date.parse("10, July, July 1995") == NaN

new Date("10, July, 1995 1995") == NaN
Date.parse("10, July, 1995 1995") == NaN

new Date("10, July, 1995, 11:22:00, PM, GMT GMT") == NaN
Date.parse("10, July, 1995, 11:22:00, PM, GMT GMT") == NaN

FF:
new Date("7/10/95") == Mon Jul 10 1995 00:00:00 GMT+0530 (India Standard Time)
Date.parse("7/10/95") == 805314600000

new Date("7-10-95") == Invalid Date
Date.parse("7-10-95") == NaN

new Date("July 10 1995") == Mon Jul 10 1995 00:00:00 GMT+0530 (India Standard Time)
Date.parse("July 10 1995") == 805314600000

new Date("10 July 1995") == Mon Jul 10 1995 00:00:00 GMT+0530 (India Standard Time)
Date.parse("10 July 1995") == 805314600000

new Date("10 1995 July") == Mon Jul 10 1995 00:00:00 GMT+0530 (India Standard Time)
Date.parse("10 1995 July") == 805314600000

new Date("10 95 July") == Mon Jul 10 1995 00:00:00 GMT+0530 (India Standard Time)
Date.parse("10 95 July") == 805314600000

new Date("July,10 1995") == Mon Jul 10 1995 00:00:00 GMT+0530 (India Standard Time)
Date.parse("July,10 1995") == 805314600000

new Date("July,,10 ,,1995") == Mon Jul 10 1995 00:00:00 GMT+0530 (India Standard Time)
Date.parse("July,,10 ,,1995") == 805314600000

new Date("July (monthOfYear) 10 (dayOfMonth) 1995 (year)") == Mon Jul 10 1995 00:00:00 GMT+0530 (India Standard Time)
Date.parse("July (monthOfYear) 10 (dayOfMonth) 1995 (year)") == 805314600000

new Date("Ju (monthOfYear) 10 (dayOfMonth) 1995 (year)") == Mon Jul 10 1995 00:00:00 GMT+0530 (India Standard Time)
Date.parse("Ju (monthOfYear) 10 (dayOfMonth) 1995 (year)") == 805314600000

new Date("Tuesday July 10 1995") == Mon Jul 10 1995 00:00:00 GMT+0530 (India Standard Time)
Date.parse("Tuesday July 10 1995") == 805314600000

new Date("10, July, 1995, 11:22, PM, PST") == Tue Jul 11 1995 12:52:00 GMT+0530 (India Standard Time)
Date.parse("10, July, 1995, 11:22, PM, PST") == 805447320000

new Date("10, July, 1995, 11:22, PM, UTC") == Tue Jul 11 1995 04:52:00 GMT+0530 (India Standard Time)
Date.parse("10, July, 1995, 11:22, PM, UTC") == 805418520000

new Date("10, July, 1995, 11:22, PM, GMT") == Tue Jul 11 1995 04:52:00 GMT+0530 (India Standard Time)
Date.parse("10, July, 1995, 11:22, PM, GMT") == 805418520000

new Date("10, July, 1995, 11:, PM, PST") == Tue Jul 11 1995 12:30:00 GMT+0530 (India Standard Time)
Date.parse("10, July, 1995, 11:, PM, PST") == 805446000000

new Date("10, July, 1995, 11:22, PM, UTC") == Tue Jul 11 1995 04:52:00 GMT+0530 (India Standard Time)
Date.parse("10, July, 1995, 11:22, PM, UTC") == 805418520000

new Date("10, July, 1995, 11:22:00, PM, GMT") == Tue Jul 11 1995 04:52:00 GMT+0530 (India Standard Time)
Date.parse("10, July, 1995, 11:22:00, PM, GMT") == 805418520000

new Date("10, July, 1995, 23:15, PM, PST") == Invalid Date
Date.parse("10, July, 1995, 23:15, PM, PST") == NaN

new Date("10, July, July 1995") == Invalid Date
Date.parse("10, July, July 1995") == NaN

new Date("10, July, 1995 1995") == Invalid Date
Date.parse("10, July, 1995 1995") == NaN

new Date("10, July, 1995, 11:22:00, PM, GMT GMT") == Tue Jul 11 1995 04:52:00 GMT+0530 (India Standard Time)
Date.parse("10, July, 1995, 11:22:00, PM, GMT GMT") == 805418520000

Opera:
new Date("7/10/95") == Mon, 10 Jul 1995 00:00:00 GMT+0530
Date.parse("7/10/95") == 805314600000

new Date("7-10-95") == Sat, 07 Oct 1995 00:00:00 GMT+0530
Date.parse("7-10-95") == 813004200000

new Date("July 10 1995") == Mon, 10 Jul 1995 00:00:00 GMT+0530
Date.parse("July 10 1995") == 805314600000

new Date("10 July 1995") == Mon, 10 Jul 1995 00:00:00 GMT+0530
Date.parse("10 July 1995") == 805314600000

new Date("10 1995 July") == Mon, 10 Jul 1995 00:00:00 GMT+0530
Date.parse("10 1995 July") == 805314600000

new Date("10 95 July") == Tue, 10 Jul 2007 00:00:00 GMT+0530
Date.parse("10 95 July") == 1184005800000

new Date("July,10 1995") == Mon, 10 Jul 1995 00:00:00 GMT+0530
Date.parse("July,10 1995") == 805314600000

new Date("July,,10 ,,1995") == Mon, 10 Jul 1995 00:00:00 GMT+0530
Date.parse("July,,10 ,,1995") == 805314600000

new Date("July (monthOfYear) 10 (dayOfMonth) 1995 (year)") == Mon, 10 Jul 1995 00:00:00 GMT+0530
Date.parse("July (monthOfYear) 10 (dayOfMonth) 1995 (year)") == 805314600000

new Date("Ju (monthOfYear) 10 (dayOfMonth) 1995 (year)") == Thu, 10 Aug 1995 00:00:00 GMT+0530
Date.parse("Ju (monthOfYear) 10 (dayOfMonth) 1995 (year)") == 807993000000

