Table of Contents

- JavaScript Language Reference
- What's New in JavaScript
- JavaScript in Visual Studio 2017
- JavaScript Fundamentals
  - Writing JavaScript Code
  - Variables
  - Data Types
  - Operators
  - Operator Precedence
  - Controlling Program Flow
  - Functions
  - Objects and Arrays
  - Intrinsic Objects
  - Creating Objects
  - Calculating Dates and Times (JavaScript)
  - Date and Time Strings
  - Displaying Text in a Webpage
- Advanced JavaScript
  - Advanced JavaScript
    - Using Constructors to Define Types
    - Prototypes and Prototype Inheritance
    - Data Properties and Accessor Properties
    - Recursion
    - Variable Scope
    - Copying, Passing, and Comparing Data
    - Using Arrays
    - Typed Arrays
    - Collections
    - Iterators and Generators

Special Characters
Template Strings
Using the bind method
Managing event listeners
Troubleshooting Your Scripts
Conditional Compilation
Conditional Compilation Variables
Strict Mode
JavaScript Reference
JavaScript Reference
Version Information
JavaScript Objects
JavaScript Constants
JavaScript Properties
JavaScript Functions
JavaScript Methods
JavaScript Operators
JavaScript Statements
JavaScript Directives
JavaScript Errors
JavaScript Reserved Words
JavaScript Future Reserved Words

JavaScript Language Reference

10/18/2017 • 1 min to read • Edit Online

JavaScript is a scripting language that can be embedded in web pages and other applications.

This documentation describes the Microsoft implementation of JavaScript, which is compliant with the ECMAScript Language Specification 5th Edition. It also provides additional features that are not included in the ECMA Standards.

You can use JavaScript code in browser applications together with HTML, CSS and the Document Object Model (DOM), which represents HTML and browser objects.

- For information about HTML, see HTML/XHTML Reference.
- For information about CSS, see Cascading Style Sheets.
- For information about the DOM, see Document Object Model (DOM).

  You can also use JavaScript code in Windows Store apps that run on Windows 8 and Windows 8.1, and in Universal Windows Platform (UWP) apps that run on Windows 10. Windows Store apps can be developed in Visual Studio 2012, Visual Studio 2013, and Visual Studio 2015; UWP apps can be developed in Visual Studio 2015.

- For information about JavaScript in Windows 8.x Store apps, see JavaScript Roadmap.
- For information about HTML and CSS in Windows 8.x Store apps, see HTML/CSS for Windows Store apps.
- For information about the Windows Runtime and Windows Library for JavaScript APIs, see API reference for Windows Runtime and Windows Library for JavaScript.
- For information about using the Windows Runtime API with JavaScript, see Using the Windows Runtime in JavaScript.

  The JavaScript editor in Visual Studio provides IntelliSense support. For more information, see JavaScript IntelliSense.

In This Section

The following sections provide more information about JavaScript.

What's New in JavaScript
Describes new features in JavaScript.

JavaScript Fundamentals
Provides an introduction to the basic structures in JavaScript.

Advanced JavaScript
Explains advanced JavaScript functionality, such as recursion, arrays, troubleshooting, and so on.

JavaScript Reference
Explains the elements of the JavaScript language .

See Also

Document Object Model

What's New in JavaScript

10/18/2017 • 2 min to read • Edit Online

This document lists new features in JavaScript that are supported in both Edge mode, Windows 8.x Store, and Windows Phone Store apps.

To find out which JavaScript elements are supported in Edge mode but deprecated in Windows 8.x Store apps, see Version Information.

> **IMPORTANT**
> For information about how to create Windows 8.x Store and Windows Phone Store apps using JavaScript, including information about the Visual Studio JavaScript editor and other features, see Develop Windows Store apps using Visual Studio 2013.

New features in JavaScript

FEATURE	DESCRIPTION
Classes	New syntax supports declaration of classes.
Promises	Promises allow easier and cleaner asynchronous coding. Promise constructors are supported, along with the `all` and `race` utility methods.
Iterators	Now you can iterate over iterable objects (including arrays, array-like objects, and iterators), invoking a custom iteration hook with statements to be executed for the value of each distinct property. For more information, see Iterators and Generators. **Note:** Generators are not yet supported.
Arrow functions	The arrow function (=>) provides shorthand syntax for the `function` keyword which features a lexical `this` binding.
New methods for built-in objects	The Array Object, Math Object, Number Object, Object Object, and String Object built-in objects include many new utility functions and properties for manipulating and inspecting data.
Object literal enhancements	Objects now support computed properties, concise method definitions, and shorthand syntax for properties whose value is initialized to a same-named variable. For more information, see Creating Objects.
Proxies	Proxies enable custom behavior for objects.
Rest parameters	Rest parameters allow you to turn consecutive arguments in a function call to an array. For more information, see Functions.
Spread operator	The spread operator (`...`) expands iterable expressions into individual arguments. For example, `a.b(...array)` is approximately the same as `a.b.apply(a, array)`.

FEATURE	DESCRIPTION
Symbols	Symbol objects allow properties to be added to existing objects with no possibility of interference with the existing object properties, with no unintended visibility, and with no other uncoordinated additions by other code.
Template strings	Template strings are string literals that allow for expressions to be evaluated and concatenated with the string literal.
Unicode enhancements	Improvements have been made to Unicode support. For example, a new escape sequence format supports astral code points (code points with more than four hexadecimal digits). For more information, see Special Characters.
WeakSet	A WeakSet is a collection of objects that will be garbage collected if they are not referenced anywhere else.

See Also

JavaScript Language Reference

JavaScript in Visual Studio 2017

3/2/2018 • 9 min to read • Edit Online

JavaScript is a first-class language in Visual Studio. You can use most or all of the standard editing aids (code snippets, IntelliSense, and so on) when you write JavaScript code in the Visual Studio IDE. You can write JavaScript code for many application types and services.

Support for ECMAScript 2015 (ES6) and beyond

Visual Studio now supports syntax for ECMAScript language updates such as ECMAScript 2015/2016.

What is ECMAScript 2015?

JavaScript is still evolving as a programming language and TC39 is the committee responsible for making updates. ECMAScript 2015 is an update to the JavaScript language that brings helpful new syntax and functionality. For a deep dive on ES6 features, check out this reference site.

In addition to support for ECMAScript 2015, Visual Studio also supports ECMAScript 2016 and will have support for future versions of ECMAScript as they are released. To keep up with TC39 and the latest changes in ECMAScript, follow their work on github.

Transpile JavaScript

A common problem with JavaScript is that you want to use the latest ES6+ language features because they help you be more productive, but your runtime environments (often browsers) don't yet support these new features. This means that you either have to keep track of which browsers support what features (which can be tedious), or you need a way to convert your ES6+ code into a version that your target runtimes understand (usually ES5). Converting your code to a version that the runtime understands is commonly referred to as "transpiling".

One of the key features of TypeScript is the ability transpile ES6+ code to ES5 or ES3 so that you can write the code that makes you most productive, but still run your code on any platform. Because JavaScript in Visual Studio 2017 uses the same language service as TypeScript, it too can take advantage of ES6+ to ES5 transpilation.

Before transpilation can be set up, some understanding of the configuration options is required. TypeScript is configured via a `tsconfig.json` file. In the absence of such a file, some default values are used. For compatibility reasons, these defaults are different in a context where only JavaScript files (and optionally `.d.ts` files) are present. To compile JavaScript files, a `tsconfig.json` file must be added, and some of these options must be set explicitly.

The required settings for the tsconfig file are as follows:

- `allowJs`: This value must be set to `true` for JavaScript files to be recognized. The default value is `false`, because TypeScript compiles to JavaScript, and the compiler should not include files it just compiled.
- `outDir`: This value should be set to a location not included in the project, in order that the emitted JavaScript files are not detected and then included in the project (see `exclude`).
- `module`: If using modules, this setting tells the compiler which module format the emitted code should use (for example `commonjs` for Node, or bundlers such as Browserify).
- `exclude`: This setting states which folders not to include in the project. The output location, as well as non-project folders such as `node_modules` or `temp`, should be added to this setting.
- `enableAutoDiscovery`: This setting enables the automatic detection and download of definition files as outlined previously.
- `compileOnSave`: This setting tells the compiler if it should recompile any time a source file is saved in Visual

Studio.

- `typeAcquisition`: This set of settings control the behavior of automatic type acquisition (further explain in this section)

In order to convert JavaScript files to CommonJS modules and place them in an `./out` folder, you could use the following `tsconfig.json` file:

```
{
  "compilerOptions": {
    "module": "commonjs",
    "allowJs": true,
    "outDir": "out"
  },
  "exclude": [
    "node_modules",
    "wwwroot",
    "out"
  ],
  "compileOnSave": true,
  "typeAcquisition": {
    "enable": true
  }
}
```

With the settings in place, if a source file (`./app.js`) existed and contained several ECMAScript 2015 language features as follows:

```
import {Subscription} from 'rxjs/Subscription';  // ES6 import

class Foo {  // ES6 Class
    sayHi(name) {
        return `Hi ${name}, welcome to Salsa!`;  // ES6 template string
    }
}

export let sqr = x => x * x;  //ES6 export, let, and arrow function
export default Subscription;  //ES6 default export
```

Then a file would be emitted to `./out/app.js` targeting ECMAScript 5 (the default) that looks something like the following:

```
"use strict";
var Subscription_1 = require('rxjs/Subscription');
var Foo = (function () {
    function Foo() {
    }
    Foo.prototype.sayHi = function (name) {
        return "Hi " + name + ", welcome to Salsa!";
    };
    return Foo;
}());
exports.sqr = function (x) { return x * x; };
Object.defineProperty(exports, "__esModule", { value: true });
exports.default = Subscription_1.Subscription;
```

Better IntelliSense

JavaScript IntelliSense in Visual Studio 2017 will now display a lot more information on parameters and member lists. This new information is provided by the TypeScript language service, which uses static analysis behind the

scenes to better understand your code. You can read more about the new IntelliSense experience and how it works here.

JSX syntax support

JavaScript in Visual Studio 2017 has rich support for the JSX syntax. JSX is a syntax set that allows HTML tags within JavaScript files.

The following illustration shows a React component defined in the `comps.tsx` TypeScript file, and then this component being used from the `app.jsx` file, complete with IntelliSense for completions and documentation within the JSX expressions. You don't need TypeScript here, this specific example just happens to contain some TypeScript code as well.

> **NOTE**
>
> To convert the JSX syntax to React calls, the setting `"jsx": "react"` must be added to the `compilerOptions` in the `tsconfig.json` file.

The JavaScript file created at `./out/app.js' upon build would contain the code:

```
"use strict";
var comps_1 = require('./comps');
var x = React.createElement(comps_1.RepoDisplay, {description: "test"});
```

Configure your JavaScript project

The language service is powered by static analysis, which means it analyzes your source code without actually executing it in order to return IntelliSense results and provide other editing features. Therefore, the larger the quantity and size of files that are included your project context, the more memory and CPU will be used during analysis. Because of this, there are a few default assumptions that are made about your project shape:

- `package.json` and `bower.json` list dependencies used by your project and by default are included in Automatic

Type Acquisition (ATA)

- A top level `node_modules` folder contains library source code and its contents are excluded from the project context by default
- Every other `.js`, `.jsx`, `.ts`, and `.tsx` file is possibly one of *your own* source files and must be included in project context

In most cases, you will be able to just open your project and have great experience using the default project configuration. However, in projects that are large or have different folder structures, it may be desirable to further configure the language service to better focus only on your own source files.

Override defaults

You can override the default configuration by adding a `tsconfig.json` file to your project root. A `tsconfig.json` has several different options that can manipulate your project context. A few of them are listed below, but for a full set of all options available, see the schema store.

Important `tsconfig.json` options

```
{
  "compilerOptions": {
    "allowJs": true,          // include .js and .jsx in project context (defaults to only .ts and .tsx)
    "noEmit": true            // turns off downlevel compiler
  },
  "files": [],                // list of explicit files to include in the project context. Highest priority.
  "include": [],              // list of folders or glob patterns to include in the project context.
  "exclude": [],              // list of folders or glob patterns to exclude. Overridden by files array.
  "typeAcquisition": {
    "enable": true,           // Defaulted to "false" with a tsconfig. Enables better IntelliSense in JS.
    "include": [ "jquery" ],  // Specific libs to fetch .d.ts files that weren't picked up by ATA
    "exclude": [ "node" ]     // Specific libs to not fetch .d.ts files for
  }
}
```

Example project configuration

Given a project with the following setup:

- project's source files are in `wwwroot/js`
- project's lib files are in `wwwrrot/lib`
- `bootstrap`, `jquery`, `jquery-validation`, and `jquery-validation-unobtrusive` are listed in the `bower.json`
- `kendo-ui` has been manually added to the lib folder

You could use the following `tsconfig.json` to make sure the language service only analyzes your source files in the `js` folder, but still fetches and uses `.d.ts` files for the libraries in your `lib` folder.

```
{
  "compilerOptions": {
    "allowJs": true,
    "noEmit": true
  },
  "exclude": ["wwwroot/lib"],  //ignore lib folders, we will get IntelliSense via ATA
  "typeAcquisition": {
    "enable": true,
    "include": [ "kendo-ui" ]  //kendo-ui wasn't added via bower, so we need to list it here
  }
}
```

Troubleshooting The JavaScript language service has been disabled for the following project(s)

When you open a JavaScript project that has a very large amount of content, you might get a message that reads "The JavaScript language service has been disabled for the following project(s)". The most common reason for having a very large amount of JavaScript source is due to including libraries with source code that exceeds a 20MB project limit.

A simple way to optimize your project is to add a `tsconfig.json` file in your project root to let the language service know which files are safe to ignore. Use the sample below to exclude the most common directories where libraries are stored:

```
{
  "compilerOptions": {
    "allowJs": true,
    "allowSyntheticDefaultImports": true,
    "maxNodeModuleJsDepth": 2,
    "noEmit": true,
    "skipLibCheck": true
  },
  "exclude": [
    "**/bin",
    "**/bower_components",
    "**/jspm_packages",
    "**/node_modules",
    "**/obj",
    "**/platforms"
  ]
}
```

Add more directories as you see fit. Some other examples include "vendor" or "wwwroot/lib" directories.

> **NOTE**
> The compiler property `disableSizeLimit` can be used as well to disable the 20MB check limit. Take special precautions when using this property because disabling the limit might crash the language service.

Notable Changes from Visual Studio 2015

As Visual Studio 2017 features a completely new language service, there are a few behaviors that will be different or absent from the previous experience. The most notable changes are the replacement of VSDoc with JSDoc, the removal of custom `.intellisense.js` extensions, and limited IntelliSense for specific code patterns.

No more `///<references/>` or `_references.js`

Previously it was fairly complicated to understand at any given moment which files were in your IntelliSense scope. Sometimes it was desirable to have all your files in scope and other times it wasn't, and this led to complex configurations involving manual reference management. Going forward you no longer need to think about reference management and so you don't need triple slash references comments or `_references.js` files.

See the JavaScript IntelliSense page for more info on how IntelliSense works.

VSDoc

XML documentation comments, sometimes referred to as VSDocs, could previously be used to decorate your source code with additional data that would be used to buff up IntelliSense results. VSDoc is no longer supported in favor of JSDoc which is easier to write and the accepted standard for JavaScript.

`.intellisense.js` extensions

Previously, you could author IntelliSense extensions which would allow you to add custom completion results for third-party libraries. These extensions were fairly difficult to write and installing and referencing them was cumbersome, so going forward the new language service won't support these files. As an easier alternative, you can write a TypeScript definition file to provide the same IntelliSense benefits as the old `.intellisense.js` extensions. You can learn more about declaration (`.d.ts`) file authoring here.

Unsupported patterns

Because the new language service is powered by static analysis rather than an execution engine (read this issue for information of the differences), there are a few JavaScript patterns that no longer can be detected. The most common pattern is the "expando" pattern. Currently the language service cannot provide IntelliSense on objects that have properties tacked on after declaration. For example:

```
var obj = {};
obj.a = 10;
obj.b = "hello world";
obj. // IntelliSense won't show properties a or b
```

You can get around this by declaring the properties during object creation:

```
var obj = {
    "a": 10,
    "b": "hello world"
}
obj. // IntelliSense shows properties a and b
```

You can also add a JSDoc comment as follows:

```
/**
 * @type {{a: number, b: string}}
 */
var obj = {};
obj.a = 10;
obj.b = "hello world";
obj. // IntelliSense shows properties a and b
```

JavaScript Fundamentals

10/18/2017 • 1 min to read • Edit Online

JavaScript is an interpreted, object-based scripting language.

The JavaScript language uses a syntax similar to that of C, and supports structured constructs, such as `if...else`, `for`, and `do...while`. Braces ({}) are used to delimit statement blocks. The language supports various data types, including `String`, `Number`, `Boolean`, `Object`, and `Array`. It includes support for enhanced date features, trigonometric functions, and regular expressions.

JavaScript uses prototypes instead of classes. You can define an object by creating a constructor function.

JavaScript is a loosely typed language, which means you do not declare the data types of variables explicitly. In many cases JavaScript performs conversions automatically when they are needed. For example, if you add a number to an item that consists of text (a string), the number is converted to text.

In This Section

Writing JavaScript Code
Explains the organization of elements within JavaScript code.

JavaScript Variables
Explains the concept of variables in JavaScript.

Data Types
Enumerates data types in JavaScript and explains the purpose of each type.

Operators
Provides a list of operators with links to information about each operator.

Operator Precedence
Explains the rules that control the order in which operations are performed when an expression is evaluated.

Controlling Program Flow
Explains conditional statements and loops in JavaScript.

Functions
Describes the built-in functions and explains how to create new functions.

Objects
Explains the concept of objects in JavaScript.

Intrinsic Objects
Lists the objects that are part of JavaScript by default and explains the purpose of each object.

Creating Your Own Objects
Explains how to create an object by defining a constructor function.

Date and Time Calculations
Describes how to perform common calendar and clock tasks, such as manipulating and comparing dates, and calculating elapsed time.

Date and Time Strings
Explains how to format date and time strings.

Displaying Text in a Web Page

Describes different ways of displaying text.

Related Sections

Advanced JavaScript

Explains advanced JavaScript functionality, such as recursion, arrays, troubleshooting, and so on.

JavaScript Language Reference

Describes the elements that make up the JavaScript language.

Writing JavaScript Code

10/18/2017 • 4 min to read • Edit Online

Like many other programming languages, JavaScript is organized into statements, blocks consisting of related sets of statements, and comments. Within a statement you can use variables, strings, numbers, and expressions.

Statements

A JavaScript program is a collection of statements. JavaScript statements combine expressions in such a way that they carry out one complete task.

A statement consists of one or more expressions, keywords, or operators (symbols). Typically, a statement is written on a single line, although a statement can be written over two or more lines. Also, two or more statements can be written on the same line by separating them with semicolons. In general, each new line begins a new statement. It is a good idea to terminate your statements explicitly. You do this with the semicolon (;), which is the JavaScript statement termination character.

Here are two examples of JavaScript statements. The sentences after the // characters are *comments*, which are explanatory remarks in the program.

```
var aBird = "Robin"; // Assign the text "Robin" to the variable aBird.
var today = new Date(); // Assign today's date to the variable today.
```

A group of JavaScript statements surrounded by braces ({}) is called a *block*. Statements grouped into a block can generally be treated as a single statement. This means you can use blocks in most places that JavaScript expects a single statement. Notable exceptions include the headers of **for** and `while` loops. Notice that the single statements within a block end in semicolons, but the block itself does not.

Generally, blocks are used in functions and conditionals. Notice that unlike C++ and some other languages, JavaScript does not consider a block to be a new scope; only functions create a new scope.

In the following example, the `else` clause contains a block of two statements surrounded by braces. The block is treated as a single statement. Also, the function itself consists of a block of statements surrounded by braces. The statements below the function are outside of the block and are therefore not part of the function definition.

```
function inchestometers(inches)
   {
   if (inches < 0)
      return -1;
   else
      {
      var meters = inches / 39.37;
      return meters;
      }
   }

var inches = 12;
var meters = inchestometers(inches);
document.write("the value in meters is " + meters);
```

Comments

A single-line JavaScript comment begins with a pair of forward slashes (//). Here is an example of a single line

comment.

```
var aGoodIdea = "Comment your code thoroughly."; // This is a single-line comment.
```

A multiline JavaScript comment begins with a forward slash and asterisk (/*), and ends with the reverse (*/).

```
/*
This is a multiline comment that explains the preceding code statement.

The statement assigns a value to the aGoodIdea variable. The value,
which is contained between the quote marks, is called a literal. A
literal explicitly and directly contains information; it does not
refer to the information indirectly. The quote marks are not part
of the literal.
*/
```

NOTE

If you attempt to embed one multiline comment within another, JavaScript interprets the resulting multiline comment in an unexpected way. The */ that marks the end of the embedded multiline comment is interpreted as the end of the whole multiline comment. This means that the text that follows the embedded multiline comment will not be commented out; instead, it will be interpreted as JavaScript code, and will generate syntax errors.

It is recommended that you write all your comments as blocks of single-line comments. This allows you to comment out large segments of code with a multiline comment later.

```
// This is another multiline comment, written as a series of single-line comments.
// After the statement is executed, you can refer to the content of the
// aGoodIdea variable by using its name.
var extendedIdea = aGoodIdea + " You never know when you'll have to figure out what it does.";
```

Assignments and Equality

The equal sign (=) is used in JavaScript statements to assign values to variables: it is the assignment operator. The left hand operand of the = operator is always an Lvalue. Examples of Lvalues are:

- variables,
- array elements,
- object properties.

  The right operand of the = operator is always an Rvalue. Rvalues can be an arbitrary value of any type, including the value of an expression. Here is an example of a JavaScript assignment statement.

```
var anInteger = 3;
```

The JavaScript compiler interprets this statement as meaning: "Assign the value 3 to the variable **anInteger**," or "**anInteger** takes the value 3."

Be certain you understand the difference between the = operator (assignment) and the == operator (equality). When you want to compare two values to find out if they are equal, use two equals signs (==). This is discussed in detail in Controlling Program Flow.

Expressions

A JavaScript expression value can be of any valid JavaScript type - a number, a string, an object, and so on. The simplest expressions are literals. Here are some examples of JavaScript literal expressions.

```
3.9                       // numeric literal
"Hello!"                  // string literal
false                     // boolean literal
null                      // literal null value
{x:1, y:2}                // Object literal
[1,2,3]                   // Array literal
function(x){return x*x;}  // function literal
```

More complicated expressions can contain variables, function calls, and other expressions. You can combine expressions to create complex expressions using operators. Examples of operators are: `+` (addition), `-` (subtraction), `*` (multiplication), and `/` (division).

Here are some examples of JavaScript complex expressions.

```
var anExpression = 3 * (4 / 5) + 6;
var aSecondExpression = Math.PI * radius * radius;
var aThirdExpression = aSecondExpression + "%" + anExpression;
var aFourthExpression = "(" + aSecondExpression + ") % (" + anExpression + ")";
```

Variables (JavaScript)

4/11/2018 • 3 min to read • Edit Online

In JavaScript, a variable contains a value, such as "hello" or 5. When you use the variable, you refer to the data it represents, for example `NumberOfDaysLeft = EndDate - TodaysDate`.

You use variables to store, retrieve, and manipulate values that appear in your code. Try to give your variables meaningful names to make it easy for other people to understand what your code does.

Declaring Variables

The first time a variable appears in your script is its declaration. The first mention of the variable sets it up in memory, so you can refer to it later on in your script. You should declare variables before using them. You do this using the `var` keyword.

```
// A single declaration.
var count;
// Multiple declarations with a single var keyword.
var count, amount, level;
// Variable declaration and initialization in one statement.
var count = 0, amount = 100;
```

If you do not initialize your variable in the `var` statement, it automatically takes on the value `undefined`.

Naming Variables

JavaScript is a case-sensitive language. This means that a variable name such as **myCounter** is different from the variable name **MYCounter**. Variable names can be of any length. The rules for creating legal variable names are as follows:

- The first character must be an ASCII letter (either uppercase or lowercase), a letter that complies with Unicode variable naming conventions, or an underscore (_) character. Note that a number cannot be used as the first character.
- Subsequent characters must be letters, numbers, or underscores (_).
- The variable name must not be a reserved word.

  Here are some examples of valid variable names:

```
_pagecount
Part9
Number_Items
```

Here are some examples of invalid variable names:

```
// Cannot begin with a number.
99Balloons
// The ampersand (&) character is not a valid character for variable names.
Alpha&Beta
```

When you want to declare a variable and initialize it, but do not want to give it any particular value, assign it the

value `null`. Here is an example.

```
var bestAge = null;
var muchTooOld = 3 * bestAge; // muchTooOld has the value 0.
```

If you declare a variable without assigning a value to it, it has the value `undefined`. Here is an example.

```
var currentCount;
// finalCount has the value NaN because currentCount is undefined.
var finalCount = 1 * currentCount;
```

The `null` value behaves like the number 0, while `undefined` behaves like the special value `NaN` (Not a Number). If you compare a `null` value and an `undefined` value, they are equal.

You can declare a variable without using the `var` keyword in the declaration, and assign a value to it. This is an implicit declaration.

```
// The variable noStringAtAll is declared implicitly.
noStringAtAll = "";
```

You cannot use a variable that has never been declared.

```
// Error. Length and width do not yet exist.
var area = length * width;
```

Coercion

JavaScript is a loosely typed language, as opposed to strongly typed languages like C++. This means that JavaScript variables have no predetermined type. Instead, the type of a variable is the type of its value. This behavior allows you to treat a value as if it were of a different type.

In JavaScript, you can perform operations on values of different types without causing an exception. The JavaScript interpreter implicitly converts, or *coerces*, one of the data types to that of the other, then performs the operation. The rules for coercion of string, number, and Boolean values are the following:

- If you add a number and a string, the number is coerced to a string.
- If you add a Boolean and a string, the Boolean is coerced to a string.
- If you add a number and a Boolean, the Boolean is coerced to a number.

  In the following example, a number added to a string results in a string.

```
var x = 2000;
var y = "Hello";
// The number is coerced to a string.
x = x + y;
document.write(x);

// Output:
// 2000Hello
```

Strings are automatically converted to equivalent numbers for comparison purposes. To explicitly convert a string to an integer, use the parseInt function. To explicitly convert a string to a number, use the parseFloat function.

Data Types (JavaScript)

10/18/2017 • 7 min to read • Edit Online

In JavaScript, there are three primary data types, two composite data types, and two special data types.

Primary Data Types

The primary (primitive) data types are:

- String
- Number
- Boolean

Composite Data Types

The composite (reference) data types are:

- Object
- Array

Special Data Types

The special data types are:

- Null
- Undefined

String Data Type

A string value is a chain of zero or more Unicode characters (letters, digits, and punctuation marks). You use the string data type to represent text in JavaScript. You include string literals in your scripts by enclosing them in single or double quotation marks. Double quotation marks can be contained in strings surrounded by single quotation marks, and single quotation marks can be contained in strings surrounded by double quotation marks. The following are examples of strings:

```
"Happy am I; from care I'm free!"
'"Avast, ye lubbers!" roared the technician.'
"45"
'c'
```

Notice that JavaScript does not have a type to represent a single character. To represent a single character in JavaScript, you create a string that consists of only one character. A string that contains zero characters ("") is an empty (zero-length) string.

JavaScript provides escape sequences that you can include in strings to create characters that you cannot type directly. For example, `\t` specifies a tab character. For more information, see Special Characters.

Number Data Type

In JavaScript, there is no distinction between integer and floating-point values; a JavaScript number can be either (internally, JavaScript represents all numbers as floating-point values).

Integer Values

Integer values can be positive whole numbers, negative whole numbers, and 0. They can be represented in base 10 (decimal), base 16 (hexadecimal), and base 8 (octal). Most numbers in JavaScript are written in decimal.

You denote hexadecimal ("hex") integers by prefixing them with a leading "0x" (zero and x|X). They can contain digits 0 through 9, and letters A through F (either uppercase or lowercase) only. The letters A through F are used to represent, as single digits, 10 through 15 in base 10. That is, 0xF is equivalent to 15, and 0x10 is equivalent to 16.

You denote octal integers by prefixing them with a leading "0" (zero). They can contain digits 0 through 7 only. A number that has a leading "0" and contains the digits "8" and/or "9" is interpreted as a decimal number.

Both hexadecimal and octal numbers can be negative, but they cannot have a decimal portion, and they cannot be written in scientific (exponential) notation.

> **NOTE**
> Starting in Internet Explorer 9 standards mode, Internet Explorer 10 standards mode, Internet Explorer 11 standards mode, and Windows Store apps, the parseInt function does not treat a string that has a prefix of "0" as octal. When you are not using the `parseInt` function, however, strings with a prefix of "0" can still be interpreted as octal.

Floating-point Values

Floating-point values can be whole numbers with a decimal portion. Additionally, they can be expressed in scientific notation. That is, an uppercase or lowercase "e" is used to represent "ten to the power of". JavaScript represents numbers using the eight byte IEEE 754 floating-point standard for numerical representation. This means you can write numbers as large as 1.79769×10^{308}, and as small as 5×10^{-324}. A number that contains a decimal point and that has a single "0" before the decimal point is interpreted as a decimal floating-point number.

Notice that a number that begins with "0x" or "00" and contains a decimal point will generate an error. Here are some examples of JavaScript numbers.

NUMBER	DESCRIPTION	DECIMAL EQUIVALENT
.0001, 0.0001, 1e-4, 1.0e-4	Four equivalent floating-point numbers.	0.0001
3.45e2	A floating-point number.	345
45	An integer.	45
0378	An integer. Although this looks like an octal number (it begins with a zero), 8 is not a valid octal digit, so the number is treated as a decimal.	378
0377	An octal integer. Notice that although it only appears to be one less than the number above, its actual value is quite different.	255
0.0001	A floating point number. Even though this begins with a zero, it is not an octal number because it has a decimal point.	0.0001

NUMBER	DESCRIPTION	DECIMAL EQUIVALENT
00.0001	This is an error. The two leading zeros mark the number as an octal, but octals are not allowed a decimal component.	N/A (compiler error)
0Xff	A hexadecimal integer.	255
0x37CF	A hexadecimal integer.	14287
0x3e7	A hexadecimal integer. Notice that the 'e' is not treated as exponentiation.	999
0x3.45e2	This is an error. Hexadecimal numbers cannot have decimal parts.	N/A (compiler error)

Additionally, JavaScript contains numbers with special values. These are:

- NaN (not a number). This is used when a mathematical operation is performed on inappropriate data, such as strings or the undefined value
- Positive Infinity. This is used when a positive number is too large to represent in JavaScript
- Negative Infinity. This is used when a negative number is too large to represent in JavaScript
- Positive and Negative 0. JavaScript differentiates between positive and negative zero.

Boolean Data Type

Whereas the string and number data types can have a virtually unlimited number of different values, the Boolean data type can only have two. They are the literals `true` and `false`. A Boolean value is a truth-value: it specifies whether the condition is true or not.

Comparisons you make in your scripts always have a Boolean outcome. Consider the following line of JavaScript code.

```
y = (x == 2000);
```

Here, the value of the variable `x` is compared to the number 2000. If it is, the result of the comparison is the Boolean value **true**, which is assigned to the variable `y`. If `x` is not equal to 2000, then the result of the comparison is the Boolean value `false`.

Boolean values are especially useful in control structures. The following code combines a comparison that creates a Boolean value directly with a statement that uses it. Consider the following JavaScript code sample.

```
if (x == 2000) {
    z = z + 1;
}
else {
    x = x + 1;
}
```

The `if/else` statement in JavaScript performs one action if a Boolean value is `true` (`z = z + 1`), and an alternate action if the Boolean value is `false` (`x = x + 1`).

You can use any expression as a comparative expression. Any expression that evaluates to 0, null, undefined, or an

empty string is interpreted as `false`. An expression that evaluates to any other value is interpreted as `true`. For example, you could use an expression such as:

```
// This may not do what you expect. See below!
if (x = y + z)
```

Note that the above line does not check whether `x` is equal to `y + z`, since only a single equal sign (the assignment operator) is used. Instead, the code above assigns the value of `y + z` to the variable `x`, and then checks whether the result of the entire expression (the value of `x`) is zero. To check whether `x` is equal to `y + z`, you need to use the following code.

```
// This is different from the code above!
if (x == y + z)
```

For more information on comparisons, see Controlling Program Flow.

The null Data Type

The `null` data type has only one value in JavaScript: null. The `null` keyword cannot be used as the name of a function or variable.

A variable that contains `null` contains no valid Number, String, Boolean, Array, or Object. You can erase the contents of a variable (without deleting the variable) by assigning it the `null` value.

Notice that in JavaScript, `null` is not the same as 0 (as it is in C and C++). Also note that the `typeof` operator in JavaScript reports `null` values as being of type `Object`, not of type `null`. This potentially confusing behavior is for backwards compatibility.

The undefined Data Type

The `undefined` value is returned when you use an object property that does not exist, or a variable that has been declared, but has never had a value assigned to it.

you can check to see if a variable exists by comparing it to `undefined`, although you can check if its type is `undefined` by comparing the type of the variable to the string "undefined". The following example shows how to find out the variable `x` has been declared:

```
var x;

// This method works.
if (x == undefined) {
    document.write("comparing x to undefined <br/>");
}
.
// This method doesn't work - you must check for the string "undefined".
if (typeof(x) == undefined) {
    document.write("comparing the type of x to undefined <br/>");
}
// This method does work.
if (typeof(x) == "undefined") {
    document.write("comparing the type of x to the string 'undefined'");
}

// Output:
// comparing x to undefined
// comparing the type of x to the string 'undefined'
```

You can also compare the undefined value to `null`. This comparison is `true` if the property `someObject.prop` is `null` or if the property `someObject.prop` does not exist.

```
someObject.prop == null;
```

To find out whether an object property exists, you can use the **in** operator:

```
if ("prop" in someObject)
    // someObject has the property 'prop'
```

See Also

Objects and Arrays
typeof Operator

Operators (JavaScript)

3/2/2018 • 1 min to read • Edit Online

JavaScript has a full range of operators, including arithmetic, logical, bitwise, assignment, as well as some miscellaneous operators. For explanations and examples, see the topics on specific operators.

Computational Operators

DESCRIPTION	SYMBOL
Unary negation	-
Increment	++
Decrement	--
Multiplication	*
Division	/
Remainder arithmetic	%
Addition	+
Subtraction	-

Logical Operators

DESCRIPTION	SYMBOL
Logical NOT	!
Less than	<
Greater than	>
Less than or equal to	<=
Greater than or equal to	>=
Equality	==
Inequality	!=
Logical AND	&&
Logical OR	||

DESCRIPTION	SYMBOL
Conditional (ternary)	?:
Comma	,
Strict Equality	===
Strict Inequality	!==

Bitwise Operators

DESCRIPTION	SYMBOL
Bitwise NOT	~
Bitwise Left Shift	<<
Bitwise Right Shift	>>
Unsigned Right Shift	>>>
Bitwise AND	&
Bitwise XOR	^
Bitwise OR	|

Assignment Operators

DESCRIPTION	SYMBOL
Assignment	=
Compound Assignment	*OP*= (such as += and &=)

Miscellaneous Operators

DESCRIPTION	SYMBOL
delete	delete
typeof	typeof
void	void
instanceof	instanceof
new	new

DESCRIPTION	SYMBOL
in	in

Equality and Strict Equality

The difference between == (equality) and === (strict equality) is that the equality operator will coerce values of different types before checking for equality. For example, comparing the string "1" with the number 1 will compare as true. The strict equality operator, on the other hand, will not coerce values to different types, and so the string "1" will not compare as equal to the number 1.

Primitive strings, numbers, and Booleans are compared by value. If they have the same value, they are equal. Objects (including `Array`, `Function`, `String`, **Number**, `Boolean`, **Error**, `Date` and `RegExp` objects) compare by reference. Even if two variables of these types have the same value, they are equal only if they refer to exactly the same object.

For example:

```
// Two strings with the same value.
var string1 = "Hello";
var string2 = "Hello";

// Two String objects with the same value.
var StringObject1 = new String(string1);
var StringObject2 = new String(string2);

if (string1 == string2)
    document.write("string1 is equal to string2 <br/>");

if (StringObject1 != StringObject2)
    document.write("StringObject1 is not equal to StringObject2 <br/>");

// To compare the values of String objects, use the toString() or valueOf() methods.
if (StringObject1.valueOf() == StringObject2.valueOf())
    document.write("The value of StringObject1 is equal to the value of StringObject2");

//Output:
// string1 is equal to string2
// StringObject1 is not equal to StringObject2
// The value of StringObject1 is equal to the value of StringObject2
```

Operator Precedence (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Operator precedence describes the order in which operations are performed when an expression is evaluated. Operations with a higher precedence are performed before those with a lower precedence. For example, multiplication is performed before addition.

JavaScript Operators

The following table lists the JavaScript operators, ordered from highest to lowest precedence. Operators with the same precedence are evaluated left to right.

OPERATOR	DESCRIPTION
.	Field access, array indexing, function calls, and expression grouping
++ -- - ~ ! delete new typeof void	Unary operators, return data type, object creation, undefined values
* / %	Multiplication, division, modulo division
+ - +	Addition, subtraction, string concatenation
<< >> >>>	Bit shifting
< <= > >= instanceof	Less than, less than or equal, greater than, greater than or equal, instanceof
== != === !==	Equality, inequality, strict equality, and strict inequality
&	Bitwise AND
^	Bitwise XOR
|	Bitwise OR
&&	Logical AND
||	Logical OR
?:	Conditional
= *OP*=	Assignment, assignment with operation (such as += and &=)
,	Multiple evaluation

Parentheses are used to alter the order of evaluation determined by operator precedence. This means an expression within parentheses is fully evaluated before its value is used in the remainder of the expression.

For example:

```
var result = 78 * 96 + 3;
document.write(result);
document.write("<br/>");

result = 78 * (9 + 3);
document.write(result);

// Output:
// 7491
// 936
```

There are three operators in the first expression: =, *, and +. According to the rules of operator precedence, they are evaluated in the following order: *, +, = (78 * 96 = 7488, 7488 + 3 = 7491).

In the second expression the ( ) operator is evaluated first, so that the addition expression is evaluated before the multiplication (9 + 3 = 12, 12 * 78 = 936).

The following example shows a statement that includes a variety of operators and resolves to `true`.

```
var num = 10;

if(5 == num / 2 && (2 + 2 * num).toString() === "22") {
    document.write(true);
}
    // Output:
    // true
```

The operators are evaluated in this order: ( ) for the grouping, *, + (within the grouping), "." for the function, ( ) for the function, /, ==, ===, and &&.

Controlling Program Flow (JavaScript)

10/18/2017 • 9 min to read • Edit Online

Normally, statements in a JavaScript script are executed one after the other, in the order in which they are written. This is called sequential execution, and is the default direction of program flow.

An alternative to sequential execution transfers the program flow to another part of your script. That is, instead of executing the next statement in the sequence, another statement is executed instead.

To make a script useful, this transfer of control must be done in a logical manner. Transfer of program control is based upon a decision, the result of which is a truth statement (returning a Boolean **true** or **false**). You create an expression, then test whether its result is true. There are two main kinds of program structures that accomplish this.

The first is the selection structure. You use it to specify alternate courses of program flow, creating a junction in your program (like a fork in a road). There are four selection structures available in JavaScript.

- the single-selection structure (**if**),
- the double-selection structure (**if/else**),
- the inline ternary operator `?:`
- the multiple-selection structure (`switch`).

  The second type of program control structure is the repetition structure. You use it to specify that an action is to be repeated while some condition remains true. When the conditions of the control statement have been met (usually after some specific number of iterations), control passes to the next statement beyond the repetition structure. There are four repetition structures available in JavaScript.

- the expression is tested at the top of the loop (`while`),
- the expression is tested at the bottom of the loop (**do/while**),
- operate on each of an object's properties (**for/in**).
- counter controlled repetition (**for**).

  You can create quite complex scripts by nesting and stacking selection and repetition control structures.

  A third form of structured program flow is provided by exception handling, which is not covered in this document.

Using Conditional Statements

JavaScript supports **if** and if...else conditional statements. In **if** statements a condition is tested, and if the condition meets the test, the relevant JavaScript code is executed. In the `if...else` statement, different code is executed if the condition fails the test. The simplest form of an **if** statement can be written on one line, but multiline **if** and `if...else` statements are much more common.

The following examples demonstrate syntaxes you can use with **if** and `if...else` statements. The first example shows the simplest kind of Boolean test. If (and only if) the item between the parentheses evaluates to (or can be coerced to) **true**, the statement or block of statements after the **if** is executed.

```
function GetReaction(newShip, color, texture, dayOfWeek)
{
   // The test succeeds if the newShip Boolean value is true.
   if (newShip)
   {
      return "Champagne Bottle";
   }

   // The test succeeds if both conditions are true.
   if (color == "deep yellow" && texture == "large and small wrinkles")
   {
      return "Is it a crenshaw melon?";
   }

   // The test succeeds if either condition is true.
   if ((dayOfWeek == "Saturday") || (dayOfWeek == "Sunday"))
   {
      return "I'm off to the beach!";
   }
   else
   {
      return "I'm going to work.";
   }
}

var reaction = GetReaction(false, "deep yellow", "smooth", "Sunday");
document.write(reaction);

// Output: I'm off to the beach!
```

Conditional Operator

JavaScript also supports an implicit conditional form. It uses a question mark after the condition to be tested (rather than the word **if** before the condition). It also specifies two alternatives, one to be used if the condition is met and one if it is not. A colon must separate these alternatives.

```
var AMorPM = (theHour >= 12) ? "PM" : "AM";
```

If you have several conditions to be tested together, and you know that one is more likely to pass or fail than the others, you can use a feature called 'short circuit evaluation' to speed the execution of your script. When JavaScript evaluates a logical expression, it only evaluates as many sub-expressions as required to get a result.

For example, if you have an AND expression such as ((x == 123) && (y == 6)), JavaScript first checks if x is 123. If it is not, the entire expression cannot be true, even if y is equal to 6. Hence, the test for y is never made, and JavaScript returns the value **false**.

Similarly, if only one of several conditions must be **true** (using the || operator), testing stops as soon as any one condition passes the test. This is effective if the conditions to be tested involve the execution of function calls or other complex expressions. With this in mind, when you write Or expressions, place the conditions most likely to be **true** first. When you write And expressions, place the conditions most likely to be **false** first.

A benefit of designing your script in this manner is that **runsecond()** will not be executed in the following example if **runfirst()** returns 0.

```
if ((runfirst() == 0) || (runsecond() == 0)) {
    // some code
}
```

Using Loops

There are several ways to execute a statement or block of statements repeatedly. In general, repetitive execution is called *looping* or *iteration*. An iteration is simply a single execution of a loop. It is typically controlled by a test of a variable, where the value of which is changed each time the loop is executed. JavaScript supports four types of loops: for loops, for...in loops, while loops, do...while loops.

Using for Loops

The **for** statement specifies a counter variable, a test condition, and an action that updates the counter. Before each iteration of the loop, the condition is tested. If the test is successful, the code inside the loop is executed. If the test is unsuccessful, the code inside the loop is not executed, and the program continues on the first line of code immediately following the loop. After the loop is executed, the counter variable is updated before the next iteration begins.

If the condition for looping is never met, the loop is never executed. If the test condition is always met, an infinite loop results. While the former may be desirable in certain cases, the latter rarely is, so be cautious when writing your loop conditions.

```
// The update expression ("icount++" in the following examples)
// is executed at the end of the loop, after the block of
// statements that forms the body of the loop is executed, and
// before the condition is tested.

// Set a limit of 10 on the loop.
var howFar = 10;

// Create an array called sum with 10 members, 0 through 9.
var sum = new Array(howFar);
sum[0] = 0;

// Iterate from 0 through 9.
var theSum = 0;
for(var icount = 0; icount < howFar; icount++)
{
   theSum += icount;
   sum[icount] = theSum;
}

// This code is not executed at all, because icount is not greater than howFar.
var newSum = 0;
for(var icount = 0; icount > howFar; icount++)
{
   newSum += icount;
}

// This is an infinite loop.
var sum = 0;
for(var icount = 0; icount >= 0; icount++)
{
   sum += icount;
}
```

Using for...in Loops

JavaScript provides a special kind of loop for stepping through all the user-defined properties of an object, or all the elements of an array. The loop counter in a `for...in` loop is a string, not a number. It contains the name of current property or the index of the current array element.

```
// Create an object with some properties
var myObject = new Object();
myObject.name = "James";
myObject.age = "22";
myObject.phone = "555 1234";

// Enumerate (loop through)_all the properties in the object
for (var prop in myObject)
{
    // This displays "The property 'name' is James", etc.
    document.write("The property '" + prop + "' is " + myObject[prop]);
    // New line.
    document.write("<br />");
}
```

Although `for...in` loops look similar to VBScript's `For Each...Next` loops, they do not work the same way. The JavaScript**for...in loop** iterates over properties of JavaScript objects. The VBScript **For Each...Next** loop iterates over items in a collection. To loop over collections in JavaScript, you need to use the Enumerator Object object or, if present, the `forEach` method of the collection object. Although some objects, such as those in Internet Explorer, support both the VBScript **For Each...Next** loop and the JavaScript**for...in** loop, most objects do not.

Using while Loops

A `while` loop is similar to a **for** loop. The difference is, a `while` loop does not have a built-in counter variable or update expression. If you want to control repetitive execution of a statement or block of statements, but need a more complex rule than simply "run this code n times", use a `while` loop. The following example uses the Internet Explorer object model and a `while` loop to ask the user a simple question.

```
var x = 0;
while ((x != 5) && (x != null))
{
    x = window.prompt("What is my favorite number?", x);
}

if (x == null)
    window.alert("You gave up!");
else
    window.alert("Correct answer!");
```

NOTE

Because `while` loops do not have explicit built-in counter variables, they are more vulnerable to infinite looping than the other types of loops. Moreover, because it is not necessarily easy to discover where or when the loop condition is updated, it is easy to write a `while` loop in which the condition never gets updated. For this reason, you should be careful when you design `while` loops.