new Date("Tuesday July 10 1995") == Mon, 10 Jul 1995 00:00:00 GMT+0530
Date.parse("Tuesday July 10 1995") == 805314600000

new Date("10, July, 1995, 11:22, PM, PST") == Mon, 10 Jul 1995 23:22:00 GMT+0530
Date.parse("10, July, 1995, 11:22, PM, PST") == 805398720000

new Date("10, July, 1995, 11:22, PM, UTC") == Tue, 11 Jul 1995 04:52:00 GMT+0530
Date.parse("10, July, 1995, 11:22, PM, UTC") == 805418520000

new Date("10, July, 1995, 11:22, PM, GMT") == Tue, 11 Jul 1995 04:52:00 GMT+0530
Date.parse("10, July, 1995, 11:22, PM, GMT") == 805418520000

new Date("10, July, 1995, 11:, PM, PST") == Mon, 10 Jul 1995 23:00:00 GMT+0530
Date.parse("10, July, 1995, 11:, PM, PST") == 805397400000

new Date("10, July, 1995, 11:22, PM, UTC") == Tue, 11 Jul 1995 04:52:00 GMT+0530
Date.parse("10, July, 1995, 11:22, PM, UTC") == 805418520000

new Date("10, July, 1995, 11:22:00, PM, GMT") == Tue, 11 Jul 1995 04:52:00 GMT+0530
Date.parse("10, July, 1995, 11:22:00, PM, GMT") == 805418520000

new Date("10, July, 1995, 23:15, PM, PST") == Mon, 10 Jul 1995 23:15:00 GMT+0530
Date.parse("10, July, 1995, 23:15, PM, PST") == 805398300000

new Date("10, July, July 1995") == Mon, 10 Jul 1995 00:00:00 GMT+0530
Date.parse("10, July, July 1995") == 805314600000

new Date("10, July, 1995 1995") == Mon, 10 Jul 1995 00:00:00 GMT+0530
Date.parse("10, July, 1995 1995") == 805314600000

new Date("10, July, 1995, 11:22:00, PM, GMT GMT") == Tue, 11 Jul 1995 04:52:00 GMT+0530
Date.parse("10, July, 1995, 11:22:00, PM, GMT GMT") == 805418520000

Safari:
new Date("7/10/95") == Mon Jul 10 1995 00:00:00 GMT+0530 (India Standard Time)

Date.parse("7/10/95") == 805314600000

new Date("7-10-95") == Invalid Date
Date.parse("7-10-95") == NaN

new Date("July 10 1995") == Mon Jul 10 1995 00:00:00 GMT+0530 (India Standard Time)
Date.parse("July 10 1995") == 805314600000

new Date("10 July 1995") == Mon Jul 10 1995 00:00:00 GMT+0530 (India Standard Time)
Date.parse("10 July 1995") == 805314600000

new Date("10 1995 July") == Invalid Date
Date.parse("10 1995 July") == NaN

new Date("10 95 July") == Invalid Date
Date.parse("10 95 July") == NaN

new Date("July,10 1995") == Mon Jul 10 1995 00:00:00 GMT+0530 (India Standard Time)
Date.parse("July,10 1995") == 805314600000

new Date("July,,10 ,,1995") == Invalid Date
Date.parse("July,,10 ,,1995") == NaN

new Date("July (monthOfYear) 10 (dayOfMonth) 1995 (year)") == Mon Jul 10 1995 00:00:00 GMT+0530 (India Standard Time)
Date.parse("July (monthOfYear) 10 (dayOfMonth) 1995 (year)") == 805314600000

new Date("Ju (monthOfYear) 10 (dayOfMonth) 1995 (year)") == Invalid Date
Date.parse("Ju (monthOfYear) 10 (dayOfMonth) 1995 (year)") == NaN

new Date("Tuesday July 10 1995") == Mon Jul 10 1995 00:00:00 GMT+0530 (India Standard Time)
Date.parse("Tuesday July 10 1995") == 805314600000

new Date("10, July, 1995, 11:22, PM, PST") == Invalid Date
Date.parse("10, July, 1995, 11:22, PM, PST") == NaN

new Date("10, July, 1995, 11:22, PM, UTC") == Invalid Date
Date.parse("10, July, 1995, 11:22, PM, UTC") == NaN

new Date("10, July, 1995, 11:22, PM, GMT") == Invalid Date
Date.parse("10, July, 1995, 11:22, PM, GMT") == NaN

new Date("10, July, 1995, 11:, PM, PST") == Invalid Date
Date.parse("10, July, 1995, 11:, PM, PST") == NaN

new Date("10, July, 1995, 11:22, PM, UTC") == Invalid Date
Date.parse("10, July, 1995, 11:22, PM, UTC") == NaN

new Date("10, July, 1995, 11:22:00, PM, GMT") == Invalid Date
Date.parse("10, July, 1995, 11:22:00, PM, GMT") == NaN

new Date("10, July, 1995, 23:15, PM, PST") == Invalid Date
Date.parse("10, July, 1995, 23:15, PM, PST") == NaN

new Date("10, July, July 1995") == Invalid Date
Date.parse("10, July, July 1995") == NaN

new Date("10, July, 1995 1995") == Invalid Date
Date.parse("10, July, 1995 1995") == NaN

new Date("10, July, 1995, 11:22:00, PM, GMT GMT") == Invalid Date
Date.parse("10, July, 1995, 11:22:00, PM, GMT GMT") == NaN

3.26 Date.UTC: §15.9.4.3

ES3 states that *"When the UTC function is called with fewer than two arguments, the behaviour is implementation-dependent."*

When called with just the year as argument, JScript returns a `Number` representing 1 Jan of that year in the current locale.

Example:

```
<script>
document.write(Date.UTC(1995));
</script>
```

Output:
IE: 788918400000
FF: same as IE
Opera: syntax error
Safari: NaN

3.27 Date.prototype.toString: §15.9.5.2

ES3 states that *"The contents of the string are implementation-dependent, but are intended to represent the Date in the current time zone in a convenient, human-readable form"*

Note that JScript uses the UTC.