As noted above, there is also a **do...while** loop in JavaScript that is similar to the while loop, except that it is guaranteed to always execute at least once, since the condition is tested at the end of the loop, rather than at the start. For example, the loop above can be re-written as:

```
var x = 0;
do
{
    x = window.prompt("What is my favorite number?", x);
} while ((x != 5) && (x != null));

if (x == null)
    window.alert("You gave up!");
else
    window.alert("Correct answer!");
```

Using break and continue Statements

In JavaScript, the break statement is used to stop the execution of a loop, if some condition is met. (Note that **break** is also used to exit a `switch` block). The continue statement can be used to jump immediately to the next iteration, skipping the rest of the code block, while updating the counter variable if the loop is a **for** or `for...in` loop.

The following example builds on the previous example to use the **break** and **continue** statements to control the loop.

```
var x = 0;
do
{
    x = window.prompt("What is my favorite number?", x);

    // Did the user cancel? If so, break out of the loop
    if (x == null)
        break;

    // Did they enter a number?
    // If so, no need to ask them to enter a number.
    if (Number(x) == x)
        continue;

    // Ask user to only enter in numbers
    window.alert("Please only enter in numbers!");

} while (x != 5)

if (x != 5)
    window.alert("You gave up!");
else
    window.alert("Correct answer!");
```

Functions (JavaScript)

1/13/2018 • 7 min to read • Edit Online

JavaScript functions perform actions; they can also return values. Sometimes these are the results of calculations or comparisons. Functions are also called "global methods".

Functions combine several operations under one name. This lets you streamline your code. You can write out a set of statements, name it, and then execute the entire set by calling it and passing to it any information it needs.

You pass information to a function by enclosing the information in parentheses after the name of the function. Pieces of information that are passed to a function are called arguments or parameters. Some functions do not take any arguments at all while others take one or more arguments. In some functions, the number of arguments depends on how you are using the function.

JavaScript supports two kinds of functions: those that are built into the language, and those you create yourself.

Built-in Functions

The JavaScript language includes several built-in functions. Some let you handle expressions and special characters, while others convert strings to numeric values.

See JavaScript Methods for information about these built-in functions.

Creating Your Own Functions

You can create your own functions and use them where needed. A function definition consists of a function statement and a block of JavaScript statements.

The **checkTriplet** function in the following example takes the lengths of the sides of a triangle as its arguments. It calculates from them whether the triangle is a right triangle by checking whether the three numbers constitute a Pythagorean triplet (the square of the length of the hypotenuse of a right triangle is equal to the sum of the squares of the lengths of the other two sides). The checkTriplet function calls one of two other functions to make the actual test.

Notice the use of a very small number ("epsilon") as a testing variable in the floating-point version of the test. Because of uncertainties and round-off errors in floating-point calculations, it is not practical to make a direct test of whether the three numbers constitute a Pythagorean triplet unless all three values in question are known to be integers. Because a direct test is more accurate, the code in this example determines whether it is appropriate and, if it is, uses it.

```
var epsilon = 0.00000000001; // Some very small number to test against.

// The test function for integers.
function integerCheck(a, b, c)
{
   // The test itself.
   if ( (a*a) == ((b*b) + (c*c)) )
      return true;

   return false;
} // End of the integer checking function.

// The test function for floating-point numbers.
function floatCheck(a, b, c)
{
```

```
   // Make the test number.
   var delta = ((a*a) - ((b*b) + (c*c)))

   // The test requires the absolute value
   delta = Math.abs(delta);

   // If the difference is less than epsilon, then it's pretty close.
   if (delta < epsilon)
      return true;

   return false;
} // End of the floating-poing check function.

// The triplet checker.
function checkTriplet(a, b, c)
{
   // Create a temporary variable for swapping values
   var d = 0;

   // First, move the longest side to position "a".

   // Swap a and b if necessary
   if (b > a)
   {
      d = a;
      a = b;
      b = d;
   }

   // Swap a and c if necessary
   if (c > a)
   {
      d = a;
      a = c;
      c = d;
   }

   // Test all 3 values. Are they integers?
   if (((a % 1) == 0) && ((b % 1) == 0) && ((c % 1) == 0))
   {
      // If so, use the precise check.
      return integerCheck(a, b, c);
   }
   else
   {
      // If not, get as close as is reasonably possible.
      return floatCheck(a, b, c);
   }
} // End of the triplet check function.

// The next three statements assign sample values for testing purposes.
var sideA = 5;
var sideB = 5;
var sideC = Math.sqrt(50.001);

// Call the function. After the call, 'result' contains the result.
var result = checkTriplet(sideA, sideB, sideC);
```

Arrow Functions

Arrow function syntax, `=>`, provides a shorthand method of specifying an anonymous function. Here is the arrow function syntax.

```
([arg] [, arg]) => {
    statements
}
```

Values to the left of the arrow, which may be enclosed by parentheses, specify the arguments passed to the function. A single argument to the function does not require parentheses. Parentheses are required if no arguments are passed in. The function definition to the right of the arrow can be either an expression, such as `v + 1`, or a block of statements enclosed by braces ({}).

> **IMPORTANT**
>
> Arrow function syntax is supported only in Microsoft Edge.

You cannot use the `new` operator with an arrow function.

The following code examples show the use of the arrow function with expressions as the function definitions. In the first example, v is passed in as the argument to the expression. In the second example, v and i are passed in as the arguments to the expression.

```
var evens = [2, 4, 6, 8];

// Using standard syntax.
var odds = evens.map(function(v) { return v + 1; });

// Using arrow function syntax.
// Add one to each value to produce output.
var odds = evens.map(v => v + 1);

// The following line of code adds the index value to the passed
// in value to produce output.
// Note: the second argument to the callback function in the map
// method is the index value (i).
var nums = evens.map((v, i) => v + i);

console.log(odds);
console.log(nums);

// Output:
// [object Array] [3, 5, 7, 9]
// [object Array] [2, 5, 8, 11]
```

The following code example shows the use of the arrow function with a statement block.

```
var fives = new Array();

// Statement block, re-using nums array from previous example.
// Note: The first argument to the callback function in forEach
// is the value of the array element (v).
nums.forEach(v => {
  if (v % 5 === 0)
    fives.push(v);
});

console.log(fives);

// Output:
// [object Array] [5]
```

Unlike standard functions, Arrow functions share the same lexical `this` object as the surrounding code, which

can be used to eliminate the need for workarounds such as `var self = this;`.

The following example shows that the value of the `this` object within the arrow function is the same as in the surrounding code (it still refers to the `bob` variable.

```
var bob = {
  _name: "Bob",
  _friends: ["Pete", "Joe", "Larry"],
  printFriends() {
    this._friends.forEach(f =>
      console.log(this._name + " knows " + f));
  }
}

// Output:
// Bob knows Pete
// Bob knows Joe
// Bob knows Larry
```

Arrow functions also share the same lexical `arguments` object as the surrounding code (just like the `this` object).

Default parameters

You can specify a default value for a parameter in a function by assigning it an initial value. The default value may be a constant value or an expression.

> **IMPORTANT**
>
> Default parameters are supported only in Microsoft Edge with experimental JavaScript features enabled (about:flags).

In the following example, the default value of y is 10, and the default value of z is 20. The function will use 10 as the value of y unless the caller passes in a distinct value (or undefined) as the second argument. The function will use 20 as the value of z unless the caller passes in a distinct value (or undefined) as the third argument.

```
var val = 20;

function f(x, y=10, z=val) {
  return x + y + z;
}

console.log(f(3));
console.log(f(3, 3));
console.log(f(3, 3, 3));

// Output:
// 33
// 26
// 9
```

Rest parameters

Rest parameters, specified by the spread operator `...`, allow you to turn consecutive arguments in a function call to an array.

Rest parameters eliminate the need for the `arguments` object. Rest parameters differ from the `arguments` object in several ways, such as:

- A rest parameter is an actual array instance and therefore supports operations that can be performed on

an array.

- A rest parameter includes only the consecutive arguments that are not passed in as separate (named) arguments (conversely, the `arguments` object contains all arguments passed into the function).

> **IMPORTANT**
>
> Rest parameters and the spread operator are supported only in Microsoft Edge.

In the following code example, "hello" and true are passed in as array values and stored in the y parameter. The rest parameter must be the last parameter of the function.

```
function f(x, ...y) {
  // y is an array.
  return x * y.length;
}

console.log(f(3, "hello", true));

// Output:
// 6
```

For additional uses of the spread operator, see Spread Operator.

See Also

JavaScript Language Reference

Objects and Arrays (JavaScript)

10/18/2017 • 3 min to read • Edit Online

JavaScript objects are collections of properties and methods. A method is a function that is a member of an object. A property is a value or set of values (in the form of an array or object) that is a member of an object. JavaScript supports four kinds of objects:

- Intrinsic objects, such as `Array` and `String`.
- Objects you create.
- Host objects, such as `window` and `document`.
- ActiveX objects.

Expando Properties and Methods

All objects in JavaScript support expando properties and methods, which can be added and removed at run time. These properties and methods can have any name and can be identified by numbers. If the name of the property or method is a simple identifier, it can be written after the object name with a period, such as `myObj.name`, `myObj.age`, and `myObj.getAge` in the following code:

```
var myObj = new Object();
myObj.name = "Fred";
myObj.age = 42;

myObj.getAge =
    function () {
        return this.age;
    };

document.write(myObj.name);
document.write("<br/>");
document.write(myObj.age);
document.write("<br/>");
document.write(myObj.getAge());

// Output:
// Fred
// 42
// 42
```

If the name of the property or method is not a simple identifier or is unknown at the time you write the script, you can use an expression inside square brackets to index the property. The names of all expando properties in JavaScript are converted to strings before being added to the object.

```
var myObj = new Object();

// Add two expando properties that cannot be written in the
// object.property syntax.
// The first contains invalid characters (spaces), so must be
// written inside square brackets.
myObj["not a valid identifier"] = "This is the property value";

// The second expando name is a number, so it also must
// be placed inside square brackets
myObj[100] = "100";
```

For information about creating an object from a definition, see Creating Objects.

Arrays as Objects

In JavaScript, objects and arrays are handled almost identically, because arrays are merely a special kind of object. Both objects and arrays can have properties and methods.

Arrays have a `length` property but objects do not. When you assign a value to an element of an array whose index is greater than its length (for example, `myArray[100] = "hello"`), the `length` property is automatically increased to the new length. Similarly, if you make the `length` property smaller, any element whose index is outside the length of the array is deleted.

```
// An array with three elements
var myArray = new Array(3);

// Add some data
myArray[0] = "Hello";
myArray[1] = 42;
myArray[2] = new Date(2000, 1, 1);

document.write("original length is: " + myArray.length);
document.write("<br/>");
// Add some expando properties
myArray.expando = "JavaScript!";
myArray["another Expando"] = "Windows";

// This will still display 3, since the two expando properties
// don't affect the length.
document.write("new length is : " + myArray.length);

// Output:
// original length is: 3
// new length is : 3
```

Arrays provide methods to iterate over and manipulate members. The following example shows how to obtain the properties of objects stored in an array.

```
var myArray = new Array(3);

// Add some data
for(var i = 1; i <= 3; i++) {
    myArray[i] = new Date(2000 + i, 1, 1);
}

myArray.forEach(function (item) {
    document.write(item.getFullYear());
});

// Output:
// 2001
// 2002
// 2003
```

Multi-Dimensional Arrays

JavaScript does not directly support multi-dimensional arrays, but you can get the behavior of multi-dimensional arrays by storing arrays within the elements of another array. (You can store any sort of data inside array elements, including other arrays.) For example, the following code builds a multiplication table for the numbers up to 5.

```
// The size of the table.
var iMaxNum = 5;
// Loop counters.
var i, j;

// Set the length of the array to iMaxNum + 1.
// The first array index is zero, not 1.
var MultiplicationTable = new Array(iMaxNum + 1);

// Loop for each major number (each row in the table)
for (i = 1; i <= iMaxNum; i++)
{
    // Create the columns in the table
    MultiplicationTable[i] = new Array(iMaxNum + 1);

    // Fill the row with the results of the multiplication
    for (j = 1; j <= iMaxNum; j++)
    {
        MultiplicationTable[i][j] = i * j;
    }
}

document.write(MultiplicationTable[3][4]);
document.write("<br/>");
document.write(MultiplicationTable[5][2]);
document.write("<br/>");
document.write(MultiplicationTable[1][4]);

// Output:
// 12
// 10
// 4
```

Intrinsic Objects (JavaScript)

10/18/2017 • 4 min to read • Edit Online

JavaScript provides intrinsic (or "built-in") objects. They are the `Array`, `Boolean`, `Date`, `Error`, `Function`, **Global**, **JSON**, **Math**, **Number**, `Object`, `RegExp`, and `String` objects. The intrinsic objects have associated methods, functions, properties, and constants that are described in detail in the language reference.

Array Object

The subscripts of an array can be thought of as properties of an object, and are referred to by their numeric index. Note that named properties added to an array cannot be indexed by number; they are separate from the array elements.

To create a new array, use the **new** operator and the **Array()** constructor, as in the following example.

```
var theMonths = new Array(12);
theMonths[0] = "Jan";
theMonths[1] = "Feb";
theMonths[2] = "Mar";
theMonths[3] = "Apr";
theMonths[4] = "May";
theMonths[5] = "Jun";
theMonths[6] = "Jul";
theMonths[7] = "Aug";
theMonths[8] = "Sep";
theMonths[9] = "Oct";
theMonths[10] = "Nov";
theMonths[11] = "Dec";
```

When you create an array using the `Array` keyword, JavaScript includes a **length** property, which records the number of entries. If you do not specify a number, the length is set to 0, and the array has no entries. If you specify a number, the length is set to that number. If you specify more than one parameter, the parameters are used as entries in the array. In addition, the number of parameters is assigned to the length property, as in the following example, which is equivalent to the preceding example.

```
var theMonths = new Array("Jan", "Feb", "Mar", "Apr", "May", "Jun",
"Jul", "Aug", "Sep", "Oct", "Nov", "Dec");
```

JavaScript automatically changes the value of **length** when you add elements to an array that you created with the `Array` keyword. Array indices in JavaScript always start at 0, not 1, so the length property is always one greater than the largest index in the array.

String Object

In JavaScript, you can treat strings (and numbers) as if they were objects. The string Object has certain built-in methods, which you can use with your strings. One of these is the substring Method, which returns part of the string. It takes two numbers as its arguments.

```
var aString = "0123456789";

// This code sets aChunk to "456".
var aChunk = aString.substring(4, 7);

// This code sets anotherChunk to "456", using
// the lower value index as the starting index.
var anotherChunk = aString.substring(7, 4);

// This code sets the firstLetter variable to "J"
// by using the array in the preceding array creation example.
firstLetter = theMonths[5].substring(0,1);
```

Another property of the `String` object is the **length** property. This property contains the number of characters in the string (0 for an empty string). This a numeric value, and can be used directly in calculations.

```
var howLong = "Hello World".length  // Sets the howLong variable to 11.
```

Math Object

The **Math** object has a number of predefined constants and functions. The constants are specific numbers. One of these specific numbers is the value of pi (approximately 3.14159...). This is the **Math.PI** constant, shown in the following example.

```
var radius = 5;
var circleArea = Math.PI * radius * radius;
```

One of the built-in functions of the **Math** object is the exponentiation method, or `Math.pow`, which raises a number to a specified power. The following example uses both pi and exponentiation to calculate the volume of a sphere.

```
var volume = (4/3)*(Math.PI*Math.pow(radius,3));
```

Date Object

The `Date` object can be used to represent arbitrary dates and times, to get the current system date, and to calculate differences between dates. It has several properties and methods, all predefined. In general, the `Date` object provides the day of the week; the month, day, and year; and the time in hours, minutes, and seconds. This information is based on the number of milliseconds since January 1, 1970, 00:00:00.000 GMT, which is Greenwich Mean Time (the preferred term is UTC, or "Universal Coordinated Time," which refers to signals issued by the World Time Standard). JavaScript can handle dates that are in the approximate range 250,000 B.C. to 255,000 A.D.

To create a new `Date` object, use the **new** operator, as shown in the following example.

```
var toDay = new Date();
var thisYear = toDay.getFullYear();
var thisMonth = theMonths[toDay.getMonth()];
var thisDay = thisMonth  + " " + toDay.getDate() + ", " + thisYear;
```

Number Object

In addition to the special numeric constants (`PI`, for example) that are available in the **Math** object, several other constants are available in JavaScript through the **Number** object.

CONSTANT	DESCRIPTION
Number.MAX_VALUE	Largest possible number, about 1.79E+308; can be positive or negative. (Value varies slightly from system to system.)
Number.MIN_VALUE	Smallest possible number, about 5.00E-324; can be positive or negative. (Value varies slightly from system to system.)
Number.NaN	Special nonnumeric value, "not a number."
Number.POSITIVE_INFINITY	Any positive value larger than the largest positive number (Number.MAX_VALUE) is automatically converted to this value; represented as infinity.
Number.NEGATIVE_INFINITY	Any value more negative than the largest negative number (-Number.MAX_VALUE) is automatically converted to this value; represented as -infinity.

Number.NaN is a special constant that is defined as "not a number." An attempt to parse a string that cannot be parsed as a number returns **Number.NaN**. `NaN` compares unequal to any number and to itself. To test for a `NaN` result, do not compare against **Number.NaN**; use the **isNaN()** function instead.

JSON Object

JSON is a lightweight data-interchange format based on a subset of the object literal notation of the JavaScript language.

The JSON object provides two functions to convert to and from JSON text format. The JSON.stringify function serializes objects and arrays into JSON text. The JSON.parse function de-serializes JSON text to produce in-memory objects. For more information, see An Introduction to JavaScript Object Notation (JSON) in JavaScript and .NET.

See Also

JavaScript Objects

Creating Objects (JavaScript)

10/18/2017 • 2 min to read • Edit Online

There are a number of ways you can create your own objects in JavaScript. You can directly instantiate an Object Object and then add your own properties and methods. Or you can use object literal notation to define your object. You can also use a constructor function to define an object. For more information about using constructor functions, see Using Constructors to Define Types.

Example

The following code shows how to instantiate an object and add some properties. In this case only the `pasta` object has the `grain`, `width`, and `shape` properties.

```
const pasta = new Object();
pasta.grain = "wheat";
pasta.width = 0.5;
pasta.shape = "round";
pasta.getShape = function() {
    return this.shape;
};
document.write(pasta.grain);
document.write("<br/>");
document.write(pasta.getShape());

// Output:
// wheat
// round
```

Object literals

You can also use object literal notation when you want to create only one instance of an object. The following code shows how to instantiate an object by using object literal notation.

```
const pasta = {
    grain: "wheat",
    width: 0.5,
    shape: "round"
};
```

You can also use an object literal inside a constructor.

Caution

The features described below are supported only in Microsoft Edge.

In Microsoft Edge, you can use shorthand syntax to create an object literal.

```
const key = 'a';
const value = 5;

// Older version
const obj1 = {
    key: key,
    value: value
};

// Edge mode
const obj2 = {key, value};

console.log(obj2);

// Output:
// [object Object] {key: "a", value: 5}
```

The following example shows the use of shorthand syntax to define methods in object literals.

```
// Older versions
const obj = {
    method1: function() {},
    method2: function() {}
};

// Edge mode
const obj = {
    method1() {},
    method2() {}
};
```

You can also set property names dynamically in object literals in Microsoft Edge. The following code example creates a property name for an object dynamically using the set syntax.

```
const propName = "prop_42";

const obj = {
    value: 0,
    set [propName](v) {
        this.value = v;
    }
}

console.log(obj.value);
// Runs the setter property.
obj.prop_42 = 777;
console.log(obj.value);

// Output:
// 0
// 777
```

The following code example creates a property name for an object dynamically using the get syntax.

```
const propName = "prop_42";

const obj = {
    get [propName]() {
        return 777;
    }
}

console.log(obj.prop_42);

// Output:
// 777
```

The following code example creates a computed property using arrow function syntax to append 42 to the property name.

```
const obj = {
    [ 'prop_' + (() => 42)() ]: 42
};
```

Calculating Dates and Times (JavaScript)

10/18/2017 • 6 min to read • Edit Online

You can use the Date object to perform common calendar and clock tasks, such as comparing dates and calculating elapsed time.

Setting a Date to the Current Date

When you create an instance of the Date object without specifying a date, it returns a value that represents the current date and time, including year, month, day, hour, minute, second, and millisecond. You can then read or modify this date and time.

The following example shows how to instantiate a date without using any parameters and display it in the format *mm-dd-yy*.

```
var dt = new Date();

// Display the month, day, and year. getMonth() returns a 0-based number.
var month = dt.getMonth()+1;
var day = dt.getDate();
var year = dt.getFullYear();
document.write(month + '-' + day + '-' + year);

// Output: current month, day, year
```

Setting a Specific Date

You can set a specific date by passing a date string to the constructor.

```
var dt = new Date('8/24/2009');
document.write(dt);

// Output: Mon Aug 24 00:00:00 PDT 2009
```

IMPORTANT

The time zone displayed in the date string corresponds to the time zone set on the local machine.

JavaScript is flexible about the format of the string you use as the parameter. For example, you can input "8-24-2009", "August 24, 2009", or "24 Aug 2009".

You can also specify a time. The following example shows one way to specify a date and time in ISO format. The "Z" indicates UTC time.

```
var dt = new Date('2010-06-09T15:20:00Z');
document.write(dt);
document.write("<br />");
document.write(dt.toISOString());

// Output:
// Wed Jun 09 2010 08:20:00 GMT-0700 (Pacific Daylight Time)
// 2010-06-09T15:20:00.000Z
```

For more information on date formats such as ISO, see Date and Time Strings.

The following example shows other ways to specify a time.

```
var dtA = new Date('8/24/2009 14:52:10');

// The parameters are year, month, day, hours, minutes, seconds.
var dtB = new Date(2009, 7, 24, 14, 52, 10);
document.write(dtA);
document.write("<br/>");
document.write(dtB);

// Output:
// Mon Aug 24 14:52:10 PDT 2009
// Mon Aug 24 14:52:10 PDT 2009
```

Adding and Subtracting Days, Months, and Years

You can use the getX and setX methods of the `Date` object to set specific dates and times.

The following example shows how you can set a date to the previous day. Note that if necessary the month and year values are also changed.

```
var myDate = new Date("1/1/1990");
var dayOfMonth = myDate.getDate();
myDate.setDate(dayOfMonth - 1);

document.write(myDate);

// Output: Sun Dec 31 00:00:00 PST 1989
```

The following example sets the date to the last day of the month by subtracting a day from the first day of the next month.

TIP

The months of the year are numbered from 0 (January) to 11 (December). The days of the week are numbered from 0 (Sunday) to 6 (Saturday).

```
var myDate = new Date("1/1/1990")
myDate.setMonth(myDate.getMonth() + 1);

myDate.setDate (myDate.getDate() - 1);

document.write(myDate);

// Output: Wed Jan 31 00:00:00 PST 1990
```

Working with Days of the Week

The getDay method gets the day of the week as a number between 0 (Sunday) and 6 (Saturday). (This is not the same as the getDate method, which gets the day of the month as a number between 1 and 31).

The following example sets the date for Thanksgiving, which in the United States is the fourth Thursday in November. The script finds November 1 of the current year, then finds the first Thursday, and then adds three weeks.

```
var myDate = new Date();
myDate.setHours(0, 0, 0, 0);

myDate.setYear(2013);

// Determine November 1.
myDate.setDate(1);
myDate.setMonth(10);

// Find Thursday.
var thursday = 4;
while(myDate.getDay() != thursday) {
    myDate.setDate(myDate.getDate() + 1);
}

// Add 3 weeks.
myDate.setDate(myDate.getDate() + 21);

document.write(myDate);

// Output: Thu Nov 28 00:00:00 <time zone> 2013
```

Calculating Elapsed Time

The getTime method returns the number of milliseconds that have elapsed since midnight on January 1, 1970. For any date before that date it returns a negative number.

You can use the getTime method to set a start and end time for calculating an elapsed time. It can be used for measuring small units, such as a few seconds, and large units, such as days.

The following example calculates elapsed time in seconds. The getTime method gets the number of milliseconds since the zero date.

```
var startTime = new Date('1/1/1990');
var startMsec = startTime.getMilliseconds();
startTime.setTime(5000000);
var elapsed = (startTime.getTime() - startMsec) / 1000;
document.write(elapsed);

// Output: 5000
```

To work with more manageable units, you can divide the milliseconds provided by the getTime method by an appropriate number. For instance, to turn milliseconds into days, divide the number by 86,400,000 (1000 milliseconds x 60 seconds x 60 minutes x 24 hours).

The following example shows how much time has elapsed since the first day of the specified year. It uses division operations to calculate elapsed time in days, hours, minutes, and seconds. It does not account for daylight savings time.

```
// Set the unit values in milliseconds.
var msecPerMinute = 1000 * 60;
var msecPerHour = msecPerMinute * 60;
var msecPerDay = msecPerHour * 24;

// Set a date and get the milliseconds
var date = new Date('6/15/1990');
var dateMsec = date.getTime();

// Set the date to January 1, at midnight, of the specified year.
date.setMonth(0);
date.setDate(1);
date.setHours(0, 0, 0, 0);

// Get the difference in milliseconds.
var interval = dateMsec - date.getTime();

// Calculate how many days the interval contains. Subtract that
// many days from the interval to determine the remainder.
var days = Math.floor(interval / msecPerDay );
interval = interval - (days * msecPerDay );

// Calculate the hours, minutes, and seconds.
var hours = Math.floor(interval / msecPerHour );
interval = interval - (hours * msecPerHour );

var minutes = Math.floor(interval / msecPerMinute );
interval = interval - (minutes * msecPerMinute );

var seconds = Math.floor(interval / 1000 );

// Display the result.
document.write(days + " days, " + hours + " hours, " + minutes + " minutes, " + seconds + " seconds.");

//Output: 164 days, 23 hours, 0 minutes, 0 seconds.
```

Determining the User's Age

The following example takes the user's birthday and calculates the user's age in years. It subtracts the birth year from the current year, and then subtracts 1 if the birthday has not occurred yet in the current year.

```
var birthday = new Date("8/1/1985");
var today = new Date();
var years = today.getFullYear() - birthday.getFullYear();

// Reset birthday to the current year.
birthday.setFullYear(today.getFullYear());

// If the user's birthday has not occurred yet this year, subtract 1.
if (today < birthday)
{
    years--;
}
document.write("You are " + years + " years old.");

// Output: You are <number of years> years old.
```

Comparing Dates

When you compare dates in JavaScript, you should keep in mind that the `==` operator returns `true` only if the dates on both sides of the operator refer to the same object. Therefore, if you have two separate `Date` objects set to the same date, `date1 == date2` returns `false`. In addition, a `Date` object set with only the date and not the

time is initialized to midnight of that date. So if you compare one `Date` set without a specified time to `Date.now`, for example, you should be aware that the first `Date` is set to midnight and `Date.now` is not.

The following example checks whether the current date is the same, before, or after a specified date. To set the current date in `todayAtMidn`, the script creates a `Date` object for the current year, month, and day.

```
// Get the current date at midnight.
var now = new Date();
var todayAtMidn = new Date(now.getFullYear(), now.getMonth(), now.getDate());

// Set specificDate to a specified date at midnight.
var specificDate = new Date("9/21/2009");

// Compare the two dates by comparing the millisecond
// representations.
if (todayAtMidn.getTime() == specificDate.getTime())
{
    document.write("Same");
}
else
{
    document.write("Different");
}

//Output: Different
```

By modifying the preceding example, we can check whether a provided date is within a particular range.

```
// Get the current date at midnight.
var now = new Date();
var todayAtMidn = new Date(now.getFullYear(), now.getMonth(), now.getDate());

// Set start/end dates to a specified date (ISO format).
var startDate = new Date("2009-06-09T15:20:00Z");
var endDate = new Date("2011-06-09T15:20:00Z");

// Compare the two dates by comparing the millisecond
// representations.
if (todayAtMidn.getTime() > startDate.getTime() &&
    todayAtMidn.getTime() < endDate.getTime()) {
    document.write("Specified date is within this range.");
}
else {
    document.write("Specified date is not in this range.");
}

// Output: Specified date is not in this range.
```

See Also

Date Object

Date and Time Strings (JavaScript)

10/18/2017 • 6 min to read • Edit Online

You can use a number of techniques to specify and format JavaScript date and time strings.

Formatting Dates using Intl.DateTimeFormat

Internet Explorer 11 introduces support for the Intl.DateTimeFormat Object, which is part of the ECMAScript Internationalization API Specification. To format dates, you can use this object directly or you can use the updated implementation of toLocaleDateString Method (Date) and toLocaleTimeString Method (Date). These methods of Date Object use `Intl.DateTimeFormat` internally to support new optional parameters for the locale and other formatting options.

The following example shows how to use `toLocaleDateString` and `toLocaleTimeString` to format dates and times. The first parameter passed to these methods is a locale value, such as "en-us". The second parameter, where present, specifies formatting options, such as the long form for the weekday.

```
var date = new Date(Date.UTC(2013, 1, 1, 14, 0, 0));
var options = {
    weekday: "long", year: "numeric", month: "short",
    day: "numeric", hour: "2-digit", minute: "2-digit"
};

document.write(date.toLocaleDateString("en-US"));
document.write(date.toLocaleTimeString("en-us", options));
document.write(date.toLocaleDateString("ja-JP"));
document.write(date.toLocaleTimeString("ja-JP", options));

// Output:
// 2/1/2013
// Friday, Feb 1, 2013 06:00 AM
// 2013年2月1日
// 2013年2月1日 金曜日 06:00
```

For a complete list of formatting options, see Intl.DateTimeFormat Object.

Formatting Dates

Before Internet Explorer 11, JavaScript did not have specific methods to format dates and times. To provide your own date formatting for previous browser versions, use the getDay Method (Date), getDate Method (Date), getMonth Method (Date), and getFullYear Method (Date) methods. (The getYear Method (Date) is obsolete and should not be used.)

```
var myDate = new Date("February 3, 2001");
var myDate = new Date("February 3 2001");
document.write((myDate.getMonth() + 1) + "-" + myDate.getDate() + "-" + myDate.getFullYear());
document.write("<br/>");
document.write((myDate.getMonth() + 1) + "/" + myDate.getDate() + "/" + myDate.getFullYear());

//Output:
// 2-3-2001
// 2/3/2001
```

You can provide your own time formatting by using the getHours Method (Date), getMinutes Method (Date),

getSeconds Method (Date), and getMilliseconds Method (Date) methods.

```
myDate = new Date();
myDate.setHours(10, 30, 53, 400);

document.write(myDate.getHours() + ":" + myDate.getMinutes() + ":" + myDate.getSeconds() +
":" + myDate.getMilliseconds());

// Output:
// 10:30:53:400
```

Converting Strings to Dates

You can specify strings to construct `Date` objects either with `Date(dateStr)` or with `Date.parse(dateStr)`. JavaScript uses the following rules to parse date strings:

- It first tries to parse a date string by using the ISO Date Format.

  > **NOTE**
  > JavaScript uses a simplified version of the ISO 8601 extended format.

- If the date string is not in ISO format, JavaScript tries to parse the date by using other Other Date Formats.

ISO Date Format

The ISO format is a simplification of the ISO 8601 extended format. The format is as follows:

`YYYY-MM-DDTHH:mm:ss.sssZ`

> **IMPORTANT**
> ISO Date Format is not supported in Internet Explorer 8 standards mode and Quirks mode.

The following table describes the parts of this format.

SYMBOL	DESCRIPTION	VALUES
`-`, `:`, `.`, `T`	Characters actually in the string. `T` specifies the start of a time.	
`YYYY`	Year. An extended year can be used instead of a 4-digit year. For more information, see Extended Years later in this topic.	
`MM`	Month	01 to 12
`DD`	Day of the month	01 to 31
`HH`	Hours	00 to 24
`mm`	Minutes	00 to 59

SYMBOL	DESCRIPTION	VALUES
`ss`	Seconds. The seconds and milliseconds are optional if a time is specified.	00 to 59
`sss`	Milliseconds	00 to 999
`Z`	The value in this position can be one of the following. If the value is omitted, UTC time is used. - `Z` indicates UTC time. - `+hh:mm` indicates that the input time is the specified offset after UTC time. - `-hh:mm` indicates that the input time is the absolute value of the specified offset before UTC time.	

The string can include the date only, as in the following formats: `YYYY`, `YYYY-MM`, `YYYY-MM-DD`.

The ISO format does not support time zone names. You can use the `Z` position to specify an offset from UTC time. If you do not include a value in the `Z` position, UTC time is used.

You can specify midnight by using 00:00, or by using 24:00 on the previous day. The following two strings specify the same time: 2010-05-25T00:00 and 2010-05-24T24:00.

To return a date in ISO format, you can use the toISOString Method (Date).

Extended Years

An *extended year* has 6 digits instead of 4 digits, and is prefixed with a plus or minus sign. An example of an extended year is `+002010`, which is equivalent to `2010`. You can use an extended year to represent years before the year 0 or after 9999.

If you use the 6-digit format, a plus or minus sign must be present. When you use the 4-digit format, the sign must be absent. Therefore, `0000` and `+000000` are accepted, but `000000` and `-0001` cause an error. The extended year 0 is considered positive and therefore prefixed with a plus sign.

The toISOString Method (Date) always uses the extended year format for years that are before 0 and after 9999.

> **NOTE**
> Supported in the following document modes: Internet Explorer 9 standards, Internet Explorer 10 standards, and Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.
> Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards.

Other Date Formats

If a date string is not in the ISO format, JavaScript uses the following rules to parse it.

Short dates

- The format must follow the month/day/year order, for example "06/08/2010".
- Either "/" or "-" can be used as a separator.

Long dates

- The year, month, and day can be in any order. "June 8 2010" and "2010 June 8" are both valid.
- The year can have two or four digits. If the year has only two digits, it must be at least 70.
- Month and day names must have at least two characters. Two character names that are not unique are resolved to the last matching name. For example, "Ju" specifies July, not June.
- A day of the week is ignored if it is inconsistent with the rest of the supplied date. For example, "Tuesday November 9 1996" resolves to "Friday November 9 1996" because Friday is the correct day of the week for that date.

  Times

- Hours, minutes, and seconds are separated by colons. However, some of the parts can be omitted. The following are valid: "10:", "10:11", and "10:11:12".
- If PM is specified and the specified hour is at least 13, NaN is returned. For example, "23:15 PM" returns NaN.

  General

- A string that contains an invalid date returns NaN. For example, a string that contains two years or two months returns NaN.
- JavaScript supports all standard time zones, and Universal Coordinated Time (UTC) and Greenwich Mean Time (GMT). (The ISO format does not support time zones.)
- Text enclosed in parentheses is treated as a comment. The parentheses can be nested.
- Commas and spaces are treated as delimiters. Multiple delimiters are permitted.

Example

The following code displays the results of parsing different date and time strings.

```
document.writeln((new Date("2010")).toUTCString());

document.writeln((new Date("2010-06")).toUTCString());

document.writeln((new Date("2010-06-09")).toUTCString());

 // Specifies Z, which indicates UTC time.
document.writeln((new Date("2010-06-09T15:20:00Z")).toUTCString());

 // Specifies -07:00 offset, which is equivalent to Pacific Daylight time.
document.writeln((new Date("2010-06-09T15:20:00-07:00")).toGMTString());

// Specifies a non-ISO Long date.
document.writeln((new Date("June 9, 2010")).toUTCString());

// Specifies a non-ISO Long date.
document.writeln((new Date("2010 June 9")).toUTCString());

// Specifies a non-ISO Short date and time.
document.writeln((new Date("6/9/2010 3:20 pm")).toUTCString());

// Output:
// Fri, 1 Jan 2010 00:00:00 UTC
// Tue, 1 Jun 2010 00:00:00 UTC
// Wed, 9 Jun 2010 00:00:00 UTC
// Wed, 9 Jun 2010 15:20:00 UTC
// Wed, 9 Jun 2010 22:20:00 UTC
// Wed, 9 Jun 2010 07:00:00 UTC
// Wed, 9 Jun 2010 07:00:00 UTC
// Wed, 9 Jun 2010 22:20:00 UTC
```

Where local times are specified, the result will vary depending on the time zone.

IMPORTANT

ISO Date Format is not supported in Internet Explorer 8 standards mode and Quirks mode.

See Also

Date Object
Date.parse Function

Displaying Text in a Webpage (JavaScript)

10/18/2017 • 2 min to read • Edit Online

There are a number of ways to display text in a webpage. Each has advantages and disadvantages and supports specific uses.

Displaying Text

- The recommended way to display text is to create an element and write to its textContent property.

```
<div id="textDiv"></div>
<script type="text/javascript">
    var div = document.getElementById("textDiv");
    div.textContent = "my text";
    var text = div.textContent;
</script>
```

  In this example, the value of `text` is "my text". However, the value resulting from getting or setting the `textContent` property on a parent node might include text content from the node's children. The following example shows that the `textContent` that is set on a child node is included in the value of `textContent` of the parent node:

```
<div id="textDiv">
    <div id="nested"></div>
</div>
<script type="text/javascript">
    var div = document.getElementById("textDiv");
    var nestedDiv = document.getElementById("nested");
    nestedDiv.textContent = "nested";

    var text = "[" + div.textContent + "]";
</script>
```

  In this example, the value of `text` is "[nested]".

- You can also create an element and write to its innerHTML or innerText properties. Setting these properties affects only the text in the element itself, not in its children. However, these properties also have some disadvantages:
  - The innerText property doesn't work in all browsers, so you might want to avoid it for reasons of compatibility.
  - The innerText property is affected by CSS styles, and doesn't appear if the element is hidden.
  - The innerHTML property gets and sets both nested nodes and text. If it isn't secured, it could be used for script-injection attacks. In addition, setting it to text without HTML tags removes all previously set nodes.
- You can use the `document.write` method without having to create an element. However, using this method causes the entire web page to be cleared, which might not be what you want.

  The following example shows one of the disadvantages of using `document.write`. The script is intended to display the time every 5 seconds, but it shows the time just twice. By the time `document.write` is called the

second time, the page has finished loading, and `document.write` then clears the entire page (it calls `document.open`). At this point, the `ShowTime` function no longer exists.

```
<!DOCTYPE html>
<html>
<head>
<script type="text/javascript">
    function ShowTime()
    {
        var dt = new Date();
        document.write(dt.toTimeString());
        // var elem = document.getElementById("divElem");
        // elem.textContent = dt.toTimeString();
        window.setTimeout("ShowTime();", 5000);
    }
</script>
</head>

<body>
<script type="text/javascript">
    ShowTime();
</script>
<div id="myDiv"></div>
</body>
</html>
```

To fix the preceding code, remove the line of code that contains `document.write` and uncomment the two commented lines of code that follow.

- You can also use an `alert``prompt`, or `confirm` function, which displays a message in a pop-up window. In most cases it's not a good idea to use a pop-up window in a web browser. Most modern browsers have settings that automatically block pop-up windows, so your message might not be seen. Moreover, it is possible to enter an infinite loop when you use pop-up windows, which makes it impossible for the user to close the web page in the usual way.

Advanced JavaScript

10/18/2017 • 1 min to read • Edit Online

These sections explain advanced JavaScript functionality, such as recursion, arrays, troubleshooting, and so on.

In This Section

Using Constructors to Create Objects
Explains how to write constructors as a way to instantiate objects.

Prototypes and Prototype Inheritance
Explains how to use prototypes to create derived objects.

Data Properties and Accessor Properties
Explains the difference between data properties and accessor properties.

Recursion
Explains how JavaScript uses recursion and looping.

Variable Scope
Describes the distinction between global and local scopes.

Copying, Passing, and Comparing Data
Explains the concept of passing by value or by reference.

Using Arrays
Describes the concept of arrays in JavaScript.

Typed Arrays
Describes typed arrays.

Collections
Describes the collection objects.

Iterators and Generators
Describes how to implement custom iterators on iterable objects.

Special Characters
Lists the characters that allow you to include in strings characters you cannot type directly.

Template Strings
Describes how to construct string literals that can include embedded expressions.

Using the bind method
Describes how to preserve the evaluation of the `this` object for functions that execute in another context.

Managing event listeners
Describes how to avoid memory leaks when using event listeners.

Troubleshooting Your Scripts
Explains how to find common script errors.

Conditional Compilation
Describes the statements that JavaScript uses compile depending on specific conditions.

Conditional Compilation Variables

Lists the variables that are available for conditional compilation.

Strict Mode

Explains the use of strict mode. Strict mode is not supported in versions of Internet Explorer earlier than Internet Explorer 10.

Using Constructors to Define Types

10/18/2017 • 1 min to read • Edit Online

A constructor is a function that instantiates a particular type of Object. You invoke a constructor with the **new** keyword. Here are a few examples of constructors with built-in JavaScript objects and custom objects.

Constructor Examples

```
// Creates a generic object.
var myObject = new Object();
// Creates a Date object.
var myBirthday = new Date(1961, 5, 10);
// Creates a user defined object.
var myCar = new Car();
```

The constructor function contains the **this** keyword, which is a reference to a newly created empty object. It initializes the new object by creating properties and giving them initial values. The constructor returns a reference to the object it constructed.

Writing Constructors

You can create objects using the **new** operator in conjunction with predefined constructor functions such as **Object()**, **Date()**, and **Function()**. You can also create custom constructor functions that define a set of properties and methods. Here is an example of a custom constructor.

```
function Circle (xPoint, yPoint, radius) {
    this.x = xPoint;  // The x component of the center of the circle.
    this.y = yPoint;  // The y component of the center of the circle.
    this.r = radius;  // The radius of the circle.
}
```

When you invoke the Circle constructor, you supply values for the circle's center point and the radius. You end up with a Circle object that contains three properties. Here is how you would instantiate a Circle object.

```
var aCircle = new Circle(5, 11, 99);
```

The type of all objects created with a custom constructor is `object`. There are only six types in JavaScript: `object`, `function`, `string`, `number`, `boolean`, and `undefined`. For more information, see typeof Operator

See Also

Using the bind method

Prototypes and Prototype Inheritance

3/15/2018 • 2 min to read • Edit Online

In JavaScript, a `prototype` is a property of functions and of objects that are created by constructor functions. The prototype of a function is an object. Its main use is when a function is used as a constructor.

```
function Vehicle(wheels, engine) {
    this.wheels = wheels;
    this.engine = engine;
}
```

In the example above, the prototype of the `Vehicle` function is the prototype of any object that is instantiated with the `Vehicle` constructor.

Using Prototypes to Add Properties and Methods

You can use the `prototype` property to add properties and methods to objects, even the ones that have already been created:

```
var testVehicle = new Vehicle(2, false);
Vehicle.prototype.color = "red";
var testColor = testVehicle.color;
```

The value of `testColor` is "red".

You can even add properties and methods to predefined objects. For example, you can define a `Trim` method on the `String` prototype object, and all the strings in your script will inherit the method.

```
String.prototype.trim = function()
{
    // Replace leading and trailing spaces with the empty string
    return this.replace(/(^\s*)|(\s*$)/g, "");
}
var s = "    leading and trailing spaces    ";
// Displays "    leading and trailing spaces     (35)"
window.alert(s + " (" + s.length + ")");
// Remove the leading and trailing spaces
s = s.trim();
// Displays "leading and trailing spaces (27)"
window.alert(s + " (" + s.length + ")");
```

Using Prototypes to Derive One Object from Another with Object.create

The prototype `Object` can be used to derive one object from another. For example, you can use the Object.create function to derive a new object `Bicycle` using the prototype of the `Vehicle` object we defined earlier (plus any new properties you need).

```
var bicycle = Object.create(Object.getPrototypeOf(Vehicle), {
    "pedals" :{value: true}
});
```

The `bicycle` object has the properties `wheels`, `engine`, `color`, and `pedals`, and its prototype is

`Vehicle.prototype`. The JavaScript engine finds the `pedals` property on `bicycle`, and it looks up the prototype chain to find the `wheels`, `engine`, and `color` properties on `Vehicle`.

Changing an Object's Prototype

In Internet Explorer 11, you can replace the internal prototype of an object or function with a new prototype by using the **proto** property. When you use this property, you inherit the properties and methods of the new prototype along with other properties and methods in its prototype chain.

> **WARNING**
>
> The `__proto__` property is a legacy feature. Use Object.getPrototypeOf instead.

The following example shows how you can change the prototype of an object. This example shows how the object's inherited properties change when you change its prototype.

```
function Friend() {
    this.demeanor = "happy";
}

function Foe() {
    this.demeanor = "suspicious";
}

var friend = new Friend();
var foe = new Foe();

var player = new Object();
player.__proto__ = foe;

friend.ally = "Tom";

if (console && console.log) {
    console.log(player.demeanor === "happy" );      // Returns false
    console.log(player.demeanor === "suspicious");  // Returns true
    console.log(player.ally === "Tom");             // Returns false
    // Turn the foe to a friend.
    player.__proto__ = friend;
    console.log(player.demeanor === "happy");       // Returns true
    console.log(player.demeanor === "suspicious");  // Returns false
    console.log(player.ally === "Tom");             // Returns true
}
```

Data Properties and Accessor Properties

10/18/2017 • 3 min to read • Edit Online

This section includes all the information you are likely to need about data properties and accessor properties.

Data Properties

A *data property* is a property that can get and set a value. Data properties contain the `value` and `writable` properties in their descriptors.

The following table lists the attributes for a data property descriptor.

DATA DESCRIPTOR ATTRIBUTE	DESCRIPTION	DEFAULT
`value`	The current value of the property.	`undefined`
`writable`	`true` or `false`. If `writable` is set to `true`, the property value can be modified.	`false`
`enumerable`	`true` or `false`. If `enumerable` is set to `true`, the property can be enumerated by a `for...in` statement.	`false`
`configurable`	`true` or `false`. If `configurable` is set to `true`, property attributes can be changed, and the property can be deleted.	`false`

If the descriptor does not have a `value`, `writable`, `get`, or `set` attribute, and the specified property name does not exist, a data property is added.

When the `configurable` attribute is `false` and `writable` is `true`, the `value` and `writable` attributes can be changed.

Data Properties Added Without Using defineProperty

If you add a data property without using the `Object.defineProperty`, `Object.defineProperties`, or `Object.create` functions, the `writable`, `enumerable`, and `configurable` attributes are all set to `true`. After the property is added, you can modify it by using the `Object.defineProperty` function.

You can use the following ways to add a data property:

- An assignment operator (=), as in `obj.color = "white";`
- An object literal, as in `obj = { color: "white", height: 5 };`
- A construction function, as described in Using Constructors to Define Types

Accessor Properties

An *accessor property* calls a user-provided function every time that the property value is set or retrieved. The descriptor for an accessor property contains a `get` attribute, a `set` attribute, or both.

The following table lists the attributes for an accessor property descriptor.

ACCESSOR DESCRIPTOR ATTRIBUTE	DESCRIPTION	DEFAULT
`get`	A function that returns the property value. The function has no parameters.	`undefined`
`set`	A function that sets the property value. It has one parameter that contains the value to be assigned.	`undefined`
`enumerable`	`true` or `false`. If `enumerable` is set to `true`, the property can be enumerated by a `for...in` statement.	`false`
`configurable`	`true` or `false`. If `configurable` is set to `true`, property attributes can be changed, and the property can be deleted.	`false`

When a `get` accessor is undefined and an attempt is made to access the property value, the value `undefined` is returned. When a `set` accessor is undefined and an attempt is made to assign a value to the accessor property, nothing occurs.

Property Modifications

If the object already has a property with the specified name, the property attributes are modified. When you modify the property, attributes that are not specified in the descriptor remain the same.

If the `configurable` attribute of an existing property is `false`, the only allowed modification is changing the `writable` attribute from `true` to `false`.

You can change a data property to an accessor property, and vice-versa. If you do this, `configurable` and `enumerable` attributes that are not specified in the descriptor are preserved in the property. Other attributes that are not specified in the descriptor are set to their default values.

You can incrementally define configurable accessor properties by using multiple calls to the `Object.defineProperty` function. For example, one `Object.defineProperty` call might define only a `get` accessor. A later call on the same property name might define a `set` accessor. The property would then have both a `get` accessor and `set` accessor.

To obtain a descriptor object that applies to an existing property, you can use the Object.getOwnPropertyDescriptor Function.

You can use the Object.seal Function and the Object.freeze Function to prevent the modification of property attributes.

Recursion (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Recursion is an important programming technique, in which a function calls itself.

An Example of Recursion

One example is the calculation of factorials. The factorial of a number *n* is calculated by multiplying 1 * 2 * 3 *... *n*. The following example shows how to calculate factorials iteratively, that is, by using a `while` loop in which the result is calculated.

```
function factorial(num)
{
    // If the number is less than 0, reject it.
    if (num < 0) {
        return -1;
    }
    // If the number is 0, its factorial is 1.
    else if (num == 0) {
        return 1;
    }
    var tmp = num;
    while (num-- > 2) {
        tmp *= num;
    }
    return tmp;
}

var result = factorial(8);
document.write(result);

// Output: 40320
```

You can make the example recursive very simply. Instead of using a `while` loop to calculate the value, you can simply call `factorial` again, passing in the next lowest value. The recursion stops when the value is 1.

```
function factorial(num)
{
    // If the number is less than 0, reject it.
    if (num < 0) {
        return -1;
    }
    // If the number is 0, its factorial is 1.
    else if (num == 0) {
        return 1;
    }
    // Otherwise, call this recursive procedure again.
    else {
        return (num * factorial(num - 1));
    }
}

var result = factorial(8);
document.write(result);

// Output: 40320
```

Variable Scope (JavaScript)

10/18/2017 • 3 min to read • Edit Online

JavaScript has two scopes: global and local. A variable that is declared outside a function definition is a global variable, and its value is accessible and modifiable throughout your program. A variable that is declared inside a function definition is local. It is created and destroyed every time the function is executed, and it cannot be accessed by any code outside the function. JavaScript does not support block scope (in which a set of braces `{. . .}` defines a new scope), except in the special case of block-scoped variables.

Scope in JavaScript

A local variable can have the same name as a global variable, but it is entirely separate; changing the value of one variable has no effect on the other. Only the local version has meaning inside the function in which it is declared.

```
// Global definition of aCentaur.
var aCentaur = "a horse with rider,";

// A local aCentaur variable is declared in this function.
function antiquities(){

   var aCentaur = "A centaur is probably a mounted Scythian warrior";
}

antiquities();
aCentaur += " as seen from a distance by a naive innocent.";

document.write(aCentaur);

// Output: "a horse with rider, as seen from a distance by a naive innocent."
```

In JavaScript, variables are evaluated as if they were declared at the beginning of the scope they exist in. Sometimes this results in unexpected behavior, as shown here.

```
var aNumber = 100;
tweak();

function tweak(){

    // This prints "undefined", because aNumber is also defined locally below.
    document.write(aNumber);

    if (false)
    {
        var aNumber = 123;
    }
}
```

When JavaScript executes a function, it first looks for all variable declarations, for example, `var someVariable;`. It creates the variables with an initial value of `undefined`. If a variable is declared with a value, for example, `var someVariable = "something";`, then it still initially has the value `undefined` and takes on the declared value only when the line that contains the declaration is executed.

JavaScript processes all variable declarations before executing any code, whether the declaration is inside a conditional block or other construct. Once JavaScript has found all the variables, it executes the code in the

function. If a variable is implicitly declared inside a function - that is, if it appears on the left side of an assignment expression but has not been declared with `var` - it is created as a global variable.

In JavaScript, an inner (nested) function stores references to the local variables that are present in the same scope as the function itself, even after the function returns. This set of references is called a closure. In the following example, the second call to the inner function outputs the same message ("Hello Bill") as the first call, because the input parameter for the outer function, `name`, is a local variable that is stored in the closure for the inner function.

```
function send(name) {
    // Local variable 'name' is stored in the closure
    // for the inner function.
    return function () {
        sendHi(name);
    }
}

function sendHi(msg) {
    console.log('Hello ' + msg);
}

var func = send('Bill');
func();
// Output:
// Hello Bill
sendHi('Pete');
// Output:
// Hello Pete
func();
// Output:
// Hello Bill
```

Block-scoped variables

Internet Explorer 11 introduces support for let and const, which are block-scoped variables. For these variables, the braces `{. . .}` define a new scope. When you set one of these variables to a particular value, the value applies only to the scope in which it is set.

The following example illustrates the use of `let` and block-scoping.

> **NOTE**
>
> The following code is supported in Internet Explorer 11 standards mode and later.

```
let x = 10;
var y = 10;
{
    let x = 5;
    var y = 5;
    {
        let x = 2;
        var y = 2;
        document.write("x: " + x + "<br/>");
        document.write("y: " + y + "<br/>");
        // Output:
        // x: 2
        // y: 2
    }
    document.write("x: " + x + "<br/>");
    document.write("y: " + y + "<br/>");
    // Output:
    // x: 5
    // y: 2
}

document.write("x: " + x + "<br/>");
document.write("y: " + y + "<br/>");
// Output:
// x: 10
// y: 2
```

Copying, Passing, and Comparing Data (JavaScript)

10/18/2017 • 3 min to read • Edit Online

In JavaScript, how data is handled depends on its data type.

By Value vs. By Reference

Numbers and Boolean values (**true** and **false**) are copied, passed, and compared *by value*. When you copy or pass by value, you allocate a space in computer memory and copy the value of the original into it. If you then change the original, the copy is not affected (and vice versa), because the two are separate entities.

Objects, arrays, and functions are copied, passed, and compared *by reference*. When you copy or pass by reference, you essentially create a pointer to the original item, and use the pointer as if it were a copy. If you then change the original, you change both the original and the copy (and vice versa). There is really only one entity; the "copy" is not actually a copy, it's just another reference to the data.

When comparing by reference, the two variables must refer to exactly the same entity for the comparison to succeed. For example, two distinct **Array** objects will always compare as unequal, even if they contain the same elements. One of the variables must be a reference to the other one for the comparison to succeed. To check if two Arrays hold the same elements, compare the results of the **toString()** method.

Last, strings are copied and passed by reference, but are compared by value. Note that if you have two **String** objects (created with **new** String("something")), they are compared by reference, but if one or both of the values is a string value, they are compared by value.

> **NOTE**
>
> Because of the way the ASCII and ANSI character sets are constructed, capital letters precede lowercase ones in sequence order. For example, "Zoo" compares as *less* than "aardvark." You can call **toUpperCase()** or **toLowerCase()** on both strings if you want to perform a case-insensitive match.

Passing Parameters to Functions

When you pass a parameter to a function by value, you are making a separate copy of that parameter, a copy that exists only inside the function. Even though objects and arrays are passed by reference, if you directly overwrite them with a new value in the function, the new value is not reflected outside the function. Only changes to properties of objects, or elements of arrays, are visible outside the function.

For example (using the Internet Explorer object model):

```
// This clobbers (over-writes) its parameter, so the change
// is not reflected in the calling code.
function Clobber(param)
{
    // clobber the parameter; this will not be seen in
    // the calling code
    param = new Object();
    param.message = "This will not work";
}

// This modifies a property of the parameter, which
// can be seen in the calling code.
function Update(param)
{
    // Modify the property of the object; this will be seen
    // in the calling code.
    param.message = "I was changed";
}

// Create an object, and give it a property.
var obj = new Object();
obj.message = "This is the original";

// Call Clobber, and print obj.message. Note that it hasn't changed.
Clobber(obj);
window.alert(obj.message); // Still displays "This is the original".

// Call Update, and print obj.message. Note that is has changed.
Update(obj);
window.alert(obj.message); // Displays "I was changed".
```

Testing Data

When you perform a test by value, you compare two distinct items to see whether they are equal to each other. Usually, this comparison is performed on a byte-by-byte basis. When you test by reference, you are checking to see whether two items are pointers to a single original item. If they are, then they compare as equal; if not, even if they contain the exact same values, byte-for-byte, they compare as unequal.

Copying and passing strings by reference saves memory; but because you cannot change strings once they are created, it becomes possible to compare them by value. This lets you test whether two strings have the same content even if one was generated entirely separately from the other.

See Also

Calculating Dates and Times (JavaScript)

Using Arrays (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Arrays in JavaScript are *sparse*. That is, if you have an array with three elements that are numbered 0, 1, and 2, you can create element 50 without worrying about elements 3 through 49. If the array has an automatic length variable (see Intrinsic Objects for an explanation of automatic monitoring of array length), the length variable is set to 51, rather than to 4. You can create arrays in which there are no gaps in the numbering of elements, but you are not required to do so.

In JavaScript, objects and arrays are almost identical to each other. The two main differences are that non-array objects do not have an automatic length property, and arrays do not have the properties and methods of an object.

Addressing Arrays

You address arrays by using brackets ([]), as shown in the following example. The brackets enclose either a numeric value or an expression that evaluates to a whole number.

```
var entryNum = 5;

sample = new Array();

sample[1] = "Maple Street";
sample[entryNum] = 25;

document.write (sample[1]);
document.write (" ");
document.write (sample[entryNum]);

// Output: Maple Street 25
```

Objects as Associative Arrays

You generally use the dot operator "." to access an object's properties. For example,

```
myObject.aProperty
```

In this case the property name is an identifier. You can also access an object's properties by using the index operator ([]). In this case you are treating the object as an *associative array* in which data values are associated with strings. For example,

```
myObject["aProperty"] // Same as above.
```

The index operator is more commonly associated with accessing array elements. When it is used with objects, the index is a string literal that represents the property name.

Notice the important differences between the two ways of accessing object properties.

1. A property name treated like an identifier (the dot (.) syntax) cannot be manipulated like data.
2. A property name treated like an index (the braces ([]) syntax) can be manipulated like data.

This difference becomes useful when you do not know what the property names will be until runtime (for example, when you are constructing objects based on user input). To extract all the properties from an associative array, you must use the `for...in` loop.

Typed Arrays (JavaScript)

10/18/2017 • 1 min to read • Edit Online

You can use typed arrays to handle binary data from sources such as network protocols, binary file formats, and raw graphics buffers. Typed arrays can also be used to manage in-memory binary data with well-known byte layouts.

Example

The following code shows how to use an ArrayBuffer Object as the response of an XMLHttpRequest. You can manipulate the bytes in the response by using the different methods of the DataView Object, or by copying the bytes into the appropriate typed array.

> **TIP**
>
> For more information about using different response types with an `XmlHttpRequest`, see XMLHttpRequest.responseType, XMLHttpRequest enhancements, and Downloading different types of content (Windows Store apps).

```
...
<div id="xhrDiv"></div>
...
var name = "http://www.microsoft.com";
var xhrDiv = document.getElementById("xhrDiv");

var req = new XMLHttpRequest();
req.open("GET", name, true);
req.responseType = "arraybuffer";
req.onreadystatechange = function () {
if (req.readyState == req.DONE) {
    var arrayResponse = req.response;
    var dataView = new DataView(arrayResponse);
    var ints = new Uint32Array(dataView.byteLength / 4);

    xhrDiv.style.backgroundColor = "#00FF00";
    xhrDiv.innerText = "Array is " + ints.length + "uints long";
    }
}
req.send();
```

ArrayBuffer

An ArrayBuffer Object represents a buffer of raw data that is used to store data for the different typed arrays. You cannot read from or write to an `ArrayBuffer`, but you can pass it to a typed array or DataView Object to interpret the raw buffer. You can use an `ArrayBuffer` to store any kind of data (or mixed types of data).

DataView

You can use a DataView Object to read and write the different kinds of binary data to any location in the `ArrayBuffer`.

Typed Arrays

The typed array types represent views of an ArrayBuffer Object that can be indexed and manipulated. All array types are of fixed length.

Name	Size (in bytes)	Description
Int8Array Object	1	Eight-bit two's complement signed integer
Uint8Array Object	1	Eight-bit unsigned integer
Int16Array Object	2	Sixteen-bit two's complement signed integer
Uint16Array Object	2	Sixteen-bit unsigned integer
Int32Array Object	4	Thirty-two-bit two's complement signed integer
Uint32Array Object	4	Thirty-two-bit unsigned integer
Float32Array Object	4	Thirty-two-bit IEEE floating point
Float64Array Object	8	Sixty-four-bit IEEE floating point

Collections (JavaScript)

1/12/2018 • 2 min to read • Edit Online

You can use the collection objects Map, Set, and WeakMap to store values and objects. These objects provide convenient methods for adding and retrieving members by using either a key or a value instead of an index. To access members of a collection by using an index, use an `Array` object. For more information, see Using Arrays.

Caution

`Map`, `Set`, and `WeakMap` are not supported in browser versions before Internet Explorer 11. For more information about version support, see Version Information.

Using collections

The `Map` and `WeakMap` objects store key/value pairs and enable you to add, retrieve, and remove members by using the key. The key and the value may be of any type. The `Set` object stores values of any type.

The `Map` and `Set` objects enable you to enumerate collection members by using the `forEach` method and to check the size of the collection by using the `size` method. The `WeakMap` object, in contrast, is not enumerable. For this collection, the key references are held weakly. Use `WeakMap` if you want the garbage collector to determine whether the app has to retain each member of the collection in memory. For example, this may be useful in caching scenarios where cached objects are very large and you don't want to hold objects in memory unnecessarily. In some scenarios, you can use this object to prevent memory leaks.

The following example shows how to use the `Map` object. In this example, you access members by using both `get` and `forEach`. The callback function in `forEach` can take up to three parameters, which provide the value of the current collection element, the key of the current element, and the collection object itself.

```
var m = new Map();
m.set(1, "black");
m.set(2, "red");
m.set("colors", 2);
m.set({x:1}, 3);

document.write(m.get(2));
document.write("<br />");

m.forEach(function (value, key, mapObj) {
    document.write(value.toString() + "<br />");
});

// Output:
// red

// black
// red
// 2
// 3
```

The use of `WeakMap` is similar to `Map`, except that you can retrieve members only by using `get`. For an example, see the WeakMap object.

The following example shows how to use the `Set` object. In this example, the callback function takes one parameter, which is the value of the current collection element.

```
var s = new Set();
s.add("Thomas Jefferson");
s.add(1776);
s.add("founding father");

s.forEach(function (value) {
    document.write(value.toString() + ", ");
});

// Output:
// Thomas Jefferson, 1776, founding father,
```

See Also

Advanced JavaScript

Iterators and Generators (JavaScript)

4/23/2018 • 4 min to read • Edit Online

An iterator is an object that is used to traverse a container object like a list. In JavaScript, an iterator object is not a distinct built-in object, but is an object that implements a `next` method to access the next item in the container object.

In Microsoft Edge, you can create your own custom iterators. However, it is generally much easier to use generators, which greatly simplify the creation of iterators. Generators are a type of function that is a factory for iterators. To create a custom iterator using a generator function, see Generators.

Caution

Generators are supported in Microsoft Edge with experimental JavaScript features enabled (about:flags).

Iterators

The implementation of a JavaScript iterator involves two or three objects that conform to specific interfaces:

- Iterable interface
- Iterator interface
- IteratorResult interface

  By using these interfaces, you can create custom iterators. This allows you to traverse an iterable object using the `for…of` statement.

Iterable interface

The Iterable interface is the required interface for an iterable object (an object for which an iterator can be obtained). For example, `C` in `for (let e of C)` must implement the Iterable interface.

An iterable object must provide the Symbol.iterator method, which returns an iterator.

```
obj[Symbol.iterator] = function() { return iterObj; }
```

This property must be a function that accepts no arguments and returns an object (`iterObject`) that conforms to the `Iterator` interface.

Many built-in types, including arrays, are now iterable. The `for…of` loop consumes an iterable object. (However, not all built-in iterables are iterators. For example, an Array object is not an iterator itself, but it is iterable, whereas an ArrayIterator is also iterable.)

Iterator interface

The object returned by the Symbol.iterator method must implement the `next` method. The `next` method has the following syntax.

```
iterObj.next() = function() { return iterResultObj; };
```

The `next` method is a function that returns a value. The function returns an object (`iterResultObj`) that conforms to the `IteratorResult` interface. If a previous call to the `next` method of an iterator returned an `IteratorResult` object whose `done` property is true, then iteration is terminated and the `next` method is not called again.

Iterators may also include a `return` method to ensure that the iterator is disposed properly when the script is finished with it.

IteratorResult interface

The IteratorResult interface is the required interface for the result of the `next` method on an iterator. The object returned by `next` must provide a `done` and `value` property.

```
var iterResultObj = { done: true|false, value: value }
```

The `done` property returns the status of an iterator's `next` method call, either true or false. If the end of the iterator was reached, `done` returns true. If the end was not reached, `done` returns false and a value is available. If the `done` property (either its own or an inherited property) does not exist, the result of `done` is treated as false.

If `done` is false, the `value` property returns the current iteration element value. If `done` is true, this is the return value of the iterator, if a return value is provided. If the iterator does not have a return value, `value` is undefined. In that case, the `value` property may be absent from the conforming object if it does not inherit an explicit value property.

Generators

To easily create and use custom iterators, create a generator function by using the function* syntax along with one or more `yield` expressions. The generator function returns an iterator (that is, a generator), which enables the generator function body to execute. The function executes to the next `yield` or `return` statement.

Call the `next` method of the iterator to return the next value from the generator function.

The following example shows a generator that returns an iterator for a string object.

```
function* stringIter() {
    var str = "bobsyouruncle";
    var idx = 0;
    while(idx < str.length)
        yield str[idx++];
}

var si = stringIter();

console.log(si.next().value);
console.log(si.next().value);
console.log(si.next().value);

// Output:
// b
// o
// b
```

In a generator, the yield operand expression terminates the call to `next` and returns an `IteratorResult` object with two properties, `done` (`done=false`) and `value` (`value=operand`). `operand` is optional and if left absent then its value is undefined.

In a generator, a `return` statement terminates the generator by returning an `IteratorResult` with `done=true` along with the optional operand result for the value property.

You can also use a `yield*` expression in place of `yield` to delegate to another generator or to another iterable object, such as an array or string.

If you append the following code to the preceding example, `yield*` delegates to the `stringIter` generator.

```
function* strIter() {
    yield "jo";
    yield* stringIter();
}

var si2 = strIter();

console.log(si2.next().value);
console.log(si2.next().value);
console.log(si2.next().value);
console.log(si2.next().value);

// Output:
// jo
// b
// o
// b
```

You can also create more advanced generators by passing an argument to `next` and using the argument to modify the state of the generator. `next` becomes the result value of the previously executed `yield` expression. In the following example, when you pass a value of 100 to the `next` method, you reset the generator's internal index value.

```
function* strIter() {
    var str = "jobob";
    var idx = 0;
    while(idx < str.length) {
        var modify = yield str[idx++];
        if(modify == 100) {
            idx = 0;
        }
    }
}

var si3 = strIter();

console.log(si3.next().value);
console.log(si3.next().value);
console.log(si3.next().value);
console.log(si3.next(100).value);

// Output:
// j
// o
// b
// j
```

Other advanced generators may call the generator's `throw` method. The thrown error appears to get thrown at the point where the generator is paused (before the next `yield` statement).

Special Characters (JavaScript)

10/18/2017 • 3 min to read • Edit Online

JavaScript provides escape sequences that you can include in strings to create characters that you cannot type directly.

Remarks

A string value is a series of zero or more Unicode characters (letters, digits, and other characters). String literals are enclosed in matching pairs of single or double quotation marks. Double quotation marks can be contained in a string that is enclosed in single quotation marks. Single quotation marks can be contained in a string that is enclosed in double quotation marks.

Each character in a string literal can be represented by an escape sequence. An escape sequence starts with a backslash (\) that informs the JavaScript interpreter that the next character is a special character.

You can specify a Unicode character by using the \u*hhhh* escape sequence, where *hhhh* is a four-digit hexadecimal number. A Unicode escape sequence can represent any 16-bit character. For additional information, see Unicode Code Point Escape Sequences.

You can use a single-character escape sequence for some characters. For example, \t specifies a tab character.

Escape Sequences

The following table lists a few examples of escape sequences for common characters.

UNICODE CHARACTER VALUE	ESCAPE SEQUENCE	MEANING	CATEGORY
\u0008	\b	Backspace	
\u0009	\t	Tab	White space
\u000A	\n	Line feed (new line)	Line terminator
\u000B	\v (See note after this table.)	Vertical tab	White space
\u000C	\f	Form feed	White space
\u000D	\r	Carriage return	Line terminator
\u0020		Space	White space
\u0022	\"	Double quotation mark (")	
\u0027	\'	Single quotation mark (')	
\u005C	\\Backslash (\)		
\u00A0		Nonbreaking space	White space

UNICODE CHARACTER VALUE	ESCAPE SEQUENCE	MEANING	CATEGORY
\u2028		Line separator	Line terminator
\u2029		Paragraph separator	Line terminator
\uFEFF		Byte order mark	White space

The Category column specifies whether the character is a white space or line terminator character. The trim Method (String) removes leading and trailing white space and line terminator characters from a string.

The backslash itself is used as the escape character. Therefore, you cannot directly type one in your script. If you want to write a backslash, you must type two of them together (\\).

> **NOTE**
> Starting in Internet Explorer 9 standards mode, Internet Explorer 10 standards mode, Internet Explorer 11 standards mode, and Windows Store apps, you cannot identify the browser as Internet Explorer by testing for the equivalence of the vertical tab (\v) and the "v". In earlier versions, the expression `"\v" === "v"` returns `true`. In Internet Explorer 9 standards mode, Internet Explorer 10 standards mode, Internet Explorer 11 standards mode, and Windows Store apps, the expression returns `false`.

Example

Description

The following example demonstrates the \\ and \' escape sequences.

Code

```
document.write('The image path is C:\\webstuff\\mypage\\gifs\\garden.gif.');
document.write ("<br />");
document.write('The caption reads, "After the snow of \'97. Grandma\'s house is covered."');
```

Unicode Code Point Escape Sequences

In Microsoft Edge, Unicode is fully supported. The most common Unicode code points are represented by four hexadecimal digits, such as /u0009 for a tab character. Astral code points, which include all symbols that require more than four hexadecimal digits, are now supported in a simplified format. By using the format "\u{*codepoint*}", the full Unicode character set can be represented in a literal format. For example, to represent the symbol "吉", you can use the following format: "\u{20BB7}".

In the following code example, the strings are all equivalent. (\uD842\uDFB7 is the workaround method to specify this symbol in previous versions.)

```
"\u{20BB7}"=="吉"=="\uD842\uDFB7"
```

RegExp now includes a /u flag to enable full support for astral code points. For example, in the following code example, the /u flag in the regular expression enables matching astral code points (the period matches any character in the provided string).

```
"吉".match(/./u)[0].length == 2
```

The /u flag enables parsing of the new format, \u{codepoint}), as a Unicode escape sequence. This is necessary because \u{xxxxx} without the /u flag has a different meaning in a regular expression.

> **NOTE**
>
> For astral code points, length is always 2. This matches behavior in previous versions.

The String object now includes two new methods, String.codePointAt and String.fromCodePoint, to support astral code points. For example, you can use codePointAt to return the code point equivalent for the "吉" symbol.

```
"吉".codePointAt(0) == 0x20BB7
```

You can also iterate code points using the `for...of` statement.

```
for(var c of "吉") {
    console.log(c);
}
```

See Also

String.fromCharCode Function

Template Strings (JavaScript)

10/18/2017 • 1 min to read • Edit Online

In Microsoft Edge, you can use template strings to construct string literals with embedded expressions. Template strings also provide built-in support for multi-line strings.

To construct a template string, use the grave accent (also called a back-tick) (`) to enclose the string instead of single or double quotes. The following code example shows a simple template string.

0
Template strings can include line breaks without requiring use of the linefeed character (\n).

1
The $ character is used to specify placeholders within a template string. The syntax is ${*expression*}, where *expression* is any JavaScript expression such as a variable or function, as shown in the following example.

2
Tagged template functions, which allow you to modify the value of a template string using a function that is invoked with arguments from the template string. The first argument is an array of string literals, delimited by the template string expressions that it contains, and the second argument is an array (a Rest parameter) that contains the values of the template string expressions.

In the following example, the tagged template function, buildURL is used to construct a URL. The syntax is to use the function name followed immediately by the template string.

3
If you need access to the raw string values passed in, the first argument passed to the tagged template function supports a `raw` property that returns the raw string form of the passed in strings.

```
function buildURL(strArray, ...valArray) {
    console.log(strArray.raw);
}

var lang = "en-us";
var a = "library";
var b = "dn771551.aspx";

// Call the tagged template function.
var url = buildURL`http://msdn.microsoft.com/${lang}/${a}/${b}`;

// Ouput:
// http://msdn.microsoft.com/
// /
// en-us
// library
```

NOTE

You can also use the String.raw function to return the raw string form of a template string.

See Also

JavaScript Language Reference

Using the bind method (JavaScript)

2/3/2018 • 1 min to read • Edit Online

The JavaScript `bind` method has several uses. Typically, it is used to preserve execution context for a function that executes in another context. `bind` creates a new function that has the same body as the original function. The first argument passed to `bind` specifies the value of the `this` keyword in the bound function. You can also pass additional, optional arguments to `bind`. For examples of additional uses, see the bind Method (Function). For an example of using `bind` to partially apply functions, see Async programming patterns and tips in Hilo JavaScript (Windows Store).

Preserving the execution context using bind

The `bind` function is often used when adding event listeners. In the following code example, `bind` is used to preserve the context of the current object (`DataObject`). The data object is passed to `bind` by using the `this` keyword, which provides access to data object properties and functions when the event handler (`dataReadyHandler`) runs. To illustrate how `bind` works, this code creates a custom event.

```
var data;

var dataReadyEvent = document.createEvent("Event");
dataReadyEvent.initEvent("dataReady", true, false);

function DataObject() {
    this.name = "Data Object";
    this.data = function () {
        return data;
    }
    this.onDataCompleted = dataReadyHandler;
    document.addEventListener('dataReady', this.onDataCompleted.bind(this));
    // To see the result of not using bind, comment out the preceding line,
    // and uncomment the following line of code.
    // document.addEventListener('dataReady', this.onDataCompleted);
}
function dataReadyHandler() {
    if (console && console.log) {
        console.log("Data object property value: " + this.name);
        console.log("Data object property value: " + this.data());
    }
}

setTimeout(function () {
    data = [0, 1, 2, 3];
    document.dispatchEvent(dataReadyEvent);
    }, 5000);

var dataObj = new DataObject();

// Output:
// Data Object
// 0,1,2,3
```

If you comment out the line of code that uses `bind`, uncomment the line of code that calls `addEventListener` without `bind`, and then rerun the code, the `dataReadyHandler` function will fail. For example, in `dataReadyHandler`, `this.name` will be undefined, and `this.data()` will result in an error because the `this` object no longer refers to the data object.

See Also

bind Method (Function)

Managing event listeners

10/18/2017 • 2 min to read • Edit Online

If the lifetime of a DOM element or object is different from the lifetime of its associated event listener, you might need to use the removeEventListenermethod to avoid memory leaks.

Event listeners and object scope

The following code example shows code that might result in a memory leak when the `dataObjFactory()` function is called. In this code, an event listener is registered for a data object each time a new data object is created. To make the current data object available in the `dataReady()` event handler, the bind Method (Function) is used in `addEventListener`.

```
var data;
var dataObj;

var dataReadyEvent = document.createEvent("Event");
dataReadyEvent.initEvent("dataReady", true, false);

function DataObject() {

    this.name = "Data Object";
    this.data = function () {
        return data;
    }
    this.onDataCompleted = dataReady;
    // this.handlerRef = this.onDataCompleted.bind(this);

    // document.addEventListener('dataReady', this.handlerRef);
    document.addEventListener('dataReady', this.onDataCompleted.bind(this));
}

// Runs when the data is available.
function dataReady() {
    if (console && console.log) {
        console.log("object value: " + this.name);
        console.log("object value: " + this.data());
    }
}

setTimeout(function () {
    // Generate data after a timeout period.
    data = [0, 1, 2, 3];
    document.dispatchEvent(dataReadyEvent);
}, 10000);

function dataObjFactory() {
     for (var x = 0; x < 100; x++) {
         if (dataObj) {
             // The following line of code has no effect.
             document.removeEventListener('dataReady', dataObj.onDataCompleted);
             dataObj = null;
         }
         dataObj = new DataObject();
     }
}

dataObjFactory();
```

The listener for each data object is registered with the `document` object, which has a different lifetime than the data objects. The registered event listener for each data object prevents it from being garbage collected as long as the `document` object remains in scope. (In some modern JavaScript design patterns, the document object will remain in scope for the lifetime of the web app.) To prevent a memory leak, `removeEventListener` is called in the `dataObjFactory` function. However, this code fails because `removeEventListener` has not been called on the bound version of the event handler that is returned by the `bind` function.

To fix this code and make the data objects available for garbage collection, you must first store a reference to the bound version of the event handler, as shown in this code, and then pass the stored reference to `addEventListener`. Here's the corrected code for `DataObject`:

```
function DataObject() {

    this.name = "Data Object";
    this.data = function () {
        return data;
    }
    this.onDataCompleted = dataReady;
    this.handlerRef = this.onDataCompleted.bind(this);

    document.addEventListener('dataReady', this.handlerRef);
    // document.addEventListener('dataReady', this.onDataCompleted.bind(this));
}
```

Finally, you need to call `removeEventListener` on the stored reference (`handlerRef`) of the bound function. Here's the corrected code for `dataObjFactory`:

```
function dataObjFactory() {
     for (var x = 0; x < 100; x++) {
         if (dataObj) {
             document.removeEventListener('dataReady', dataObj.handlerRef);
         }
         dataObj = new DataObject();
     }
 }
```

Now the call to `removeEventListener` works, and the unneeded data objects can be garbage collected even while the `document` object remains in scope.

Troubleshooting Your Scripts (JavaScript)

10/18/2017 • 2 min to read • Edit Online

There are places in any programming language that have surprises. For example, the `null` value in JavaScript does not behave the same as the `Null` value in the C or C++ languages.

Here are some of the trouble areas that you may run into as you write JavaScript scripts.

Syntax Errors

It is important to pay attention to detail when you write scripts. For example, strings must be enclosed in quotation marks.

Order of Script Interpretation

JavaScript interpretation is part of the your Web browser's HTML parsing process. If you place a script inside the <HEAD> tag in a document, it is interpreted before any part of the <BODY> tag. If you have objects that are created in the <BODY> tag, they do not exist at the time the <HEAD> is being parsed, and cannot be manipulated by the script.

> **NOTE**
>
> This behavior is specific to Internet Explorer. ASP and WSH have different execution models (as do other hosts).

Automatic Type Coercion

JavaScript is a loosely-typed language with automatic coercion. Even though values having different types are not equal, the expressions in the following example evaluate to **true**.

```
"100" == 100;
false == 0;
```

To check that both the type and value are the same, use the strict equality operator (===). The following both evaluate to false:

```
"100" === 100;
false === 0;
```

Operator Precedence

Operator precedence determines when an operation is performed during the evaluation of an expression. In the following example multiplication is performed before subtraction, even though the subtraction appears first in the expression.

```
theRadius = aPerimeterPoint - theCenterpoint * theCorrectionFactor;
```

Using for...in Loops with Objects

When you iterate through the properties of an object with a for...in loop, you cannot predict or control the order in which the fields of the object are assigned to the loop counter variable. Moreover, the order may be different in different implementations of the language.

with Keyword

The with statement is convenient for accessing properties that already exist in a specified object, but cannot be used to add properties to an object. To create new properties in an object, you must refer to the object specifically.

this Keyword

Although you use the `this` keyword inside the definition of an object to refer to the object itself, you cannot use `this` or similar keywords to refer to the currently executing function when that function is not an object definition. If the function is to be assigned to an object as a method, you can use the `this` keyword within the function to refer to the object.

Writing a Script That Writes a Script in Internet Explorer

The </SCRIPT> tag terminates the current script if the interpreter encounters it. To display "</SCRIPT>" itself, rewrite this as at least two strings, for example, "</SCR" and "IPT>", which you can then concatenate together in the statement that writes them out.

Implicit Window References in Internet Explorer

Because more than one window can be open at a time, any implicit window reference points to the current window. For other windows, you must use an explicit reference.

Conditional Compilation (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Conditional compilation allows the use of new JavaScript language features without sacrificing compatibility with older versions that do not support the features.

> **WARNING**
> Conditional compilation is supported in all versions of Internet Explorer prior to Internet Explorer 11. Starting with Internet Explorer 11 Standards mode, and in Windows 8.x Store apps, conditional compilation is not supported.

Statements

Conditional compilation is activated by using the `@cc_on` statement, or using an `@if` or `@set` statement. Some typical uses for conditional compilation include using new features in JavaScript, embedding debugging support into a script, and tracing code execution.

Always place conditional compilation code in comments, so that hosts (like Netscape Navigator) that do not support conditional compilation will ignore it. Here is an example.

```
/*@cc_on @*/
/*@if (@_jscript_version >= 4)
    alert("JavaScript version 4 or better");
    @else @*/
    alert("Conditional compilation not supported by this scripting engine.");
/*@end @*/
```

This example uses special comment delimiters that are used only if conditional compilation is activated by the `@cc_on` statement. Scripting engines that do not support conditional compilation see only the message that says conditional compilation is not supported.

Conditional Compilation Variables (JavaScript)

10/18/2017 • 1 min to read • Edit Online

The following predefined variables are available for conditional compilation. If a variable is not **true**, it is not defined and behaves as `NaN` when accessed.

> **WARNING**
> Conditional compilation is supported in all versions of Internet Explorer prior to Internet Explorer 11. Starting with Internet Explorer 11 Standards mode, and in Windows 8.x Store apps, conditional compilation is not supported.

Variables

VARIABLE	DESCRIPTION
@_win32	True if running on a Win32 system.
@_win16	True if running on a Win16 system.
@_mac	True if running on an Apple Macintosh system.
@_alpha	True if running on a DEC Alpha processor.
@_x86	True if running on an Intel processor.
@_mc680x0	True if running on a Motorola 680x0 processor.
@_PowerPC	True if running on a Motorola PowerPC processor.
@_jscript	Always true.
@_jscript_build	Contains the build number of the JavaScript scripting engine.
@_jscript_version	Contains the JavaScript version number in major.minor format.

Strict Mode (JavaScript)

10/18/2017 • 4 min to read • Edit Online

Strict mode is a way to introduce better error-checking into your code. When you use strict mode, you cannot, for example, use implicitly declared variables, or assign a value to a read-only property, or add a property to an object that is not extensible. The restrictions are listed in the Restrictions on Code in Strict Mode section later in this topic. For additional information on strict mode, see ECMAScript Language Specification, 5th edition.

> **WARNING**
>
> Strict mode is not supported in versions of Internet Explorer earlier than Internet Explorer 10.

Declaring Strict Mode

You can declare strict mode by adding `"use strict";` at the beginning of a file, a program, or a function. This kind of declaration is known as a *directive prologue*. The scope of a strict mode declaration depends on its context. If it is declared in a global context (outside the scope of a function), all the code in the program is in strict mode. If it is declared in a function, all the code in the function is in strict mode. For example, in the following example all the code is in strict mode, and the variable declaration outside the function causes the syntax error "Variable undefined in strict mode."

```
"use strict";
function testFunction(){
    var testvar = 4;
    return testvar;
}

// This causes a syntax error.
testvar = 5;
```

In the following example, only the code inside `testFunction` is in strict mode. The variable declaration outside the function does not cause a syntax error, but the declaration inside the function does.

```
function testFunction(){
    "use strict";
    // This causes a syntax error.
    testvar = 4;
    return testvar;
}
testvar = 5;
```

Restrictions on Code in Strict Mode

The following table lists the most important restrictions that apply in strict mode.

Language element	Restriction	Error	Example
Variable	Using a variable without declaring it.	SCRIPT5042: Variable undefined in strict mode	`testvar = 4;`

Read-only property	Writing to a read-only property.	SCRIPT5045: Assignment to read-only properties is not allowed in strict mode	`var testObj = Object.defineProperties({}, { prop1: { value: 10, writable: false // by default }, prop2: { get: function () { } } }); testObj.prop1 = 20; testObj.prop2 = 30;`
Non-extensible property	Adding a property to an object whose `extensible` attribute is set to `false`.	SCRIPT5046: Cannot create property for a non-extensible object	`var testObj = new Object(); Object.preventExtensions(testObj) testObj.name = "Bob";`
`delete`	Deleting a variable, a function, or an argument. Deleting a property whose `configurable` attribute is set to `false`.	SCRIPT1045: Calling delete on <expression>is not allowed in strict mode	`var testvar = 15; function testFunc() {}; delete testvar; delete testFunc; Object.defineProperty(testObj, "testvar", { value: 10, configurable: false }); delete testObj.testvar;`
Duplicating a property	Defining a property more than once in an object literal.	SCRIPT1046: Multiple definitions of a property not allowed in strict mode	`var testObj = { prop1: 10, prop2: 15, prop1: 20 };`
Duplicating a parameter name	Using a parameter name more than once in a function.	SCRIPT1038: Duplicate formal parameter names not allowed in strict mode	`function testFunc(param1, param1) { return 1; };`
Future reserved keywords	Using a future reserved keyword as a variable or function name.	SCRIPT1050: The use of a future reserved word for an identifier is invalid. The identifier name is reserved in strict mode.	- `implements` - `interface` - `package` - `private` - `protected` - `public` - `static` - `yield`
Octals	Assigning an octal value to a numeric literal, or attempting to use an escape on an octal value.	SCRIPT1039: Octal numeric literals and escape characters not allowed in strict mode	`var testoctal = 010; var testescape = \010;`
`this`	The value of `this` is not converted to the global object when it is `null` or `undefined`.		`function testFunc() { return this; } var testvar = testFunc();` In non-strict mode, the value of `testvar` is the global object, but in strict mode the value is `undefined`.

`eval` as an identifier	The string "eval" cannot be used as an identifier (variable or function name, parameter name, and so on).		`var eval = 10;`
Function declared inside a statement or a block	You cannot declare a function inside a statement or a block.	SCRIPT1047: In strict mode, function declarations cannot be nested inside a statement or block. They may only appear at the top level or directly inside a function body.	`var arr = [1, 2, 3, 4, 5]; var index = null; for (index in arr) { function myFunc() {}; }`

Variable declared inside an `eval` function	If a variable is declared inside an `eval` function, it cannot be used outside that function.	SCRIPT1041: Invalid usage of 'eval' in strict mode	`eval("var testvar = 10"); testvar = 15;`
			
			Indirect evaluation is possible, but you still cannot use a variable declared outside the `eval` function.
			
			`var indirectEval = eval; indirectEval("var testvar = 10;"); document.write(testVar);`
			
			This code causes an error SCRIPT5009: 'testVar' is undefined.

`Arguments` as an identifier	The string "arguments" cannot be used as an identifier (variable or function name, parameter name, and so on).	SCRIPT1042: Invalid usage of 'arguments' in strict mode	`var arguments = 10;`
`arguments` inside a function	You cannot change the values of members of the local `arguments` object.		`function testArgs(oneArg) { arguments[0] = 20; }` In non-strict mode, you can change the value of the `oneArg` parameter by changing the value of `arguments[0]`, so that the value of both `oneArg` and `arguments[0]` is 20. In strict mode, changing the value of `arguments[0]` does not affect the value of `oneArg`, because the `arguments` object is merely a local copy.
`arguments.callee`	Not allowed.		`function (testInt) { if (testInt-- == 0) return; arguments.callee(testInt--); }`
`with`	Not allowed.	SCRIPT1037: 'with' statements are not allowed in strict mode	`with (Math){ x = cos(3); y = tan(7); }`

JavaScript Reference

10/18/2017 • 1 min to read • Edit Online

These sections explain the elements that make up the JavaScript language.

JavaScript code in Internet Explorer can interact with HTML, CSS and the Document Object Model (DOM), which represents HTML and browser objects.

- For information about HTML, see HTML/XHTML Reference.
- For information about CSS, see Cascading Style Sheets.
- For information about the DOM, see Document Object Model (DOM).

  JavaScript code can be used in browser applications as well as Windows 8.x Store apps. Windows 8.x Store apps are supported in Windows 8 using Visual Studio 2012 and in Windows 8.1 using Visual Studio 2013.

- For information about JavaScript in Windows 8.x Store apps, see JavaScript roadmap.
- For information about HTML and CSS in Windows 8.x Store apps, see HTML/CSS for Windows Store apps.
- For information about Windows 8.x Store APIs, see API reference for Windows Runtime and Windows Library for JavaScript.

  The JavaScript editor in Visual Studio provides IntelliSense support. For more information, see JavaScript IntelliSense.

In This Section

Version Information
Provides a list of JavaScript language features and the Internet Explorer versions in which they were introduced.

Objects
Provides a list of objects with links to information about each object.

Constants
Provides a list of constants and links to information about each constant.

Properties
Provides a list of properties with links to information about each property.

Functions
Provides a list of functions with links to information about each function.

Methods
Provides a list of methods with links to information about each method.

Operators
Provides a list of operators with links to information about each operator.

Statements
Provides a list of statements with links to information about each statement.

JavaScript Directives
Includes links to directives.

Errors

Includes links to run-time and syntax errors.

JavaScript Reserved Words

Provides a list of reserved words. These words may not be used as identifiers.

JavaScript Future Reserved Words

Provides a list of future reserved words. These words may not be used as identifiers.

Related Sections

JavaScript Fundamentals

Provides information about how to use JavaScript.

HTML and DHTML Reference

Provides reference information about Dynamic HTML (DHTML) API .

JavaScript Version Information

3/2/2018 • 19 min to read • Edit Online

Different versions of JavaScript support different sets of JavaScript elements. Windows 8.x Store apps support a slightly different set of features from Internet Explorer.

> **IMPORTANT**
>
> A Windows 8.x Store app is a new type of application that runs on Windows 8 devices. To find out more about Windows 8.x Store apps, see What's a Windows Store app?

Standards mode (the mode used in all versions of Internet Explorer up to Internet Explorer 11 when there is a `<!doctype>` directive) supports a different set of elements than quirks mode (the mode used when there is no `<!doctype>` directive). For more information about versioning, see Defining Document Compatibility.

The table that follows shows the Internet Explorer document modes (and Store apps representing Windows 8.x Store and Windows Phone Store) that support specific language elements. Document modes that support a given element are shown with the letter **Y**, and document modes that do not support a given element are shown with the letter **N**.

> **IMPORTANT**
>
> Microsoft Edge (Edge browser in Windows 10) does not include support for legacy document modes. Support for Windows Phone Store apps starts with Windows Phone 8.1. Experimental features (about:flags) are indicated by "Exp."

The table contains summary info. For more specific information, see the documentation for the language element.

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
_proto_Property(Object)	N	N	N	N	Y	Y	v8(Win): N v8.1(Win): Y v8.1(Phone): Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
$1...$9 Properties (RegExp)	Y	Y	Y	Y	Y	Y	Y
0n Property	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STORE APPS
absFunction	Y	Y	Y	Y	Y	Y	Y
acosFunction	Y	Y	Y	Y	Y	Y	Y
acoshFunction	N	N	N	N	N	Y	v8.1: N v10: Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
ActiveXObject Object	Y	Y	Y	Y	Y	Y	N

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STORE APPS
Addition Assignment Operator (+=)	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
AdditionOperator(+)	Y	Y	Y	Y	Y	Y	Y
applyMethod	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
argumentsObject	Y	Y	Y	Y	Y	Y	Y
argumentsProperty	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STORE APPS
Array Object	Y	Y	Y	Y	Y	Y	Y
Array.from Function (Array)	N	N	N	N	N	N	v8.1: N v10: Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STORE APPS
Array.isArrayFunction	N	N	Y	Y	Y	Y	Y
Array.ofFunction(Array)	N	N	N	N	N	N	v8.1: N v10: Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
ArrayBufferObject	N	N	N	Y	Y	Y	Y
Functions	N	N	N	N	N	N	v8.1: N v10: Y
asinFunction	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
Object.assignFunction(Object)	N	N	N	N	N	N	v8.1: N v10: Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STORE APPS
Assignment Operator (=)	Y	Y	Y	Y	Y	Y	Y
atan Function	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STORE APPS
atan2 Function	Y	Y	Y	Y	Y	Y	Y
atEndMethod	Y	Y	Y	Y	Y	Y	N
bindMethod	N	N	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STORE APPS
Bitwise AND Assignment Operator (&=)	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STORE APPS
Bitwise AND Operator (&)	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
Bitwise Left Shift Operator (<<)	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STORE APPS
Bitwise NOT Operator (~)	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STORE APPS
Bitwise OR Assignment Operator (|=)	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
Bitwise OR Operator (|)	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STORE APPS
Bitwise Right Shift Operator (>>)	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STORE APPS
Bitwise YOR Assignment Operator (^=)	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STORE APPS
Bitwise YOR Operator (^)	Y	Y	Y	Y	Y	Y	Y
blinkMethod	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
boldMethod	Y	Y	Y	Y	Y	Y	Y
BooleanObject	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STORE APPS
breakStatement	Y	Y	Y	Y	Y	Y	Y
callMethod	Y	Y	Y	Y	Y	Y	Y
calleeProperty	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
callerProperty	Y	Y	Y	Y	Y	Y	Y
catchStatement	Y	Y	Y	Y	Y	Y	Y
ceilFunction	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
charAtMethod	Y	Y	Y	Y	Y	Y	Y
charCodeAtMethod	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
classStatement	N	N	N	N	N	Exp.	v8.1: N v10: Exp.
codePointAtMethod (String)	N	N	N	N	N	Y	v8.1: N v10: Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
Comma Operator (,)	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STORE APPS
// (Single-line Comment Statement)	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STORE APPS
/*..*/ (Multiline Comment Statement)	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
Comparison Operators	Y	Y	Y	Y	Y	Y	Y
compileMethod	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
concatMethod(Array)	Y	Y	Y	Y	Y	Y	Y
concatMethod(String)	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
Conditional Compilation	Y	Y	Y	Y	N	N	N

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
Conditional Compilation Variables	Y	Y	Y	Y	N	N	N

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STORE APPS
Conditional (Ternary) Operator (?:)	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
constructor Property	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STORE APPS
constStatement	N	N	N	N	Y	Y	v8 (Win): N v8.1 (Win): Y v8.1 (Phone): Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
continueStatement	Y	Y	Y	Y	Y	Y	Y
cosFunction	Y	Y	Y	Y	Y	Y	Y
createFunction	N	N	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
DataViewObject	N	N	N	Y	Y	Y	Y
DateObject	Y	Y	Y	Y	Y	Y	Y
DebugObject	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
Debug.setNonUserCodeExceptionsProperty	N	N	N	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
Debug.setNonUserCodeExceptionsProperty	N	N	N	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
debuggerStatement	Y	Y	Y	Y	Y	Y	Y
decodeURIFunction	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STORE APPS
decodeURIComponentFunction	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
Decrement Operator (--)	Y	Y	Y	Y	Y	Y	Y
Functions	N	N	N	N	N	Exp.	v8.1: N v10: Exp.