Example:

```
<script>
var d = new Date("4 July 1776");
document.write(d.toString());
</script>
```

Output:
IE: Thu Jul 4 00:00:00 UTC+0530 1776
FF: Thu Jul 04 1776 00:00:00 GMT+0530 (India Standard Time)
Opera: Thu, 04 Jul 1776 00:00:00 GMT+0530
Safari: same as FF

3.28 Date.prototype.toDateString: §15.9.5.3

ES3 states that *"The contents of the string are implementation-dependent, but are intended to represent the specified portion of the Date"*

Example:

```
<script>
var d = new Date("July 10 1995");
document.write(d.toDateString());
</script>
```

Output:
IE: Mon Jul 10 1995
FF: same as IE
Opera: Mon, 10 Jul 1995
Safari: same as IE

3.29 Date.prototype.toTimeString: §15.9.5.4

ES3 states that *"The contents of the string are implementation-dependent, but are intended to represent the specified portion of the Date"*

Example:

```
<script>
var d = new Date("July 10 1995");
document.write(d.toTimeString());
</script>
```

Output:
IE: 00:00:00 UTC+0530
FF: 00:00:00 GMT+0530 (India Standard Time)
Opera: 00:00:00 GMT+0530
Safari: same as FF

3.30 Date.prototype.toLocaleString: §15.9.5.5

ES3 states that *"The contents of the string are implementation-dependent, but are intended to represent the specified portion of the Date"*

```
<script>
var d = new Date("July 10 1995");
document.write(d.toLocaleString());
</script>
```

Output:
IE: Monday, July 10, 1995 12:00:00 AM
FF: same as IE
Opera: 7/10/1995 12:00:00 AM
Safari: Monday, July 10, 1995 00:00:00

3.31 Date.prototype.toLocaleDateString: §15.9.5.6

ES3 states that *"The contents of the string are implementation-dependent, but are intended to represent the specified portion of the Date"*

Example:

```
<script>
var d = new Date("July 10 1995");
document.write(d.toLocaleDateString());
</script>
```

Output:
IE: Monday, July 10, 1995
FF: same as IE
Opera: 7/10/1995
Safari: same as IE

3.32 Date.prototype.toLocaleTimeString: §15.9.5.7

ES3 states that *"The contents of the string are implementation-dependent, but are intended to represent the specified portion of the Date"*

Example:

```
<script>
var d = new Date("July 10 1995");
document.write(d.toLocaleTimeString());
</script>
```

Output:
IE: 12:00:00 AM
FF: same as IE
Opera: same as IE

Safari: 00:00:00

3.33 Date.prototype.toUTCString: §15.9.5.42

ES3 states that *"The contents of the string are implementation-dependent, but are intended to represent the Date in a convenient, human-readable form in UTC."*

Example:

```
<script>
var d = new Date("July 10 1995");
document.write(d.toUTCString());
</script>
```

Output:
IE: Sun, 9 Jul 1995 18:30:00 UTC
FF: Sun, 09 Jul 1995 18:30:00 GMT
Opera: same as FF
Safari: same as FF

3.34 new RegExp(pattern, flags): §15.10.4.1

ES3 states that *"The source property of the newly constructed object is set to an implementation-defined string value in the form of a Pattern based on P."*

Example:

```
<script>
document.write(new RegExp(/(.*?)a(?!(a+)b\2c)\2(.*)/).source +
'<br>');
</script>
```

Output:
IE: (.*?)a(?!(a+)b\2c)\2(.*)
FF: same as IE
Opera: same as IE
Safari: same as IE

3.35 RegExp.prototype.toString: §15.10.6.4

ES3 states that *"An implementation may choose to take advantage of src being allowed to be different from the source passed to the RegExp constructor to escape special characters in src. For example, in the regular expression obtained from new RegExp("/"), src could be, among other possibilities, "/" or "\/"."*

Example:

```
<script>
document.write(new RegExp().toString() + '<br>');
document.write(new RegExp(undefined).toString() + '<br>');
document.write(new RegExp("/").toString() + '<br>');
</script>
```

Output:
IE:
//
//
///

FF:
/(?:)/
/undefined/
/\//

Opera: same as IE
Safari: same as IE

Furthermore, ES3 does not state the order in which flags must be concatenated into the final string.

Example:

```
<script>
document.write(new RegExp('foo', 'mgi').toString() + '<br>');
</script>
```

Output:
IE: /foo/igm
FF: /foo/gim
Opera: same as FF
Safari: same as FF

3.36 Error.prototype.message: §15.11.4.3

ES3 states that *"The initial value of Error.prototype.message is an implementation-defined string"*. In §15.11.2.1, ES3 states that *"If the argument message is not undefined, the message property of the newly constructed object is set to ToString(message)."*

In JScript this is the empty string. If the argument `message` is explicitly passed in as `undefined`, the initial value is set to `undefined`.

Example:

```
<script>
function test(e) {
    document.write('e.message: ' + e.message + '<br>');
}
```

```
test(new Error());
test(new Error(undefined));
</script>
```

Output:
IE:
e.message:
e.message: undefined

FF: same as IE

Opera:
e.message: Generic error
e.message: undefined

Safari:
e.message: Unknown error
e.message: Unknown error

3.37 Error.prototype.toString: §15.11.4.4

ES3 states that this function *"Returns an implementation defined string."*

Example:

```
<script>
function test(e) {
//    document.write('e.message: ' + e.message + '<br>');
    document.write('e.toString(): ' + e.toString() + '<br>');
}

test(new Error());
test(new Error(undefined));
test(new Error('Complex error'));
</script>
```

Output:
IE:
e.toString(): [object Error]
e.toString(): [object Error]
e.toString(): [object Error]

FF:
e.toString(): Error
e.toString(): Error: undefined
e.toString(): Error: Complex error

Opera:

```
e.toString(): [Error: name: Error message: Generic error ]
e.toString(): [Error: name: Error ]
e.toString(): [Error: name: Error message: Complex error ]
```

Safari:

```
e.toString(): Error: Unknown error
e.toString(): Error: Unknown error
e.toString(): Error: Complex error
```

3.38 NativeError.prototype.message: §15.11.7.10

ES3 states that *"The initial value of the message property of the prototype for a given NativeError constructor is an implementation-defined string."*

In JScript these have the empty string as the initial value.