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
defineProperties Function	N	Y*	Y	Y	Y	Y	Y
defineProperty Function	N	Y*	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
deleteOperator	Y	Y	Y	Y	Y	Y	Y
descriptionProperty	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
dimensionsMethod	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
Division Assignment Operator (/=)	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STORE APPS
Division Operator (/)	Y	Y	Y	Y	Y	Y	Y
do...while Statement	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
EConstant	Y	Y	Y	Y	Y	Y	Y
encodeURIFunction	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STORE APPS
encodeURIComponentFunction	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
entriesMethod(Array)	N	N	N	N	N	N	v8.1: N v10: Y
EnumeratorObject	Y	Y	Y	Y	Y	Y	N

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
Number Constants	N	N	N	N	N	N	v8.1: N v10: Y
Equality Operator (==)	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
Error Object	Y	Y	Y	Y	Y	Y	Y
stackProperty (Error)	N	N	N	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STORE APPS
stackTraceLimitProperty (Error)	N	N	N	Y	Y	Y	Y
escapeFunction	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STORE APPS
evalFunction	Y	Y	Y	Y	Y	Y	Y
execMethod	Y	Y	Y	Y	Y	Y	Y
everyMethod	N	N	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STORE APPS
expFunction	Y	Y	Y	Y	Y	Y	Y
fillMethod(Array)	N	N	N	N	N	N	v8.1: N v10: Y
filterMethod	N	N	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
finally Statement	Y	Y	Y	Y	Y	Y	Y
findIndexMethod(Array)	N	N	N	N	N	N	v8.1: N v10: Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
fixedMethod	Y	Y	Y	Y	Y	Y	Y
Float32ArrayObject	N	N	N	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
Float64ArrayObject	N	N	N	Y	Y	Y	Y
floorFunction	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
fontcolorMethod	Y	Y	Y	Y	Y	Y	Y
fontsizeMethod	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STORE APPS
forStatement	Y	Y	Y	Y	Y	Y	Y
forEachMethod	N	N	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
for...in Statement	Y	Y	Y	Y	Y	Y	Y
for...of Statement	N	N	N	N	N	Y	v8.1: N v10: Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
freezeFunction	N	N	Y	Y	Y	Y	Y
fromCharCodeFunction	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
fromCodePoint Function	N	N	N	N	N	Y	v8.1: N v10: Y
Function Object	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STORE APPS
function Statement	Y	Y	Y	Y	Y	Y	Y
Generators	N	N	N	N	N	Exp.	v8.1: N v10: Exp.

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
getDateMethod	Y	Y	Y	Y	Y	Y	Y
getDayMethod	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STORE APPS
getFullYearMethod	Y	Y	Y	Y	Y	Y	Y
getHoursMethod	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
getItemMethod	Y	Y	Y	Y	Y	Y	Y
getMillisecondsMethod	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
getMinutesMethod	Y	Y	Y	Y	Y	Y	Y
getMonthMethod	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
GetObjectFunction	Y	Y	N	N	N	N	N

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
getOwnPropertyDescriptorFunction	N	Y*	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STORE APPS
getOwnPropertyNamesFunction	N	N	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
getPrototypeOfFunction	N	N	Y	Y	Y	Y	Y
getSecondsMethod	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
getTimeMethod	Y	Y	Y	Y	Y	Y	Y
getTimezoneOffsetMethod	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
getUTCDateMethod	Y	Y	Y	Y	Y	Y	Y
getUTCDayMethod	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
getUTCFullYearMethod	Y	Y	Y	Y	Y	Y	Y
getUTCHoursMethod	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STORE APPS
getUTCMillisecondsMethod	Y	Y	Y	Y	Y	Y	Y
getUTCMinutesMethod	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
getUTCMonthMethod	Y	Y	Y	Y	Y	Y	Y
getUTCSecondsMethod	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
getVarDateMethod	Y	Y	Y	Y	Y	Y	N
getYearMethod	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
Global Object	Y	Y	Y	Y	Y	Y	Y
globalProperty	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
Greater than Operator (>)	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
Greater than or equal to Operator (>=)	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STORE APPS
hasOwnProperty Method	Y	Y	Y	Y	Y	Y	Y
HTML Tag Methods	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
hypotFunction	N	N	N	N	N	Y	v8.1: N v10: Y
IdentityOperator(===)	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
if...elseStatement	Y	Y	Y	Y	Y	Y	Y
ignoreCaseProperty	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STORE APPS
imulFunction	N	N	N	N	N	Y	v8.1: N v10: Y
InOperator	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STORE APPS
includesMethod (String)	N	N	N	N	N	Y	v8.1: N v10: Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
Increment Operator (++)	Y	Y	Y	Y	Y	Y	Y
indexProperty	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
indexOfMethod(Array)	N	N	Y	Y	Y	Y	Y
indexOfMethod(String)	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
Inequality Operator (!=)	Y	Y	Y	Y	Y	Y	Y
Infinity Constant	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
inputProperty($_)	Y	Y	Y	Y	Y	Y	Y
instanceofOperator	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
Int8ArrayObject	N	N	N	Y	Y	Y	Y
Int16ArrayObject	N	N	N	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
Int32ArrayObject	N	N	N	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STORE APPS
Intl.Collator Object	N	N	N	N	Y	Y	v8 (Win): N v8.1 (Win): Y v8.1 (Phone): Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STORE APPS
Intl.DateTimeFormatObject	N	N	N	N	Y	Y	v8: N v8.1: Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
Intl.NumberFormatObject	N	N	N	N	Y	Y	v8: N v8.1: Y
isFiniteFunction	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
isArrayFunction	N	N	Y	Y	Y	Y	Y
IsExtensibleFunction	N	N	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
isFrozenFunction	N	N	Y	Y	Y	Y	Y
isIntegerFunction	N	N	N	N	N	Y	v8.1: N v10: Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
isNaNFunction	Y	Y	Y	Y	Y	Y	Y
isNaNFunction(Number)	N	N	N	N	N	Y	v8.1: N v10: Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
ISODateFormat	N	N	Y	Y	Y	Y	Y
IsPrototypeOfMethod	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STORE APPS
isSealedFunction	N	N	Y	Y	Y	Y	Y
italicsMethod	Y	Y	Y	Y	Y	Y	Y
Iterators	N	N	N	N	N	Y	v8.1: N v10: Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STORE APPS
itemMethod	Y	Y	Y	Y	Y	Y	Y
joinMethod	Y	Y	Y	Y	Y	Y	Y
JSONObject	N	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
keysFunction	N	N	Y	Y	Y	Y	Y
keysMethod(Array)	N	N	N	N	N	Y	v8.1: N v10: Y

LANGUAGEELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
Labeled Statement	Y	Y	Y	Y	Y	Y	Y
lastIndexProperty	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
lastIndexOfMethod(Array)	N	N	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
lastIndexOfMethod(String)	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
lastMatchProperty($&)	Y	Y	Y	Y	Y	Y	Y
lastParenProperty($+)	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
lboundMethod	Y	Y	Y	Y	Y	Y	Y
leftContextProperty($')	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
Left Shift Assignment Operator (<<=)	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
length Property (Arguments)	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
length Property (Array)	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
lengthProperty (Function)	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
lengthProperty (String)	Y	Y	Y	Y	Y	Y	Y
LessthanOperator (<)	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STORE APPS
Less than or equal to Operator (<=)	Y	Y	Y	Y	Y	Y	Y
let Statement	N	N	N	N	Y	Y	v8: N v8.1: Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STORE APPS
linkMethod	Y	Y	Y	Y	Y	Y	Y
LN2Constant	Y	Y	Y	Y	Y	Y	Y
LN10Constant	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
localeCompareMethod	Y	Y	Y	Y	Y	Y	Y
logFunction	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
LOG2EConstant	Y	Y	Y	Y	Y	Y	Y
LOG10EConstant	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
Logical AND Operator (&&)	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STORE APPS
Logical NOT Operator (!)	Y	Y	Y	Y	Y	Y	Y
Logical OR Operator (||)	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
mapMethod	N	N	Y	Y	Y	Y	Y
MapObject	N	N	N	N	Y	Y	v8: N v8.1: Y
matchMethod	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
MathObject	Y	Y	Y	Y	Y	Y	Y
maxFunction	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STORE APPS
MAX_VALUE Constant	Y	Y	Y	Y	Y	Y	Y
messageProperty	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
minFunction	Y	Y	Y	Y	Y	Y	Y
MIN_VALUE Constant	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STORE APPS
Remainder Assignment Operator (%=)	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
Remainder Operator (%)	Y	Y	Y	Y	Y	Y	Y
moveFirstMethod	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STORE APPS
moveNextMethod	Y	Y	Y	Y	Y	Y	Y
multilineProperty	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
Multiplication Assignment Operator (*=)	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STORE APPS
Multiplication Operator (*)	Y	Y	Y	Y	Y	Y	Y
name Property	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
NaNConstant(Global)	Y	Y	Y	Y	Y	Y	Y
NaNConstant(Number)	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STORE APPS
NEGATIVE_INFINITY Constant	Y	Y	Y	Y	Y	Y	Y
newOperator	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STORE APPS
Nonidentity Operator (!==)	Y	Y	Y	Y	Y	Y	Y
nowFunction	N	N	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STORE APPS
Number Object	Y	Y	Y	Y	Y	Y	Y
number Property	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
Object Object	Y	Y	Y	Y	Y	Y	Y
Operator Precedence	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STORE APPS
Date.parseFunction	Y	Y	Y	Y	Y	Y	Y
JSON.parseFunction	N	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
parseFloatFunction	Y	Y	Y	Y	Y	Y	Y
parseIntFunction	Y	Y	Y	Y	Y	Y	Y
PIConstant	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
popMethod	Y	Y	Y	Y	Y	Y	Y
POSITIVE_INFINITY Constant	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
powFunction	Y	Y	Y	Y	Y	Y	Y
preventExtensionsFunction	N	N	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
PromiseObject	N	N	N	N	N	Y	v8.1: N v10: Y
prototype Property	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
propertyIsEnumerableMethod	Y	Y	Y	Y	Y	Y	Y
ProxyObject	N	N	N	N	N	Y	v8.1: N v10: Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
pushMethod	Y	Y	Y	Y	Y	Y	Y
randomFunction	Y	Y	Y	Y	Y	Y	Y
rawFunction	N	N	N	N	N	Y	v8.1: N v10: Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
reduceMethod	N	N	Y	Y	Y	Y	Y
reduceRightMethod	N	N	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
RegExpObject	Y	Y	Y	Y	Y	Y	Y
RegularExpressionObject	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
Regular Expression Syntax	Y	Y	Y	Y	Y	Y	Y
Regular Expression /y flag	N	N	N	N	N	Exp.	v8.1: N v10: Exp.

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
repeatMethod (String)	N	N	N	N	N	Y	v8.1: N v10: Y
replaceMethod	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STORE APPS
Functions	N	N	N	N	N	N	v8.1: N v10: Y
returnStatement	Y	Y	Y	Y	Y	Y	Y
reverseMethod	Y	Y	Y	Y	Y	Y	Y

LANGUAGEELEMENT	QUIRKS, INTERNETEXPLORER6STANDARDS, INTERNETEXPLORER7STANDARDS	INTERNETEXPLORER8STANDARDS	INTERNETEXPLORER9STANDARDS	INTERNETEXPLORER10STANDARDS	INTERNETEXPLORER11STANDARDS	EDGE	STOREAPPS
rightContextProperty($')	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STORE APPS
Right Shift Assignment Operator (>>=)	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
roundFunction	Y	Y	Y	Y	Y	Y	Y
ScriptEngineFunction	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
Script Engine Build Version Function	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
Script Engine Major Version Function	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
ScriptEngineMinorVersion Function	Y	Y	Y	Y	Y	Y	Y
seal Function	N	N	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STORE APPS
searchMethod	Y	Y	Y	Y	Y	Y	Y
SetObject	N	N	N	N	Y	Y	v8: N v8.1: Y
setDateMethod	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STORE APPS
setFullYearMethod		Y	Y	Y	Y	Y	Y
setHoursMethod	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STORE APPS
setMilliseconds Method	Y	Y	Y	Y	Y	Y	Y
setMinutes Method	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
setMonthMethod	Y	Y	Y	Y	Y	Y	Y
setSecondsMethod	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STORE APPS
setTimeMethod	Y	Y	Y	Y	Y	Y	Y
setUTCDateMethod	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
setUTCFullYearMethod	Y	Y	Y	Y	Y	Y	Y
setUTCHoursMethod	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STORE APPS
setUTCMilliseconds Method	Y	Y	Y	Y	Y	Y	Y
setUTCMinutes Method	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STORE APPS
setUTCMonthMethod	Y	Y	Y	Y	Y	Y	Y
setUTCSecondsMethod	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STORE APPS
setYearMethod	Y	Y	Y	Y	Y	Y	Y
shiftMethod	Y	Y	Y	Y	Y	Y	Y
sinFunction	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
sliceMethod(Array)	Y	Y	Y	Y	Y	Y	Y
sliceMethod(String)	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STORE APPS
smallMethod	Y	Y	Y	Y	Y	Y	Y
someMethod	N	N	Y	Y	Y	Y	Y
sortMethod	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
sourceProperty	Y	Y	Y	Y	Y	Y	Y
spliceMethod	Y	Y	Y	Y	Y	Y	Y
splitMethod	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STORE APPS
Functions	N	N	N	N	N	Y	v8.1: N v10: Y
sqrtFunction	Y	Y	Y	Y	Y	Y	Y
SQRT1_2Constant	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STORE APPS
SQRT2 Constant	Y	Y	Y	Y	Y	Y	Y
use strict Directive	N	N	N	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
strikeMethod	Y	Y	Y	Y	Y	Y	Y
StringObject	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STORE APPS
JSON.stringifyFunction	N	Y	Y	Y	Y	Y	Y
subMethod	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
substrMethod	Y	Y	Y	Y	Y	Y	Y
substringMethod	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STORE APPS
Subtraction Assignment Operator (-=)	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
Subtraction Operator (-)	Y	Y	Y	Y	Y	Y	Y
supMethod	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
switchStatement	Y	Y	Y	Y	Y	Y	Y
SymbolObject	N	N	N	N	N	Y	v8.1: N v10: Y
tanFunction	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STORE APPS
Template strings	N	N	N	N	N	Y	v8.1: N v10: Y
testMethod	Y	Y	Y	Y	Y	Y	Y
thisStatement	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STORE APPS
throwStatement	Y	Y	Y	Y	Y	Y	Y
toArrayMethod	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STORE APPS
toDateString Method	Y	Y	Y	Y	Y	Y	Y
toExponential Method	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
toFixedMethod	Y	Y	Y	Y	Y	Y	Y
toGMTStringMethod	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STORE APPS
toISOString Method	N	N	Y	Y	Y	Y	Y
toJSON Method	N	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
toLocaleDateString Method	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
toLocaleLowerCaseMethod	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
toLocaleString Method	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STORE APPS
toLocaleTimeString Method	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
toLocaleUppercaseMethod	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STORE APPS
toLowerCaseMethod	Y	Y	Y	Y	Y	Y	Y
toPrecisionMethod	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
toStringMethod	Y	Y	Y	Y	Y	Y	Y
toTimeStringMethod	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
toUpperCaseMethod	Y	Y	Y	Y	Y	Y	Y
toUTCStringMethod	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STORE APPS
trimMethod	N	N	Y	Y	Y	Y	Y
tryStatement	Y	Y	Y	Y	Y	Y	Y
typeofOperator	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
uboundMethod	Y	Y	Y	Y	Y	Y	Y
Uint8ArrayObject	N	N	N	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
Uint16ArrayObject	N	N	N	Y	Y	Y	Y
Uint32ArrayObject	N	N	N	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STORE APPS
Uint8ClampedArrayObject	N	N	N	N	Y	Y	v8: No v8.1 (Win): Yes v8.1 (Phone): No v10: Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
Unary Negation Operator (-)	Y	Y	Y	Y	Y	Y	Y
undefined Constant	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
unescapeFunction	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STORE APPS
Unicodecodepointescapecharacters	N	N	N	N	N	Y	v8.1: N v10: Y
unshiftMethod	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STORE APPS
Unsigned Right Shift Assignment Operator (>>>=)	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STORE APPS
Unsigned Right Shift Operator (>>>)	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
use strict Directive	N	N	N	Y	Y	Y	Y
UTC Function	Y	Y	Y	Y	Y	Y	Y
valueOf Method	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
valuesMethod (Array)	N	N	N	N	N	Y	v8.1: N v10: Y
varStatement	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
VBArrayObject	Y	Y	Y	Y	Y	Y	N
voidOperator	Y	Y	Y	Y	Y	Y	Y
WeakMapObject	N	N	N	N	Y	Y	v8: N v8.1: Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
WeakSetObject	N	N	N	N	N	Y	v8.1: N v10: Y
while Statement	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
WinRTError Object	N	N	N	Y	Y	Y	Y
withStatement	Y	Y	Y	Y	Y	Y	Y
writeFunction	Y	Y	Y	Y	Y	Y	Y

LANGUAGE ELEMENT	QUIRKS, INTERNET EXPLORER 6 STANDARDS, INTERNET EXPLORER 7 STANDARDS	INTERNET EXPLORER 8 STANDARDS	INTERNET EXPLORER 9 STANDARDS	INTERNET EXPLORER 10 STANDARDS	INTERNET EXPLORER 11 STANDARDS	EDGE	STOREAPPS
writeln Function	Y	Y	Y	Y	Y	Y	Y

* Supports DOM objects but not user-defined objects. The `enumerable` and `configurable` attributes can be specified, but they are not used.

See Also

Defining Document Compatibility

JavaScript Objects

10/18/2017 • 2 min to read • Edit Online

The following table lists JavaScript Objects.

Objects

DESCRIPTION	LANGUAGE ELEMENT
Enables and returns a reference to an Automation object. This object is supported in Internet Explorer only.	ActiveXObject Object
Provides support for creation of arrays of any data type.	Array Object
Represents a raw buffer of binary data, which is used to store data for the different typed arrays. ArrayBuffers cannot be read from or written to directly, but can be passed to a typed array or DataView to interpret the raw buffer as needed.	ArrayBuffer Object
An object representing the arguments to the currently executing function, and the functions that called it.	arguments Object
Creates a new Boolean value.	Boolean Object
Used to read and write different kinds of binary data to any location in the ArrayBuffer.	DataView Object
Enables basic storage and retrieval of dates and times.	Date Object
An intrinsic object that can send output to a script debugger.	Debug Object
Enables enumeration of items in a collection. This object is supported in Internet Explorer only.	Enumerator Object
An object that contains information about errors that occur while JavaScript code is running.	Error Object
A typed array of 32-bit float values.	Float32Array Object
A typed array of 64-bit float values.	Float64Array Object
Creates a new function.	Function Object
An intrinsic object whose purpose is to collect global methods into one object.	Global Object
A typed array of 8-bit integer values	Int8Array Object
A typed array of 16-bit integer values	Int16Array Object

DESCRIPTION	LANGUAGE ELEMENT
A typed array of 32-bit integer values	Int32Array Object
Provides locale-specific string comparisons.	Intl.Collator Object
Provides locale-specific date and time formatting.	Intl.DateTimeFormat Object
Provides locale-specific number formatting.	Intl.NumberFormat Object
An intrinsic object that provides two methods to convert to and from the JavaScript Object Notation (JSON) format.	JSON Object
A collection of key/value pairs.	Map Object
An intrinsic object that provides basic mathematics functionality and constants.	Math Object
An object representation of the number data type and placeholder for numeric constants.	Number Object
Provides functionality common to allJavaScript objects.	Object Object
Provides a mechanism to schedule work to be done on a value that has not yet been computed.	Promise Object
Enables custom behavior for an object.	Proxy Object
Provides methods for use in operations that are intercepted.	Reflect Object
Stores information on regular expression pattern searches.	RegExp Object
Contains a regular expression pattern.	Regular Expression Object
A collection of unique values that may be of any type.	Set Object
Allows manipulation and formatting of text strings and determination and location of substrings within strings.	String Object
Allows you to create a unique identifier.	Symbol Object
A typed array of 8-bit unsigned integer values,	Uint8Array Object
A typed array of 8-bit unsigned integers with clamped values.	Uint8ClampedArray Object
A typed array of 16-bit unsigned integer values	Uint16Array Object
A typed array of 32-bit unsigned integer values	Uint32Array Object
Provides access to Visual Basic safe arrays.	VBArray Object
A collection of key/value pairs in which each key is an object reference.	WeakMap Object

DESCRIPTION	LANGUAGE ELEMENT
A collection of unique objects.	WeakSet Object
An error that originates in Windows Runtime functions and methods.	WinRTError Object

Related Reference

HTML and DHTML Reference

ActiveXObject Object (JavaScript)

10/18/2017 • 2 min to read • Edit Online

Enables and returns a reference to an Automation object.

This object is used only to instantiate Automation objects, and has no members.

> **WARNING**
>
> This object is a Microsoft extension and is supported in Internet Explorer only, not in Windows 8.x Store apps.

Syntax

```
newObj = new ActiveXObject(servername.typename[, location])
```

Parameters

`newObj`
Required. The variable name to which the `ActiveXObject` is assigned.

`servername`
Required. The name of the application providing the object.

`typename`
Required. The type or class of the object to create.

`location`
Optional. The name of the network server where the object is to be created.

Remarks

Automation servers provide at least one type of object. For example, a word-processing application may provide an application object, a document object, and a toolbar object.

You may be able to identify `servername.typename` values on a host PC in the `HKEY_CLASSES_ROOT` registry key. For example, here are a few examples of values you may find there, depending on which programs are installed:

- Excel.Application
- Excel.Chart
- Scripting.FileSystemObject
- WScript.Shell
- Word.Document

> **IMPORTANT**
> ActiveX objects may present security issues. To use the `ActiveXObject`, you may need to adjust security settings in Internet Explorer for the relevant security zone. For example, for the Local intranet zone, you typically need to change a custom setting to "Initialize and script ActiveX controls not marked as safe for scripting."

To identify members of an Automation object that you can use in your code, you may need to use a COM object browser, such as the OLE/COM Object Viewer, if no reference documentation is available for the Automation object.

To create an Automation object, assign the new `ActiveXObject` to an object variable:

```
var ExcelApp = new ActiveXObject("Excel.Application");
var ExcelSheet = new ActiveXObject("Excel.Sheet");
```

This code starts the application creating the object (in this case, a Microsoft Excel worksheet). Once an object is created, you refer to it in code using the object variable you defined. In the following example, you access properties and methods of the new object using the object variable `ExcelSheet` and other Excel objects, including the Application object and the ActiveSheet.Cells collection.

```
// Make Excel visible through the Application object.
ExcelSheet.Application.Visible = true;
// Place some text in the first cell of the sheet.
ExcelSheet.ActiveSheet.Cells(1,1).Value = "This is column A, row 1";
// Save the sheet.
ExcelSheet.SaveAs("C:\\TEST.XLS");
// Close Excel with the Quit method on the Application object.
ExcelSheet.Application.Quit();
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Not supported in Windows 8.x Store apps. See Version Information.

> **NOTE**
> Creating an `ActiveXObject` on a remote server is not supported in Internet Explorer 9 standards mode, Internet Explorer 10 standards mode, Internet Explorer 11 standards mode, and Windows Store apps or later.

See Also

GetObject Function
Unique authentication using Magic of HTML5/WCF sample app

Array Object (JavaScript)

10/18/2017 • 4 min to read • Edit Online

Provides support for creation of arrays of any data type.

Syntax

```
arrayObj = new Array()
arrayObj = new Array([size])
arrayObj = new Array([element0[, element1[, ...[, elementN]]]])
```

Parameters

`arrayObj`
Required. The variable name to which the `Array` object is assigned.

`size`
Optional. The size of the array. As arrays are zero-based, created elements will have indexes from zero to `size` - 1.

`element0,...,elementN`
Optional. The elements to place in the array. This creates an array with *n* + 1 elements, and a `length` of *n* + 1. Using this syntax, you must supply more than one element.

Remarks

After an array is created, you can access the individual elements of the array by using [ ] notation. Note that arrays in JavaScript are zero-based.

```
var my_array = new Array();
for (i = 0; i < 10; i++) {
    my_array[i] = i;
}
x = my_array[4];
document.write(x);

// Output: 4
```

You can pass an unsigned 32-bit integer to the `Array` constructor to specify the size of the array. If the value is negative or not an integer, a run-time error occurs. If you run the following code, you should see this error in the Console.

```
var arr = new Array(10);
document.write(arr.length);

// Output: 10

// Don't do this
var arr = new Array(-1);
arr = new Array(1.50);
```

If a single value is passed to the `Array` constructor, and it is not a number, the `length` property is set to 1, and the value of the only element becomes the single, passed-in argument.

```
var arr = new Array("one");
document.write(arr.length);
document.write("<br/>");
document.write(arr[0]);

// Output:
1
one
```

JavaScript arrays are sparse arrays, which means that not all the elements in an array may contain data. In JavaScript, only the elements that actually contain data exist in the array. This reduces the amount of memory used by the array.

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

Some members in the following lists were introduced in later versions. For more information, see Version Information or the documentation for the individual members.

Properties

The following table lists the properties of the `Array` object.

PROPERTY	DESCRIPTION
constructor Property	Specifies the function that creates an array.
length Property (Array)	Returns an integer value that is one higher than the highest element defined in an array.
prototype Property	Returns a reference to the prototype for an array.

Functions

The following table describes the functions of the `Array` object.

FUNCTION	DESCRIPTION
Array.from Function	Returns an array from an array-like or iterable object.
Array.isArray Function	Returns a Boolean value that indicates whether an object is an array.
Array.of Function	Returns an array from the passed in arguments.

Methods

The following table lists the methods of the `Array` object.

METHOD	DESCRIPTION
concat Method (Array)	Returns a new array consisting of a combination of two arrays.
entries Method	Returns an iterator that contains the key/value pairs of the array.
every Method	Checks whether a defined callback function returns `true` for all elements in an array.
fill Method	Populates an array with a specified value.
filter Method	Calls a defined callback function on each element of an array, and returns an array of values for which the callback function returns `true`.
findIndex Method	Returns an index value for the first array element that meets test criteria specified in a callback function.
forEach Method	Calls a defined callback function for each element in an array.
hasOwnProperty Method	Returns a Boolean value that indicates whether an object has a property with the specified name.
indexOf Method (Array)	Returns the index of the first occurrence of a value in an array.
isPrototypeOf Method	Returns a Boolean value that indicates whether an object exists in another object's prototype chain.
join Method	Returns a `String` object consisting of all the elements of an array concatenated together.
keys Method	Returns an iterator that contains the index values of the array.
lastIndexOf Method (Array)	Returns the index of the last occurrence of a specified value in an array.
map Method	Calls a defined callback function on each element of an array, and returns an array that contains the results.
pop Method	Removes the last element from an array and returns it.
propertyIsEnumerable Method	Returns a Boolean value that indicates whether a specified property is part of an object and whether it is enumerable.
push Method	Appends new elements to an array, and returns the new length of the array.

METHOD	DESCRIPTION
reduce Method	Accumulates a single result by calling a defined callback function for all elements in an array. The return value of the callback function is the accumulated result, and is provided as an argument in the next call to the callback function.
reduceRight Method	Accumulates a single result by calling a defined callback function for all elements in an array, in descending order. The return value of the callback function is the accumulated result, and is provided as an argument in the next call to the callback function.
reverse Method	Returns an `Array` object with the elements reversed.
shift Method	Removes the first element from an array and returns it.
slice Method (Array)	Returns a section of an array.
some Method	Checks whether a defined callback function returns `true` for any element of an array.
sort Method	Returns an `Array` object with the elements sorted.
splice Method	Removes elements from an array and, if necessary, inserts new elements in their place, returning the deleted elements.
toLocaleString Method	Returns a string using the current locale.
toString Method	Returns a string representation of an array.
unshift Method	Inserts new elements at the start of an array.
valueOf Method	Gets a reference to the array.
values Method	Returns an iterator that contains the values of the array.

See Also

Scrolling, panning, and zooming sample app

constructor Property (Array)

10/18/2017 • 1 min to read • Edit Online

Specifies the function that creates an array.

Syntax

```
array.constructor
```

Remarks

The required `array` is the name of an array.

The `constructor` property is a member of the prototype of every object that has a prototype. This includes all intrinsic JavaScript objects except the `Global` and `Math` objects. The `constructor` property contains a reference to the function that constructs instances of that particular object.

Example

The following example illustrates the use of the constructor property.

```
var x = new Array();

if (x.constructor == Array)
    document.write("Object is an Array.");
else
    document.write("Object is not an Array.");

// Output:
// Object is an Array.
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

length Property (Array) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Gets or sets the length of the array. This is a number one higher than the highest element defined in an array.

Syntax

```
numVar = arrayObj.length
```

Parameters

`numVar`
Required. Any number.

`arrayObj`
Required. Any `Array` object.

Remarks

In JavaScript arrays are sparse, and the elements in an array do not have to be contiguous. The `length` property is not necessarily the number of elements in the array. For example, in the following array definition, `my_array.length` contains 7, not 2:

```
var my_array = new Array( );
my_array[0] = "Test";
my_array[6] = "Another Test";
```

If you make the `length` property smaller than its previous value, the array is truncated, and any elements with array indexes equal to or greater than the new value of the `length` property are lost.

If you make the length property larger than its previous value, the array is expanded, and any new elements created have the value `undefined`.

The following example illustrates the use of the `length` property:

```
var a;
a = new Array(0,1,2,3,4);
document.write(a.length);

// Output
// 5
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

NOTE

Starting with Internet Explorer 9 Standards mode, trailing commas included in the initialization of an Array are handled differently.

prototype Property (Array)

10/18/2017 • 1 min to read • Edit Online

Returns a reference to the prototype for a class of array.

Syntax

```
array.prototype
```

Remarks

The `array` argument is the name of an array.

Use the `prototype` property to provide a base set of functionality to a class of objects. New instances of an object "inherit" the behavior of the prototype assigned to that object.

For example, to add a method to the `Array` object that returns the value of the largest element of the array, declare the function, add it to `Array.prototype`, and then use it.

```
function array_max( ){
    var i, max = this[0];
    for (i = 1; i < this.length; i++)
    {
    if (max < this[i])
    max = this[i];
    }
    return max;
}
Array.prototype.max = array_max;
var myArray = new Array(7, 1, 3, 11, 25, 9
);
document.write(myArray.max());

// Output: 25
```

All intrinsic JavaScript objects have a `prototype` property that is read-only. Properties and methods may be added to the prototype, but the object may not be assigned a different prototype. However, user-defined objects may be assigned a new prototype.

The method and property lists for each intrinsic object in this language reference indicate which ones are part of the object's prototype, and which are not.

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

Array.from Function (Array) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Returns an array from an array-like or iterable object.

Syntax

```
Array.from (arrayLike [ , mapfn [ , thisArg ] ] );
```

Parameters

`arrayLike`

Required. An array-like object or an iterable object.

`mapfn`

Optional. A mapping function to call on each element in `arrayLike`.

`thisArg`

Optional. Specifies the `this` object in the mapping function.

Remarks

The `arrayLike` parameter must be either an object with indexed elements and a `length` property or an iterable object, such as a `Set` object.

The optional mapping function is called on each element in the array.

Example

The following example returns an array from a collection of DOM element nodes.

```
var elemArr = Array.from(document.querySelectorAll('*'));
var elem = elemArr[0]; // elem contains a reference to the first DOM element
```

Example

The following example returns an array of characters.

```
var charArr = Array.from("abc");
// charArr[0] == "a";
```

Example

The following example returns an array of objects contained in the collection.

```
var setObj = new Set("a", "b", "c");
var objArr = Array.from(setObj);
// objArr[1] == "b";
```

Example

The following example shows the use of arrow syntax and a mapping function to change the value of elements.

```
var arr = Array.from([1, 2, 3], x => x * 10);
// arr[0] == 10;
// arr[1] == 20;
// arr[2] == 30;
```

Requirements

Supported in Microsoft Edge (Edge browser). Also supported in Store apps (Microsoft Edge on Windows 10). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Not supported in Windows 8.1.

Array.isArray Function (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Determines whether an object is an array.

Syntax

```
Array.isArray(object)
```

Parameters

`object`

Required. The object to test.

Return Value

`true` if `object` is an array; otherwise, `false`. If the `object` argument is not an object, `false` is returned.

Example

The following example illustrates the use of the `Array.isArray` function.

```
var ar = [];
var result = Array.isArray(ar);
// Output: true

var ar = new Array();
var result = Array.isArray(ar);
// Output: true

var ar = [1, 2, 3];
var result = Array.isArray(ar);
// Output: true

var result = Array.isArray("an array");
document.write(result);
// Output: false
```

Requirements

Supported in the following document modes: Internet Explorer 9 standards, Internet Explorer 10 standards, and Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards.

See Also

Array Object

Using Arrays

typeof Operator

Array.of Function (Array) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Returns an array from the passed in arguments.

Syntax

```
Array.of(element0[, element1][, ...][,elementN]);
```

Parameters

`element0,...,elementN`

Optional. The elements to place in the array. This creates an array with n + 1 elements, and a length of n + 1.

Remarks

This function is similar to calling `new Array(args)`, but `Array.of` does not include special behavior when one argument is passed in.

Example

The following example creates an array from passed in numbers.

```
var arr = Array.of(1, 2, 3);
// arr[0] == 1
```

Example

The following example shows the difference between using `Array.of` and `new Array`.

```
var arr1 = Array.of(3);
// arr1[0] == 3

// With new Array, a single argument specifies
// the length of the new array.
var arr2 = new Array(3);
// arr2[0] is undefined
// arr2.length == 3
```

Requirements

Supported in Microsoft Edge (Edge browser). Also supported in Store apps (Microsoft Edge on Windows 10). See Version Information.

Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Not supported in Windows 8.1.

concat Method (Array) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Combines two or more arrays.

Syntax

```
array1.concat([item1[, item2[, . . . [, itemN]]]])
```

Parameters

`array1`
Required. The `Array` object to which the other arrays are concatenated.

`item1,. . ., itemN`
Optional. Additional items to add to the end of `array1`.

Remarks

The `concat` method returns an `Array` object containing the concatenation of `array1` and any other supplied items.

The items to be added (*item1 itemN*) to the array are added, in order, starting from the first item in the list. If one of the items is an array, its contents are added to the end of `array1`. If the item is anything other than an array, it is added to the end of the array as a single array element.

Elements of source arrays are copied to the resulting array as follows:

- If an object is copied from any of the arrays being concatenated to the new array, the object reference continues to point to the same object. A change in either the new array or the original array will result in a change to the other.
- If a number or string value is added to the new array, only the value is copied. Changing the value in one array does not affect the value in the other.

Example

The following example shows how to use the `concat` method when used with an array:

```
var a, b, c, d;
a = new Array(1,2,3);
b = "dog";
c = new Array(42, "cat");
d = a.concat(b, c);
document.write(d);

//Output:
1, 2, 3, "dog", 42, "cat"
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

See Also

concat Method (String)
join Method (Array)

entries Method (Array) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Returns an iterator that returns the key/value pairs of the array.

Syntax

```
arrayObj.entries();
```

Parameters

`arrayObj`

Required. The array object.

Remarks

Example

The following example shows how to get the key/value pairs of an array.

```
var entries = ["a", "b", "c"].entries();
// entries.next().value == [0, "a"]
// entries.next().value == [1, "b"]
// entries.next().value == [2, "c"]
```

Requirements

Supported in Microsoft Edge (Edge browser). Also supported in Store apps (Microsoft Edge on Windows 10). See Version Information.

Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Not supported in Windows 8.1.

every Method (Array) (JavaScript)

10/18/2017 • 3 min to read • Edit Online

Determines whether all the members of an array satisfy the specified test.

Syntax

```
array1.every(callbackfn[, thisArg])
```

Parameters

PARAMETER	DEFINITION
`array1`	Required. An array object.
`callbackfn`	Required. A function that accepts up to three arguments. The `every` method calls the `callbackfn` function for each element in `array1` until the `callbackfn` returns `false`, or until the end of the array.
`thisArg`	Optional. An object to which the `this` keyword can refer in the `callbackfn` function. If `thisArg` is omitted, `undefined` is used as the `this` value.

Return Value

`true` if the `callbackfn` function returns `true` for all array elements; otherwise, `false`. If the array is has no elements, the `every` method returns `true`.

Exceptions

If the `callbackfn` argument is not a function object, a `TypeError` exception is thrown.

Remarks

The `every` method calls the `callbackfn` function one time for each array element, in ascending index order, until the `callbackfn` function returns `false`. If an element that causes `callbackfn` to return `false` is found, the `every` method immediately returns `false`. Otherwise, the `every` method returns `true`.

The callback function is not called for missing elements of the array.

In addition to array objects, the `every` method can be used by any object that has a `length` property and that has numerically indexed property names.

> **NOTE**
>
> You can use the some Method (Array) to check whether the callback function returns `true` for any element of an array.

Callback Function Syntax

The syntax of the callback function is as follows:

```
function callbackfn(value, index, array1)
```

You can declare the callback function with up to three parameters.

The following table lists the callback function parameters.

CALLBACK PARAMETER	DEFINITION
`value`	The value of the array element.
`index`	The numeric index of the array element.
`array1`	The array object that contains the element.

Modifying the Array Object

The array object can be modified by the callback function.

The following table describes the results of modifying the array object after the `every` method starts.

CONDITION AFTER THE `EVERY` METHOD STARTS	ELEMENT PASSED TO CALLBACK FUNCTION?
Element is added beyond the original length of the array.	No.
Element is added to fill in a missing element of the array.	Yes, if that index has not yet been passed to the callback function.
Element is changed.	Yes, if that element has not yet been passed to the callback function.
Element is deleted from the array.	No, unless that element has already been passed to the callback function.

Example

The following example illustrates the use of the `every` method.

```
// Define the callback function.
function CheckIfEven(value, index, ar) {
    document.write(value + " ");

    if (value % 2 == 0)
        return true;
    else
        return false;
}

// Create an array.
var numbers = [2, 4, 5, 6, 8];

// Check whether the callback function returns true for all of the
// array values.
if (numbers.every(CheckIfEven))
    document.write("All are even.");
else
    document.write("Some are not even.");

// Output:
// 2 4 5 Some are not even.
```

Example

The following example illustrates the use of the `thisArg` argument, which specifies an object to which the `this` keyword can refer.

```
// Create a function that returns true if the value is
// numeric and within range.
var checkNumericRange = function(value) {
    if (typeof value !== 'number')
        return false;
    else
        return value >= this.minimum && value <= this.maximum;
}

// Create an array of numbers.
var numbers = [10, 15, 19];

// Check whether the callback function returns true for
// all of the array values.
// The obj argument enables use of the this value
// within the callback function.

var obj = { minimum: 10, maximum: 20 }

if (numbers.every(checkNumericRange, obj))
    document.write ("All are within range.");
else
    document.write ("Some are not within range.");

// Output:
//   All are within range.
```

Requirements

Supported in the following document modes: Internet Explorer 9 standards, Internet Explorer 10 standards, and Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7

standards, Internet Explorer 8 standards.

See Also

some Method (Array)
filter Method (Array)
Array Object
Using Arrays

fill Method (Array) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Populates an array with a specified value.

Syntax

```
arrayObj.fill(value [ , start [ , end ] ]);
```

Parameters

`arrayObj`
Required. The array object.

`value`
Required. The value used to populate the array.

`start`
Optional. The starting index used to populate array values. The default value is 0.

`end`
Optional. The ending index used to populate array values. The default value is the length property of the `this` object.

Remarks

If `start` is negative, `start` is treated as `length` + `start`, where `length` is the length of the array. If `end` is negative, `end` is treated as `length` + `end`.

Example

The following code examples populate an array with values.

```
[0, 0, 0].fill(7, 1);
// Array contains [0,7,7]

[0, 0, 0].fill(7);
// Array contains [7,7,7]
```

Requirements

Supported in Microsoft Edge (Edge browser). Also supported in Store apps (Microsoft Edge on Windows 10). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Not supported in Windows 8.1.

filter Method (Array) (JavaScript)

10/18/2017 • 4 min to read • Edit Online

Returns the elements of an array that meet the condition specified in a callback function.

Syntax

```
array1.filter(callbackfn[, thisArg])
```

Parameters

PARAMETER	DEFINITION
`array1`	Required. An array object.
`callbackfn`	Required. A function that accepts up to three arguments. The `filter` method calls the `callbackfn` function one time for each element in the array.
`thisArg`	Optional. An object to which the `this` keyword can refer in the `callbackfn` function. If `thisArg` is omitted, `undefined` is used as the `this` value.

Return Value

A new array that contains all the values for which the callback function returns `true`. If the callback function returns `false` for all elements of `array1`, the length of the new array is 0.

Exceptions

If the `callbackfn` argument is not a function object, a `TypeError` exception is thrown.

Remarks

The `filter` method calls the `callbackfn` function one time for each element in the array, in ascending index order. The callback function is not called for missing elements of the array.

In addition to array objects, the `filter` method can be used by any object that has a `length` property and that has numerically indexed property names.

Callback Function Syntax

The syntax of the callback function is as follows:

```
function callbackfn(value, index, array1)
```

You can declare the callback function by using up to three parameters.

The following table lists the callback function parameters.

CALLBACK ARGUMENT	DEFINITION
`value`	The value of the array element.
`index`	The numeric index of the array element.
`array1`	The array object that contains the element.

Modifying the Array Object

The `filter` method does not directly modify the original array, but the callback function might modify it. The following table describes the results of modifying the array object after the `filter` method starts.

CONDITION AFTER THE `FILTER` METHOD STARTS	ELEMENT PASSED TO CALLBACK FUNCTION?
Element is added beyond the original length of the array.	No.
Element is added to fill in a missing element of the array.	Yes, if that index has not yet been passed to the callback function.
Element is changed.	Yes, if that element has not yet been passed to the callback function.
Element is deleted from the array.	No, unless that element has already been passed to the callback function.

Example

The following example shows how to use the `filter` method.

```
// Define a callback function.
function CheckIfPrime(value, index, ar) {
    high = Math.floor(Math.sqrt(value)) + 1;

    for (var div = 2; div <= high; div++) {
        if (value % div == 0) {
            return false;
        }
    }
    return true;
}

// Create the original array.
var numbers = [31, 33, 35, 37, 39, 41, 43, 45, 47, 49, 51, 53];

// Get the prime numbers that are in the original array.
var primes = numbers.filter(CheckIfPrime);

document.write(primes);
// Output: 31,37,41,43,47,53
```

Example

In the following example, the `callbackfn` argument includes the code of the callback function.

```
// Create the original array.
var arr = [5, "element", 10, "the", true];

// Create an array that contains the string
// values that are in the original array.
var result = arr.filter(
    function (value) {
        return (typeof value === 'string');
    }
);

document.write(result);
// Output: element, the
```

Example

The following example displays the names of properties that start with the letter "css" in the `window` DOM object.

```
var filteredNames = Object.getOwnPropertyNames(window).filter(IsC);

    for (i in filteredNames)
        document.write(filteredNames[i] + "<br/>");

// Check whether the string starts with "css".
function IsC(value) {
    var firstChar = value.substr(0, 3);
    if (firstChar.toLowerCase() == "css")
        return true;
    else
        return false;
    }

// Output:
// CSSRule
// CSSFontFaceRule
// CSSImportRule
// CSSMediaRule
// CSSNamespaceRule
// CSSPageRule
// CSSRuleList
// CSSStyleDeclaration
// CSSStyleRule
// CSSStyleSheet
```

Example

The following example illustrates the use of the `thisArg` argument, which specifies an object to which the `this` keyword can refer.

```
var checkNumericRange = function(value) {
    if (typeof value !== 'number')
        return false;
    else
        return value >= this.minimum && value <= this.maximum;
}

var numbers = [6, 12, "15", 16, "the", -12];

// The obj argument enables use of the this value
// within the callback function.
var obj = { minimum: 10, maximum: 20 }

var result = numbers.filter(checkNumericRange, obj);

document.write(result);
// Output: 12,16
```

Example

The `filter` method can be applied to a string instead of an array. The following example shows how to do this.

```
// Define a callback function that returns true
// if the current array element follows a space
// or is the first character.
function CheckValue(value, index, ar) {
    if (index == 0)
        return true;
    else
        return ar[index - 1] === " ";
}

// Create a string.
var sentence = "The quick brown fox jumps over the lazy dog.";

// Create an array that contains all characters that follow a space.
var subset = [].filter.call(sentence, CheckValue);

// You can use this alternative syntax.
//var subset = Array.prototype.filter.call(sentence, CheckValue);

document.write(subset);
// Output: T,q,b,f,j,o,t,l,d
```

Requirements

Supported in the following document modes: Internet Explorer 9 standards, Internet Explorer 10 standards, and Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards.