Example:

```
<script>
document.write(new EvalError().message + '<br>');
document.write(new RangeError().message + '<br>');
document.write(new ReferenceError().message + '<br>');
document.write(new SyntaxError().message + '<br>');
document.write(new TypeError().message + '<br>');
document.write(new URIError().message + '<br>');
</script>
```

Output:
IE: the empty string in each case
FF: same as IE

Opera:

```
Use of eval as a value
Illegal manipulation of array or string length
Undefined variable or property
Mis-constructed program text
Incorrect value to a primitive operation
Generic error in a URI
```

Safari:

```
EvalError
RangeError
ReferenceError
SyntaxError
TypeError
URIError
```

3.39 Errors: §16

ES3 states that "*An implementation may provide additional types, values, objects, properties, and functions beyond those described in this specification. This may cause constructs (such as looking up a variable in the global scope) to have implementation-defined behaviour instead of throwing an error (such as **ReferenceError**).*"

This can hinder interoperability.

Example:

```
<script>
var x = {};
x.__proto__ = 3;
document.write(x.__proto__);
</script>
```

Output:
IE: 3
FF: [object Object]
Opera: same as IE
Safari: same as IE

Note that in this case the user defined **__proto__** property clashes with the property of the same name that FF adds.

4 JScript Extensions

4.1 debugger statement

JScript recognizes `debugger` as a statement, and treats it as a request for a breakpoint.

Example:

```
<script>
debugger;
</script>
```

Output:
IE: invokes and jumps into a script debugger (if more than one such debugger is available on the system, then the user is given the option to choose).
FF: no-op; however, jumps into FireBug iff installed.
Opera: no-op
Safari; same as Opera

4.2 conditional compilation

JScript supports conditional compilation.

Example:

```
<script>
@cc_on   // turn condition compilation ON

@set @foo = 0;
@if (@foo == 0) {
    document.write(@foo);
}
@end

@set @foo = 1;
@if (@foo == 0) {
    document.write("unreachable!");
}
@else {
    document.write(@foo);
}
@end

@set @foo = 2;
@if (@foo == 0) {
    document.write("unreachable!");
}
@elif (@foo == 2) {
    document.write(@foo);
}
@else {
    document.write("still unreachable!");
}
@end
document.write("<br>");

// the following are the pre-defined conditionalcompilation
// variables available to the JScript programmer.
var ccvars = [@_win32,     // true if running on a Win32.
              @_win16,     // true if running on a Win16.
              @_mac,       // true if running on an Mac.
              @_alpha,     // true if running on a DECAlpha proc.
              @_x86,       // true if running on an Intel proc.
              @_mc680x0,   // true if running on a Motorola 680x0
                           // proc.
              @_PowerPC,   // true if running on a Motorola
                           // PowerPC proc.
              @_win64,     // true if running on Win64
              @_ia64,      // true if running on IA64 proc
              @_amd64,     // true if running on AMD64 proc
              @_unix,      // true if running on HP or Sparc
              @_sparc,     // true if running on Sparc
              @_hp,        // true if running on HP
              @_microsoft,         // always true
              @_jscript,           // always true.
              @_jscript_build,     // contains the build number
                                   // of the JScript scripting
                                   // engine.
              @_jscript_version];  // contains the JScript
                                   // version number in
                                   // major.minor format.

for (var i = 0; i < ccvars.length; i++) {
    document.write(ccvars[i] + "<br>");
}

</script>
```

Output:
IE: prints

```
012
true
NaN
NaN
NaN
true
NaN
NaN
NaN
NaN
NaN
NaN
NaN
NaN
true
true
5730
5.7
```

FF: syntax error (after the @cc_on directive).
Opera: syntax error
Safari: conditional compilation is not supported but no error is reported.

It is recommended that conditional compilation code be placed within comments, so that hosts that do not understand conditional compilation will ignore it.

4.3 *dynamic scoping across script engines*

Consider the following example:

Example:

File: foo.html
This is the startup page. Open this in IE.

```
<html>
<head>
<script>
function foo(x) {
    var i = document.getElementById("i");
    alert(i.contentWindow.e1('e param', 'x'));
    alert(i.contentWindow.e2('e param', 'x'));
}
</script>
</head>

<body onload="foo('foo param')">
<iframe id="i" src="bar.html">
</iframe>
</body>

</html>
```

File: bar.html

```
<html>
<script>
function e1(x, s) {
    return eval(s)
}

function e2(x, s) {
    return top.eval(s)
}
</script>
</html>
```

Output:
IE: displays "e param" followed by "foo param"
FF: displays "e param" followed by "e param" (a bug as per Brendan)
Opera: displays "e param" (also a bug).
Safari: same as FF

In IE we maintain 1 script engine per document. So, in this case, there are 2 script engines in existence - one in the iframe's document, and one in the parent's document. When we do an **`eval(s)`** (in bar.html), the eval executes in the iframe's script engine. When we do a **`top.eval(s)`**, the eval executes in the parent document's script engine. Hence, they bind to different scopes.

In bar.html, when we execute **`eval(s)`** the value of '**`s`**' is from the scope of the '**`function e`**'; however, when we execute **`top.eval(s)`** the value of '**`s`**' is from the scope of '**`function foo`**'.

If you view the stack of activations as a single engine's runtime state, instead of split across two engines, it becomes as follows:
for **`o.eval(s)`**, find the nearest activation of a function scoped by **`o`** from the top of (single) stack, and use that activation as dynamic scope.

4.4 Function Declaration

JScript permits function names to be qualified. This allows for function `foo.prototype()` to be syntactic sugar for `foo.prototype = function (){}`
Note that the object on LHS of the '.' must already be defined.

Example:

```
<script>
function foo() {}

function foo.prototype.bar() { return 5;}
// equivalent ES3 form:
// foo.prototype.bar = function () { return 5;}

var f = new foo();
document.write(f.bar());
</script>
```

Output:
IE: compiles fine
FF: syntax error
Opera: syntax error
Safari: syntax error

JScript permits special syntax to declare event handlers (using `::`). There is no syntax to raise an event, however. This can typically be used for VBScript-style event binding on named host objects.