See Also

Array Object
Using Arrays
map Method (Array)
forEach Method (Array)

findIndex Method (Array) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Returns an index value for the first array element that meets test criteria specified in a callback function.

Syntax

```
arrayObj.findIndex(callbackfn [, thisArg]);
```

Parameters

`arrayObj`
Required. The array object.

`callbackfn`
Required. The callback function to test each element in the array.

`thisArg`
Optional. Specifies the `this` object in the callback function. If not specified, the `this` object is undefined.

Remarks

The `findIndex` calls the callback function one time for each element in the array, in ascending order, until an element returns `true`. As soon as an element returns true, `findIndex` returns the index value of the element that returns true. If no elements in the array return true, `findIndex` returns -1.

`findIndex` does not mutate the array object.

Callback Function Syntax

The syntax of the callback function is as follows:

`function callbackfn(value, index, thisArg)`

You can declare the callback function by using up to three parameters.

The callback function parameters are as follows.

CALLBACK ARGUMENT	DEFINITION
`value`	The value of the array element.
`index`	The numeric index of the array element.
`arrayObj`	The array object to be traversed.

Example

In the following example, the callback function tests whether each element in the array is equal to 2.

```
[1,2,3].findIndex(function(x) { x == 2; });
// Returns an index value of 1.
```

Example

The following example uses arrow syntax to test each element. In this example, no elements return `true`, and `findIndex` returns -1.

```
[1,2,3].findIndex(x => x == 4);
// Returns an index value of -1.
```

Requirements

Supported in Microsoft Edge (Edge browser). Also supported in Store apps (Microsoft Edge on Windows 10). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Not supported in Windows 8.1.

forEach Method (Array) (JavaScript)

10/18/2017 • 3 min to read • Edit Online

Performs the specified action for each element in an array.

Syntax

```
array1.forEach(callbackfn[, thisArg])
```

Parameters

PARAMETER	DEFINITION
`array1`	Required. An array object.
`callbackfn`	Required. A function that accepts up to three arguments. `forEach` calls the `callbackfn` function one time for each element in the array.
`thisArg`	Optional. An object to which the `this` keyword can refer in the `callbackfn` function. If `thisArg` is omitted, `undefined` is used as the `this` value.

Exceptions

If the `callbackfn` argument is not a function object, a `TypeError` exception is thrown.

Remarks

The `forEach` method calls the `callbackfn` function one time for each element present in the array, in ascending index order. The callback function is not called for missing elements of the array.

In addition to array objects, the `forEach` method can be used by any object that has a `length` property and that has numerically indexed property names.

Callback Function Syntax

The syntax of the callback function is as follows:

`function callbackfn(value, index, array1)`

You can declare the callback function by using up to three parameters.

The callback function parameters are as follows.

CALLBACK ARGUMENT	DEFINITION
`value`	The value of the array element.

CALLBACK ARGUMENT	DEFINITION
`index`	The numeric index of the array element.
`array1`	The array object that contains the element.

Modifying the Array Object

The `forEach` method does not directly modify the original array, but the callback function might modify it. The following table describes the results of modifying the array object after the `forEach` method starts.

CONDITION AFTER `FOREACH` METHOD STARTS	ELEMENT PASSED TO CALLBACK FUNCTION?
Element is added beyond the original length of the array.	No.
Element is added to fill in a missing element of the array.	Yes, if that index has not yet been passed to the callback function.
Element is changed.	Yes, if that element has not yet been passed to the callback function.
Element is deleted from the array.	No, unless that element has already been passed to the callback function.

Example

The following example illustrates use of the `forEach` method.

```
// Define the callback function.
function ShowResults(value, index, ar) {
    document.write("value: " + value);
    document.write(" index: " + index);
    document.write("<br />");
}

// Create an array.
var letters = ['ab', 'cd', 'ef'];

// Call the ShowResults callback function for each
// array element.
letters.forEach(ShowResults);

// Output:
//  value: ab index: 0
//  value: cd index: 1
//  value: ef index: 2
```

Example

In the following example, the `callbackfn` argument includes the code of the callback function.

```
// Create an array.
var numbers = [10, 11, 12];

// Call the addNumber callback function for each array element.
var sum = 0;
numbers.forEach(
    function addNumber(value) { sum += value; }
);

document.write(sum);
// Output: 33
```

Example

The following example illustrates the use of the `thisArg` argument, which specifies an object that can be referred to with the `this` keyword.

```
// Define the object that contains the callback function.
var obj = {
    showResults: function(value, index) {
        // Call calcSquare by using the this value.
        var squared = this.calcSquare(value);

        document.write("value: " + value);
        document.write(" index: " + index);
        document.write(" squared: " + squared);
        document.write("<br />");
    },
    calcSquare: function(x) { return x * x }
};

// Define an array.
var numbers = [5, 6];

// Call the showResults callback function for each array element.
// The obj is the this value within the
// callback function.
numbers.forEach(obj.showResults, obj);

// Embed the callback function in the forEach statement.
// The obj argument is the this value within the obj object.
// The output is the same as for the previous statement.
numbers.forEach(function(value, index) { this.showResults(value, index) }, obj);

// Output:
//  value: 5 index: 0 squared: 25
//  value: 6 index: 1 squared: 36
//  value: 5 index: 0 squared: 25
//  value: 6 index: 1 squared: 36
```

Requirements

Supported in the following document modes: Internet Explorer 9 standards, Internet Explorer 10 standards, and Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards.

See Also

filter Method (Array)
map Method (Array)
some Method (Array)
Array Object
Using Arrays
Hilo JavaScript sample app (Windows Store)

indexOf Method (Array) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Returns the index of the first occurrence of a value in an array.

Syntax

```
array1.indexOf(searchElement[, fromIndex])
```

Parameters

PARAMETER	DEFINITION
`array1`	Required. An array object.
`searchElement`	Required. The value to locate in `array1`.
`fromIndex`	Optional. The array index at which to begin the search. If `fromIndex` is omitted, the search starts at index 0.

Return Value

The index of the first occurrence of `searchElement` in the array, or -1 if `searchElement` is not found.

Remarks

The `indexOf` method searches an array for a specified value. The method returns the index of the first occurrence, or -1 if the specified value is not found.

The search occurs in ascending index order.

The array elements are compared to the `searchElement` value by strict equality, similar to the `===` operator. For more information, see Comparison Operators.

The optional `fromIndex` argument specifies the array index at which to begin the search. If `fromIndex` is greater than or equal to the array length, -1 is returned. If `fromIndex` is negative, the search starts at the array length plus `fromIndex`.

Example

The following examples illustrate the use of the `indexOf` method.

```
// Create an array. (The elements start at index 0.)
var ar = ["ab", "cd", "ef", "ab", "cd"];

// Determine the first location of "cd".
document.write(ar.indexOf("cd") + "<br/>");

// Output: 1

// Find "cd" starting at index 2.
document.write(ar.indexOf("cd", 2) + "<br/>");

// Output: 4

// Find "gh" (which is not found).
document.write (ar.indexOf("gh")+ "<br/>");

// Output: -1

// Find "ab" with a fromIndex argument of -2.
// The search starts at index 3, which is the array length plus -2.
document.write (ar.indexOf("ab", -2) + "<br/>");
// Output: 3
```

Requirements

Supported in the following document modes: Internet Explorer 9 standards, Internet Explorer 10 standards, and Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards.

See Also

JavaScript Methods
Array Object
Using Arrays

join Method (Array) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Adds all the elements of an array separated by the specified separator string.

Syntax

```
arrayObj.join([separator])
```

Parameters

`arrayObj`
Required. An `Array` object.

`separator`
Optional. A string used to separate one element of an array from the next in the resulting `String`. If omitted, the array elements are separated with a comma.

Remarks

If any element of the array is **undefined** or `null`, it is treated as an empty string.

Example

The following example illustrates the use of the **join** method.

```
var a, b;
a = new Array(0,1,2,3,4);
b = a.join("-");
document.write(b);

// Output:
// 0-1-2-3-4
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

See Also

String Object

keys Method (Array) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Returns an iterator that returns the index values of the array.

Syntax

```
arrayObj.keys();
```

Parameters

`arrayObj`

Required. The array object.

Remarks

Example

The following example shows how to get the key values of an array.

```
var k = ["a", "b", "c"].keys();
// k.next().value == 0
// k.next().value == 1
// k.next().value == 2
```

Requirements

Supported in Microsoft Edge (Edge browser). Also supported in Store apps (Microsoft Edge on Windows 10). See Version Information.

Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Not supported in Windows 8.1.

lastIndexOf Method (Array) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Returns the index of the last occurrence of a specified value in an array.

Syntax

```
array1.lastIndexOf(searchElement[, fromIndex])
```

Parameters

PARAMETER	DEFINITION
`array1`	Required. The array object to search.
`searchElement`	Required. The value to locate in `array1`.
`fromIndex`	Optional. The array index at which to begin the search. If `fromIndex` is omitted, the search starts at the last index in the array.

Return Value

The index of the last occurrence of `searchElement` in the array, or -1 if `searchElement` is not found.

Remarks

The `lastIndexOf` method searches an array for a specified value. The method returns the index of the first occurrence, or -1 if the specified value is not found.

The search occurs in descending index order (last member first). To search in ascending order, use the indexOf Method (Array).

The array elements are compared to the `searchElement` value by strict equality, similar to the comparison made by the `===` operator. For more information, see Comparison Operators.

The optional `fromIndex` argument specifies the array index at which to begin the search. If `fromIndex` is greater than or equal to the array length, the whole array is searched. If `fromIndex` is negative, the search starts at the array length plus `fromIndex`. If the computed index is less than 0, -1 is returned.

Example

The following examples illustrate the use of the `lastIndexOf` method.

```
// Create an array.
var ar = ["ab", "cd", "ef", "ab", "cd"];

// Determine the first location, in descending order, of "cd".
document.write(ar.lastIndexOf("cd") + "<br/>");

// Output: 4

// Find "cd" in descending order, starting at index 2.
document.write(ar.lastIndexOf("cd", 2) + "<br/>");

// Output: 1

// Search for "gh" (which is not found).
document.write(ar.lastIndexOf("gh")+ "<br/>");

// Output: -1

// Find "ab" with a fromIndex argument of -3.
// The search in descending order starts at index 3,
// which is the array length minus 2.
document.write(ar.lastIndexOf("ab", -3) + "<br/>");
// Output: 0
```

Requirements

Supported in the following document modes: Internet Explorer 9 standards, Internet Explorer 10 standards, and Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards.

See Also

indexOf Method (Array)
Array Object
Using Arrays

map Method (Array) (JavaScript)

10/18/2017 • 3 min to read • Edit Online

Calls a defined callback function on each element of an array, and returns an array that contains the results.

Syntax

```
array1.map(callbackfn[, thisArg])
```

Parameters

PARAMETER	DEFINITION
`array1`	Required. An array object.
`callbackfn`	Required. A function that accepts up to three arguments. The `map` method calls the `callbackfn` function one time for each element in the array.
`thisArg`	Optional. An object to which the `this` keyword can refer in the `callbackfn` function. If `thisArg` is omitted, `undefined` is used as the `this` value.

Return Value

A new array in which each element is the callback function return value for the associated original array element.

Exceptions

If the `callbackfn` argument is not a function object, a `TypeError` exception is thrown.

Remarks

The `map` method calls the `callbackfn` function one time for each element in the array, in ascending index order. The callback function is not called for missing elements of the array.

In addition to array objects, the `map` method can be used by any object that has a `length` property and that has numerically indexed property names.

Callback Function Syntax

The syntax of the callback function is as follows:

`function callbackfn(value, index, array1)`

You can declare the callback function by using up to three parameters.

The following table lists the callback function parameters.

CALLBACK ARGUMENT	DEFINITION
`value`	The value of the array element.
`index`	The numeric index of the array element.
`array1`	The array object that contains the element.

Modifying the Array Object

The array object can be modified by the callback function.

The following table describes the results of modifying the array object after the `map` method starts.

CONDITION AFTER THE `MAP` METHOD STARTS	ELEMENT PASSED TO CALLBACK FUNCTION?
Element is added beyond the original length of the array.	No.
Element is added to fill in a missing element of the array.	Yes, if that index has not yet been passed to the callback function.
Element is changed.	Yes, if that element has not yet been passed to the callback function.
Element is deleted from the array.	No, unless that element has already been passed to the callback function.

Example

The following example illustrates the use of the `map` method.

```
// Define the callback function.
function AreaOfCircle(radius) {
    var area = Math.PI * (radius * radius);
    return area.toFixed(0);
}

// Create an array.
var radii = [10, 20, 30];

// Get the areas from the radii.
var areas = radii.map(AreaOfCircle);

document.write(areas);

// Output:
// 314,1257,2827
```

Example

The following example illustrates the use of the `thisArg` argument, which specifies an object to which the `this` keyword can refer.

```
// Define an object that contains a divisor property and
// a remainder function.
var obj = {
    divisor: 10,
    remainder: function (value) {
        return value % this.divisor;
    }
}

// Create an array.
var numbers = [6, 12, 25, 30];

// Get the remainders.
// The obj argument specifies the this value in the callback function.
var result = numbers.map(obj.remainder, obj);
document.write(result);

// Output:
// 6,2,5,0
```

Example

In the following example, a built-inJavaScript method is used as the callback function.

```
// Apply Math.sqrt(value) to each element in an array.
var numbers = [9, 16];
var result = numbers.map(Math.sqrt);

document.write(result);
// Output: 3,4
```

Example

The `map` method can be applied to a string. The following example illustrates this.

```
// Define the callback function.
function threeChars(value, index, str) {
    // Create a string that contains the previous, current,
    // and next character.
    return str.substring(index - 1, index + 2);
}

// Create a string.
var word = "Thursday";

// Apply the map method to the string.
// Each array element in the result contains a string that
// has the previous, current, and next character.
// The commented out statement shows an alternative syntax.
var result = [].map.call(word, threeChars);
// var result = Array.prototype.map.call(word, threeChars);

document.write(result);

// Output:
// Th,Thu,hur,urs,rsd,sda,day,ay
```

Requirements

Supported in the following document modes: Internet Explorer 9 standards, Internet Explorer 10 standards, and

Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards.

See Also

JavaScript Methods
Array Object
Using Arrays
filter Method (Array)
forEach Method (Array)
Hilo JavaScript sample app (Windows Store)

pop Method (Array) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Removes the last element from an array and returns it.

Syntax

```
arrayObj.pop( )
```

Remarks

The push and `pop` methods enable you to simulate a stack, which uses the principle of last in, first out (LIFO) to store data.

The required `arrayObj` reference is an `Array` object.

If the array is empty, `undefined` is returned.

Example

The following example illustrates the use of the `pop` method.

```
var number;
var my_array = new Array();

my_array.push (5, 6, 7);
my_array.push (8, 9);

number = my_array.pop();
while (number != undefined)
   {
   document.write (number + " ");
   number = my_array.pop();
   }

// Output: 9 8 7 6 5
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

See Also

push Method (Array)

push Method (Array) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Appends new elements to an array, and returns the new length of the array.

Syntax

```
arrayObj.push([item1 [item2 [. . . [itemN ]]]])
```

Parameters

`arrayObj`
Required. An `Array` object.

`item, item2,. . ., itemN`
Optional. New elements of the `Array`.

Remarks

The `push` and `pop` methods allow you to simulate a last in, first out stack.

The `push` method appends elements in the order in which they appear. If one of the arguments is an array, it is added as a single element. Use the `concat` method to join the elements from two or more arrays.

Example

The following example illustrates the use of the `push` method.

```
var number;
var my_array = new Array();

my_array.push (5, 6, 7);
my_array.push (8, 9);

number = my_array.pop();
while (number != undefined)
   {
   document.write (number + " ");
   number = my_array.pop();
   }

// Output:
// 9 8 7 6 5
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

See Also

concat Method (Array)
pop Method (Array)

reduce Method (Array) (JavaScript)

10/18/2017 • 5 min to read • Edit Online

Calls the specified callback function for all the elements in an array. The return value of the callback function is the accumulated result, and is provided as an argument in the next call to the callback function.

Syntax

```
array1.reduce(callbackfn[, initialValue])
```

Parameters

PARAMETER	DEFINITION
`array1`	Required. An array object.
`callbackfn`	Required. A function that accepts up to four arguments. The `reduce` method calls the `callbackfn` function one time for each element in the array.
`initialValue`	Optional. If `initialValue` is specified, it is used as the initial value to start the accumulation. The first call to the `callbackfn` function provides this value as an argument instead of an array value.

Return Value

The accumulated result from the last call to the callback function.

Exceptions

A `TypeError` exception is thrown when either of the following conditions is true:

- The `callbackfn` argument is not a function object.
- The array contains no elements and `initialValue` is not provided.

Remarks

If an `initialValue` is provided, the `reduce` method calls the `callbackfn` function one time for each element present in the array, in ascending index order. If an `initialValue` is not provided, the `reduce` method calls the `callbackfn` function on each element, starting with the second element.

The return value of the callback function is provided as the `previousValue` argument on the next call to the callback function. The return value of the last call to the callback function is the return value of the `reduce` method.

The callback function is not called for missing elements of the array.

> **NOTE**
> The reduceRight Method (Array) processes the elements in descending index order.

Callback Function Syntax

The syntax of the callback function is as follows:

```
function callbackfn(previousValue, currentValue, currentIndex, array1)
```

You can declare the callback function by using up to four parameters.

The following table lists the callback function parameters.

CALLBACK ARGUMENT	DEFINITION
`previousValue`	The value from the previous call to the callback function. If an `initialValue` is provided to the `reduce` method, the `previousValue` is `initialValue` the first time the function is called.
`currentValue`	The value of the current array element.
`currentIndex`	The numeric index of the current array element.
`array1`	The array object that contains the element.

First Call to the Callback Function

The first time the callback function is called, the values provided as arguments depend on whether the `reduce` method has an `initialValue` argument.

If an `initialValue` is provided to the reduce method:

- The `previousValue` argument is `initialValue`.
- The `currentValue` argument is the value of the first element present in the array.

  If an `initialValue` is not provided:

- The `previousValue` argument is the value of the first element present in the array.
- The `currentValue` argument is the value of the second element present in the array.

Modifying the Array Object

The array object can be modified by the callback function.

The following table describes the results of modifying the array object after the `reduce` method starts.

CONDITION AFTER THE `REDUCE` METHOD STARTS	ELEMENT PASSED TO CALLBACK FUNCTION?
Element is added beyond the original length of the array.	No.

CONDITION AFTER THE `REDUCE` METHOD STARTS	ELEMENT PASSED TO CALLBACK FUNCTION?
Element is added to fill in a missing element of the array.	Yes, if that index has not yet been passed to the callback function.
Element is changed.	Yes, if that element has not yet been passed to the callback function.
Element is deleted from the array.	No, unless that element has already been passed to the callback function.

Example

The following example concatenates array values into a string, separating the values with "::". Because no initial value is provided to the `reduce` method, the first call to the callback function has "abc" as the `previousValue` argument and "def" as the `currentValue` argument.

```
// Define the callback function.
function appendCurrent (previousValue, currentValue) {
    return previousValue + "::" + currentValue;
    }

// Create an array.
var elements = ["abc", "def", 123, 456];

// Call the reduce method, which calls the callback function
// for each array element.
var result = elements.reduce(appendCurrent);

document.write(result);

// Output:
//  abc::def::123::456
```

Example

The following example adds the values of an array after they have been rounded. The `reduce` method is called with an initial value of 0.

```
// Define the callback function.
function addRounded (previousValue, currentValue) {
    return previousValue + Math.round(currentValue);
    }

// Create an array.
var numbers = [10.9, 15.4, 0.5];

// Call the reduce method, starting with an initial value of 0.
var result = numbers.reduce(addRounded, 0);

document.write (result);
// Output: 27
```

Example

The following example adds the values in an array. The `currentIndex` and `array1` parameters are used in the callback function.

```
function addDigitValue(previousValue, currentDigit, currentIndex, array) {
    var exponent = (array.length - 1) - currentIndex;
    var digitValue = currentDigit * Math.pow(10, exponent);
    return previousValue + digitValue;
    }

var digits = [4, 1, 2, 5];

// Determine an integer that is computed from values in the array.
var result = digits.reduce(addDigitValue, 0);

document.write (result);
// Output: 4125
```

Example

The following example gets an array that contains only those values that are between 1 and 10 in another array. The initial value provided to the `reduce` method is an empty array.

```
function Process(previousArray, currentValue) {
    // If currentValue is between 1 and 10,
    // append currentValue to the array.
    var nextArray;
    if (currentValue >= 1 && currentValue <= 10)
        nextArray = previousArray.concat(currentValue);
    else
        nextArray = previousArray;

    // If this is not the last call by the reduce method,
    // the returned array is previousArray on the next call.
    // If this is the last call by the reduce method, the
    // returned array is the return value of the reduce method.
    return nextArray;
}

// Create an array.
var numbers = [20, 1, -5, 6, 50, 3];

// Call the reduce method, starting with an initial empty array.
var emptyArray = new Array();
var resultArray = numbers.reduce(Process, emptyArray);

document.write("result array=" + resultArray);

// Output:
// result array=1,6,3
```

Requirements

Supported in the following document modes: Internet Explorer 9 standards, Internet Explorer 10 standards, and Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards.

See Also

reduceRight Method (Array)

reduceRight Method (Array) (JavaScript)

10/18/2017 • 5 min to read • Edit Online

Calls the specified callback function for all the elements in an array, in descending order. The return value of the callback function is the accumulated result, and is provided as an argument in the next call to the callback function.

Syntax

```
array1.reduceRight(callbackfn[, initialValue])
```

Parameters

PARAMETER	DEFINITION
`array1`	Required. An array object.
`callbackfn`	Required. A function that accepts up to four arguments. The `reduceRight` method calls the `callbackfn` function one time for each element in the array.
`initialValue`	Optional. If `initialValue` is specified, it is used as the initial value to start the accumulation. The first call to the `callbackfn` function provides this value as an argument instead of an array value.

Return Value

An object that contains the accumulated result from the last call to the callback function.

Exceptions

A `TypeError` exception is thrown when either of the following conditions is true:

- The `callbackfn` argument is not a function object.
- The array contains no elements and `initialValue` is not provided.

Remarks

If an `initialValue` is provided, the `reduceRight` method calls the `callbackfn` function one time for each element in the array, in descending index order. If no `initialValue` is provided, the `reduceRight` method calls the `callbackfn` function on each element, starting with the second-to-last element, in descending index order.

The return value of the callback function is provided as the `previousValue` argument on the next call to the callback function. The return value of the last call to the callback function is the return value of the `reduceRight` method.

The callback function is not called for missing elements of the array.

To accumulate a result in ascending index order, use the reduce Method (Array).

Callback Function Syntax

The syntax of the callback function is as follows:

```
function callbackfn(previousValue, currentValue, currentIndex, array1)
```

You can declare the callback function by using up to four parameters.

The following table lists the callback function parameters.

CALLBACK ARGUMENT	DEFINITION
`previousValue`	The value from the previous call to the callback function. If an `initialValue` is provided to the `reduceRight` method, the `previousValue` is `initialValue` the first time the function is called.
`currentValue`	The value of the current array element.
`currentIndex`	The numeric index of the current array element.
`array1`	The array object that contains the element.

First Call to the Callback Function

The first time the callback function is called, the values provided as arguments depend on whether the `reduceRight` method has an `initialValue` argument.

If an `initialValue` is provided to the `reduceRight` method:

- The `previousValue` argument is `initialValue`.
- The `currentValue` argument is the value of the last element present in the array.

  If an `initialValue` is not provided:

- The `previousValue` argument is the value of the last element present in the array.
- The `currentValue` argument is the value of the second-to-last element present in the array.

Modifying the Array Object

The array object can be modified by the callback function.

The following table describes the results of modifying the array object after the `reduceRight` method starts.

CONDITION AFTER THE `REDUCERIGHT` METHOD STARTS	ELEMENT PASSED TO CALLBACK FUNCTION?
Element is added beyond the original length of the array.	No.
Element is added to fill in a missing element of the array.	Yes, if that index has not yet been passed to the callback function.
Element is changed.	Yes, if that element has not yet been passed to the callback function.

CONDITION AFTER THE `REDUCERIGHT` METHOD STARTS	ELEMENT PASSED TO CALLBACK FUNCTION?
Element is deleted from the array.	No, unless that element has already been passed to the callback function.

Example

The following example concatenates array values into a string, separating the values with "::". Because no initial value is provided to the `reduceRight` method, the first call to the callback function has 456 as the `previousValue` argument and 123 as the `currentValue` argument.

```
// Define the callback function.
function appendCurrent (previousValue, currentValue) {
    return previousValue + "::" + currentValue;
    }

// Create an array.
var elements = ["abc", "def", 123, 456];

// Call the reduceRight method, which calls the callback function
// for each array element, in descending index order.
var result = elements.reduceRight(appendCurrent);

document.write(result);

// Output:
//  456::123::def::abc
```

Example

The following example finds the sum of the squares of the array elements. The `reduceRight` method is called with an initial value of 0.

```
// Define the callback function.
function Process(previousValue, currentValue, index, array) {
    // Add the previous value to the current value squared.
    var nextValue = previousValue + (currentValue * currentValue);

    // If this is not the last call by the reduceRight method,
    // the return value is previousValue on the next call.
    // If this is the last call by the reduceRight method, the
    // return value is the return value of the reduceRight method.
    return nextValue;
}

// Create an array.
var numbers = [3, 4, 5];

// Call the reduceRight method with an initial value of 0.
var sumOfSquares = numbers.reduceRight(Process, 0);

document.write("sum of squares=" + sumOfSquares);

// Output:
//  sum of squares=50
```

Example

The following example gets those elements of an array whose values are between 1 and 10. The initial value

provided to the `reduceRight` method is an empty array.

```
function Process2(previousArray, currentValue) {
    // If currentValue is between 1 and 10,
    // append currentValue to the array.
    var nextArray;
    if (currentValue >= 1 && currentValue <= 10)
        nextArray = previousArray.concat(currentValue);
    else
        nextArray = previousArray;

    // If this is not the last call by the reduceRight method,
    // the returned array is previousArray on the next call.
    // If this is the last call by the reduceRight method, the
    // returned array is the return value of the reduceRight method.
    return nextArray;
}

// Create an array.
var numbers = [20, 1, -5, 6, 50, 3];

// Call the reduceRight method, starting with an empty array.
var emptyArray = new Array();
var resultArray = numbers.reduceRight(Process2, emptyArray);

document.write("result array=" + resultArray);

// Output:
//  result array=3,6,1
```

Example

The `reduceRight` method can be applied to a string. The following example shows how to use this method to reverse the characters in a string.

```
// Define the callback function.
function AppendToArray(previousValue, currentValue) {
    return previousValue + currentValue;
}

var word = "retupmoc";

// Create a string that reverses the characters of another string.
// The commented-out statement shows an alternative syntax.
var result = [].reduceRight.call(word, AppendToArray, "the ");
// var result = Array.prototype.reduceRight.call(word, AppendToArray, "the ");

document.write(result);

// Output:
// the computer
```

Requirements

Supported in the following document modes: Internet Explorer 9 standards, Internet Explorer 10 standards, and Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards.

See Also

reduce Method (Array)

reverse Method (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Reverses the elements in an `Array`.

Syntax

```
arrayObj.reverse()
```

Parameters

`arrayObj`
Required. Any `Array` object.

Return Value

The reversed array.

Remarks

The required `arrayObj` reference is an `Array` object.

The `reverse` method reverses the elements of an `Array` object in place. It does not create a new `Array` object during execution.

If the array is not contiguous, the `reverse` method creates elements in the array that fill the gaps in the array. Each of these created elements has the value `undefined`.

Example

The following example illustrates the use of the `reverse` method.

```
var arr = new Array(0,1,2,3,4);
var reverseArr = arr.reverse();
document.write(reverseArr);

// Output:
// 4,3,2,1,0
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

See Also

concat Method (Array)

shift Method (Array) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Removes the first element from an array and returns it.

Syntax

```
arrayObj.shift( )
```

Parameters

The required `arrayObj` reference is an `Array` object.

Return Value

Returns the element removed from the array.

Remarks

The following example illustrates the use of the `shift` method.

```
var arr = new Array(10, 11, 12);
while (arr.length > 0)
    {
    var i = arr.shift();
    document.write (i.toString() + " ");
    }

// Output:
// 10 11 12
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

See Also

unshift Method (Array)

slice Method (Array) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Returns a section of an array.

Syntax

```
arrayObj.slice(start, [end])
```

Parameters

`arrayObj`
Required. An `Array` object.

`start`
Required. The beginning of the specified portion of `arrayObj`.

`end`
Optional. The end of the specified portion of `arrayObj`.

Remarks

The `slice` method returns an `Array` object containing the specified portion of `arrayObj`.

The `slice` method copies up to, but not including, the element indicated by `end`. If `start` is negative, it is treated as `length` + `start`, where `length` is the length of the array. If `end` is negative, it is treated as `length` + `end` where `length` is the length of the array. If `end` is omitted, extraction continues to the end of `arrayObj`. If `end` occurs before `start`, no elements are copied to the new array.

Example

The following examples show how to use the `slice` method. In the first example, all but the last element of `myArray` is copied into `newArray`. In the second example, only the last two elements of `myArray` are copied into `newArray`.

```
var origArray = [3, 5, 7, 9];
var newArray = origArray. slice(0, -1);
document.write(origArray);
document.write("<br/>");
newArray = origArray. slice(-2);
document.write(newArray);

// Output:
// 3,5,7,9
// 7,9
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards,

Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

See Also

slice Method (String)
String Object

some Method (Array) (JavaScript)

10/18/2017 • 3 min to read • Edit Online

Determines whether the specified callback function returns `true` for any element of an array.

Syntax

```
array1.some(callbackfn[, thisArg])
```

Parameters

PARAMETER	DEFINITION
`array1`	Required. An array object.
`callbackfn`	Required. A function that accepts up to three arguments. The `some` method calls the `callbackfn` function for each element in `array1` until the `callbackfn` returns `true`, or until the end of the array.
`thisArg`	Optional. An object to which the `this` keyword can refer in the `callbackfn` function. If `thisArg` is omitted, `undefined` is used as the `this` value.

Return Value

`true` if the `callbackfn` function returns `true` for any array element; otherwise, `false`.

Exceptions

If the `callbackfn` argument is not a function object, a `TypeError` exception is thrown.

Remarks

The `some` method calls the `callbackfn` function on each array element, in ascending index order, until the `callbackfn` function returns `true`. If an element that causes `callbackfn` to return `true` is found, the `some` method immediately returns `true`. If the callback does not return `true` on any element, the `some` method returns `false`.

The callback function is not called for missing elements of the array.

In addition to array objects, the `some` method can be used by any object that has a `length` property and that has numerically indexed property names.

> **NOTE**
>
> You can use the every Method (Array) to check whether the callback function returns `true` for all the elements of an array.

Callback Function Syntax

The syntax of the callback function is as follows:

`function callbackfn(value, index, array1)`

You can declare the callback function with up to three parameters.

The following table lists the callback function parameters.

CALLBACK PARAMETER	DEFINITION
`value`	The value of the array element.
`index`	The numeric index of the array element.
`array1`	The array object that contains the element.

Modifying the Array Object

The array object can be modified by the callback function.

The following table describes the results of modifying the array object after the `some` method starts.

CONDITION AFTER THE `SOME` METHOD STARTS	ELEMENT PASSED TO CALLBACK FUNCTION?
Element is added beyond the original length of the array.	No.
Element is added to fill in a missing element of the array.	Yes, if that index has not yet been passed to the callback function.
Element is changed.	Yes, if that element has not yet been passed to the callback function.
Element is deleted from the array.	No, unless that element has already been passed to the callback function.

Example

The following example uses the `some` method to find out if any elements in an array are even.

```
// The callback function.
function CheckIfEven(value, index, ar) {
    if (value % 2 == 0)
        return true;
}

var numbers = [1, 15, 4, 10, 11, 22];

var evens = numbers.some(CheckIfEven);
document.write(evens);

// Output:
// true
```

Example

The following example shows how to use the `thisArg` parameter, which specifies an object to which the `this` keyword can refer. It checks whether any of the numbers in an array are outside the range provided by an object passed

```
// Create a function that returns true if the value is
// outside the range.
var isOutsideRange = function (value) {
    return value < this.minimum || value > this.maximum;
}

// Create an array of numbers.
var numbers = [6, 12, 16, 22, -12];

// The range object is to be the 'this' object.
var range = { minimum: 10, maximum: 20 };

document.write(numbers.some(isOutsideRange, range));

// Output: true
```

Requirements

Supported in the following document modes: Internet Explorer 9 standards, Internet Explorer 10 standards, and Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards.

See Also

every Method (Array)
filter Method (Array)
map Method (Array)

sort Method (Array) (JavaScript)

2/3/2018 • 1 min to read • Edit Online

Sorts an `Array`.

Syntax

```
arrayobj.sort(sortFunction)
```

Parameters

`arrayObj`
Required. Any `Array` object.

`sortFunction`
Optional. The name of the function used to determine the order of the elements. If omitted, the elements are sorted in ascending, ASCII character order.

Return Value

The sorted array.

Remarks

The `sort` method sorts the `Array` object in place; no new `Array` object is created during execution.

`sortFunction` takes two arguments and must return one of the following values:

- A negative value (less than 0) if the first argument passed is less than the second argument. The first argument is sorted to a lower index.
- Zero (0) if the two arguments are equivalent. The two arguments are sorted with respect to other elements in the array, but are not sorted with respect to each other.
- A positive value (greater than 0) if the first argument is greater than the second argument. The second argument is sorted to a lower index.

Example

The following example shows how to use the `sort` method.

```
var a = new Array(4, 11, 2, 10, 3, 1);

var b = a.sort();
document.write(b);
document.write("<br/>");

// This is ASCII character order.
// Output: 1,10,11,2,3,4)

// Sort the array elements with a function that compares array elements.
b = a.sort(CompareForSort);
document.write(b);
document.write("<br/>");
// Output: 1,2,3,4,10,11.

// Sorts array elements in ascending order numerically.
function CompareForSort(first, second)
{
    if (first == second)
        return 0;
    if (first < second)
        return -1;
    else
        return 1;
}
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

splice Method (Array) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Removes elements from an array and, if necessary, inserts new elements in their place, returning the deleted elements.

Syntax

```
arrayObj.splice(start, deleteCount, [item1[, item2[, . . . [,itemN]]]])
```

Parameters

`arrayObj`
Required. An `Array` object.

`start`
Required. The zero-based location in the array from which to start removing elements.

`deleteCount`
Required. The number of elements to remove.

`item1, item2,. . ., itemN`
Optional. Elements to insert into the array in place of the deleted elements.

Remarks

The `splice` method modifies `arrayObj` by removing the specified number of elements from position `start` and inserting new elements. The deleted elements are returned as a new `Array` object.

Example

The following code shows how to use the `splice` method.

```
var arr = new Array("4", "11", "2", "10", "3", "1");
arr.splice(2, 2, "21", "31");
document.write(arr);

// Output: 4,11,21,31,3,1
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

See Also

slice Method (Array)

toString Method (Array)

10/18/2017 • 1 min to read • Edit Online

Returns a string representation of an array.

Syntax

```
array.toString()
```

Parameters

`array`
Required. The array to represent as a string.

Return Value

The string representation of the array.

Remarks

The elements of an `Array` are converted to strings. The resulting strings are concatenated and separated by commas.

Example

The following example illustrates the use of the **toString** method with an array.

```
var arr = [1, 2, 3, 4];
var s = arr.toString();
document.write(s);

// Output: 1,2,3,4
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

unshift Method (Array) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Inserts new elements at the start of an array.

Syntax

```
arrayObj.unshift([item1[, item2 [, . . . [, itemN]]]])
```

Parameters

`arrayObj`
Required. An `Array` object.

`item1, item2,. . ., itemN`
Optional. Elements to insert at the start of the `Array`.

Remarks

The `unshift` method inserts elements into the start of an array, so they appear in the same order in which they appear in the argument list.

Example

The following example illustrates the use of the `unshift` method.

```
var ar = new Array();
ar.unshift(10, 11);
ar.unshift(12, 13, 14);
document.write(ar.toString());

// Output: 12,13,14,10,11
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

See Also

shift Method (Array)

valueOf Method (Array)

10/18/2017 • 1 min to read • Edit Online

Returns the primitive value of the specified object.

Syntax

```
array.valueOf()
```

Parameters

This method has no parameters.

Return Value

Returns the array instance.

Remarks

In the following example, the instantiated array object is the same as the return value of this method.

```
var arr = [1, 2, 3, 4];
var s = arr.valueOf();

if (arr === s)
document.write("same");
else
document.write("different");

// Output:
// same
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

values Method (Array) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Returns an iterator that returns the values of the array.

Syntax

```
arrayObj.values();
```

Parameters

`arrayObj`

Required. The array object.

Remarks

Example

The following example shows how to get the values of an array.

```
var v = ["a", "b", "c"].values();
// v.next().value == "a"
// v.next().value == "b"
// v.next().value == "c"
```

Requirements

Supported in Microsoft Edge (Edge browser). Also supported in Store apps (Microsoft Edge on Windows 10). See Version Information.

Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Not supported in Windows 8.1.

ArrayBuffer Object

10/18/2017 • 1 min to read • Edit Online

Represents a raw buffer of binary data, which is used to store data for the different typed arrays. `ArrayBuffers` cannot be read from or written to directly, but can be passed to a typed array or DataView Object to interpret the raw buffer as needed.

For more information about typed arrays, see Typed Arrays.

Syntax

```
arrayBuffer = new ArrayBuffer(length);
```

Parameters

`arrayBuffer`
Required. The variable name to which the `ArrayBuffer` object is assigned.

`length`
The length of the buffer. The contents of the ArrayBuffer are initialized to 0. If the requested number of bytes could not be allocated an exception is raised.

Properties

The following table lists the properties of the `ArrayBuffer` object.

PROPERTY	DESCRIPTION
byteLength Property	Read-only. The length of the ArrayBuffer (in bytes).

Functions

The following table lists the functions of the `ArrayBuffer` object.

PROPERTY	DESCRIPTION
ArrayBuffer.isView Function	Determines whether an object provides a view of the buffer.

Methods

The following table lists the methods of the `ArrayBuffer` object.

PROPERTY	DESCRIPTION
slice Method	Returns a section of an `ArrayBuffer`.

Example

The following example shows how to use an ArrayBuffer object to process the binary data acquired from an XMLHttpRequest. You can use a DataView Object to get the individual values.

```
var req = new XMLHttpRequest();
    req.open('GET', "http://www.example.com");
    req.responseType = "arraybuffer";
    req.send();

    req.onreadystatechange = function () {
        if (req.readyState === 4) {
            var buffer = req.response;
            var dataview = new DataView(buffer);
            var ints = new Int32Array(buffer.byteLength / 4);
            for (var i = 0; i < ints.length; i++) {
                ints[i] = dataview.getInt32(i * 4);
            }
        alert(ints[10]);
        }
    }
```

Remarks

For more information about using `XmlHttpRequest`, see XMLHttpRequest enhancements.

Requirements

Supported in the following document modes: Internet Explorer 10 standards and Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards.

byteLength Property (ArrayBuffer)

10/18/2017 • 1 min to read • Edit Online

Read-only. The length of the ArrayBuffer (in bytes).

Syntax

```
var arrayLength = arrayBuffer.byteLength;
```

Remarks

Example

The following example shows how to get the byte length of the ArrayBuffer.

```
var req = new XMLHttpRequest();
    req.open('GET', "http://www.example.com");
    req.responseType = "arraybuffer";
    req.send();

    req.onreadystatechange = function () {
        if (req.readyState === 4) {
            var buffer = req.response;

        alert(buffer.byteLength);
        }
    }
```

Requirements

Supported in the following document modes: Internet Explorer 10 standards and Internet Explorer 11 standards.
Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards.

ArrayBuffer.isView Function (ArrayBuffer)

10/18/2017 • 1 min to read • Edit Online

Determines whether an object provides a view of the buffer.

Syntax

```
ArrayBuffer.isView(object)
```

Parameters

`object`

Required. The object to test.

Return Value

`true` if either of the following is true:

- `object` is a `DataView` object.
- `object` is a typed array.

  Otherwise, the method returns `false`.

Remarks

Example

The following example illustrates the use of the `isView` function to test a typed array and a `DataView` object.

```
var uint = new UInt8ClampedArray(10);

if(console && console.log) {
    console.log( ArrayBuffer.isView(uint) ); // Outputs true
{
var dataView = new DataView(uint.buffer);

if(console && console.log) {
    console.log( ArrayBuffer.isView(dataView) ); // Outputs true.
}
```

Requirements

Supported in the Internet Explorer 11 standards document mode. Also supported in Windows Store apps (Windows 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards. Not supported in Windows 8 or Windows Phone 8.1.

See Also

Uint8ClampedArray Object

slice Method (ArrayBuffer)

10/18/2017 • 1 min to read • Edit Online

Returns a section of an ArrayBuffer.

Syntax

```
arrayBufferObj.slice(start, [end])
```

Parameters

`arrayBufferObj`
Required. The ArrayBuffer object the section will be copied from.

`start`
Required. The byte index of the beginning of the section to be copied.

`end`
Optional. The byte index of the end of the section to be copied.

Remarks

The `slice` method returns an ArrayBuffer object that contains the specified portion of `arrayBufferObj`.

The `slice` method copies up to, but not including, the byte indicated by `end`. If `start` or `end` is negative, the specified index is treated as `length` + `start` or `end`, respectively, where `length` is the length of the ArrayBuffer. If `end` is omitted, extraction continues to the end of `arrayBufferObj`. If `end` occurs before `start`, no bytes are copied to the new ArrayBuffer.

Example

The following examples show how to use the `slice` method.

```
var req = new XMLHttpRequest();
    req.open('GET', "http://www.example.com");
    req.responseType = "arraybuffer";
    req.send();

    req.onreadystatechange = function () {
        if (req.readyState === 4) {
            var buffer1 = req.response;
            var buffer2 = buffer1.slice(40, 48);
            var dataview = new DataView(buffer2);
            var ints = new Int32Array(buffer2.byteLength / 4);
            for (var i = 0; i < ints.length; i++) {
                ints[i] = dataview.getInt32(i * 4);
            }
        alert(ints[1]);
        }
    }
```

Requirements

Supported in the Internet Explorer 11 standards document mode. Also supported in Windows Store apps (Windows 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards. Not supported in Windows 8 or Windows Phone 8.1.

See Also

ArrayBuffer Object

arguments Object (JavaScript)

10/18/2017 • 1 min to read • Edit Online

An object representing the arguments to the currently executing function, and the functions that called it.

Syntax

```
[function.]arguments[n]
```

Parameters

function
Optional. The name of the currently executing `Function` object.

n
Required. The zero-based index to argument values passed to the `Function` object.

Remarks

You cannot explicitly create an **arguments** object. The **arguments** object only becomes available when a function begins execution. The **arguments** object of the function is not an array, but the individual arguments are accessed the same way array elements are accessed. The index *n* is actually a reference to one of the **0** ***n*** properties of the **arguments** object.

Example

The following example illustrates the use of the **arguments** object.

```
function ArgTest(a, b)
{
   var s = "";

   s += "Expected Arguments: " + ArgTest.length;
   s += "<br />";
   s += "Passed Arguments: " + arguments.length;
   s += "<br />";

   s += "The individual arguments are: "
   for (n = 0; n < arguments.length; n++)
   {
      s += ArgTest.arguments[n];
      s += " ";
   }

   document.write(s);
}

ArgTest(1, 2, "hello", new Date())

// Output:
// Expected Arguments: 2
// Passed Arguments: 4
// The individual arguments are: 1 2 hello Tues Jan 8 08:27:09 PST 20xx
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

See Also

0...n Properties (arguments)
callee Property (arguments)
length Property (arguments)

0...n Properties (arguments) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Returns the actual value of individual arguments from an **arguments** object returned by the **arguments** property of an executing function.

Syntax

```
[function.]arguments[[0|1|2|...|n]]
```

Parameters

function
Optional. The name of the currently executing `Function` object.

0, 1, 2, , *n*
Required. Non-negative integer in the range of 0 to *n* where 0 represents the first argument and *n* represents the final argument. The value of the final argument *n* is **arguments.length-1**.

Remarks

The values returned by the 0 . . . n properties are the actual values passed to the executing function. While not actually an array of arguments, the individual arguments that comprise the **arguments** object are accessed the same way that array elements are accessed.

Example

The following example illustrates the use of the **0 . . . *n*** properties of the **arguments** object. To fully understand the example, pass one or more arguments to the function:

```
function ArgTest(){
   var s = "";
   s += "The individual arguments are: "
   for (n = 0; n < arguments.length; n++){
      s += ArgTest.arguments[n];
      s += " ";
   }
   return(s);
}
document.write(ArgTest(1, 2, "hello", new Date()));
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

Applies To: arguments Object| Function Object

See Also

length Property (arguments)
length Property (Function)

callee Property (arguments) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Returns the `Function` object being executed, that is, the body text of the specified `Function` object.

Syntax

```
[function.]arguments.callee
```

Remarks

The optional *function* argument is the name of the currently executing `Function` object.

The `callee` property is a member of the **arguments** object that becomes available only when the associated function is executing.

The initial value of the `callee` property is the `Function` object being executed. This allows anonymous functions to be recursive.

Example

```
function factorial(n){
  if (n <= 0)
     return 1;
  else
     return n * arguments.callee(n - 1)
}
document.write(factorial(4));
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

Applies To: arguments Object| Function Object

See Also

caller Property (Function)

length Property (arguments) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Returns the actual number of arguments passed to a function by the caller.

Syntax

```
[function.]arguments.length
```

Remarks

The optional *function* argument is the name of the currently executing `Function` object.

The **length** property of the **arguments** object is initialized by the scripting engine to the actual number of arguments passed to a `Function` object when execution begins in that function.

Example

The following example illustrates the use of the **length** property of the **arguments** object. To fully understand the example, pass more arguments to the function than the 2 arguments expected:

```
function ArgTest(a, b){
   var s = "";

   s += "Expected Arguments: " + ArgTest.length;
   s += "<br />";
   s += "Passed Arguments: " + arguments.length;

   document.write (s);
}
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

Applies To: arguments Object| Function Object

See Also

arguments Property (Function)
length Property (Array)
length Property (String)

Boolean Object (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Creates a new Boolean value.

Syntax

```
boolObj = new Boolean([boolValue])
```

Parameters

`boolObj`
Required. The variable name to which the `Boolean` object is assigned.

`boolValue`
Optional. The initial Boolean value for the new object. If `boolvalue` is omitted, or is `false`, 0, `null`, `NaN`, or an empty string, the initial value of the Boolean object is `false`. Otherwise, the initial value is `true`.

Remarks

The `Boolean` object is a wrapper for the Boolean data type. JavaScript implicitly uses the `Boolean` object whenever a Boolean data type is converted to a `Boolean` object.

You rarely instantiate the `Boolean` object explicitly.

Properties

The following table lists the properties of the `Boolean` object.

PROPERTY	DESCRIPTION
constructor Property	Specifies the function that creates a Boolean.
prototype Property	Returns a reference to the prototype for a Boolean.

Methods

The following table lists the methods of the `Boolean` object.

METHOD	DESCRIPTION
toString Method	Returns a string representation of a Boolean.
valueOf Method	Gets a reference to the Boolean.

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

constructor Property (Boolean)

10/18/2017 • 1 min to read • Edit Online

Specifies the function that creates a Boolean.

Syntax

```
boolean.constructor([[value])
```

Parameters

`boolean`

The name of the Boolean.

`value`

Optional. Specifies the value of the Boolean. This can be the numbers 1 or 0, or the strings "true" or "false".

Remarks

The `constructor` property contains a reference to the function that constructs instances of the Boolean object.

Example

The following example illustrates the use of the constructor property.

```
var x = new Boolean("true");

if (x.constructor == Boolean)
    document.write("Object is a Boolelan.");

// Output:
// Object is a Boolean.
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

prototype Property (Boolean)

10/18/2017 • 1 min to read • Edit Online

Returns a reference to the prototype for a Boolean.

Syntax

```
boolean.prototype
```

Remarks

The `boolean` argument is the name of an object.

The `prototype` property provides a base set of functionality to a class of objects. New instances of an object "inherit" the behavior of the prototype assigned to that object. Properties and methods may be added to the prototype, but builtin objects may not be assigned a different prototype.

For example, to add a method to the `Boolean` object that returns the value of the largest element of the array, declare the function, add it to `Boolean.prototype`, and then use it.

```
function isFalse( ){
    if (this.toString() == "false")
         return true;
    else
        return false;
}
Boolean.prototype.isFalse = isFalse;
var bool = new Boolean(1);
document.write(bool.isFalse());

// Output:
// false
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

toString Method (Boolean) 1

10/18/2017 • 1 min to read • Edit Online

Returns a string representation of an object.

Syntax

```
boolean.toString()
```

Parameters

`boolean`
Required. An object for which to get a string representation.

Return Value

If the Boolean value is `true`, returns "true". Otherwise, returns "false".

Remarks

Example

The following example illustrates the use of the **toString** method.

```
var s = new Boolean(0);
document.write(s.toString());

// Output: false;
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

valueOf Method (Boolean)

10/18/2017 • 1 min to read • Edit Online

Returns the primitive value of the specified boolean.

Syntax

```
boolean.valueOf()
```

Return Value

The primitive value (true or false) of the Boolean.

Remarks

The following code shows how to use this method.

```
var bool = new Boolean("true");
var s = bool.valueOf();
document.write(s);

// Output: true
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

DataView Object

10/18/2017 • 3 min to read • Edit Online

You can use a DataView object to read and write the different kinds of binary data to any location in the ArrayBuffer.

Syntax

```
dataView = new DataView(buffer, byteOffset, byteLength);
```

Parameters

`dataView`
Required. The variable name to which the **DataView** object is assigned.

`buffer`
The ArrayBuffer that the DataView represents.

`byteOffset`
Optional. Specifies the offset in bytes from the start of the buffer at which the DataView should begin.

`byteLength`
Optional. Specifies the length (in bytes) of the section of the buffer that the DataView should represent.

Remarks

If both byteOffset and byteLength are omitted, the DataView spans the entire ArrayBuffer range. If the byteLength is omitted, the DataView extends from the given byteOffset until the end of the ArrayBuffer. If the given byteOffset and byteLength references an area beyond the end of the ArrayBuffer, an exception is raised.

Properties

The following table lists the properties of the `DataView` object.

PROPERTY	DESCRIPTION
buffer Property	Read-only. Gets the ArrayBuffer that is referenced by this view.
byteLength Property	Read-only. The length of this view from the start of its ArrayBuffer, in bytes, as fixed at construction time.
byteOffset Property	Read-only. The offset of this view from the start of its ArrayBuffer, in bytes, as fixed at construction time.

Methods

The following table lists the methods of the `DataView` object.

METHOD	DESCRIPTION
getInt8 Method	Gets the Int8 value at the specified byte offset from the start of the view.
getUint8 Method (DataView)	Gets the Uint8 value at the specified byte offset from the start of the view.
getInt16 Method (DataView)	Gets the Int16 value at the specified byte offset from the start of the view.
getUint16 Method (DataView)	Gets the Uint16 value at the specified byte offset from the start of the view.
getInt32 Method (DataView)	Gets the Int32 value at the specified byte offset from the start of the view.
getUint32 Method (DataView)	Gets the Uint32 value at the specified byte offset from the start of the view.
getFloat32 Method (DataView)	Gets the Float32 value at the specified byte offset from the start of the view.
getFloat64 Method (DataView)	Gets the Float64 value at the specified byte offset from the start of the view.
setInt8 Method (DataView)	Stores a Int8 value at the specified byte offset from the start of the view.
setUint8 Method (DataView)	Stores a Uint8 value at the specified byte offset from the start of the view.
setInt16 Method (DataView)	Stores a Int16 value at the specified byte offset from the start of the view.
setUint16 Method (DataView)	Stores a Uint16 value at the specified byte offset from the start of the view.
setInt32 Method (DataView)	Stores a Int32 value at the specified byte offset from the start of the view.
setUint32 Method (DataView)	Stores a Uint32 value at the specified byte offset from the start of the view.
setFloat32 Method (DataView)	Stores a Float32 value at the specified byte offset from the start of the view.
setFloat64 Method (DataView)	Stores a Float64 value at the specified byte offset from the start of the view.

Example

The following example shows how to use a DataView object to process the binary data acquired from an XmlHttpRequest:

```
var req = new XMLHttpRequest();
    req.open('GET', "http://www.example.com");
    req.responseType = "arraybuffer";
    req.send();

    req.onreadystatechange = function () {
        if (req.readyState === 4) {
            var buffer = req.response;
            var dataView = new DataView(buffer);
            var ints = new Int32Array(dataView.byteLength / 4);
            for (var i = 0; i < ints.length; i++) {
                ints[i] = dataview.getInt32(i * 4);
            }
        alert(ints[10]);
        }
    }
```

Requirements

Supported in the following document modes: Internet Explorer 10 standards and Internet Explorer 11 standards.
Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards.

buffer Property (DataView)

10/18/2017 • 1 min to read • Edit Online

Read-only. Gets the ArrayBuffer that is referenced by this view.

Syntax

```
var arrayBuffer = dataView.buffer;
```

Example

The following example shows how to get the length of the ArrayBuffer underlying the DataView.

```
var req = new XMLHttpRequest();
    req.open('GET', "http://www.example.com");
    req.responseType = "arraybuffer";
    req.send();

    req.onreadystatechange = function () {
        if (req.readyState === 4) {
            var buffer = req.response;
            var dataView = new DataView(buffer);
            alert(dataView.buffer.byteLength)
        }
    }
```

Requirements

Supported in the following document modes: Internet Explorer 10 standards and Internet Explorer 11 standards.
Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards.

byteLength Property (DataView)

10/18/2017 • 1 min to read • Edit Online

Read-only. The length of this view from the start of its ArrayBuffer, in bytes, as fixed at construction time.

Syntax

```
var byteLength = dataView.byteLength;
```

Example

The following example shows how get the length of a DataView from an XMLHttpRequest.

```
var req = new XMLHttpRequest();
    req.open('GET', "http://www.example.com");
    req.responseType = "arraybuffer";
    req.send();

    req.onreadystatechange = function () {
        if (req.readyState === 4) {
            var buffer = req.response;
            var dataView = new DataView(buffer);
            alert(dataView.byteLength);
        }
    }
```

Requirements

Supported in the following document modes: Internet Explorer 10 standards and Internet Explorer 11 standards.
Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards.

byteOffset Property (DataView)

10/18/2017 • 1 min to read • Edit Online

Read-only. The offset of this view from the start of its ArrayBuffer, in bytes, as fixed at construction time.

Syntax

```
var arrayOffset = dataView.byteOffset;
```

Example

The following example shows how to get the length of a DataView.

```
var req = new XMLHttpRequest();
    req.open('GET', "http://www.example.com");
    req.responseType = "arraybuffer";
    req.send();

    req.onreadystatechange = function () {
        if (req.readyState === 4) {
            var buffer = req.response;
            var dataView = new DataView(buffer);
            alert(dataView.byteOffset)
        }
    }
```

Requirements

Supported in the following document modes: Internet Explorer 10 standards and Internet Explorer 11 standards.
Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards.

getInt8 Method (DataView)

10/18/2017 • 1 min to read • Edit Online

Gets the Int8 value at the specified byte offset from the start of the view. There is no alignment constraint; multi-byte values may be fetched from any offset.

Syntax

```
var testInt = dataView.getInt8(byteOffset);
```

Parameters

`testInt`

Required. The Int8 value that is returned from the method.

`byteOffset`

The place in the buffer at which the value should be retrieved.

Remarks

These methods raise an exception if they would read beyond the end of the view.

Example

The following example shows how to get the first Int8 in the DataView.

```
var req = new XMLHttpRequest();
    req.open('GET', "http://www.example.com");
    req.responseType = "arraybuffer";
    req.send();

    req.onreadystatechange = function () {
        if (req.readyState === 4) {
            var buffer = req.response;
            var dataView = new DataView(buffer);
            alert(dataView.getInt8(0));
        }
    }
```

Requirements

Supported in the following document modes: Internet Explorer 10 standards and Internet Explorer 11 standards.
Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards.

getUint8 Method (DataView)

10/18/2017 • 1 min to read • Edit Online

Gets the Uint8 value at the specified byte offset from the start of the view. There is no alignment constraint; multi-byte values may be fetched from any offset.

Syntax

```
var testInt = dataView.getUint8(byteOffset);
```

Parameters

`testInt`

Required. The Uint8 value that is returned from the method.

`byteOffset`

The place in the buffer at which the value should be retrieved.

Remarks

These methods raise an exception if they would read beyond the end of the view.

Example

The following example shows how to get the first Uint8 in the DataView.

```
var req = new XMLHttpRequest();
    req.open('GET', "http://www.example.com");
    req.responseType = "arraybuffer";
    req.send();

    req.onreadystatechange = function () {
        if (req.readyState === 4) {
            var buffer = req.response;
            var dataView = new DataView(buffer);
            alert(dataView.getUint8(0));
        }
    }
```

Requirements

Supported in the following document modes: Internet Explorer 10 standards and Internet Explorer 11 standards.
Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards.

getInt16 Method (DataView)

10/18/2017 • 1 min to read • Edit Online

Gets the Int16 value at the specified byte offset from the start of the view. There is no alignment constraint; multi-byte values may be fetched from any offset.

Syntax

```
var testInt = dataView.getInt16(byteOffset, littleEndian);
```

Parameters

`testInt`

Required. The Int16 value that is returned from the method.

`byteOffset`

The place in the buffer at which the value should be retrieved.

`littleEndian`

Optional. If false or undefined, a big-endian value should be read, otherwise a little-endian value should be read.

Remarks

These methods raise an exception if they would read beyond the end of the view.

Example

The following example shows how to get the first Int16 in the DataView.

```
var req = new XMLHttpRequest();
    req.open('GET', "http://www.example.com");
    req.responseType = "arraybuffer";
    req.send();

    req.onreadystatechange = function () {
        if (req.readyState === 4) {
            var buffer = req.response;
            var dataView = new DataView(buffer);
            alert(dataView.getInt16(0));
        }
    }
```

Requirements

Supported in the following document modes: Internet Explorer 10 standards and Internet Explorer 11 standards.
Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards.

getUint16 Method (DataView)

10/18/2017 • 1 min to read • Edit Online

Gets the Uint16 value at the specified byte offset from the start of the view. There is no alignment constraint; multi-byte values may be fetched from any offset.

Syntax

```
var testInt = dataView.getUint16(byteOffset, littleEndian);
```

Parameters

`testInt`

Required. The Uint16 value that is returned from the method.

`byteOffset`

The place in the buffer at which the value should be retrieved.

`littleEndian`

Optional. If false or undefined, a big-endian value should be read, otherwise a little-endian value should be read.

Remarks

These methods raise an exception if they would read beyond the end of the view.

Example

The following example shows how to get the first Uint16 in the DataView.

```
var req = new XMLHttpRequest();
    req.open('GET', "http://www.example.com");
    req.responseType = "arraybuffer";
    req.send();

    req.onreadystatechange = function () {
        if (req.readyState === 4) {
            var buffer = req.response;
            var dataView = new DataView(buffer);
            alert(dataView.getUint16(0));
        }
    }
```

Requirements

Supported in the following document modes: Internet Explorer 10 standards and Internet Explorer 11 standards.
Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards.

getInt32 Method (DataView)

10/18/2017 • 1 min to read • Edit Online

Gets the Int32 value at the specified byte offset from the start of the view. There is no alignment constraint; multi-byte values may be fetched from any offset.

Syntax

```
var testInt = dataView.getInt32(byteOffset, littleEndian);
```

Parameters

`testInt`

Required. The Int32 value that is returned from the method.

`byteOffset`

The place in the buffer at which the value should be retrieved.

`littleEndian`

Optional. If false or undefined, a big-endian value should be read, otherwise a little-endian value should be read.

Remarks

These methods raise an exception if they would read beyond the end of the view.

Example

The following example shows how to get the first Int32 in the DataView.

```
var req = new XMLHttpRequest();
    req.open('GET', "http://www.example.com");
    req.responseType = "arraybuffer";
    req.send();

    req.onreadystatechange = function () {
        if (req.readyState === 4) {
            var buffer = req.response;
            var dataView = new DataView(buffer);
            alert(dataView.getInt32(0));
        }
    }
```

Requirements

Supported in the following document modes: Internet Explorer 10 standards and Internet Explorer 11 standards.
Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards.

getUint32 Method (DataView)

10/18/2017 • 1 min to read • Edit Online

Gets the Uint32 value at the specified byte offset from the start of the view. There is no alignment constraint; multi-byte values may be fetched from any offset.

Syntax

```
var testInt = dataView.get Uint32 (byteOffset, littleEndian);
```

Parameters

`testInt`

Required. The Uint32 value that is returned from the method.

`byteOffset`

The place in the buffer at which the value should be retrieved.

`littleEndian`

Optional. If false or undefined, a big-endian value should be read, otherwise a little-endian value should be read.

Remarks

These methods raise an exception if they would read beyond the end of the view.

Example

The following example shows how to get the first Uint32 in the DataView.

```
var req = new XMLHttpRequest();
    req.open('GET', "http://www.example.com");
    req.responseType = "arraybuffer";
    req.send();

    req.onreadystatechange = function () {
        if (req.readyState === 4) {
            var buffer = req.response;
            var dataView = new DataView(buffer);
            alert(dataView.getUint32(0));
        }
    }
```

Requirements

Supported in the following document modes: Internet Explorer 10 standards and Internet Explorer 11 standards.
Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards.

getFloat32 Method (DataView)

10/18/2017 • 1 min to read • Edit Online

Gets the Float32 value at the specified byte offset from the start of the view. There is no alignment constraint; multi-byte values may be fetched from any offset.

Syntax

```
var testFloat = dataView.getFloat32(byteOffset, littleEndian);
```

Parameters

`testFloat`

Required. The Float32 value that is returned from the method.

`byteOffset`

The place in the buffer at which the value should be retrieved.

`littleEndian`

Optional. If false or undefined, a big-endian value should be read, otherwise a little-endian value should be read.

Remarks

These methods raise an exception if they would read beyond the end of the view.

Example

The following example shows how to get the first Float32 in the DataView.

```
var req = new XMLHttpRequest();
    req.open('GET', "http://www.example.com");
    req.responseType = "arraybuffer";
    req.send();

    req.onreadystatechange = function () {
        if (req.readyState === 4) {
            var buffer = req.response;
            var dataView = new DataView(buffer);
            alert(dataView.getFloat32(0));
        }
    }
```

Requirements

Supported in the following document modes: Internet Explorer 10 standards and Internet Explorer 11 standards.
Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards.

getFloat64 Method (DataView)

10/18/2017 • 1 min to read • Edit Online

Gets the Float64 value at the specified byte offset from the start of the view. There is no alignment constraint; multi-byte values may be fetched from any offset.

Syntax

```
var testFloat = dataView.getFloat64(byteOffset, littleEndian);
```

Parameters

`testFloat`

Required. The Float64 value that is returned from the method.

`byteOffset`

The place in the buffer at which the value should be retrieved.

`littleEndian`

Optional. If false or undefined, a big-endian value should be read, otherwise a little-endian value should be read.

Remarks

These methods raise an exception if they would read beyond the end of the view.

Example

The following example shows how to get the first Float64 in the DataView.

```
var req = new XMLHttpRequest();
    req.open('GET', "http://www.example.com");
    req.responseType = "arraybuffer";
    req.send();

    req.onreadystatechange = function () {
        if (req.readyState === 4) {
            var buffer = req.response;
            var dataView = new DataView(buffer);
            alert(dataView.getFloat64(0));
        }
    }
```

Requirements

Supported in the following document modes: Internet Explorer 10 standards and Internet Explorer 11 standards.
Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards.

setInt8 Method (DataView)

10/18/2017 • 1 min to read • Edit Online

Stores an Int8 value at the specified byte offset from the start of the view.

Syntax

```
dataView.setInt8(byteOffset, value);
```

Parameters

`byteOffset`

The place in the buffer at which the value should be set.

`value`

The value to set.

Remarks

These methods raise an exception if they would write beyond the end of the view.

Example

The following example shows how to set the first Int8 in the DataView.

```
var req = new XMLHttpRequest();
    req.open('GET', "http://www.example.com");
    req.responseType = "arraybuffer";
    req.send();

    req.onreadystatechange = function () {
        if (req.readyState === 4) {
            var buffer = req.response;
            var dataView = new DataView(buffer);
            dataView.setInt8(0, 9);
        }
    }
```

Requirements

Supported in the following document modes: Internet Explorer 10 standards and Internet Explorer 11 standards.
Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards.

setUint8 Method (DataView)

10/18/2017 • 1 min to read • Edit Online

Stores a Uint8 value at the specified byte offset from the start of the view.

Syntax

```
dataView.setUint8(byteOffset, value);
```

Parameters

`byteOffset`

The place in the buffer at which the value should be set.

`value`

The value to set.

Remarks

These methods raise an exception if they would write beyond the end of the view.

Example

The following example shows how to set the first Uint8 in the DataView.

```
var req = new XMLHttpRequest();
    req.open('GET', "http://www.example.com");
    req.responseType = "arraybuffer";
    req.send();

    req.onreadystatechange = function () {
        if (req.readyState === 4) {
            var buffer = req.response;
            var dataView = new DataView(buffer);
            dataView.setUint8(0, 9);
        }
    }
```

Requirements

Supported in the following document modes: Internet Explorer 10 standards and Internet Explorer 11 standards.
Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards.

setInt16 Method (DataView)

10/18/2017 • 1 min to read • Edit Online

Sets the Int16 value at the specified byte offset from the start of the view. There is no alignment constraint; multi-byte values may be set at any offset.

Syntax

```
dataView.setInt16(byteOffset, value, littleEndian);
```

Parameters

`byteOffset`
The place in the buffer at which the value should be retrieved.

`value`
The value to set.

`littleEndian`
Optional. If false or undefined, a big-endian value should be written, otherwise a little-endian value should be written.

Remarks

These methods raise an exception if they would write beyond the end of the view.

Example

The following example shows how to set the first Int16 in the DataView.

```
var req = new XMLHttpRequest();
    req.open('GET', "http://www.example.com");
    req.responseType = "arraybuffer";
    req.send();

    req.onreadystatechange = function () {
        if (req.readyState === 4) {
            var buffer = req.response;
            var dataView = new DataView(buffer);
            dataView.setInt16(0, 9);
        }
    }
```

Requirements

Supported in the following document modes: Internet Explorer 10 standards and Internet Explorer 11 standards.
Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards.

setUint16 Method (DataView)

10/18/2017 • 1 min to read • Edit Online

Sets the Uint16 value at the specified byte offset from the start of the view. There is no alignment constraint; multi-byte values may be set at any offset.

Syntax

```
dataView.setUint16(byteOffset, value, littleEndian);
```

Parameters

`testInt`

Required. The Uint16 value that is returned from the method.

`value`

The value to set.

`byteOffset`

The place in the buffer at which the value should be retrieved.

`littleEndian`

Optional. If false or undefined, a big-endian value should be written, otherwise a little-endian value should be written.

Remarks

These methods raise an exception if they would write beyond the end of the view.

Example

The following example shows how to set the first Uint16 in the DataView.

```
var req = new XMLHttpRequest();
    req.open('GET', "http://www.example.com");
    req.responseType = "arraybuffer";
    req.send();

    req.onreadystatechange = function () {
        if (req.readyState === 4) {
            var buffer = req.response;
            var dataView = new DataView(buffer);
            dataView.setUint16(0, 9);
        }
    }
```

Requirements

Supported in the following document modes: Internet Explorer 10 standards and Internet Explorer 11 standards.
Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7

standards, Internet Explorer 8 standards, Internet Explorer 9 standards.

setInt32 Method (DataView)

10/18/2017 • 1 min to read • Edit Online

Sets the Int32 value at the specified byte offset from the start of the view. There is no alignment constraint; multi-byte values may be set at any offset.

Syntax

```
dataView.setInt32 (byteOffset, value, littleEndian);
```

Parameters

`byteOffset`
The place in the buffer at which the value should be retrieved.

`value`
The value to set.

`littleEndian`
Optional. If false or undefined, a big-endian value should be written, otherwise a little-endian value should be written.

Remarks

These methods raise an exception if they would write beyond the end of the view.

Example

The following example shows how to set the first Int32 in the DataView.

```
var req = new XMLHttpRequest();
    req.open('GET', "http://www.example.com");
    req.responseType = "arraybuffer";
    req.send();

    req.onreadystatechange = function () {
        if (req.readyState === 4) {
            var buffer = req.response;
            var dataView = new DataView(buffer);
            dataView.setInt32(0, 9);
        }
    }
```

Requirements

Supported in the following document modes: Internet Explorer 10 standards and Internet Explorer 11 standards.
Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards.

setUint32 Method (DataView)

10/18/2017 • 1 min to read • Edit Online

Sets the Uint32 value at the specified byte offset from the start of the view. There is no alignment constraint; multi-byte values may be set at any offset.

Syntax

```
dataView.setUint32 (byteOffset, value, littleEndian);
```

Parameters

`byteOffset`

The place in the buffer at which the value should be retrieved.

`value`

The value to set.

`littleEndian`

Optional. If false or undefined, a big-endian value should be written, otherwise a little-endian value should be written.

Remarks

These methods raise an exception if they would write beyond the end of the view.

Example

The following example shows how to set the first Uint32 in the DataView.

```
var req = new XMLHttpRequest();
    req.open('GET', "http://www.example.com");
    req.responseType = "arraybuffer";
    req.send();

    req.onreadystatechange = function () {
        if (req.readyState === 4) {
            var buffer = req.response;
            var dataView = new DataView(buffer);
            dataView.setUint32(0, 9);
        }
    }
```

Requirements

Supported in the following document modes: Internet Explorer 10 standards and Internet Explorer 11 standards.
Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards.

setFloat32 Method (DataView)

10/18/2017 • 1 min to read • Edit Online

Sets the Float32 value at the specified byte offset from the start of the view. There is no alignment constraint; multi-byte values may be set at any offset.

Syntax

```
dataView.setFloat32 (byteOffset, value, littleEndian);
```

Parameters

`byteOffset`

The place in the buffer at which the value should be retrieved.

`value`

The value to set.

`littleEndian`

Optional. If false or undefined, a big-endian value should be written, otherwise a little-endian value should be written.

Remarks

These methods raise an exception if they would write beyond the end of the view.

Example

The following example shows how to set the first Float32 in the DataView.

```
var req = new XMLHttpRequest();
    req.open('GET', "http://www.example.com");
    req.responseType = "arraybuffer";
    req.send();

    req.onreadystatechange = function () {
        if (req.readyState === 4) {
            var buffer = req.response;
            var dataView = new DataView(buffer);
            dataView.setFloat32(0, 9.1);
        }
    }
```

Requirements

Supported in the following document modes: Internet Explorer 10 standards and Internet Explorer 11 standards.
Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards.

setFloat64 Method (DataView)

10/18/2017 • 1 min to read • Edit Online

Sets the Float64 value at the specified byte offset from the start of the view. There is no alignment constraint; multi-byte values may be set at any offset.

Syntax

```
dataView.setFloat64 (byteOffset, value, littleEndian);
```

Parameters

`byteOffset`

The place in the buffer at which the value should be retrieved.

`value`

The value to set.

`littleEndian`

Optional. If false or undefined, a big-endian value should be written, otherwise a little-endian value should be written.

Remarks

These methods raise an exception if they would write beyond the end of the view.

Example

The following example shows how to set the first Float64 in the DataView.

```
var req = new XMLHttpRequest();
    req.open('GET', "http://www.example.com");
    req.responseType = "arraybuffer";
    req.send();

    req.onreadystatechange = function () {
        if (req.readyState === 4) {
            var buffer = req.response;
            var dataView = new DataView(buffer);
            dataView.setFloat64(0, 9.1);
        }
    }
```

Requirements

Supported in the following document modes: Internet Explorer 10 standards and Internet Explorer 11 standards.
Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards.

Date Object (JavaScript)

10/18/2017 • 5 min to read • Edit Online

Enables basic storage and retrieval of dates and times.

Syntax

```
      dateObj = new Date()
dateObj = new Date(dateVal)
dateObj = new Date(year, month, date[, hours[, minutes[, seconds[,ms]]]])
```

Parameters

`dateObj`
Required. The variable name to which the `Date` object is assigned.

`dateVal`
Required. If a numeric value, `dateVal` represents the number of milliseconds in Universal Coordinated Time between the specified date and midnight January 1, 1970. If a string, `dateVal` is parsed according to the rules in Date and Time Strings. The `dateVal` argument can also be a VT_DATE value as returned from some ActiveX objects.

`year`
Required. The full year, for example, 1976 (and not 76).

`month`
Required. The month as an integer between 0 and 11 (January to December).

`date`
Required. The date as an integer between 1 and 31.

`hours`
Optional. Must be supplied if `minutes` is supplied. An integer from 0 to 23 (midnight to 11pm) that specifies the hour.

minutes
Optional. Must be supplied if `seconds` is supplied. An integer from 0 to 59 that specifies the minutes.

`seconds`
Optional. Must be supplied if `milliseconds` is supplied. An integer from 0 to 59 that specifies the seconds.

`ms`
Optional. An integer from 0 to 999 that specifies the milliseconds.

Remarks

A `Date` object contains a number representing a particular instant in time to within a millisecond. If the value of an argument is greater than its range or is a negative number, other stored values are modified accordingly. For example, if you specify 150 seconds, JavaScript redefines that number as two minutes and 30 seconds.

If the number is `NaN`, the object does not represent a specific instant of time. If you pass no parameters to the `Date` object, it is initialized to the current time (UTC). A value must be given to the object before you can use it.

The range of dates that can be represented in a `Date` object is approximately 285,616 years on either side of January 1, 1970.

See Calculating Dates and Times (JavaScript) for more information about how to use the `Date` object and related methods.

Example

The following example illustrates the use of the `Date` object.

```
var dateString = "Today's date is: ";

var newDate = new Date();

// Get the month, day, and year.
dateString += (newDate.getMonth() + 1) + "/";
dateString += newDate.getDate() + "/";
dateString += newDate.getFullYear();

document.write(dateString);

// Output: Today's date is: <date>
```

Requirements

The `Date object` was introduced in Internet Explorer before Internet Explorer 6. Some members in the following lists were introduced in later versions. For more information, see Version Information or the documentation for the individual members.

Properties

The following table lists properties of the `Date Object`.

PROPERTY	DESCRIPTION
constructor Property	Specifies the function that creates an object.
prototype Property	Returns a reference to the prototype for a class of objects.

Functions

The following table lists functions of the `Date Object`.

FUNCTIONS	DESCRIPTION
Date.now Function	Returns the number of milliseconds between January 1, 1970, and the current date and time.
Date.parse Function	Parses a string containing a date, and returns the number of milliseconds between that date and midnight, January 1, 1970.

FUNCTIONS	DESCRIPTION
Date.UTC Function	Returns the number of milliseconds between midnight, January 1, 1970 Universal Coordinated Time (UTC) (or GMT) and the supplied date.

Methods

The following table lists methods of the `Date Object`.

METHOD	DESCRIPTION
getDate Method	Returns the day-of-the-month value using local time.
getDay Method	Returns the day-of-the-week value using local time.
getFullYear Method	Returns the year value using local time.
getHours Method	Returns the hours value using local time.
getMilliseconds Method	Returns the milliseconds value using local time.
getMinutes Method	Returns the minutes value using local time.
getMonth Method	Returns the month value using local time.
getSeconds Method	Returns seconds value using local time.
getTime Method	Returns the time value in a `Date` Object as the number of milliseconds since midnight January 1, 1970.
getTimezoneOffset Method	Returns the difference in minutes between the time on the host computer and Universal Coordinated Time (UTC).
getUTCDate Method	Returns the day-of-the-month value using UTC.
getUTCDay Method	Returns the day-of-the-week value using UTC.
getUTCFullYear Method	Returns the year value using UTC.
getUTCHours Method	Returns the hours value using UTC.
getUTCMilliseconds Method	Returns the milliseconds value using UTC.
getUTCMinutes Method	Returns the minutes value using UTC.
getUTCMonth Method	Returns the month value using UTC.
getUTCSeconds Method	Returns the seconds value using UTC.
getVarDate Method	Returns the VT_DATE value in a `Date` object.

METHOD	DESCRIPTION
getYear Method	Returns the year value .
hasOwnProperty Method	Returns a Boolean value that indicates whether an object has a property with the specified name.
isPrototypeOf Method	Returns a Boolean value that indicates whether an object exists in another object's prototype chain.
propertyIsEnumerable Method	Returns a Boolean value that indicates whether a specified property is part of an object and whether it is enumerable.
setDate Method	Sets the numeric day of the month using local time.
setFullYear Method	Sets the year value using local time.
setHours Method	Sets the hours value using local time.
setMilliseconds Method	Sets the milliseconds value using local time.
setMinutes Method	Sets the minutes value using local time.
setMonth Method	Sets the month value using local time.
setSeconds Method	Sets the seconds value using local time.
setTime Method	Sets the date and time value in the `Date` object.
setUTCDate Method	Sets the numeric day of the month using UTC.
setUTCFullYear Method	Sets the year value using UTC.
setUTCHours Method	Sets the hours value using UTC.
setUTCMilliseconds Method	Sets the milliseconds value using UTC.
setUTCMinutes Method	Sets the minutes value using UTC.
setUTCMonth Method	Sets the month value using UTC.
setUTCSeconds Method	Sets the seconds value using UTC.
setYear Method	Sets the year value using local time.
toDateString Method	Returns a date as a string value.
toGMTString Method	Returns a date converted to a string using Greenwich Mean Time (GMT).
toISOString Method	Returns a date as a string value in ISO format.

METHOD	DESCRIPTION
toJSON Method	Used to transform data of an object type before the JSON serialization.
toLocaleDateString Method	Returns a date as a string value appropriate to the host environment's current locale.
toLocaleString Method	Returns an object converted to a string using the current locale.
toLocaleTimeString Method	Returns a time as a string value appropriate to the host environment's current locale.
toString Method	Returns a string representation of an object.
toTimeString Method	Returns a time as a string value.
toUTCString Method	Returns a date converted to a string using UTC.
valueOf Method	Returns the primitive value of the specified object.

See Also

Calculating Dates and Times (JavaScript)
Date and Time Strings

constructor Property (Date)

10/18/2017 • 1 min to read • Edit Online

Specifies the function that creates a date.

Syntax

```
date.constructor
```

Remarks

The required `date` is the name of a date object.

The `constructor` property is a member of the prototype of every object that has a prototype. The `constructor` property contains a reference to the function that constructs instances of that particular object.

Example

The following example illustrates the use of the constructor property.

```
var x = new Date("Hi");

if (x.constructor == Date)
    document.write("Object is a date.");

// Output:
// Object is a date.
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

prototype Property (Date)

10/18/2017 • 1 min to read • Edit Online

Returns a reference to the prototype for a date.

Syntax

```
date.prototype
```

Remarks

The `date` argument is the name of an object.

Use the `prototype` property to provide a base set of functionality to a Date. New instances of an object "inherit" the behavior of the prototype assigned to that object.

For example, to add a method to the `Date` object that returns the value of the largest element of the array, declare the function, add it to `Date.prototype`, and then use it.

```
function max( ){
    var max = new Date();
    max.setFullYear(2200, 01, 01);
    return max;
}
Date.prototype.maxDate = max;
var myDate = new Date();

if (myDate < myDate.maxDate())
    document.write("today isn't the max");
else if (myDate == myDate.maxDate())
    document.write("today is the max");

// Output:
// today isn't the max
```

All intrinsic JavaScript objects have a `prototype` property that is read-only. Properties and methods may be added to the prototype, but the object may not be assigned a different prototype. However, user-defined objects may be assigned a new prototype.

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

Date.now Function (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Gets the current date and time.

Syntax

```
Date.now()
```

Return Value

The number of milliseconds between midnight, January 1, 1970, and the current date and time.

Remarks

The getTime method returns the number of milliseconds between January 1, 1970, and a specified date.

For information about how to calculate elapsed time and compare dates, see Calculating Dates and Times (JavaScript).

Example

The following example illustrates the use of the `now` method.

```
var start = Date.now();
var response = prompt("What is your name?", "");
var end = Date.now();
var elapsed = (end - start) / 1000;
document.write("You took " + elapsed + " seconds" + " to type: " + response);

// Output:
// You took <seconds> seconds to type: <name>
```

Requirements

Not supported in installed versions earlier than Internet Explorer 9. However, it is supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards. Also supported in Windows 8.x Store apps.

See Also

getTime Method (Date)
Date Object
Calculating Dates and Times (JavaScript)
JavaScript Methods

Date.parse Function (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Parses a string containing a date, and returns the number of milliseconds between that date and midnight, January 1, 1970.

Syntax

```
Date.parse(dateVal)
```

Remarks

The required `dateVal` argument is either a string containing a date or a VT_DATE value retrieved from an ActiveX object or other object. For information about date strings that the `Date.parse` function can parse. see Date and Time Strings.

The `Date.parse` function returns an integer value representing the number of milliseconds between midnight, January 1, 1970 and the date supplied in `dateVal`.

Example

The following example illustrates the use of the `Date.parse` function.

```
var dateString = "November 1, 1997 10:15 AM";
var mSec = Date.parse(dateString);
document.write(mSec);
// Output: 878404500000
```

Example

The following example returns the difference between the date provided and 1/1/1970.

```
var minMilli = 1000 * 60;
var hrMilli = minMilli * 60;
var dyMilli = hrMilli * 24;

var testDate = new Date("June 1, 1990");
var ms = Date.parse(testDate);
var days = Math.round(ms / dyMilli);

var dateStr = "";
dateStr += "There are " + days + " days ";
dateStr += "between 01/01/1970 and " + testDate;
document.write(dateStr);

// Output: There are 7456 days between 01/01/1970 and Fri Jun 1 00:00:00 PDT 1990
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards,

Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

See Also

getDate Method (Date)

Date.UTC Function (JavaScript)

10/18/2017 • 2 min to read • Edit Online

Returns the number of milliseconds between midnight, January 1, 1970 Universal Coordinated Time (UTC) (or GMT) and the specified date.

Syntax

```
Date.UTC(year, month, day[, hours[, minutes[, seconds[,ms]]]])
```

Parameters

`year`
Required. The full year designation is required for cross-century date accuracy. If `year` is between 0 and 99 is used, then `year` is assumed to be 1900 + `year`.

`month`
Required. The month as an integer between 0 and 11 (January to December).

`day`
Required. The date as an integer between 1 and 31.

`hours`
Optional. Must be supplied if `minutes` is supplied. An integer from 0 to 23 (midnight to 11pm) that specifies the hour.

`minutes`
Optional. Must be supplied if `seconds` is supplied. An integer from 0 to 59 that specifies the minutes.

`seconds`
Optional. Must be supplied if `milliseconds` is supplied. An integer from 0 to 59 that specifies the seconds.

`ms`
Optional. An integer from 0 to 999 that specifies the milliseconds.

Remarks

The `Date.UTC` function returns the number of milliseconds between midnight, January 1, 1970 UTC and the supplied date. This return value can be used in the `setTime` method and in the `Date` object constructor. If the value of an argument is greater than its range, or is a negative number, other stored values are modified accordingly. For example, if you specify 150 seconds, JavaScript redefines that number as two minutes and 30 seconds.

The difference between the `Date.UTC` function and the `Date` object constructor that accepts a date is that the `Date.UTC` function assumes UTC, and the `Date` object constructor assumes local time.

Example

The following example illustrates the use of the `Date.UTC` function.

```
// Determine the milliseconds per day.
 var MinMilli = 1000 * 60;
var HrMilli = MinMilli * 60;
var DyMilli = HrMilli * 24;

var date = new Date("June 1, 1990");
var year = date.getFullYear();
var month = date.getMonth();
var day = date.getDay();

var newDay = new Date("January 16, 2020");
var yeartoday = newDay.getUTCFullYear();
var monthtoday = newDay.getUTCMonth();
var dayofmonthtoday = newDay.getUTCDate();

// Get the milliseconds since 1/1/1970 UTC.
var t1 = Date.UTC(year, month - 1, day)
var t2 = Date.UTC(yeartoday, monthtoday, dayofmonthtoday);

// Determine the difference in days.
var days = (t2 - t1) / DyMilli;

document.write(days);
// Output: 10848
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

See Also

setTime Method (Date)

getDate Method (Date) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Gets the day-of-the-month, using local time.

Syntax

```
dateObj.getDate()
```

Parameters

The required `dateObj` reference is a `Date` object.

Return Value

An integer between 1 and 31 that represents the day-of-the-month.

Remarks

To get the day-of-the-month using Universal Coordinated Time (UTC), use the `getUTCDate` method.

Example

The following example illustrates the use of the `getDate` method.

```
var date = new Date("Jan 01, 2001");
var str = "Today's date is: ";
   str += (date.getMonth() + 1) + "/";
   str += date.getDate() + "/";
   str += date.getFullYear();
document.write(str);

// Output: Today's date is: 1/1/2001
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

Applies To: Date Object

See Also

getUTCDate Method (Date)
setDate Method (Date)
setUTCDate Method (Date)

getDay Method (Date) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Gets the day of the week, using local time.

Syntax

```
dateObj. getDay()
```

Parameters

The required `dateObj` reference is a `Date` object.

Return Value

An integer between 0 and 6 representing the day of the week (Sunday is 0, Saturday is 6).

Remarks

To get the day using Universal Coordinated Time (UTC), use the `getUTCDay` method.

The following example shows how to use the `getDay` method.

```
var date = new Date("Saturday, February 9, 2008");
day = date.getDay();
document.write(day);

// Output: 6
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

Applies To: Date Object

See Also

getUTCDay Method (Date)

getFullYear Method (Date) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Gets the year, using local time.

Syntax

```
dateObj.getFullYear()
```

Parameters

The required `dateObj` reference is a `Date` object.

Return Value

The year as a four-digit number. For example, the year 1976 is returned as 1976. Years specified as two digits in the `Date` constructor or in `setFullYear` are assumed to be in the twentieth century, so given "5/14/12", `getFullYear` returns "1912".

Remarks

To get the year using Universal Coordinated Time (UTC), use the `getUTCFullYear` method.

Example

The following example illustrates the use of the **getFullYear** method.

```
var date = new Date("1/1/01");
document.write(date.getFullYear());

// Output: 1901
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

Applies To: Date Object

See Also

getUTCFullYear Method (Date)
setFullYear Method (Date)
setUTCFullYear Method (Date)

getHours Method (Date) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Gets the hours in a date, using local time.

Syntax

```
dateObj.getHours()
```

Parameters

The required `dateObj` reference is a `Date` object.

Return Value

An integer between 0 and 23, indicating the number of hours since midnight. Zero is returned if the time is before 1:00:00 am. If a `Date` object was created without specifying the time, by default the hour is 0.

Remarks

To get the hours value using Universal Coordinated Time (UTC), use the `getUTCHours` method.

Example

The following example shows how to use the `getHours` method.

```
var date = new Date("1/1/2001");
document.write(date.getHours());
document.write("<br/>");

date.setTime(50000000);
document.write(date.getHours());

// Output:
// 0
// 5
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

Applies To: Date Object

See Also

getUTCHours Method (Date)
setHours Method (Date)
setUTCHours Method (Date)

getMilliseconds Method (Date) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Gets the milliseconds of a Date, using local time.

Syntax

```
dateObj.getMilliseconds()
```

Parameters

The required `dateObj` reference is a `Date` object.

Return Value

Returns the milliseconds of a date. The value can range from 0-999. If a date has been constructed without specifying the milliseconds, the value returned is 0.

Remarks

To get the number of milliseconds in Universal Coordinated Time (UTC), use the `getUTCMilliseconds` method.

Example

The following example shows how to use the **getMilliseconds** method.

```
var date = new Date("1/1/2001");
document.write(date.getMilliseconds());
document.write("<br/>");

date.setMilliseconds(5);
document.write(date.getMilliseconds());
// Output:
// 0
// 5
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

Applies To: Date Object

See Also

getUTCMilliseconds Method (Date)
setMilliseconds Method (Date)
setUTCMilliseconds Method (Date)

getMinutes Method (Date) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Gets the minutes of a `Date` object, using local time.

Syntax

```
dateObj.getMinutes()
```

Parameters

The required `dateObj` reference is a `Date` object.

Return Value

Returns an integer between 0 and 59. Zero is returned the time is less than one minute after the hour. If a `Date` object was created without specifying the time, by default the minute value is 0.

Remarks

To get the minutes value using Universal Coordinated Time (UTC), use the `getUTCMinutes` method.

Example

The following example shows how to the `getMinutes` method.

```
var date = new Date("1/1/2001");
document.write(date.getMinutes());
document.write("<br/>");

date.setMinutes(5);
document.write(date.getMinutes());

// Output:
// 0
// 5
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

Applies To: Date Object

See Also

getUTCMinutes Method (Date)
setMinutes Method (Date)
setUTCMinutes Method (Date)

getMonth Method (Date) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Gets the month, using local time.

Syntax

```
dateObj.getMonth()
```

Parameters

The required `dateObj` reference is a `Date` object.

Return Value

The `getMonth` method returns an integer between 0 (January) and 11 (December). For a `Date` constructed with "Jan 5, 1996", `getMonth` returns 0.

Remarks

To get the month value using Universal Coordinated Time (UTC), use the `getUTCMonth` method.

Example

The following example shows how to use the `getMonth` method.

```
var date = new Date("1/1/2001");
document.write(date.getMonth());

// Output: 0
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

Applies To: Date Object

See Also

getUTCMonth Method (Date)
setMonth Method (Date)
setUTCMonth Method (Date)

getSeconds Method (Date) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Gets the seconds of a `Date` object, using local time.

Syntax

```
dateObj.getSeconds()
```

Parameters

The required `dateObj` reference is a `Date` object.

Return Value

Returns an integer between 0 and 59. Zero is returned when the time is less than one second into the current minute. If a `Date` object was created without specifying the time, by default the seconds value is 0.

Remarks

To get the seconds value using Universal Coordinated Time (UTC), use the `getUTCSeconds` method.

Example

The following example shows how to use the `getSeconds` method.

```
var date = new Date("1/1/2001");
document.write(date.getSeconds());
document.write("<br/>");

date.setSeconds(5);
document.write(date.getSeconds());

// Output:
// 0
// 5
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

Applies To: Date Object

See Also

getUTCSeconds Method (Date)
setSeconds Method (Date)
setUTCSeconds Method (Date)

getTime Method (Date) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Gets the time value in milliseconds.

Syntax

```
dateObj.getTime()
```

Parameters

The required `dateObj` reference is a `Date` object.

Return Value

Returns the number of milliseconds between midnight, January 1, 1970 and the time value in the `Date` object. The range of dates is approximately 285,616 years from either side of midnight, January 1, 1970. Negative numbers indicate dates prior to 1970.

Remarks

When doing multiple date and time calculations, you may want to define variables equal to the number of milliseconds in a day, hour, or minute. For example:

```
var minute = 1000 * 60;
var hour = minute * 60;
var day = hour * 24;
```

See Calculating Dates and Times (JavaScript) for more information about how to use the `getTime` method.

Example

The following example shows how to use the `getTime` method.

```
var minute = 1000 * 60;
var hour = minute * 60;
var day = hour * 24;

date = new Date("1/1/2001");
var time = date.getTime();

document.write(Math.round(time / day) + " days from 1/1/1970 to 1/1/2001");

// Output: 11323 days from 1/1/1970 to 1/1/2001
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version

Information.

Applies To: Date Object

See Also

setTime Method (Date)

getTimezoneOffset Method (Date) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Gets the difference in minutes between the time on the local computer and Universal Coordinated Time (UTC).

Syntax

```
dateObj.getTimezoneOffset()
```

Parameters

The required `dateObj` reference is a `Date` object.

Return Value

Returns the number of minutes between the time on the current computer (either the client machine or, if this method is called from a server script, the server machine) and UTC. It is positive if the current computer's local time is behind UTC (e.g., Pacific Daylight Time), and negative if the current computer's local time is ahead of UTC (e.g., Japan). If a server in New York City is contacted by a client in Los Angeles on December 1, `getTimezoneOffset` returns 480 if executed on the client, or 300 if executed on the server.

Remarks

Example

The following example shows how to use the `getTimezoneOffset` method.