Example:

```
<script>
function window::onfocus() {
    document.write("window has focus");
}

/*
Here is an eventhandler for the event "bar" on a "foo".
function foo(){}
function foo::bar() {}
```

```
*/

</script>
```

Output:
IE: displays “window has focus”
FF: gives a syntax error for this code.
Opera: same as FF
Safari: same as FF

JScript also supports unnamed function expressions.

Example:

```
<script>
function(){}
</script>
```

Output:
IE: no error
FF: same as IE
Opera: Syntax error
Safari: same as IE

4.5 function “caller” identification

`functionName.caller` returns a reference to the function that invoked the current function.

The `caller` property is only defined for a function while that function is executing. If the function is called from the top level of a JScript program, caller contains **`null`**.

If the caller property is used in a string context, the result is the same as functionName`.toString`, that is, the decompiled text of the function is displayed. The functionName object is the name of any *executing* function.

Example:

```
<script>
function foo() {
    document.write(foo.caller);
    document.write('<br>');
    if (foo.caller != null) {
        foo.caller(0);
    }
}

foo();
bar(1);
```

```
function bar(x) {
    if (x == 1) {
        foo();
    }
    else {
        document.write("done");
        document.write("<br>");
    }
}
</script>
```

Output:
IE: prints
null
function bar(x) { if (x == 1) { foo(); } else { document.write("done"); document.write("
"); } }
done

FF: same as JScript
Opera: prints,
undefined
undefined

Safari: Same as JScript

4.6 lastIndex property on the RegExp Captures Array

JScript adds a property `lastIndex` to the captures array that contains the next position in the string following the match.

The property has the attributes { DontEnum, DontDelete, ReadOnly }.

Example:

```
<script>
var src = "The rain in Spain falls mainly in the plain.";
var re = /\w+/g;
var arr;

while ((arr = re.exec(src)) != null) {
   document.write(arr.index + '-' + arr.lastIndex + ':' + arr +
';');
}
</script>
```

Output:
IE: 0-3:The;4-8:rain;9-11:in;12-17:Spain;18-23:falls;24-30:mainly;31-33:in;34-37:the;38-43:plain;

FF: 0-undefined:The;4-undefined:rain;9-undefined:in;12-undefined:Spain;18-undefined:falls;24-undefined:mainly;31-undefined:in;34-undefined:the;38-undefined:plain;
Opera: same as FF
Safari: same as FF

4.7 RegExp.index

JScript adds this property to the `RegExp` constructor even for non global regular expressions, and sets it value to the index in the string of the last match.

The property has the attributes { DontEnum, DontDelete, ReadOnly }.

Example:

```
<script>
var s  = "Able was I ere I saw Elba.";
var re = /((\w+) (\w+) (\w+) (\w+) (\w+) (\w+) (\w+)(.*))/;
re.exec(s);
document.write('RegExp.index == ' + RegExp.index + '<br>');
</script>
```

Output:
IE: RegExp.index == 0
FF: RegExp.index == undefined
Opera: same as FF
Safari: same as FF

4.8 RegExp.input and RegExp.$_

JScript adds these properties to the `RegExp` constructor even for non global regular expressions, and sets it value to the string last searched by the regular expression. `RegExp.$_` is a synonym for `RegExp.input`.

These properties have the attributes { DontDelete, DontEnum }.

Example:

```
<script>
var s  = "Able was I ere I saw Elba.";
var re = /((\w+) (\w+) (\w+) (\w+) (\w+) (\w+) (\w+)(.*))/;
re.exec(s);
document.write('RegExp.input == ' + RegExp.input + '<br>');
document.write('RegExp.$_ == ' + RegExp.$_ + '<br>');
</script>
```

Output:
IE:

RegExp.input == Able was I ere I saw Elba.
RegExp.$_ == Able was I ere I saw Elba.

FF: same as IE

Opera:
RegExp.input == undefined
RegExp.$_ == undefined

Safari: same as IE

4.9 RegExp.lastIndex

JScript adds this property to the `RegExp` constructor even for non global regular expressions, and sets it value to the index in the string following the match, or to -1 if there is no match.

The property has the attributes { DontEnum, DontDelete, ReadOnly }.

Example:

```
<script>
var s  = "Able was I ere I saw Elba.";
var re = /((\w+) (\w+) (\w+) (\w+) (\w+) (\w+) (\w+)(.*))/;
re.exec(s);
document.write('RegExp.lastIndex == ' + RegExp.lastIndex +
'<br>');
</script>
```

Output:
IE: RegExp.lastIndex == 26
FF: RegExp.lastIndex == undefined
Opera: same as FF
Safari: same as FF

Example:

```
<script>
var s  = "Able was I ere I saw Elba.";
var re = /foo/;
re.exec(s);
document.write('RegExp.lastIndex == ' + RegExp.lastIndex +
'<br>');
</script>
```

Output:
IE: RegExp.lastIndex == -1
FF: RegExp.lastIndex == undefined
Opera: same as FF

Safari: same as FF

4.10 RegExp.lastMatch and RegExp.$&

JScript adds these properties to the `RegExp` constructor, and sets it value to the string value of the last match made by a regular expression. `RegExp['$&']` is a synonym for `RegExp.lastMatch`.

These properties have the attributes { DontDelete, DontEnum }.

```
<script>
var s  = "Able was I ere I saw Elba.";
var re = /((\w+) (\w+) (\w+) (\w+) (\w+) (\w+) (\w+)(.*))/;

re.exec(s);

document.write('RegExp.lastMatch == ' + RegExp.lastMatch +
'<br>');
document.write('RegExp[\'$&\'] == ' + RegExp['$&'] + '<br>');
</script>
```

Output:
IE:
RegExp.lastMatch == Able was I ere I saw Elba.
RegExp['$&'] == Able was I ere I saw Elba.

FF: same as IE

Opera:
RegExp.lastMatch == undefined
RegExp['$&'] == undefined

Safari: same as IE

4.11 RegExp.lastParen and RegExp.$+

JScript adds these properties to the `RegExp` constructor, and sets its value to the string value of the last item in the capturing array. `RegExp.$+` is a synonym for `RegExp.lastParen`.

These properties have the attributes { DontDelete, DontEnum }.

Example:

```
<script>
var s  = "Able was I ere I saw Elba.";
var re = /((\w+) (\w+) (\w+) (\w+) (\w+) (\w+) (\w+)(.*))/;

re.exec(s);
```

```
document.write('RegExp.lastParen == ' + RegExp.lastParen +
'<br>');
document.write('RegExp[\'$+\'] == ' + RegExp['$+'] + '<br>');
</script>
```

Output:
IE:
RegExp.lastParen == .
RegExp['$+'] == .

FF: same as IE

Opera:
RegExp.lastParen == undefined
RegExp['$+'] == undefined

Safari: same as IE

4.12 RegExp.leftContext and RegExp.$`

JScript adds these properties to the `RegExp` constructor, and sets its value to the leading portion of the string before the match. `RegExp.$`` ` is a synonym for `RegExp.leftContext`.

These properties have the attributes { DontDelete, DontEnum }.

Example:

```
<script>
var s  = "Able was I ere I saw Elba.";
var re = /((\w+) (\w+) (\w+) (\w+) (\w+) (\w+) (\w+)(.*))/;

re.exec(s);

document.write('RegExp.leftContext == ' + RegExp.leftContext +
'<br>');
document.write('RegExp[\'$`\'] == ' + RegExp['$`'] + '<br>');
</script>
```

Output:
IE:
RegExp.leftContext ==
RegExp['$`'] ==

FF: same as IE

Opera:
RegExp.leftContext ==

RegExp['$`'] == undefined

Safari: same as IE

4.13 RegExp.rightContext and RegExp.$'

JScript adds these properties to the `RegExp` constructor, and sets its value to the trailing portion of the string after the match. `RegExp.$'` is a synonym for `RegExp.rightContext`.

These properties have the attributes { DontDelete, DontEnum }.

Example:

```
<script>
var s  = "Able was I ere I saw Elba.";
var re = /((\w+) (\w+) (\w+) (\w+) (\w+) (\w+) (\w+)(.*))/;

re.exec(s);

document.write('RegExp.rightContext == ' + RegExp.rightContext +
'<br>');
document.write('RegExp[\'$\'\'] == ' + RegExp['$\''] + '<br>');
</script>
```

Output:
IE:
RegExp.rightContext ==
RegExp['$''] ==

FF: same as IE

Opera:
RegExp.rightContext ==
RegExp['$''] == undefined

Safari: same as IE

4.14 RegExp.$1 – RegExp.$n

JScript adds these properties to the `RegExp` constructor, and sets its value to the corresponding item from the captures array.

These properties have the attributes { DontDelete, DontEnum }.

Example:

```
<script>
var s  = "Able was I ere I saw Elba.";
var re = /((\w+) (\w+) (\w+) (\w+) (\w+) (\w+) (\w+)(.*))/;

var a = re.exec(s);

for (var i = 1; i < a.length; i++) {
    document.write('RegExp[\'$' + i + '\'] == ' + RegExp['$' + i]
+
      '; ' +
      'a[' + i + '] == ' + a[i] + '<br>');
}
</script>
```

Output:
IE:
RegExp['$1'] == Able was I ere I saw Elba.; a[1] == Able was I ere I saw Elba.
RegExp['$2'] == Able; a[2] == Able
RegExp['$3'] == was; a[3] == was
RegExp['$4'] == I; a[4] == I
RegExp['$5'] == ere; a[5] == ere
RegExp['$6'] == I; a[6] == I
RegExp['$7'] == saw; a[7] == saw
RegExp['$8'] == Elba; a[8] == Elba
RegExp['$9'] == .; a[9] == .

FF: same as IE
Opera: same as IE
Safari: same as IE

4.15 RegExp.compile(pattern, flags)

The compile method converts pattern into an internal format for faster execution. This allows for more efficient use of regular expressions in loops, for example. A compiled regular expression speeds things up when reusing the same expression repeatedly. No advantage is gained, however, if the regular expression changes.

This is a property on RegExp instances and has the attributes { DontDelete, DontEnum }.

The semantics of the parameters are the same as that for the RegExp constructor. Refer §3.24 within this document for JScript deviations.

Example:

```
<script>
var re = new RegExp();
var s = "AaBbCcDdEeFfGgHhIiJjKkLlMmNnOoPp"
var p = '[A-Z]';
var f = 'g';

// Compile the regular expression for uppercase only.
```

```
if (re.compile) {
    re.compile(p, f);
    var a = s.match(re)              // Find matches.
    document.write(a);
}
else {
    document.write('RegExp.compile undefined');
}
</script>
```

Output:
IE: A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P
FF: same as IE
Opera: same as IE
Safari: RegExp. compile undefined

4.16 Properties on the Error Object

In JScript, instances of the Native Error object have two additional properties: an error description (`description` property) and an error number (`number` property).

In JS 5.5 the `message` property as defined in ES3 used to be called `description`. It is retained in the current version for compatibility.

An error number is a 32-bit value. The upper 16-bit word is the facility code, and the lower word is the actual error code.

Example:

```
<script>
var x;

try {
    x = y;
}
catch (e) {
    document.write(e.number & 0xFFFF);
    document.write(": ");
    document.write(e.description);
}
</script>
```

Output:
IE: 5009: 'y' is undefined
FF: 0: undefined
Opera:0: undefined (followed by reference errors in lines 7, 10
Safari: same as Opera but no error diagnostics seen.