```
var date =  new Date();
var minutes = date.getTimezoneOffset();

if (minutes < 0)
    document.write(minutes / 60 + " hours after UTC");
else
    document.write(minutes / 60 + " hours before UTC");

// Output (for example, where local time is PST):
7 hours before UTC
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

Applies To: Date Object

See Also

getTime Method (Date)

getUTCDate Method (Date) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Gets the day-of-the-month, using Universal Coordinated Time (UTC).

Syntax

```
dateObj.getUTCDate()
```

Parameters

The required `dateObj` reference is a `Date` object.

Return Value

Returns an integer between 1 and 31 that represents the day-of-the-month.

Remarks

To get the day of the month using local time, use the `getDate` method.

Example

The following example shows how to use the `getUTCDate` method.

```
var date = new Date("1/23/2001");
document.write(date.getUTCDate());

// Output: 23
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

Applies To: Date Object

See Also

getDate Method (Date)
setDate Method (Date)
setUTCDate Method (Date)

getUTCDay Method (Date) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Gets the day of the week using Universal Coordinated Time (UTC).

Syntax

```
dateObj.getUTCDay()
```

Parameters

The required `dateObj` reference is a `Date` object.

Return Value

Returns an integer between 0 (Sunday) and 6 (Saturday) that represents the day of the week.

Remarks

To get the day of the week using local time, use the `getDate` method.

Example

The following example shows how to use the `getUTCDay` method.

```
var date = new Date("2/6/2001");
var day = date.getUTCDay();
document.write(day);

// Output: 2
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

Applies To: Date Object

See Also

getDay Method (Date)

getUTCFullYear Method (Date) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Gets the year using Universal Coordinated Time (UTC).

Syntax

```
dateObj.getUTCFullYear()
```

Parameters

The required `dateObj` reference is a `Date` object.

Return Value

Returns the year as a four-digit number. Years specified as two digits in the `Date` constructor or in `setFullYear` are assumed to be in the twentieth century, so given "5/14/12", `getUTCFullYear` returns "1912".

Remarks

To get the year using local time, use the `getFullYear` method.

Example

The following example shows how to use the `getUTCFullYear` method.

```
var date = new Date("1/9/36");
document.write(date.getUTCFullYear());

// Output: 1936
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

Applies To: Date Object

See Also

getFullYear Method (Date)
setFullYear Method (Date)
setUTCFullYear Method (Date)

getUTCHours Method (Date) (JavaScript)

2/1/2018 • 1 min to read • Edit Online

Gets the hours value in a `Date` object using Universal Coordinated Time (UTC).

Syntax

```
dateObj.getUTCHours()
```

Parameters

The required `dateObj` reference is a `Date` object.

Return Value

Returns an integer between 0 and 23 indicating the number of hours since midnight. Zero is returned if the time is before 1:00:00 am. If a `Date` object was created without specifying the time, by default the hour is 0 in UTC time. This time may be non-zero in other time zones.

Remarks

To get the number of hours elapsed since midnight using local time, use the `getHours` method.

Example

The following example illustrates the use of the `getUTCHours` method.

```
var date = new Date("1/1/2001");
document.write(date.getUTCHours());
document.write("<br/>");

var date2 = new Date("1/1/2001 11:22:33");
document.write(date2.getUTCHours());

// Output (in the PST time zone):
// 15
// 2
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

Applies To: Date Object

See Also

getHours Method (Date)
setHours Method (Date)
setUTCHours Method (Date)

getUTCMilliseconds Method (Date) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Gets the milliseconds of a `Date` object using Universal Coordinated Time (UTC).

Syntax

```
dateObj.getUTCMilliseconds()
```

Parameters

The required `dateObj` reference is a `Date` object.

Return Value

Returns a millisecond value that can range from 0-999.

Remarks

To get the number of milliseconds in local time, use the `getMilliseconds` method.

Example

The following example illustrates the use of the `getUTCMilliseconds` method.

```
var date = new Date("1/1/2001");
document.write(date.getUTCMilliseconds());
document.write("<br/>");

date.setMilliseconds(34);
document.writedate.getUTCMilliseconds());

// Output:
// 0
// 34
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

Applies To: Date Object

See Also

getMilliseconds Method (Date)
setMilliseconds Method (Date)
setUTCMilliseconds Method (Date)

getUTCMinutes Method (Date) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Gets the minutes of a `Date` object using Universal Coordinated Time (UTC).

Syntax

```
dateObj.getUTCMinutes()
```

Parameters

The required `dateObj` reference is a `Date` object.

Return Value

Returns an integer between 0 and 59. Zero is returned the time is less than one minute after the hour. If a `Date` object was created without specifying the time, by default the UTC minute value is 0. However, in other time zones it may be different.

Remarks

To get the number of minutes stored using local time, use the `getMinutes` method.

Example

The following example illustrates the use of the `getUTCMinutes` method.

```
var date = new Date("1/1/2001");
document.write(date.getUTCMinutes());
document.write("<br/>");

date.setMinutes(5);
document.write(date.getUTCMinutes());

// Output:
// 0
// 5
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

Applies To: Date Object

See Also

getMinutes Method (Date)
setMinutes Method (Date)
setUTCMinutes Method (Date)

getUTCMonth Method (Date) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Gets the month of a `Date` object using Universal Coordinated Time (UTC).

Syntax

```
dateObj.getUTCMonth()
```

Parameters

The required `dateObj` reference is a `Date` object.

Return Value

Returns an integer between 0 (January) and 11 (December).

Remarks

To get the month in local time, use the **getMonth** method.

Example

The following example shows how to use the `getUTCMonth` method.

```
var date = new Date("2/2/2002");
document.write(date.getUTCMonth());

// Output: 1
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

Applies To: Date Object

See Also

getMonth Method (Date)
setMonth Method (Date)
setUTCMonth Method (Date)

getUTCSeconds Method (Date) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Gets the seconds of a `Date` object using Universal Coordinated Time (UTC).

Syntax

```
dateObj.getUTCSeconds()
```

Parameters

The required `dateObj` reference is a `Date` object.

Return Value

Returns an integer between 0 and 59. Zero is returned when the time is less than one second into the current minute. If a `Date` object was created without specifying the time, by default the UTC seconds value is 0.

Remarks

To get the number of seconds in local time, use the `getSeconds` method.

Example

The following example shows how to use the `getUTCSeconds` method.

```
var date = new Date("1/1/2001");
document.write(date. getUTCSeconds());

// Output: 0
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

Applies To: Date Object

See Also

getSeconds Method (Date)
setSeconds Method (Date)
setUTCSeconds Method (Date)

getVarDate Method (Date) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Returns a VT_DATE value from a `Date` object.

> **WARNING**
>
> This method is supported in Internet Explorer only.

Syntax

```
dateObj.getVarDate()
```

Parameters

The required `dateObj` reference is a `Date` object.

Return Value

Returns a VT_DATE value.

Remarks

The `getVarDate` method is used when JavaScript code interacts with COM objects, ActiveX objects, or other objects that accept and return date values in VT_DATE format. These include objects in Visual Basic and Visual Basic Scripting Edition (VBScript). The actual format of the returned value depends on regional settings.

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, and Internet Explorer 10 standards. Not supported in Windows 8.x Store apps. See Version Information.

Applies To: Date Object

See Also

getDate Method (Date)
Date.parse Function

getYear Method (Date) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Gets the year of a `Date` object.

Syntax

```
dateObj.getYear()
```

Parameters

The required `dateObj` reference is a `Date` object.

Return Value

Returns the year.

Remarks

> **IMPORTANT**
>
> This method is obsolete, and is provided for backward compatibility only. Use the `getFullYear` method instead.

In Internet Explorer 3.0, and then in Internet Explorer versions starting with Internet Explorer 9 standards mode, the value returned is the stored year minus 1900. For example, the year 1899 is returned as -1 and the year 2000 is returned as 100.

In Internet Explorer 4.0 through Internet Explorer 8 standards mode, the formula depends on the year. For the years 1900 through 1999, the value returned is a 2-digit value that is the stored year minus 1900. For dates outside that range, the 4-digit year is returned. For example, 1996 is returned as 96, but 1825 and 2025 are returned as is.

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

Applies To: Date Object

See Also

getFullYear Method (Date)
getUTCFullYear Method (Date)
setFullYear Method (Date)
setUTCFullYear Method (Date)
setYear Method (Date)

setDate Method (Date) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Sets the numeric day-of-the-month value of the `Date` object using local time.

Syntax

```
dateObj.setDate(numDate)
```

Parameters

`dateObj`
Required. Any `Date` object.

`numDate`
Required. A numeric value equal to the day of the month.

Remarks

To set the day-of-the-month value using Universal Coordinated Time (UTC), use the `setUTCDate` method.

If the value of `numDate` is greater than the number of days in the month, the date rolls over to a later month and/or year. For example, if the stored date is January 5, 1996 and `setDate(32)` is called, the date changes to February 1, 1996. If `numDate` is a negative number, the date rolls back to an earlier month and/or year. For example, if the stored date is January 5, 1996 and `setDate(-32)` is called, the date changes to November 29, 1995.

The setFullYear Method (Date) can be used to set the year, month, and day of the month.

Example

The following example shows how to use the `setDate` method.

```
var date = new Date("12/15/1990");
date.setDate(30);
document.write(date);

// Output (for the PST time zone): Sun Dec 30 00:00:00 PST 1990
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

Applies To: Date Object

See Also

getDate Method (Date)
setFullYear Method (Date)
setMonth Method (Date)
setUTCDate Method (Date)

setFullYear Method (Date) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Sets the year of the `Date` object using local time.

Syntax

```
dateObj.setFullYear(numYear[, numMonth[, numDate]])
```

Parameters

`dateObj`
Required. Any `Date` object.

`numYear`
Required. A numeric value for the year.

`numMonth`
Optional. A zero-based numeric value for the month (0 for January, 11 for December). Must be specified if `numDate` is specified.

`numDate`
Optional. A numeric value equal for the day of the month.

Remarks

All **set** methods taking optional arguments use the value returned from corresponding **get** methods, if you do not specify the optional argument. For example, if the `numMonth` argument is optional, but not specified, JavaScript uses the value returned from the **getMonth** method.

In addition, if the value of an argument is greater than its calendar range or is negative, the date rolls forward or backward as appropriate.

To set the year using Universal Coordinated Time (UTC), use the `setUTCFullYear` method.

The range of years supported in the date object is approximately 285,616 years before and after 1970.

Example

The following example illustrates the use of the `setFullYear` method:

```
var date1 = new Date("1/1/2001");
date1.setFullYear(2007);

var date2 = new Date("1/1/2001");
date2.setFullYear(2008, 10, 3);

document.write (date1.toLocaleString());
document.write ("<br />");
document.write (date2.toLocaleString());

// Output:
// Monday, January 01, 2007 12:00:00 AM
// Monday, November 03, 2008 12:00:00 AM
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

Applies To: Date Object

See Also

getFullYear Method (Date)
getUTCFullYear Method (Date)
setUTCFullYear Method (Date)

setHours Method (Date) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Sets the hour value in the `Date` object using local time.

Syntax

```
dateObj.setHours(numHours[, numMin[, numSec[, numMilli]]])
```

Parameters

`dateObj`
Required. Any `Date` object.

`numHours`
Required. A numeric value equal to the hours value.

`numMin`
Optional. A numeric value equal to the minutes value. Must be supplied if either of the following arguments is used.

`numSec`
Optional. A numeric value equal to the seconds value. Must be supplied if the following argument is used.

`numMilli`
Optional. A numeric value equal to the milliseconds value.

Remarks

All **set** methods taking optional arguments use the value returned from corresponding **get** methods, if you do not specify an optional argument. For example, if the `numMinutes` argument is not specified, JavaScript uses the value returned from the `getMinutes` method.

To set the hours value using Universal Coordinated Time (UTC), use the `setUTCHours` method.

If the value of an argument is greater than its range or is a negative number, other stored values are modified accordingly. For example, if the stored date is "Jan 5, 1996 00:00:00", and **setHours(30)** is called, the date is changed to "Jan 6, 1996 06:00:00." Negative numbers have a similar behavior.

Example

The following example illustrates the use of the `setHours` method.

```
function SetHoursDemo(nhr, nmin, nsec){
   var d, s;                     //Declare variables.
   d = new Date();               //Create Date object.
   d.setHours(nhr, nmin, nsec);  //Set hours, minutes, & seconds.
   s = "Current setting is " + d.toLocaleString()
   return(s);                    //Return new date setting.
}
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

Applies To: Date Object

See Also

getHours Method (Date)
getUTCHours Method (Date)
setUTCHours Method (Date)

setMilliseconds Method (Date) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Sets the milliseconds value in the `Date` object using local time.

Syntax

```
dateObj.
setMilliseconds(
numMilli
)
```

Parameters

`dateObj`
Required. Any `Date` object.

`numMilli`
Required. A numeric value equal to the millisecond value.

Remarks

To set the milliseconds value using Universal Coordinated Time (UTC), use the `setUTCMilliseconds` method.

If the value of `numMilli` is greater than 999 or is a negative number, the stored number of seconds (and minutes, hours, and so forth if necessary) is incremented an appropriate amount.

Example

The following example illustrates the use of the `setMilliseconds` method.

```
function SetMSecDemo(nmsec){
   var d, s;                    // Declare variables.
   d = new Date();              // Create Date object.
   d.setMilliseconds(nmsec);    // Set milliseconds.
   s = "Current setting is ";
   s += d.toLocaleString();
   s += " and " + d.getMilliseconds();
   s += " milliseconds";
   return(s);                   // Return new date setting.
}
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

Applies To: Date Object

See Also

getMilliseconds Method (Date)
getUTCMilliseconds Method (Date)
setUTCMilliseconds Method (Date)

setMinutes Method (Date) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Sets the minutes value in the **Date** object using local time.

Syntax

```
dateObj.setMinutes(numMinutes[, numSeconds[, numMilli]])
```

Parameters

`dateObj`
Required. Any `Date` object.

`numMinutes`
Required. A numeric value equal to the minutes value. Must be supplied if either of the following arguments is used.

numSeconds
Optional. A numeric value equal to the seconds value. Must be supplied if the `numMilli` argument is used.

`numMilli`
Optional. A numeric value equal to the milliseconds value.

Remarks

All **set** methods taking optional arguments use the value returned from corresponding **get** methods, if you do not specify an optional argument. For example, if the *numSeconds* argument not specified, JavaScript uses the value returned from the `getSeconds` method.

To set the minutes value using Universal Coordinated Time (UTC), use the `setUTCMinutes` method.

If the value of an argument is greater than its range or is a negative number, other stored values are modified accordingly. For example, if the stored date is "Jan 5, 1996 00:00:00" and **setMinutes(90)** is called, the date is changed to "Jan 5, 1996 01:30:00." Negative numbers have a similar behavior.

Example

The following example illustrates the use of the `setMinutes` method.

```
function SetMinutesDemo(nmin, nsec){
   var d, s;                    // Declare variables.
   d = new Date();              // Create Date object.
   d.setMinutes(nmin, nsec);    // Set minutes.
   s = "Current setting is " + d.toLocaleString()
   return(s);                   // Return new setting.
}
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

Applies To: Date Object

See Also

getMinutes Method (Date)
getUTCMinutes Method (Date)
setUTCMinutes Method (Date)

setMonth Method (Date) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Sets the month value in the `Date` object using local time.

Syntax

```
dateObj. setMonth(numMonth[, dateVal])
```

Parameters

`dateObj`
Required. Any `Date` object.

`numMonth`
Required. A numeric value equal to the month. The value for January is 0, and other month values follow consecutively.

`dateVal`
Optional. A numeric value representing the day of the month. If this value is not supplied, the value from a call to the `getDate` method is used.

Remarks

To set the month value using Universal Coordinated Time (UTC), use the `setUTCMonth` method.

If the value of `numMonth` is greater than 11 (January is month 0) or is a negative number, the stored year is modified accordingly. For example, if the stored date is "Jan 5, 1996" and **setMonth(14)** is called, the date is changed to "Mar 5, 1997."

The **setFullYear** method can be used to set the year, month, and day of the month.

Example

The following example illustrates the use of the `setMonth` method.

```
date = new Date('1/1/1990');
date.setMonth(14);
document.write(date);

// Output: Fri Mar 1 00:00:00 PST 1991
// Note that the time zone corresponds to the time zone on the local computer.
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

Applies To: Date Object

See Also

getMonth Method (Date)
getUTCMonth Method (Date)
setUTCMonth Method (Date)

setSeconds Method (Date) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Sets the seconds value in the `Date` object using local time.

Syntax

```
dateObj
.setSeconds(
numSeconds[, numMilli])
```

Parameters

`dateObj`
Required. Any `Date` object.

numSeconds
Required. A numeric value equal to the seconds value.

`numMilli`
Optional. A numeric value equal to the milliseconds value.

Remarks

All **set** methods taking optional arguments use the value returned from corresponding **get** methods, if you do not specify an optional argument. For example, if the `numMilli` argument is not specified, JavaScript uses the value returned from the **getMilliseconds** method.

To set the seconds value using Universal Coordinated Time (UTC), use the `setUTCSeconds` method.

If the value of an argument is greater than its range or is a negative number, other stored values are modified accordingly. For example, if the stored date is "Jan 5, 1996 00:00:00" and **setSeconds(150)** is called, the date is changed to "Jan 5, 1996 00:02:30."

The `setHours` method can be used to set the hours, minutes, seconds, and milliseconds.

Example

The following example illustrates the use of the `setSeconds` method.

```
function SetSecondsDemo(nsec){
   var d = new Date();         //Create Date object.
   d.setSeconds(nsec);         //Set seconds.
   var s = "Current setting is ";
   s += d.toLocaleString();
   return(s);                  //Return new setting.
}
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

Applies To: Date Object

See Also

getSeconds Method (Date)
getUTCSeconds Method (Date)
setUTCSeconds Method (Date)

setTime Method (Date) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Sets the date and time value in the `Date` object.

Syntax

```
dateObj.setTime(milliseconds)
```

Parameters

`dateObj`
Required. Any `Date` object.

milliseconds
Required. A numeric value representing the number of elapsed milliseconds since midnight, January 1, 1970 GMT.

Remarks

If *milliseconds* is negative, it indicates a date before 1970. The range of available dates is approximately 285,616 years from either side of 1970.

Setting the date and time with the `setTime` method is independent of the time zone.

Example

The following example illustrates the use of the `setTime` method.

```
function SetTimeTest(newtime){
   var d, s;                  //Declare variables.
   d = new Date();            //Create Date object.
   d.setTime(newtime);        //Set time.
   s = "Current setting is ";
   s += d.toUTCString();
   return(s);                 //Return new setting.
}
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

Applies To: Date Object

See Also

getTime Method (Date)

setUTCDate Method (Date) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Sets the numeric day of the month in the `Date` object using Universal Coordinated Time (UTC).

Syntax

```
dateObj.setUTCDate(numDate)
```

Parameters

`dateObj`
Required. Any `Date` object.

numDate
Required. A numeric value equal to the day of the month.

Remarks

To set the day of the month using local time, use the `setDate` method.

If the value of *numDate* is greater than the number of days in the month stored in the **Date** object or is a negative number, the date is set to a date equal to *numDate* minus the number of days in the stored month. For example, if the stored date is January 5, 1996, and **setUTCDate(32)** is called, the date changes to February 1, 1996. Negative numbers have a similar behavior.

The **setUTCFullYear** method can be used to set the year, month, and day of the month.

Example

The following example illustrates the use of the `setUTCDate` method.

```
function SetUTCDateDemo(newdayofmonth){
   var d = new Date();           // Create Date object.
   d.setUTCDate(newdayofmonth);  // Set UTC day of month.
   var s = "Current setting is ";
   s += d.toUTCString();
   return(s);                    // Return new setting.
}
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

Applies To: Date Object

See Also

getDate Method (Date)
getUTCDate Method (Date)
setDate Method (Date)

setUTCFullYear Method (Date) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Sets the year value in the `Date` object using Universal Coordinated Time (UTC).

Syntax

```
dateObj.setUTCFullYear(numYear[, numMonth[, numDate]])
```

Parameters

`dateObj`
Required. Any `Date` object.

`numYear`
Required. A numeric value equal to the year.

`numMonth`
Optional. A numeric value equal to the month. The value for January is 0, and other month values follow consecutively. Must be supplied if *numDate* is supplied.

numDate
Optional. A numeric value equal to the day of the month.

Remarks

All **set** methods taking optional arguments use the value returned from corresponding **get** methods, if you do not specify an optional argument. For example, if the `numMonth` argument is not specified, JavaScript uses the value returned from the `getUTCMonth` method.

In addition, if the value of an argument is greater that its range or is a negative number, other stored values are modified accordingly.

To set the year using local time, use the `setFullYear` method.

The range of years supported in the `Date` object is approximately 285,616 years from either side of 1970.

Example

The following example illustrates the use of the `setUTCFullYear` method.

```
var dtFirst = new Date();
dtFirst.setUTCFullYear(2007);

var dtSecond = new Date();
// 10 is the value for November.
dtSecond.setUTCFullYear(2008, 10, 3);

document.write (dtFirst.toUTCString());
document.write ("<br />");
document.write (dtSecond.toUTCString());
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

Applies To: Date Object

See Also

getFullYear Method (Date)
getUTCFullYear Method (Date)
setFullYear Method (Date)

setUTCHours Method (Date) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Sets the hours value in the `Date` object using Universal Coordinated Time (UTC).

Syntax

```
dateObj.setUTCHours(numHours[, numMin[, numSec[, numMilli]]])
```

Parameters

`dateObj`
Required. Any `Date` object.

`numHours`
Required. A numeric value equal to the hours value.

`numMin`
Optional. A numeric value equal to the minutes value. Must be supplied if either `numSec` or `numMilli` are used.

`numSec`
Optional. A numeric value equal to the seconds value. Must be supplied if `numMilli` argument is used.

`numMilli`
Optional. A numeric value equal to the milliseconds value.

Remarks

All **set** methods taking optional arguments use the value returned from corresponding **get** methods, if you do not specify an optional argument. For example, if the `numMin` argument is not specified, JavaScript uses the value returned from the `getUTCMinutes` method.

To set the hours value using local time, use the `setHours` method.

If the value of an argument is greater than its range, or is a negative number, other stored values are modified accordingly. For example, if the stored date is "Jan 5, 1996 00:00:00.00", and **setUTCHours(30)** is called, the date is changed to "Jan 6, 1996 06:00:00.00."

Example

The following example illustrates the use of the `setUTCHours` method.

```
function SetUTCHoursDemo(nhr, nmin, nsec){
   var d, s;                        // Declare variables.
   d = new Date();                  // Create Date object.
   d.setUTCHours(nhr, nmin, nsec);  // Set UTC hours, minutes, seconds.
   s = "Current setting is " + d.toUTCString()
   return(s);                       // Return new setting.
}
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

Applies To: Date Object

See Also

getHours Method (Date)
getUTCHours Method (Date)
setHours Method (Date)

setUTCMilliseconds Method (Date) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Sets the milliseconds value in the `Date` object using Universal Coordinated Time (UTC).

Syntax

```
dateObj.setUTCMilliseconds(numMilli)
```

Parameters

`dateObj`
Required. Any `Date` object.

`numMilli`
Required. A numeric value equal to the millisecond value.

Remarks

To set the milliseconds using local time, use the `setMilliseconds` method.

If the value of `numMilli` is greater than 999, or is a negative number, the stored number of seconds (and minutes, hours, and so forth, if necessary) is incremented an appropriate amount.

Example

The following example illustrates the use of the `setUTCMilliseconds` method.

```
function SetUTCMSecDemo(nmsec){
// Create Date object.
   var d = new Date();
// Set UTC milliseconds.
   d.setUTCMilliseconds(nmsec);

   var s = "Current setting is ";
   s += d.toUTCString();
   s += " and " + d.getUTCMilliseconds();
   s += " milliseconds"
   return(s);
}
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

Applies To: Date Object

See Also

getMilliseconds Method (Date)
getUTCMilliseconds Method (Date)
setMilliseconds Method (Date)

setUTCMinutes Method (Date) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Sets the minutes value in the `Date` object using Universal Coordinated Time (UTC).

Syntax

```
dateObj.setUTCMinutes(numMinutes[, numSeconds[, numMilli]])
```

Parameters

`dateObj`
Required. Any `Date` object.

`numMinutes`
Required. A numeric value equal to the minutes value. Must be supplied if either of the following arguments is used.

numSeconds
Optional. A numeric value equal to the seconds value. Must be supplied if `numMilli` is used.

`numMilli`
Optional. A numeric value equal to the milliseconds value.

Remarks

All **set** methods taking optional arguments use the value returned from corresponding **get** methods, if you do not specify an optional argument. For example, if the *numSeconds* argument is not specified, JavaScript uses the value returned from the `getUTCSeconds` method.

To modify the minutes value using local time, use the `setMinutes` method.

If the value of an argument is greater than its range, or is a negative number, other stored values are modified accordingly. For example, if the stored date is "Jan 5, 1996 00:00:00.00", and **setUTCMinutes(70)** is called, the date is changed to "Jan 5, 1996 01:10:00.00."

The **setUTCHours** method can be used to set the hours, minutes, seconds, and milliseconds.

Example

The following example illustrates the use of the `setUTCMinutes` method:

```
function SetUTCMinutesDemo(nmin, nsec){
   var d, s;                    // Declare variables.
   d = new Date();              // Create Date object.
   d.setUTCMinutes(nmin,nsec);  // Set UTC minutes.
   s = "Current setting is " + d.toUTCString()
   return(s);                   // Return new setting.
}
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

Applies To: Date Object

See Also

getMinutes Method (Date)
getUTCMinutes Method (Date)
setMinutes Method (Date)

setUTCMonth Method (Date) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Sets the month value in the `Date` object using Universal Coordinated Time (UTC).

Syntax

```
dateObj.setUTCMonth(numMonth[, dateVal])
```

Parameters

`dateObj`
Required. Any `Date` object.

`numMonth`
Required. A numeric value equal to the month. The value for January is 0, and other month values follow consecutively.

`dateVal`
Optional. A numeric value representing the day of the month. If it is not supplied, the value from a call to the `getUTCDate` method is used.

Remarks

To set the month value using local time, use the `setMonth` method.

If the value of `numMonth` is greater than 11 (January is month 0), or is a negative number, the stored year is incremented or decremented appropriately. For example, if the stored date is "Jan 5, 1996 00:00:00.00", and **setUTCMonth(14)** is called, the date is changed to "Mar 5, 1997 00:00:00.00."

The **setUTCFullYear** method can be used to set the year, month, and day of the month.

Example

The following example illustrates the use of the `setUTCMonth` method.

```
function SetUTCMonthDemo(newmonth){
   var d, s;                        // Declare variables.
   d = new Date();                  // Create Date object.
   d.setUTCMonth(newmonth);         // Set UTC month.
   s = "Current setting is ";
   s += d.toUTCString();
   return(s);                       // Return new setting.
}
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11

standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

Applies To: Date Object

See Also

getMonth Method (Date)
getUTCMonth Method (Date)
setMonth Method (Date)

setUTCSeconds Method (Date) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Sets the seconds value in the `Date` object using Universal Coordinated Time (UTC).

Syntax

```
dateObj.setUTCSeconds(numSeconds[, numMilli])
```

Parameters

`dateObj`
Required. Any `Date` object.

numSeconds
Required. A numeric value equal to the seconds value.

`numMilli`
Optional. A numeric value equal to the milliseconds value.

Remarks

All **set** methods taking optional arguments use the value returned from corresponding **get** methods, if you do not specify an optional argument. For example, if the `numMilli` argument is not specified, JavaScript uses the value returned from the `getUTCMilliseconds` method.

To set the seconds value using local time, use the `setSeconds` method.

If the value of an argument is greater than its range or is a negative number, other stored values are modified accordingly. For example, if the stored date is "Jan 5, 1996 00:00:00.00" and **setSeconds(150)** is called, the date is changed to "Jan 5, 1996 00:02:30.00."

The **setUTCHours** method can be used to set the hours, minutes, seconds, and milliseconds.

Example

The following example illustrates the use of the `setUTCSeconds` method.

```
function SetUTCSecondsDemo(nsec){
// Create Date object.
    var d = new Date();
// Set UTC seconds.
    d.setUTCSeconds(nsec);
    var s = "Current setting is ";
    s += d.toUTCString();
// Return new setting.
    return(s);
}
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

Applies To: Date Object

See Also

getSeconds Method (Date)
getUTCSeconds Method (Date)
setSeconds Method (Date)

setYear Method (Date) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Sets the year value in the `Date` object.

Syntax

```
dateObj.setYear(numYear)
```

Parameters

`dateObj`
Required. Any `Date` object.

`numYear`
Required. For the years 1900 through 1999, this is a numeric value equal to the year minus 1900. For dates outside that range, this is a 4-digit numeric value.

Remarks

This method is obsolete, and is maintained for backwards compatibility only. Use the `setFullYear` method instead.

To set the year of a `Date` object to 1997, call **setYear(97)**. To set the year to 2010, call **setYear(2010)**. Finally, to set the year to a year in the range 0-99, use the `setFullYear` method.

> **NOTE**
> For JavaScript version 1.0, `setYear` uses a value that is the result of the addition of 1900 to the year value provided by `numYear`, regardless of the value of the year. For example, to set the year to 1899 `numYear` is -1 and to set the year 2000 `numYear` is 100.

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

Applies To: Date Object

See Also

getFullYear Method (Date)
getUTCFullYear Method (Date)
getYear Method (Date)
setFullYear Method (Date)
setUTCFullYear Method (Date)

toDateString Method (Date) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Returns a date as a string value.

Syntax

```
objDate.toDateString( )
```

Remarks

The required *objDate* reference is a `Date` object.

The `toDateString` method returns a string value containing the date, in the current time zone, in a convenient, easily read format.

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

Applies To: Date Object

See Also

toTimeString Method (Date)
toLocaleDateString Method (Date)

toGMTString Method (Date) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Returns a date converted to a string using Greenwich Mean Time(GMT).

Syntax

```
dateObj.toGMTString()
```

Remarks

The required `dateObj` reference is any `Date` object.

The `toGMTString` method is obsolete, and is provided for backwards compatibility only. It is recommended that you use the `toUTCString` method instead.

The `toGMTString` method returns a `String` object that contains the date formatted using GMT convention. The format of the return value is as follows: "05 Jan 1996 00:00:00 GMT."

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

Applies To: Date Object

See Also

toUTCString Method (Date)

toISOString Method (Date) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Returns a date as a string value in ISO format.

Syntax

```
objDate.toISOString()
```

Return Value

A string representation of the date in International Organization for Standardization (ISO) format.

Exceptions

If `objDate` does not contain a valid date, a `RangeError` exception is thrown.

Remarks

The ISO format is a simplification of the ISO 8601 format. For more information, see Date and Time Strings.

The time zone is always UTC, denoted by the suffix Z in the output.

Example

The following example illustrates the use of the `toISOString` method.

```
var dt = new Date("30 July 2010 15:05 UTC");
document.write(dt.toISOString());
document.write("<br />");
document.write(dt.toUTCString());

// Output:
//  2010-07-30T15:05:00.000Z
//  Fri, 30 Jul 2010 15:05:00 UTC
```

Requirements

Supported in the following document modes: Internet Explorer 9 standards, Internet Explorer 10 standards, and Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards.

See Also

Date Object
Date and Time Strings

toJSON Method (Date) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Used by the JSON.stringify method to enable the transformation of an object's data for JavaScript Object Notation (JSON) serialization.

Syntax

```
objectname.toJSON()
```

Parameters

`objectname`
Required. An object for which JSON serialization is wanted.

Remarks

The `toJSON` method is used by the `JSON.stringify` function. `JSON.stringify` serializes a JavaScript value into JSON text. If a `toJSON` method is provided to `JSON.stringify`, the `toJSON` method is called when `JSON.stringify` is called.

The `toJSON` method is a built-in member of the DateJavaScript object. It returns an ISO-formatted date string for the UTC time zone (denoted by the suffix Z).

You can override the `toJSON` method for the `Date` type, or define a `toJSON` method for other object types to achieve transformation of data for the specific object type before JSON serialization.

Example

The following example uses the `toJSON` method to serialize string member values in uppercase. The `toJSON` method is called when `JSON.stringify` is called.

```
var contact = new Object();
contact.firstname = "Jesper";
contact.surname = "Aaberg";
contact.phone = ["555-0100", "555-0120"];

contact.toJSON = function(key)
 {
    var replacement = new Object();
    for (var val in this)
    {
        if (typeof (this[val]) === 'string')
            replacement[val] = this[val].toUpperCase();
        else
            replacement[val] = this[val]
    }
    return replacement;
};

var jsonText = JSON.stringify(contact);

/* The value of jsonText is:
'{"firstname":"JESPER","surname":"AABERG","phone":["555-0100","555-0120"]}'
*/
```

Example

The following example illustrates how to use the `toJSON` method that is a built-in member of the Date object.

```
var dt = new Date('8/24/2009');
dt.setUTCHours(7, 30, 0);
var jsonText = JSON.stringify(dt);

/* The value of jsonText is:
'"2009-08-24T07:30:00Z"'
*/
```

Requirements

Supported in the following document modes: Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards. **Applies To:** Date Object

See Also

JSON Object
JSON.parse Function
JSON.stringify Function
JavaScript Methods

toLocaleDateString Method (Date) (JavaScript)

10/18/2017 • 2 min to read • Edit Online

Returns a date as a string value that is appropriate to the host environment's current locale or the specified locale.

Syntax

```
dateObj.toLocaleDateString( [locales][, options])
```

Parameters

`dateObj`
Required. A `Date` object.

`locales`
Optional. An array of locale strings that contain one or more language or locale tags. If you include more than one locale string, list them in descending order of priority so that the first entry is the preferred locale. If you omit this parameter, the default locale of the JavaScript runtime is used.

`options`
Optional. An object that contains one or more properties that specify comparison options.

Remarks

Starting in Internet Explorer 11, `toLocaleDateString` uses `Intl.DateTimeFormat` internally to format the date, which adds support for the `locales` and `options` parameters. For more information about these parameters, see Intl.DateTimeFormat.

> **IMPORTANT**
> The `locales` and `options` parameters are not supported in all document modes and browser versions. For more information, see the Requirements section.

When you use `toLocaleDateString` in Internet Explorer 10 standards document mode, earlier document modes, and quirks mode:

- it returns a string value that contains a date in the current time zone.
- The returned date is in the default format of the host environment's current locale.

  If you omit the `locales` parameter, the return value of this method cannot be relied upon in scripting, because it will vary from computer to computer. In this scenario, use the method only to format displayed text - never as part of a computation.

Example

The following example shows how to use the `toLocaleDateString` method with a specified locale and comparison options.

```
var date = new Date(Date.UTC(2013, 1, 1, 14, 0, 0));
var options = { weekday: "long", year: "numeric", month: "short",
    day: "numeric" };

// Using I18N toLocaleString
document.write(date.toLocaleDateString("en-US"));
document.write(date.toLocaleDateString("ja-JP"));
document.write(date.toLocaleDateString("ar-SA", options));
document.write(date.toLocaleDateString("hi-IN", options));

// Output:
// 2/1/2013
// 2013年2月1日
// الجمعة, ٢٠ ربيع الأول, ١٤٣٤
// शुक्रवार, 01 फरवरी 2013
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

`locales` and `options` parameters:

Supported in the Internet Explorer 11 standards document mode. Also supported in Store apps (Windows 8.1 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards. Not supported in Windows 8.

See Also

toDateString Method (Date)
toLocaleTimeString Method (Date)

toLocaleString (Date)

10/18/2017 • 2 min to read • Edit Online

Converts a date to a string by using the current or specified locale.

Syntax

```
dateObj.toLocaleString([locales][, options])
```

Parameters

`dateObj`
Required. The `Date` object to convert.

`locales`
Optional. An array of locale strings that contain one or more language or locale tags. If you include more than one locale string, list them in descending order of priority so that the first entry is the preferred locale. If you omit this parameter, the default locale of the JavaScript runtime is used.

`options`
Optional. An object that contains one or more properties that specify comparison options.

Remarks

Starting in Internet Explorer 11, `toLocaleString` uses `Intl.DateTimeFormat` internally to make comparisons, which adds support for the `locales` and `options` parameters. For more information about these parameters, see Intl.DateTimeFormat.

> **IMPORTANT**
> The `locales` and `options` parameters are not supported in all document modes and browser versions. For more information, see the Requirements section.

When you use `toLocaleString` in Internet Explorer 10 standards document mode, earlier document modes, and quirks mode:

- It returns a `String` object that contains the date written in the current locale's long default format.
- For dates between 1601 and 1999 A.D., the date is formatted according to the user's Control Panel regional settings.

> **NOTE**
> If you omit the `locales` parameter, use `toLocaleString` only to display results to a user; never use it to compute values within a script, because the returned result is machine-specific.

Example

The following example shows how to use the `toLocaleString` method.

```
function toLocaleStrDemo(){
   var d, s;                      //Declare variables.
   d = new Date();                //Create Date object.
   s = "Current setting is ";
   s += d.toLocaleString();       //Convert to current locale.
   return(s);                     //Return converted date
}
```

Example

The following example shows how to use the `toLocaleString` method with a specified locale and comparison options.

```
var date = new Date(Date.UTC(2013, 1, 1, 14, 0, 0));
var options = { weekday: "long", year: "numeric", month: "short",
    day: "numeric" };

// Using I18N toLocaleString
document.write(date.toLocaleString("en-US"));
document.write(date.toLocaleString("ja-JP"));
document.write(date.toLocaleString("ar-SA", options));
document.write(date.toLocaleString("hi-IN", options));

// Output:
// 2/1/2013 6:00:00 AM
// 2013年2月1日 6:00:00
// الجمعة, ٢٠ ربيع الأول, ١٤٣٤
// शुक्रवार, 01 फरवरी 2013
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

`locales` and `options` parameters:

Supported in the Internet Explorer 11 standards document mode. Also supported in Store apps (Windows 8.1 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards. Not supported in Windows 8.

See Also

toLocaleDateString Method (Date)

toLocaleTimeString Method (Date) (JavaScript)

10/18/2017 • 2 min to read • Edit Online

Returns a time as a string value that is appropriate to the host environment's current locale or a specified locale.

Syntax

```
dateObj.toLocaleTimeString([locales][, options])
```

Parameters

`dateObj`
Required. A `Date` object.

`locales`
Optional. An array of locale strings that contain one or more language or locale tags. If you include more than one locale string, list them in descending order of priority so that the first entry is the preferred locale. If you omit this parameter, the default locale of the JavaScript runtime is used.

`options`
Optional. An object that contains one or more properties that specify comparison options.

Remarks

Starting in Internet Explorer 11, `toLocaleTimeString` uses `Intl.DateTimeFormat` internally to format the time, which adds support for the `locales` and `options` parameters. For more information about these parameters, see Intl.DateTimeFormat.

> **IMPORTANT**
> The `locales` and `options` parameters are not supported in all document modes and browser versions. For more information, see the Requirements section.

When you use `toLocaleTimeString` in Internet Explorer 10 standards document mode, earlier document modes, and quirks mode:

- It returns a string value that contains a time in the current time zone.
- The returned time is in the default format of the host environment's current locale.

  If you omit the `locales` parameter, the return value of this method cannot be relied upon in scripting, because it will vary from computer to computer. In this scenario, use the method only to format displayed text - never as part of a computation.

Example

The following example shows how to use the `toLocaleTimeString` method with a specified locale and comparison options.

```
var date = new Date(Date.UTC(2013, 1, 1, 14, 0, 0));
var options = { weekday: "long", year: "numeric", month: "short",
    day: "numeric" };

// Using I18N toLocaleString
document.write(date.toLocaleTimeString("en-US"));
document.write(date.toLocaleTimeString("ja-JP"));
document.write(date.toLocaleTimeString("ar-SA", options));
document.write(date.toLocaleTimeString("hi-IN", options));

// Output:
// 6:00:00 AM
// 6:00:00
// ٠٦ ١٤٣٤ ,الأول ربيع ٢٠ ,الجمعة:٠٠:٠٠ ص
// शुक्रवार, 01 फरवरी 2013 06:00:00
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

`locales` and `options` parameters:

Supported in the Internet Explorer 11 standards document mode. Also supported in Store apps (Windows 8.1 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards. Not supported in Windows 8.

See Also

toTimeString Method (Date)
toLocaleDateString Method (Date)

toTimeString Method (Date) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Returns a time as a string value.

Syntax

```
objDate. toTimeString( )
```

Remarks

The required `objDate` reference is a `Date` object.

The `toTimeString` method returns a string value containing the time in the current time zone.

Example

In the following example, the time is set to 2000 milliseconds after midnight January 1, 1970 UTC, and then it is written out.

```
var aDate = new Date();
     aDate.setTime(2000);
     document.write(aDate.toTimeString());

// Output depends on the time in the current time zone.
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

See Also

toDateString Method (Date)
toLocaleTimeString Method (Date)

toUTCString Method (Date) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Returns a date converted to a string using Universal Coordinated Time (UTC).

Syntax

```
dateObj.toUTCString()
```

Remarks

The required `dateObj` reference is any `Date` object.

The `toUTCString` method returns a `String` object that contains the date formatted using UTC convention in a convenient, easily read form.

Example

The following example illustrates the use of the `toUTCString` method.

```
function toUTCStrDemo(){
   var d, s;                    //Declare variables.
   d = new Date();              //Create Date object.
   s = "Current setting is ";
   s += d.toUTCString();        //Convert to UTC string.
   return(s);                   //Return UTC string.
}
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

Applies To: Date Object

See Also

toGMTString Method (Date)

toString Method (Date)

10/18/2017 • 1 min to read • Edit Online

Returns a string representation of a date. The format of the string depends on the locale. For U.S. English (en-us), it is as follows:

day of the week month day hour: *minute*:*second time zone year*

Syntax

```
date.toString()
```

Parameters

`date`
Required. The date to represent as a string.

Return Value

Returns the string representation of the date.

Example

The following example illustrates the use of the `toString` method with a date.

```
var myDate = new Date();
myDate.setFullYear(2100, 5, 5);
var dateString = myDate.toString();
document.write(dateString);

// Output: <date>
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

valueOf Method (Date)

10/18/2017 • 1 min to read • Edit Online

Returns the stored time value in milliseconds since midnight, January 1, 1970 UTC.

Syntax

```
date.valueOf()
```

Parameters

The `date` object is any instance of a Date.

Return Value

The stored time value in milliseconds since midnight, January 1, 1970 UTC. This is the same value as `getTime`.

Example

The following example illustrates the use of the `valueOf` method with a date.

```
var myDate = new Date();
myDate.setFullYear(2100, 5, 5);
if (myDate.getTime() == myDate.valueOf())
    document.write("values are the same");
else
    document.write("values are different");

// Output: values are the same
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

Debug Object (JavaScript)

10/18/2017 • 1 min to read • Edit Online

An intrinsic global object that sends output to a debugger.

Syntax

```
Debug.function
```

Remarks

You do not instantiate the Debug object. You can access all its properties and methods by calling `function`.

There are different ways to debug Internet Explorer and Windows 8.x Store apps. In Windows 8.x Store apps, the `write` and `writeln` functions of the `Debug` object display strings in the Visual Studio **Output** window at run time. For more information about debugging Windows 8.x Store apps, see Debug Windows Universal Apps in Visual Studio.

To debug Internet Explorer scripts, you must have a script debugger installed and the script must run in debug mode. Internet Explorer 8 and later versions include the JavaScript debugger. If you are using an earlier version of Internet Explorer, see How to: Enable and Start Script Debugging from Internet Explorer.

If the script is not being debugged, the functions have no effect.

Example

This example uses the `write` function to display the value of the variable.

```
var counter = 42;
Debug.write("The value of counter is " + counter);
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

Constants

Debug Constants

Properties

Debug.debuggerEnabled Property | Debug.setNonUserCodeExceptions Property

Functions

Debug.msTraceAsyncCallbackStarting Function | Debug.msTraceAsyncCallbackCompleted Function | Debug.msTraceAsyncOperationCompleted Function | Debug.msTraceAsyncOperationStarting Function |

Debug.msUpdateAsyncCallbackRelation Function | Debug.write Function | Debug.writeln Function

See Also

debugger Statement

Debug Constants

10/18/2017 • 1 min to read • Edit Online

Debug constants return constant values that are properties of the `Debug` object.

Debug Object Constants

The following table lists constant values that are properties of the Debug object.

CONSTANT	DESCRIPTION	VALUE
`Debug.MS_ASYNC_CALLBACK_STATUS_ASSIGN_DELEGATE`	The synchronous work item assigned a callback or continuation to be run by an asynchronous operation.	0
`Debug.MS_ASYNC_CALLBACK_STATUS_JOIN`	The synchronous work item satisfied part of a join asynchronous operation.	1
`Debug.MS_ASYNC_CALLBACK_STATUS_CHOOSEANY`	The synchronous work item satisfied a choice asynchronous operation.	2
`Debug.MS_ASYNC_CALLBACK_STATUS_CANCEL`	The synchronous work item was canceled.	3
`Debug.MS_ASYNC_CALLBACK_STATUS_ERROR`	The synchronous work item caused an error in an asynchronous operation.	4
`Debug.MS_ASYNC_OP_STATUS_SUCCESS`	The asynchronous operation was successful.	1
`Debug.MS_ASYNC_OP_STATUS_CANCELED`	The asynchronous operation was canceled.	2
`Debug.MS_ASYNC_OP_STATUS_ERROR`	The asynchronous operation resulted in an error.	3

Requirements

Supported in the Internet Explorer 11 standards document mode. Also supported in Store apps (Windows 8.1 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards. Not supported in Windows 8.

See Also

Debug.msTraceAsyncOperationCompleted Function
Debug.msUpdateAsyncCallbackRelation Function

Debug.debuggerEnabled Property

10/18/2017 • 1 min to read • Edit Online

Determines whether debugging is enabled for the script context. Debugging may be enabled or disabled whether or not a debugger is attached.

Syntax

```
var dbgEnabled = Debug.debuggerEnabled;
```

Requirements

Supported in the following document modes: Internet Explorer 10 standards and Internet Explorer 11 standards.
Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards.

Debug.setNonUserCodeExceptions Property

10/18/2017 • 1 min to read • Edit Online

Determines whether any try-catch blocks in this scope are to be treated by the debugger as user-unhandled. Exceptions can be classified as thrown, user-unhandled or unhandled.

If this property is set to `true` in a given scope, the debugger can then determine to take some action (for example, break) on exceptions thrown inside that scope if the developer wishes to break on user-unhandled exceptions. If this property is set to `false` is the same as if the property was never set.

For more information on debugging, see Active Script Debugging Overview.

Syntax

```
Debug.setNonUserCodeExceptions [= bool];
```

Example

The following code shows how to set this property.