4.17 The Error constructor

JScript supports a two argument error constructor. ES3 (§15.11.3) states that the Error constructor has a single param which is to be used as the 'message' property. However, in JS 5.1 the Error constructor had 2 params which were used as the 'number' and 'message' property. JScript uses ES3 semantics only when the the first parameter cannot be converted to a number and no second param is specified.

```
new Error(number, message)
```

Example:

```
<script>
var err = new Error(101, "describe the error");
for (var i in err) {
    document.write(i + ' : ' + err[i] + '<br>');
}
</script>
```

Output:
IE:
name : Error
number : 101
description : describe the error
message : describe the error

FF:
message : 101
fileName : describe the error
lineNumber : 2
stack : Error("101","describe the error")@:0 @file:///D:/jsJavaScript/test.html:2
name : Error

Opera: message : 101
Safari: same as Opera

4.18 Script engine identification functions

The following methods are made available to identify details related to the script engine in use.

```
ScriptEngine();  // return a string representing the
                 // scripting language in use;
                 // "JScript" or "VBScript" or "VBA"
ScriptEngineMajorVersion(); // Returns the major version number
                            // of the script engine in use.
ScriptEngineMinorVersion(); // Returns the minor version number
                            // of the script engine in use.
ScriptEngineBuildVersion(); // Returns the build version number
                            // of the script engine in use.
```

These functions are documented in MSDN at http://msdn2.microsoft.com/en-us/library/efy5bay1.aspx

4.19 ActiveXObject

The `ActiveXObject` represents a reference to a OLE Automation object and can be used to programmatically access applications that support an OLE Automation model.

For example, Microsoft Excel supports an OLE automation model that can be programmatically accessed through COM. The `ActiveXObject` object provides the same access to JScript programmers. For the purpose of accessing the applications automation model, `ActiveXObect` creates a new instance of the application.

Example:

```
<script>
/*
Invoke Excel and get a reference to its "Application" object
Get a reference to the "Sheet" object (also creates a sheet)
Make the application visible
Quit the application
*/
var xlApp   = new ActiveXObject("Excel.Application");
var xlSheet = new ActiveXObject("Excel.Sheet");

xlSheet.Application.Visible = true;
xlSheet.Application.Quit();
</script>
```

Output:
IE7:
This brings up Excel, creates a sheet, makes the application visible, and then quits the application.
FF: Syntax error (ActiveXObject undefined)
Opera: same as FF
Safari: no output (not supported), but no error is reported

4.20 GetObject

This is similar to `ActiveXObject` except that while `ActiveXObect` creates a new instance of the application `GetObject(...)` grabs an instance of the application that is already running.
This is no longer supported. It was withdrawn from IE5 onwards for security reasons.

4.21 VBArray and its members

JScript provides access to Visual Basic safe arrays. The `VBArray` object is read-only, and cannot be created directly. The `safeArray` argument must have obtained a `VBArray` value before being passed to the `VBArray` constructor. This can only be done by retrieving the value from an existing ActiveX or other object.

VBArrays can have multiple dimensions. The indices of each dimension can be different. The `dimensions` method retrieves the number of dimensions in the array; the `lbound` and `ubound` methods retrieve the range of indices used by each dimension.

VBArray supports the following methods (as shown in the example code below):

```
dimensions()
lbound(i)
ubound(i)
toArray()
getItem(i)
```

Example:

```
<script language="VBScript">
Function CreateVBArray()
    Dim i, j, k
    Dim a(2, 2)

    k = 1
    For i = 0 To 2
        For j = 0 To 2
            a(j, i) = k
            document.writeln(k)
            k = k + 1
        Next
        document.writeln("<BR>")
    Next
    CreateVBArray = a
End Function
</script>

<script language="Javascript">
function VBArrayTest(vbarray) {
    var a = new VBArray(vbarray);
    var d = a.dimensions();
    document.write("vbarray has " +d +" dimensions");
    document.write("<br>");

    var i, j;
    for (i = 1; i <= d; i++) {
        document.write("dimension: " +i);
        document.write(" lower bound: " +a.lbound(i));
        document.write(" upper bound: " +a.ubound(i));
        document.write("<br>");
    }
    var b = a.toArray();
    for (j = 0; j < 9; j++) {
        document.writeln(b[j]);
    }

    document.write("<br>");
    for (i = 0; i <= 2; i++) {
        for (j =0; j <= 2; j++) {
            document.writeln(a.getItem(i, j));
        }
    }
}

VBArrayTest(CreateVBArray());
</script>
```

Output:
IE:
1 2 3
4 5 6
7 8 9
vbarray has 2 dimensions
dimension: 1 lower bound: 0 upper bound: 2
dimension: 2 lower bound: 0 upper bound: 2
1 2 3 4 5 6 7 8 9
1 4 7 2 5 8 3 6 9

FF: syntax error (CreateVBArray() not defined)
Opera: same as FF
Safari: no output (not supported), but no error is reported.

4.22 Enumertor and its methods

JScript provides an **`Enumerator`** object that can enumerate over a collection.

The Enumerator object supports the following methods:

```
moveFirst()
moveNext()
atEnd()
item()
```

Example

```
<script>
var a = new Array();
a.push(1);
a.push(2);

var e = new Enumerator(a);
```

```
for (; !e.atEnd(); e.moveNext()) {
    o = e.item();
    document.write(o + "<br>");
}
</script>
```

Output:
IE:
1
2

FF: Syntax error
Opera: same as FF
Safari: no output (not supported), but no error is reported

4.23 FileSystemObject and its methods and properties

The **`FileSystemObject`**, its methods and properties are documented in MSDN at http://msdn2.microsoft.com/en-us/library/z9ty6h50.aspx

4.24 Dictionary and its methods and properties

The **`Dictionary`** object, its methods and properties are documented in MSDN at http://msdn2.microsoft.com/en-us/library/x4k5wbx4.aspx

5 References

ES3 spec
JScript Conditional Compilation

6 Appendix

ES3 implementation loopholes:

Section name	**Section Text**
Identifiers	"implementations may allow additional legal identifier characters based on the category assignment from later versions of Unicode"
Regular Expression Literals	"An implementation may extend the regular expression constructor's grammar, but it should not extend the RegularExpressionBody and RegularExpressionFlags productions or the productions used by these productions."
Regular Expression Literals / Semantics	"If the call to new RegExp generates an error, an implementation may, at its discretion, either report the error immediately while scanning the program, or it may defer the error until the regular expression literal is evaluated in the course of program execution"

The Number Type	"external code might be able to detect a difference between various Non-a-Number values, but such behaviour is implementation-dependent"
Internal Properties and Methods	"Whether or not a native object can have a host object as its [[Prototype]] depends on the implementation."
Function Objects	"An implementation may also provide implementation-dependent internal functions that are not described in this specification. "
Function Calls	"Whether calling a host object can return a value of type Reference is implementation-dependent."
The typeof Operator	typeof result for "host objects" is implementation dependent
The for-in Statement	"The mechanics of enumerating the properties (step 5 in the first algorithm, step 6 in the second) is implementation dependent. The order of enumeration is defined by the object."
Creating Function Objects	"If there is more than one object E satisfying these criteria, choose one at the implementation's discretion." ; "13. At the implementation's discretion, go to either step 2 or step 14." ; "Step 1 allows an implementation to optimise the common case of a function A that has a nested function B where B is not dependent on A. In this case the implementation is allowed to reuse the same object for B instead of creating a new one every time A is called. Step 13 makes this optimisation optional; an implementation that chooses not to implement it will go to step 2." ; plus additional verbiage about "joined" functions
Native ECMAScript Objects	"Unless otherwise specified in the description of a particular function, if a function or constructor described in this section is given more arguments than the function is specified to allow, the behaviour of the function or constructor is undefined. In particular, an implementation is permitted (but not required) to throw a TypeError exception in this case."
The Global Object	"The values of the [[Prototype]] and [[Class]] properties of the global object are implementation-dependent"
parseInt (string , radix)	"When radix is 0 or undefined and the string's number begins with a 0 digit not followed by an x or X, then the implementation may, at its discretion, interpret the number either as being octal or as being decimal. Implementations are encouraged to interpret numbers in this case as being decimal."

parseInt (string , radix)	". (However, if R is 10 and Z contains more than 20 significant digits, every significant digit after the 20th may be replaced by a 0 digit, at the option of the implementation; and if R is not 2, 4, 8, 10, 16, or 32, then Result(16) may be an implementation-dependent approximation to the mathematical integer value that is represented by Z in radix-R notation.)"
new Object ([value])	"4. If the value is a host object, then actions are taken and a result is returned in an implementation-dependent manner that may depend on the host object."
Object.prototype. valueOf ( )	"If the object is the result of calling the Object constructor with a host object (section 15.2.2.1), it is implementation-defined whether valueOf returns its this value or another value such as the host object originally passed to the constructor."
Function.prototype. toString ( )	"An implementation-dependent representation of the function is returned. This representation has the syntax of a FunctionDeclaration. Note in particular that the use and placement of white space, line terminators, and semicolons within the representation string is implementation-dependent."
Array.prototype. toLocaleString ( )	"a separator string that has been derived in an implementation-defined locale-specific way"
Array.prototype.concat Array.prototype.join Array.prototype.pop Array.prototype.push Array.prototype.reverse Array.prototype.shift Array.prototype.slice Array.prototype.sort Array.prototype.splice Array.prototype.unshift	"Whether the *XXX* function can be applied successfully to a host object is implementation-dependent."
Array.prototype.sort	"If comparefn is not undefined and is not a consistent comparison function for the elements of this array (see below), the behaviour of sort is implementation-defined. " ;"If there exist integers i and j and an object P such that all of the conditions below are satisfied then the behaviour of sort is implementation-defined:"
String.prototype. localeCompare	"The two strings are compared in an implementation-defined fashion. " ; "The actual return values are left implementation-defined to permit implementers to encode additional information in the result value"
String.prototype.replace	$n: "If n>m, the result is implementation-defined." ; $nn: "If nn>m, the result is implementation-defined"

Number.prototype.toString	"If radix is an integer from 2 to 36, but not 10, the result is a string, the choice of which is implementation-dependent."
Number.prototype. toLocaleString	"This function is implementation-dependent, and it is permissible, but not encouraged, for it to return the same thing as toString."
Number.prototype.toFixed Number.prototype. toExponential	"An implementation is permitted to extend the behaviour of XXX for values of fractionDigits less than 0 or greater than 20. In this case XXX would not necessarily throw RangeError for such values."
Number.prototype. toPrecision	"An implementation is permitted to extend the behaviour of toPrecision for values of precision less than 1 or greater than 21."
Function properties of math object	
Daylight Saving Time Adjustment	"An implementation of ECMAScript is expected to determine the daylight saving time algorithm"
TimeClip (time)	"The point of step 3 is that an implementation is permitted a choice of internal representations of time values, for example as a 64-bit signed integer or as a 64-bit floating-point value. Depending on the implementation, this internal representation may or may not distinguish -0 and +0"
Date.parse (string)	"the value produced by Date.parse is implementation-dependent when given any string value that could not be produced in that implementation by the toString or toUTCString method."
Date.UTC	"When the UTC function is called with fewer than two arguments, the behaviour is implementation-dependent."
Date.prototype.toString	"The contents of the string are implementation-dependent, but are intended to represent the Date in the current time zone in a convenient, human-readable form"

Date.prototype.toDateString Date.prototype.toTimeString Date.prototype.toLocaleString Date.prototype.toLocaleDateString Date.prototype.toLocaleTimeString	"The contents of the string are implementation-dependent, but are intended to represent the “XXX” portion of the Date"
Date.prototype.toUTCString	"The contents of the string are implementation-dependent, but are intended to represent the Date in a convenient, human-readable form in UTC."
new RegExp(pattern, flags)	"The source property of the newly constructed object is set to an implementation-defined string value in the form of a Pattern based on P."
Error.prototype.message	"The initial value of Error.prototype.message is an implementation-defined string."
Error.prototype.toString	"Returns an implementation defined string."
NativeError Object Structure	"and in the implementation-defined message property of the prototype object."
NativeError.prototype.message	"The initial value of the message property of the prototype for a given NativeError constructor is an implementation-defined string."
Errors	Various allowances for implementation dependent erorr behavior (or lack there of) related to implementation dependent extensions
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."

7 Revision history

9 Sep 2007	pratapL	Creation