```
(function () {
    Debug.setNonUserCodeExceptions = true;
    try{
        var x = null;
        x.y();
    } catch (e) {
    // Catch the exception.
    }
})();
```

Requirements

Supported in the following document modes: Internet Explorer 10 standards and Internet Explorer 11 standards.
Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards.

Debug.msTraceAsyncCallbackCompleted Function

10/18/2017 • 1 min to read • Edit Online

Indicates that the callback stack associated with a previously specified asynchronous operation has completed.

Syntax

```
Debug.msTraceAsyncCallbackCompleted()
```

Remarks

Call this function after the call to `Debug.msTraceAsyncCallbackStarting`.

> **NOTE**
>
> Some debugging tools do not display the information sent to the debugger.

Example

The following code provides an example of tracing an asynchronous call for a Windows 8.x Store app.

```
function asyncWrapperFunction() {
    var opID = Debug.msTraceAsyncOperationStarting('async trace');
    doSomethingAsync().then(function (result) {
        Debug.msTraceAsyncOperationCompleted(opID, Debug.MS_ASYNC_OP_STATUS_SUCCESS);
        Debug.msTraceAsyncCallbackStarting(opID);
        // Process result of async operation.
    }, function (error) {
        Debug.msTraceAsyncOperationCompleted(opID, Debug.MS_ASYNC_OP_STATUS_ERROR);
        Debug.msTraceAsyncCallbackStarting(opID);
    });

    Debug.msTraceAsyncCallbackCompleted();
}

function doSomethingAsync() {
    return WinJS.Promise.as(true);
}

asyncWrapperFunction();
```

Requirements

Supported in the Internet Explorer 11 standards document mode. Also supported in Store apps (Windows 8.1 and Windows Phone 8.1). See Version Information.

Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards. Not supported in Windows 8.

Debug.msTraceAsyncCallbackStarting Function

10/18/2017 • 1 min to read • Edit Online

Associates the callback stack with a previously specified asynchronous operation.

Syntax

```
Debug.msTraceAsyncCallbackStarting(asyncOperationId)
```

Parameters

`asyncOperationId`

Required. The ID associated with the asynchronous operation.

Remarks

Call this function in the callback function for the asynchronous operation after the call to `Debug.msTraceAsyncOperationCompleted`.

> **NOTE**
>
> Some debugging tools do not display the information sent to the debugger.

`asyncOperationId` must correspond to the operation name previously returned from `Debug.msTraceAsyncOperationStarting`.

Example

The following code provides an example of tracing an asynchronous call for a Windows 8.x Store app.

```
function asyncWrapperFunction() {
    var opID = Debug.msTraceAsyncOperationStarting('async trace');
    doSomethingAsync().then(function (result) {
        Debug.msTraceAsyncOperationCompleted(opID, Debug.MS_ASYNC_OP_STATUS_SUCCESS);
        Debug.msTraceAsyncCallbackStarting(opID);
        // Process result of async operation.
    }, function (error) {
        Debug.msTraceAsyncOperationCompleted(opID, Debug.MS_ASYNC_OP_STATUS_ERROR);
        Debug.msTraceAsyncCallbackStarting(opID);
    });

    Debug.msTraceAsyncCallbackCompleted();
}

function doSomethingAsync() {
    return WinJS.Promise.as(true);
}

asyncWrapperFunction();
```

Requirements

Supported in the Internet Explorer 11 standards document mode. Also supported in Store apps (Windows 8.1 and

Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards. Not supported in Windows 8.

Debug.msTraceAsyncOperationCompleted Function

10/18/2017 • 1 min to read • Edit Online

Indicates that an asynchronous operation has completed.

Syntax

```
Debug.msTraceAsyncOperationCompleted(asyncOperationId, status)
```

Parameters

`asyncOperationId`

Required. The ID associated with an asynchronous operation.

`status`

Optional. The status of the asynchronous operation. If not specified, `Debug.MS_ASYNC_OP_STATUS_SUCCESS` is used.

Remarks

Call this function when the asynchronous operation completes.

`asyncOperationId` must correspond to the operation ID previously returned from `Debug.msTraceAsyncOperationStarting`.

The possible values for `status` include:

- `Debug.MS_ASYNC_OP_STATUS_SUCCESS`
- `Debug.MS_ASYNC_OP_STATUS_CANCELED`
- `Debug.MS_ASYNC_OP_STATUS_ERROR`

> **NOTE**
>
> Some debugging tools do not display the information sent to the debugger.

Example

The following code provides an example of tracing an asynchronous call for a Windows 8.x Store app.

```
function asyncWrapperFunction() {
    var opID = Debug.msTraceAsyncOperationStarting('async trace');
    doSomethingAsync().then(function (result) {
        Debug.msTraceAsyncOperationCompleted(opID, Debug.MS_ASYNC_OP_STATUS_SUCCESS);
        Debug.msTraceAsyncCallbackStarting(opID);
        // Process result of async operation.
    }, function (error) {
        Debug.msTraceAsyncOperationCompleted(opID, Debug.MS_ASYNC_OP_STATUS_ERROR);
        Debug.msTraceAsyncCallbackStarting(opID);
    });

    Debug.msTraceAsyncCallbackCompleted();
}

function doSomethingAsync() {
    return WinJS.Promise.as(true);
}

asyncWrapperFunction();
```

Requirements

Supported in the Internet Explorer 11 standards document mode. Also supported in Store apps (Windows 8.1 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards. Not supported in Windows 8.

Debug.msTraceAsyncOperationStarting Function

10/18/2017 • 1 min to read • Edit Online

Initiates a trace for an asynchronous operation.

Syntax

```
Debug.msTraceAsyncOperationStarting(operationName)
```

Parameters

`operationName`

Required. A string that identifies the asynchronous operation. If `operationName` is null or undefined, an empty string is used for the operation name.

Return Value

An integer representing the operation ID.

Remarks

Call this method before the asynchronous operation starts.

> **NOTE**
>
> Some debugging tools do not display the information sent to the debugger.

Example

The following code provides an example of instrumenting an asynchronous call for a Windows 8.x Store app.

```
function asyncWrapperFunction() {
    var opID = Debug.msTraceAsyncOperationStarting('async trace');
    doSomethingAsync().then(function (result) {
        Debug.msTraceAsyncOperationCompleted(opID, Debug.MS_ASYNC_OP_STATUS_SUCCESS);
        Debug.msTraceAsyncCallbackStarting(opID);
        // Process result of async operation.
    }, function (error) {
        Debug.msTraceAsyncOperationCompleted(opID, Debug.MS_ASYNC_OP_STATUS_ERROR);
        Debug.msTraceAsyncCallbackStarting(opID);
    });

    Debug.msTraceAsyncCallbackCompleted();
}

function doSomethingAsync() {
    return WinJS.Promise.as(true);
}

asyncWrapperFunction();
```

Requirements

Supported in the Internet Explorer 11 standards document mode. Also supported in Store apps (Windows 8.1 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards. Not supported in Windows 8.

Debug.msUpdateAsyncCallbackRelation Function

10/18/2017 • 1 min to read • Edit Online

Updates the relationship status between a synchronous work item and the associated asynchronous operation.

Syntax

```
Debug.msUpdateAsyncCallbackRelation(relatedAsyncOperationId, relationType)
```

Parameters

`relatedAsyncOperationId`

Required. The ID associated with the asynchronous operation.

`relationType`

Optional. The value that specifies the relationship status.

Remarks

The synchronous work item is typically the callback function for the asynchronous operation. This function may be called when an asynchronous operation is aborted, when a join operation is used, or in other scenarios.

The possible values for `relationType` include:

- `Debug.MS_ASYNC_CALLBACK_STATUS_ASSIGN_DELEGATE`
- `Debug.MS_ASYNC_CALLBACK_STATUS_JOIN`
- `Debug.MS_ASYNC_CALLBACK_STATUS_CHOOSEANY`
- `Debug.MS_ASYNC_CALLBACK_STATUS_CANCEL`
- `Debug.MS_ASYNC_CALLBACK_STATUS_ERROR`

  For more information, see Debug Constants.

> **NOTE**
>
> Some debugging tools do not display the information sent to the debugger by this function.

Requirements

Supported in the Internet Explorer 11 standards document mode. Also supported in Store apps (Windows 8.1 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards. Not supported in Windows 8.

Debug.write Function (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Sends strings to the script debugger.

Syntax

```
Debug.write([str1 [, str2 [, ... [, strN]]]])
```

Parameters

str1, str2, ... , strN
Optional. Strings to send to the script debugger.

Remarks

The `Debug.write` function sends strings to the Immediate window of a script debugger at run time. If the script is not being debugged, the `Debug.write` function has no effect.

The `Debug.write` function is almost identical to the `Debug.writeln` function. The only difference is that the `Debug.writeln` function sends a newline character after the strings are sent.

Example

This example uses the `Debug.write` function to display the value of the variable in the Immediate window of the script debugger.

> **NOTE**
>
> To run this example, you must have a script debugger installed and the script must run in debug mode.
>
> Internet Explorer 8 includes the JavaScript debugger. If you are using an earlier version of Internet Explorer, see How to: Enable and Start Script Debugging from Internet Explorer.

```
var counter = 42;
Debug.write("The value of counter is " + counter);
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

Applies To: Debug Object

See Also

Debug.writeln Function

Debug.writeln Function (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Sends strings to the script debugger, followed by a newline character.

Syntax

```
Debug.writeln([str1 [, str2 [, ... [, strN]]]])
```

Parameters

str1, str2, ... , strN
Optional. Strings to send to the script debugger.

Remarks

The `Debug.writeln` function sends strings, followed by a newline character, to the Immediate window of the Microsoft Script Debugger at run time. If the script is not being debugged, the `Debug.writeln` function has no effect.

The `Debug.writeln` function is almost identical to the `Debug.write` function. The only difference is that the `Debug.write` function does not send a newline character after sending the strings.

Example

This example uses the `Debug.writeln` function to display the value of the variable in the Immediate window of the Microsoft Script Debugger.

> **NOTE**
>
> To run this example, you must have a script debugger installed and the script must run in debug mode.
>
> Internet Explorer 8 includes the JavaScript debugger. If you are using an earlier version of Internet Explorer, see How to: Enable and Start Script Debugging from Internet Explorer.

```
var counter = 42;
Debug.writeln("The value of counter is " + counter);
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

Applies To: Debug Object

See Also

Debug.write Function

Enumerator Object (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Enables enumeration of items in a collection.

> **WARNING**
> This object is supported in Internet Explorer only, not in Windows 8.x Store apps.

Syntax

```
enumObj = new Enumerator([collection])
```

Parameters

`enumObj`
Required. The variable name to which the `Enumerator` object is assigned.

`collection`
Optional. Any Collection object.

Remarks

Collections differ from arrays in that the members of a collection are not directly accessible. Instead of using indexes, as you would with arrays, you can move the current item pointer only to the first or next element of a collection.

The `Enumerator` object provides a way to access any member of a collection and behaves similarly to the `For...Each` statement in VBScript.

Example

The following code shows the usage of the `Enumerator` object:

```
var bytesPerGB = 1024 * 1024 * 1024;

var fso = new ActiveXObject("Scripting.FileSystemObject");

document.write(fso.Drives);
var e = new Enumerator(fso.Drives);

var driveString = "";

e.moveFirst();
while (e.atEnd() == false)
{
    var drv = e.item();

    driveString += drv.Path + " - ";

    if (drv.IsReady){
        var freeGB = drv.FreeSpace / bytesPerGB;
        var totalGB = drv.TotalSize / bytesPerGB;

        driveString += freeGB.toFixed(3) + " GB free of ";
        driveString += totalGB.toFixed(3) + " GB";
    }
    else{
        driveString += "Not Ready";
    }

    driveString += "<br />";;

    e.moveNext();
}
document.write(driveString);

// Output: <drive information
```

Properties

The `Enumerator` object has no properties.

Methods

atEnd Method | item Method | moveFirst Method | moveNext Method

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, and Internet Explorer 10 standards. Not supported in Windows 8.x Store apps. See Version Information.

See Also

Boolean Object

atEnd Method (Enumerator) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Returns a Boolean value indicating if the enumerator is at the end of the collection.

> **WARNING**
>
> This object is supported in Internet Explorer only.

Syntax

```
myEnum.atEnd()
```

Remarks

The required *myEnum* reference is any `Enumerator` object.

The `atEnd` method returns **true** if the current item is the last one in the collection, the collection is empty, or the current item is undefined. Otherwise, it returns **false**.

Example

In following code, the `atEnd` method is used to determine if the end of a list of drives has been reached:

```
function ShowDrives()
{
    var s = "";
    var bytesPerGB = 1024 * 1024 * 1024;

    var fso = new ActiveXObject("Scripting.FileSystemObject");
    var e = new Enumerator(fso.Drives);

    e.moveFirst();
    while (e.atEnd() == false)
    {
        var drv = e.item();

        s += drv.Path + " - ";

        if (drv.IsReady)
        {
            var freeGB = drv.FreeSpace / bytesPerGB;
            var totalGB = drv.TotalSize / bytesPerGB;

            s += freeGB.toFixed(3) + " GB free of ";
            s += totalGB.toFixed(3) + " GB";
        }
        else
        {
            s += "Not Ready";
        }

        s += "<br />";

        e.moveNext();
    }
    return(s);
}
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, and Internet Explorer 10 standards. Not supported in Windows 8.x Store apps. See Version Information.

Applies To: Enumerator Object

See Also

item Method (Enumerator)
moveFirst Method (Enumerator)
moveNext Method (Enumerator)
Enumerator Object

item Method (Enumerator) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Returns the current item in the collection.

> **WARNING**
>
> This object is supported in Internet Explorer only.

Syntax

```
enumObj.item()
```

Remarks

The required *enumObj* reference is any `Enumerator` object.

The **item** method returns the current item. If the collection is empty or the current item is undefined, it returns **undefined**.

Example

In following code, the **item** method is used to return a member of the `Drives` collection.

```
function ShowDrives()
{
    var s = "";
    var bytesPerGB = 1024 * 1024 * 1024;

    var fso = new ActiveXObject("Scripting.FileSystemObject");
    var e = new Enumerator(fso.Drives);

    e.moveFirst();
    while (e.atEnd() == false)
    {
        var drv = e.item();

        s += drv.Path + " - ";

        if (drv.IsReady)
        {
            var freeGB = drv.FreeSpace / bytesPerGB;
            var totalGB = drv.TotalSize / bytesPerGB;

            s += freeGB.toFixed(3) + " GB free of ";
            s += totalGB.toFixed(3) + " GB";
        }
        else
        {
            s += "Not Ready";
        }

        s += "<br />";

        e.moveNext();
    }
    return(s);
}
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, and Internet Explorer 10 standards. Not supported in Windows 8.x Store apps. See Version Information.

Applies To: Enumerator Object

See Also

atEnd Method (Enumerator)
moveFirst Method (Enumerator)
moveNext Method (Enumerator)

moveFirst Method (Enumerator) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Resets the current item in the collection to the first item.

> **WARNING**
>
> This object is supported in Internet Explorer only.

Syntax

```
enumObj.moveFirst( )
```

Remarks

The required *enumObj* reference is any `Enumerator` object.

If there are no items in the collection, the current item is set to undefined.

Example

In following example, the `moveFirst` method is used to evaluate members of the `Drives` collection from the beginning of the list:

```
function ShowDrives()
{
    var s = "";
    var bytesPerGB = 1024 * 1024 * 1024;

    var fso = new ActiveXObject("Scripting.FileSystemObject");
    var e = new Enumerator(fso.Drives);

    e.moveFirst();
    while (e.atEnd() == false)
    {
        var drv = e.item();

        s += drv.Path + " - ";

        if (drv.IsReady)
        {
            var freeGB = drv.FreeSpace / bytesPerGB;
            var totalGB = drv.TotalSize / bytesPerGB;

            s += freeGB.toFixed(3) + " GB free of ";
            s += totalGB.toFixed(3) + " GB";
        }
        else
        {
            s += "Not Ready";
        }

        s += "<br />";

        e.moveNext();
    }
    return(s);
}
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, and Internet Explorer 10 standards. Not supported in Windows 8.x Store apps. See Version Information.

Applies To: Enumerator Object

See Also

atEnd Method (Enumerator)
item Method (Enumerator)
moveNext Method (Enumerator)

moveNext Method (Enumerator) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Moves the current item to the next item in the collection.

> **WARNING**
>
> This object is supported in Internet Explorer only.

Syntax

```
enumObj.moveNext( )
```

Remarks

The required *enumObj* reference is any `Enumerator` object.

If the enumerator is at the end of the collection or the collection is empty, the current item is set to undefined.

Example

In following example, the **moveNext** method is used to move to the next drive in the `Drives` collection:

```
function ShowDrives()
{
    var s = "";
    var bytesPerGB = 1024 * 1024 * 1024;

    var fso = new ActiveXObject("Scripting.FileSystemObject");
    var e = new Enumerator(fso.Drives);

    e.moveFirst();
    while (e.atEnd() == false)
    {
        var drv = e.item();

        s += drv.Path + " - ";

        if (drv.IsReady)
        {
            var freeGB = drv.FreeSpace / bytesPerGB;
            var totalGB = drv.TotalSize / bytesPerGB;

            s += freeGB.toFixed(3) + " GB free of ";
            s += totalGB.toFixed(3) + " GB";
        }
        else
        {
            s += "Not Ready";
        }

        s += "<br />";

        e.moveNext();
    }
    return(s);
}
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, and Internet Explorer 10 standards. Not supported in Windows 8.x Store apps. See Version Information.

Applies To: Enumerator Object

See Also

atEnd Method (Enumerator)
item Method (Enumerator)
moveFirst Method (Enumerator)

Error Object (JavaScript)

10/18/2017 • 2 min to read • Edit Online

Contains information about errors.

Syntax

```
      errorObj = new Error()
errorObj = new Error([number])
errorObj = new Error([number[, description]])
```

Parameters

`errorObj`
Required. The variable name to which the `Error` object is assigned. The variable assignment is omitted when you create the error using a `throw` statement.

`number`
Optional. Numeric value assigned to an error. Zero if omitted.

`description`
Optional. Brief string that describes an error. Empty string if omitted.

Remarks

Whenever a run-time error occurs, an instance of the `Error` object is created to describe the error. This instance has two intrinsic properties that contain the description of the error (`description` property) and the error number (`number` property). For more information, see JavaScript Run-time Errors.

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

`Error` objects can also be explicitly created, using the syntax shown above, or thrown using the `throw` statement. In both cases, you can add any properties you choose to expand the capability of the `Error` object.

Typically, the local variable that is created in a **try...catch** statement refers to the implicitly created `Error` object. As a result, you can use the error number and description in any way you choose.

Example

The following example illustrates the use of the `Error` object.

```
function checkInput(x) {
    try
    {
        if (isNaN(parseInt(x))) {
            throw new Error("Input is not a number.");
        }
    }
    catch(e)
    {
        document.write(e.description);
    }
}

checkInput("not a number");
```

Example

The following example illustrates the use of the implicitly created `Error` object.

```
try
   {
   // Cause an error.
   x = y;
   }
catch(e)
   {
      document.write(e);
      document.write ("<br />");

      document.write ("Number: ");
      document.write (e.number & 0xFFFF);
      document.write ("<br />");

      document.write ("Facility Code: ");
      document.write(e.number>>16 & 0x1FFF);
      document.write ("<br />");

      document.write ("Description: ");
      document.write (e.description);
   }

// Output:
// ReferenceError: 'y' is undefined
// Number: 5009
// Facility Code: 10
// Description: 'y' is undefined
```

Methods

toString Method (Error) | valueOf Method (Date)

Properties

constructor Property (Error) | description Property | message Property | name Property | number Property | prototype Property (Error) | stack Property (Error) | stackTraceLimit Property (Error)

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet

Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

See Also

new Operator
throw Statement
try...catch...finally Statement
var Statement
Hilo JavaScript sample app (Windows Store)

constructor Property (Error)

10/18/2017 • 1 min to read • Edit Online

Specifies the function that creates an Error.

Syntax

```
error.constructor
```

Remarks

The required `error` is the name of an error object.

The `constructor` property is a member of the prototype of every object that has a prototype. The `constructor` property contains a reference to the function that constructs instances of that particular object.

Example

The following example illustrates the use of the constructor property.

```
var x = new Error("This is an error");

if (x.constructor == Error)
    document.write("Object is an error.");

// Output:
// Object is an error.
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

prototype Property (Error)

10/18/2017 • 1 min to read • Edit Online

Returns a reference to the prototype for an error.

Syntax

```
error.prototype
```

Remarks

The `error` argument is the name of an error.

Use the `prototype` property to provide a base set of functionality to an Error. New instances of an object "inherit" the behavior of the prototype assigned to that object.

For example, to add a method to the `Error` object that returns the value of the largest element of the array, declare the function, add it to `Error.prototype`, and then use it.

```
function getSeverity(){
    if (this.number > 1000)
        return "high";
    else
        return "low";
}
Error.prototype.getSev = getSeverity;
var myError = new Error();
myError.number = 5000;

document.write(myError.getSev());

// Output: high
```

All intrinsic JavaScript objects have a `prototype` property that is read-only. Properties and methods may be added to the prototype, but the object may not be assigned a different prototype. However, user-defined objects may be assigned a new prototype.

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

description Property (Error) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Returns or sets the descriptive string associated with a specific error.

Syntax

```
object
.description [= stringExpression]
```

Parameters

object
Required. Any instance of an `Error` object.

`stringExpression`
Optional. A string expression containing a description of the error.

Remarks

The **description** property contains the error message string associated with a specific error. Use the value contained in this property to alert a user to an error.

The **description** and **message** properties provide the same functionality; the **description** property provides backward compatibility; the **message** property complies with the ECMA standard.

Example

The following example illustrates the use of the **description** property.

```
try
{
// Cause an error:
    x = y
}
catch(e)
{
// Prints "[object Error]":
    document.write(e)
    document.write (" ");
// Prints 5009:
    document.write((e.number & 0xFFFF))
    document.write (" ");
// Prints "'y' is undefined":
    document.write(e.description);
    document.write (" ");
// Prints "'y' is undefined":
    document.write(e.message)
}
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

Applies To: Error Object

See Also

number Property (Error)
message Property (Error)
name Property (Error)

message Property (Error) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Returns an error message string.

Syntax

```
errorObj.message
```

Parameters

`errorObj`
Required. Instance of `Error` object.

Remarks

The `message` property returns a string that contains an error message associated with a specific error.

The `description` and `message` properties provide the same functionality. The `description` property provides backwards compatibility; the `message` property complies with the ECMA standard.

Example

The following example causes a TypeError exception to be thrown and displays the name of the error and its message.

```
try
{
    // Cause an error.
    var x = y;
}
catch(e)
{
    document.write ("Error Message: " + e.message);
    document.write ("<br />");
    document.write ("Error Code: ");
    document.write (e.number & 0xFFFF)
    document.write ("<br />");
    document.write ("Error Name: " + e.name);
}
```

Example

The output of this code is as follows.

```
Error Message: 'y' is undefined
Error Code: 5009
Error Name: TypeError
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

Applies To: Error Object

See Also

description Property (Error)
name Property (Error)

name Property (Error) (JavaScript)

10/18/2017 • 2 min to read • Edit Online

Returns the name of an error.

Syntax

```
errorObj.
name
```

Parameters

`errorObj`
Required. Instance of `Error` object.

Remarks

The **name** property returns the name or exception type of an error. When a runtime error occurs, the name property is set to one of the following native exception types:

EXCEPTION TYPE	MEANING
ConversionError	This error occurs whenever there is an attempt to convert an object into something to which it cannot be converted.
RangeError	This error occurs when a function is supplied with an argument that has exceeded its allowable range. For example, this error occurs if you attempt to construct an `Array` object with a length that is not a valid positive integer.
ReferenceError	This error occurs when an invalid reference has been detected. This error will occur, for example, if an expected reference is `null`.
RegExpError	This error occurs when a compilation error occurs with a regular expression. Once the regular expression is compiled, however, this error cannot occur. This example will occur, for example, when a regular expression is declared with a pattern that has an invalid syntax, or flags other than **i**, **g**, or **m**, or if it contains the same flag more than once.
SyntaxError	This error occurs when source text is parsed and that source text does not follow correct syntax. This error will occur, for example, if the `eval` function is called with an argument that is not valid program text.
TypeError	This error occurs whenever the actual type of an operand does not match the expected type. An example of when this error occurs is a function call made on something that is not an object or does not support the call.

EXCEPTION TYPE	MEANING
URIError	This error occurs when an illegal Uniform Resource Indicator (URI) is detected. For example, this is error occurs when an illegal character is found in a string being encoded or decoded.

Example

The following example causes a TypeError exception to be thrown and displays the name of the error and its message.

```
try
{
    var x = y;
}
catch(e)
{
    document.write ("Error Message: " + e.message);
    document.write ("<br />");
    document.write ("Error Code: ");
    document.write (e.number & 0xFFFF)
    document.write ("<br />");
    document.write ("Error Name: " + e.name);
}
```

Example

The output of this code is as follows.

```
Error Message: 'y' is undefined
Error Code: 5009
Error Name: TypeError
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

Applies To: Error Object

See Also

description Property (Error)
message Property (Error)
number Property (Error)

number Property (Error) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Returns or sets the numeric value associated with a specific error. The `Error` object's default property is **number**.

Syntax

```
object
.number [= errorNumber]
```

Parameters

Object
Any instance of the `Error` object.

errorNumber
An integer representing an error.

Remarks

An error number is a 32-bit value. The upper 16-bit word is the facility code, and the lower word is the error code. To determine the error code, use the `&` (bitwise And) operator to combine the number property with the hexadecimal number `0xFFFF`.

Example

The following example causes an exception to be thrown and displays the error code that is derived from the error number.

```
try
    {
    // Cause an error.
    var x = y;
    }
catch(e)
    {
    document.write ("Error Code: ");
    document.write (e.number & 0xFFFF)
    document.write ("<br />");

    document.write ("Facility Code: ")
    document.write(e.number>>16 & 0x1FFF)
    document.write ("<br />");

    document.write ("Error Message: ")
    document.write (e.message)
    }
```

Example

The output of this code is as follows.

```
Error Code: 5009
Facility Code: 10
Error Message: 'y' is undefined
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

Applies To: Error Object

See Also

description Property (Error)
message Property (Error)
name Property (Error)

stack Property (Error) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Gets or sets the error stack as a string that contains the stack trace frames.

Syntax

```
object
.stack
```

Remarks

The `stack` property is set to `undefined` when the error is constructed, and gets the trace information when the error is raised. If an error is raised multiple times, the `stack` property is updated each time the error is raised.

Stack frames are displayed in the following format: **at FunctionName (<Fully-qualified name/URL>:<line number>:<column number>)**

If you create your own Error object and set the stack trace to a value, the value won't be overwritten when the error is thrown.

The `stack` property does not show inline functions in its frames. It shows only the physical stack.

Example

The following example shows how to get the stack when you're catching an error.

```
try
    {
        var x = y.name;
    }
catch(e)
    {
        document.write ("Error stack: ")
        document.write (e.stack);
    }
```

Example

The following example shows how to set and then get the stack.

```
try
    {
        var err = Error("my error");
        err.stack = "my stack trace";
        throw err;
    }
catch(e)
    {
        document.write ("Error stack: ")
        document.write (e.stack);
    }
```

Requirements

Supported in the following document modes: Internet Explorer 10 standards and Internet Explorer 11 standards.
Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards.

Applies To: Error Object

See Also

description Property (Error)
message Property (Error)
name Property (Error)
stackTraceLimit Property (Error)

stackTraceLimit Property (Error) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Gets or sets the stack trace limit, which is equivalent to the number of error frames to display. The default limit is 10.

Syntax

```
Error
.stackTraceLimit
```

Remarks

You can set the `stackTraceLimit` property to any positive value between 0 and `Infinity`. If the `stackTraceLimit` property is set to 0 at the time an error is thrown, no stack trace is shown. If the property is set to a negative value or a non-numeric value, the value is converted to 0. If the stackTraceLimit is set to `Infinity`, the entire stack is shown. Otherwise, `ToUint32` is used to convert the value.

Example

The following example shows how to set and then get the stack trace limit.

```
try
    {
    var err = new Error("my error");
    Error.stackTraceLimit = 7;
    throw err;
    }
catch(e)
    {
    document.write ("Error stack trace limit: ")
    document.write (Error.stackTraceLimit);
    }
```

Requirements

Supported in Internet Explorer 10 and in Windows 8.x Store apps.

Applies To: Error Object

See Also

description Property (Error)
message Property (Error)
name Property (Error)
stack Property (Error)

toString Method (Error)

10/18/2017 • 1 min to read • Edit Online

Returns a string representation of an error.

Syntax

```
error.toString()
```

Parameters

`date`

Required. The error to represent as a string.

Return Value

Returns "Error: " plus the error message.

Example

The following example illustrates the use of the `toString` method with an error.

```
var myError = new Error();
myError.message = "My Error";
var errorString = myError.toString();
document.write(errorString);

// Output: Error: My Error
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

valueOf Method (Error)

10/18/2017 • 1 min to read • Edit Online

Returns the string value of an error.

Syntax

```
error.valueOf()
```

Parameters

The `error` object is any instance of an Error.

Return Value

The string "Error: " plus the error message.

Example

The following example illustrates the use of the `valueOF` method with a date.

```
var myError = new Error();
myError.message = "This is an error.";
var value = myError.valueOf();
document.write(value);

// Output: Error: This is an error.
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

Float32Array Object

10/18/2017 • 2 min to read • Edit Online

A typed array of 32-bit float values. The contents are initialized to 0. If the requested number of bytes could not be allocated an exception is raised.

Syntax

```
      float32Array = new Float32Array( length );
float32Array = new Float32Array( array );
float32Array = new Float32Array( buffer, byteOffset, length);
```

Parameters

`float32Array`
Required. The variable name to which the **Float32Array** object is assigned.

`length`
Specifies the number of elements in the array.

`array`
The array (or typed array) that is contained in this array. The contents are initialized to the contents of the given array or typed array, with each element converted to the Float32 type.

`buffer`
The ArrayBuffer that the Float32Array represents.

`byteOffset`
Optional. Specifies the offset in bytes from the start of the buffer at which the Float32Array should begin.

`length`
The number of elements in the array.

Constants

The following table lists the constants of the `Float32Array` object.

CONSTANT	DESCRIPTION
BYTES_PER_ELEMENT Constant	The size in bytes of each element in the array.

Properties

The following table lists the constants of the `Float32Array` object.

PROPERTY	DESCRIPTION
buffer Property	Read-only. Gets the ArrayBuffer that is referenced by this array.

PROPERTY	DESCRIPTION
byteLength Property	Read-only. The length of this array from the start of its ArrayBuffer, in bytes, as fixed at construction time.
byteOffset Property	Read-only. The offset of this array from the start of its ArrayBuffer, in bytes, as fixed at construction time.
length Property	The length of the array.

Methods

The following table lists the methods of the `Float32Array` object.

METHOD	DESCRIPTION
get Method	Omittable. Gets the element at the specified index.
set Method (Float32Array)	Sets a value or an array of values.
subarray Method (Float32Array)	Gets a new Float32Array view of the ArrayBuffer store for this array.

Example

The following example shows how to use a Float32Array object to process the binary data acquired from an XmlHttpRequest:

```
var req = new XMLHttpRequest();
    req.open('GET', "http://www.example.com");
    req.responseType = "arraybuffer";
    req.send();

    req.onreadystatechange = function () {
        if (req.readyState === 4) {
            var buffer = req.response;
            var dataview = new DataView(buffer);
            var ints = new Float32Array(buffer.byteLength / 4);
            for (var i = 0; i < ints.length; i++) {
                ints[i] = dataview.getFloat32(i * 4);
            }
        alert(ints[10]);
        }
    }
```

Requirements

Supported in the following document modes: Internet Explorer 10 standards and Internet Explorer 11 standards.
Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards.

BYTES_PER_ELEMENT Constant (Float32Array)

10/18/2017 • 1 min to read • Edit Online

The size in bytes of each element in the array.

Syntax

```
var arraySize = int8Array.BYTES_PER_ELEMENT;
```

Example

The following example shows how to get the size of the array elements.

```
var req = new XMLHttpRequest();
    req.open('GET', "http://www.example.com");
    req.responseType = "arraybuffer";
    req.send();

    req.onreadystatechange = function () {
        if (req.readyState === 4) {
            var buffer = req.response;
            var dataView = new DataView(buffer);
            var floatArr = new Float32Array(buffer.byteLength / 4);
            alert(floatArr.BYTES_PER_ELEMENT);
        }
    }
```

Requirements

Supported in the following document modes: Internet Explorer 10 standards and Internet Explorer 11 standards.
Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards.

buffer Property (Float32Array)

10/18/2017 • 1 min to read • Edit Online

Read-only. Gets the ArrayBuffer that is referenced by this array.

Syntax

```
var arrayBuffer = float32Array.buffer;
```

Example

The following example shows how to get the ArrayBuffer of the array.

```
var req = new XMLHttpRequest();
    req.open('GET', "http://www.example.com");
    req.responseType = "arraybuffer";
    req.send();

    req.onreadystatechange = function () {
        if (req.readyState === 4) {
            var buffer = req.response;
            var dataView = new DataView(buffer);
            var floatArr = new Float32Array(buffer.byteLength / 4);
            alert(floatArr.buffer.byteLength);
        }
    }
```

Requirements

Supported in the following document modes: Internet Explorer 10 standards and Internet Explorer 11 standards.
Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards.

byteLength Property (Float32Array)

10/18/2017 • 1 min to read • Edit Online

Read-only. The length of this array from the start of its ArrayBuffer, in bytes, as fixed at construction time.

Syntax

```
var arrayByteLength = float32Array.byteLength;
```

Example

The following example shows how to get the length of the array.

```
var req = new XMLHttpRequest();
    req.open('GET', "http://www.example.com");
    req.responseType = "arraybuffer";
    req.send();

    req.onreadystatechange = function () {
        if (req.readyState === 4) {
            var buffer = req.response;
            var dataView = new DataView(buffer);
            var floatArr = new Float32Array(buffer.byteLength) / 4);
            alert(floatArr.byteLength);
        }
    }
```

Requirements

Supported in the following document modes: Internet Explorer 10 standards and Internet Explorer 11 standards.
Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards.

byteOffset Property (Float32Array)

10/18/2017 • 1 min to read • Edit Online

Read-only. The offset of this array from the start of its ArrayBuffer, in bytes, as fixed at construction time.

Syntax

```
var arrayOffset = float32Array.byteOffset;
```

Example

The following example shows how to get the offset of the array.

```
var req = new XMLHttpRequest();
    req.open('GET', "http://www.example.com");
    req.responseType = "arraybuffer";
    req.send();

    req.onreadystatechange = function () {
        if (req.readyState === 4) {
            var buffer = req.response;
            var dataView = new DataView(buffer);
            var floatArr = new Float32Array(buffer.byteLength / 4);
            alert(floatArr.byteOffset);
        }
    }
```

Requirements

Supported in the following document modes: Internet Explorer 10 standards and Internet Explorer 11 standards.
Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards.

length Property (Float32Array)

10/18/2017 • 1 min to read • Edit Online

The length of the array.

Syntax

```
var arrayLength = float32Array.length;
```

Example

The following example shows how to get the length of the array.

```
var req = new XMLHttpRequest();
    req.open('GET', "http://www.example.com");
    req.responseType = "arraybuffer";
    req.send();

    req.onreadystatechange = function () {
        if (req.readyState === 4) {
            var buffer = req.response;
            var dataView = new DataView(buffer);
            var floatArr = new Float32Array(buffer.byteLength / 4);
            alert(floatArr.length);
        }
    }
```

Requirements

Supported in the following document modes: Internet Explorer 10 standards and Internet Explorer 11 standards.
Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards.

get Method (Float32Array)

10/18/2017 • 1 min to read • Edit Online

Omittable. Gets the element at the specified index.

Syntax

```
var value = float32Array.get(index);
```

Parameters

`value`

The value returned by this method.

`index`

The index at which to get the element of the array.

Example

The following example shows how to get the first element of the array.

```
var req = new XMLHttpRequest();
    req.open('GET', "http://www.example.com");
    req.responseType = "arraybuffer";
    req.send();

    req.onreadystatechange = function () {
        if (req.readyState === 4) {
            var buffer = req.response;
            var dataView = new DataView(buffer);
            var floatArr = new Float32Array(buffer.byteLength / 4);
            var element = floatArr.getFloat32(0);
        }
    }
```

Requirements

Supported in the following document modes: Internet Explorer 10 standards and Internet Explorer 11 standards.
Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards.

set Method (Float32Array)

10/18/2017 • 1 min to read • Edit Online

Sets a value or an array of values.

Syntax

```
float32Array.set(index, value);
float32Array.set(array, offset);
```

Parameters

`index`
The index of the location to set.

`value`
The value to set.

`array`
A typed or untyped array of values to set.

`offset`
The index in the current array at which the values are to be written.

Remarks

If the input array is a TypedArray, the two arrays may use the same underlying ArrayBuffer. In this situation, setting the values takes place as if all the data is first copied into a temporary buffer that does not overlap either of the arrays, and then the data from the temporary buffer is copied into the current array.

If the offset plus the length of the given array is out of range for the current TypedArray, an exception is raised.

Example

The following example shows how to set the first element of the array.

```
var req = new XMLHttpRequest();
    req.open('GET', "http://www.example.com");
    req.responseType = "arraybuffer";
    req.send();

    req.onreadystatechange = function () {
        if (req.readyState === 4) {
            var buffer = req.response;
            var dataView = new DataView(buffer);
            var floatArr = new Float32Array(buffer.byteLength / 4);
            floatArr.set(0, 9);
        }
    }
```

Requirements

Supported in the following document modes: Internet Explorer 10 standards and Internet Explorer 11 standards.
Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards.

subarray Method (Float32Array)

10/18/2017 • 1 min to read • Edit Online

Gets a new Float32Array view of the ArrayBuffer Object store for this array, specifying the first and last members of the subarray.

Syntax

```
var newFloat32Array = float32Array.subarray(begin, end);
```

Parameters

`newFloat32Array`
The subarray returned by this method.

`begin`
The index of the beginning of the array.

`end`
The index of the end of the array.

Remarks

If either `begin` or `end` is negative, it refers to an index from the end of the array, as opposed to from the beginning. If `end` is unspecified, the subarray contains all elements from `begin` to the end of the typed array. The range specified by the `begin` and `end` values is clamped to the valid index range for the current array. If the computed length of the new typed array would be negative, it is clamped to zero. The returned array is of the same type as the array on which this method is invoked.

Example

The following example shows how to get a subarray three elements long, starting with the first element of the array.

```
var req = new XMLHttpRequest();
    req.open('GET', "http://www.example.com");
    req.responseType = "arraybuffer";
    req.send();

    req.onreadystatechange = function () {
        if (req.readyState === 4) {
            var buffer = req.response;
            var floatArr = new Float32Array(buffer.byteLength / 4);
            var subArr = floatArr.subarray(0, 2);
        }
    }
```

Requirements

Supported in the following document modes: Internet Explorer 10 standards and Internet Explorer 11 standards.

Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards.

Float64Array Object

10/18/2017 • 2 min to read • Edit Online

A typed array of 64-bit float values. The contents are initialized to 0. If the requested number of bytes could not be allocated an exception is raised.

Syntax

```
      float64Array = new Float64Array( length );
float64Array = new Float64Array( array );
float64Array = new Float64Array( buffer, byteOffset, length);
```

Parameters

`float64Array`
Required. The variable name to which the **Float64Array** object is assigned.

`length`
Specifies the number of elements in the array.

`array`
The array (or typed array) that is contained in this array.. The contents are initialized to the contents of the given array or typed array, with each element converted to the Float64 type.

`buffer`
The ArrayBuffer that the Float64Array represents.

`byteOffset`
Optional. Specifies the offset in bytes from the start of the buffer at which the Float64Array should begin.

`length`
The number of elements in the array.

Constants

The following table lists the constants of the `Float64Array` object.

CONSTANT	DESCRIPTION
BYTES_PER_ELEMENT Constant	The size in bytes of each element in the array.

Properties

The following table lists the constants of the `Float64Array` object.

PROPERTY	DESCRIPTION
buffer Property	Read-only. Gets the ArrayBuffer that is referenced by this array.

PROPERTY	DESCRIPTION
byteLength Property	Read-only. The length of this array from the start of its ArrayBuffer, in bytes, as fixed at construction time.
byteOffset Property	Read-only. The offset of this array from the start of its ArrayBuffer, in bytes, as fixed at construction time.
length Property	The length of the array.

Methods

The following table lists the methods of the `Float64Array` object.

METHOD	DESCRIPTION
get Method	Omittable. Gets the element at the specified index.
set Method (Float64Array)	Sets a value or an array of values.
subarray Method (Float64Array)	Gets a new Float64Array view of the ArrayBuffer store for this array.

Example

The following example shows how to use a Float64Array object to process the binary data acquired from an XmlHttpRequest:

```
var req = new XMLHttpRequest();
    req.open('GET', "http://www.example.com");
    req.responseType = "arraybuffer";
    req.send();

    req.onreadystatechange = function () {
        if (req.readyState === 4) {
            var buffer = req.response;
            var dataview = new DataView(buffer);
            var ints = new Float64Array(buffer.byteLength / 8);
            for (var i = 0; i < ints.length; i++) {
                ints[i] = dataview.getFloat64(i * 8);
            }
        alert(ints[10]);
        }
    }
```

Requirements

Supported in the following document modes: Internet Explorer 10 standards and Internet Explorer 11 standards.
Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards.

BYTES_PER_ELEMENT Constant (Float64Array)

10/18/2017 • 1 min to read • Edit Online

The size in bytes of each element in the array.

Syntax

```
var arraySize = int8Array.BYTES_PER_ELEMENT;
```

Example

The following example shows how to get the size of the array elements.

```
var req = new XMLHttpRequest();
    req.open('GET', "http://www.example.com");
    req.responseType = "arraybuffer";
    req.send();

    req.onreadystatechange = function () {
        if (req.readyState === 4) {
            var buffer = req.response;
            var dataView = new DataView(buffer);
            var floatArr = new Float64Array(buffer.byteLength / 8);
            alert(floatArr.BYTES_PER_ELEMENT);
        }
    }
```

Requirements

Supported in the following document modes: Internet Explorer 10 standards and Internet Explorer 11 standards.
Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards.

buffer Property (Float64Array)

10/18/2017 • 1 min to read • Edit Online

Read-only. Gets the ArrayBuffer that is referenced by this array.

Syntax

```
var arrayBuffer = float64Array.buffer;
```

Example

The following example shows how to get the ArrayBuffer of the array.

```
var req = new XMLHttpRequest();
    req.open('GET', "http://www.example.com");
    req.responseType = "arraybuffer";
    req.send();

    req.onreadystatechange = function () {
        if (req.readyState === 4) {
            var buffer = req.response;
            var dataView = new DataView(buffer);
            var floatArr = new Float64Array(buffer.byteLength / 8);
            alert(floatArr.buffer.byteLength);
        }
    }
```

Requirements

Supported in the following document modes: Internet Explorer 10 standards and Internet Explorer 11 standards.
Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards.

byteLength Property (Float64Array)

10/18/2017 • 1 min to read • Edit Online

Read-only. The length of this array from the start of its ArrayBuffer, in bytes, as fixed at construction time.

Syntax

```
var arrayByteLength = float64Array.byteLength;
```

Example

The following example shows how to get the byte length of the array.

```
var req = new XMLHttpRequest();
    req.open('GET', "http://www.example.com");
    req.responseType = "arraybuffer";
    req.send();

    req.onreadystatechange = function () {
        if (req.readyState === 4) {
            var buffer = req.response;
            var dataView = new DataView(buffer);
            var floatArr = new Float64Array(buffer.byteLength / 8);
            alert(floatArr.byteLength);
        }
    }
```

Requirements

Supported in the following document modes: Internet Explorer 10 standards and Internet Explorer 11 standards.
Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards.

bufferOffset Property (Float64Array)

10/18/2017 • 1 min to read • Edit Online

Read-only. The offset of this array from the start of its ArrayBuffer, in bytes, as fixed at construction time.

Syntax

```
var arrayOffset = float64Array.byteOffset;
```

Example

The following example shows how to get the offset of the array.

```
var req = new XMLHttpRequest();
    req.open('GET', "http://www.example.com");
    req.responseType = "arraybuffer";
    req.send();

    req.onreadystatechange = function () {
        if (req.readyState === 4) {
            var buffer = req.response;
            var dataView = new DataView(buffer);
            var floatArr = new Float64Array(buffer.byteLength / 8);
            alert(floatArr.byteOffset);
        }
    }
```

Requirements

Supported in the following document modes: Internet Explorer 10 standards and Internet Explorer 11 standards.
Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards.

length Property (Float64Array)

10/18/2017 • 1 min to read • Edit Online

The length of the array.

Syntax

```
var arrayLength = float64Array.length;
```

Example

The following example shows how to get the length of the array.

```
var req = new XMLHttpRequest();
    req.open('GET', "http://www.example.com");
    req.responseType = "arraybuffer";
    req.send();

    req.onreadystatechange = function () {
        if (req.readyState === 4) {
            var buffer = req.response;
            var dataView = new DataView(buffer);
            var floatArr = new Float64Array(buffer.byteLength / 8);
            alert(floatArr.length);
        }
    }
```

Requirements

Supported in the following document modes: Internet Explorer 10 standards and Internet Explorer 11 standards.
Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards.

get Method (Float64Array)

10/18/2017 • 1 min to read • Edit Online

Omittable. Gets the element at the specified index.

Syntax

```
var value = float64Array.get(index);
```

Parameters

`value`

The value returned by this method.

`index`

The index at which to get the element of the array.

Example

The following example shows how to get the first element of the array.

```
var req = new XMLHttpRequest();
    req.open('GET', "http://www.example.com");
    req.responseType = "arraybuffer";
    req.send();

    req.onreadystatechange = function () {
        if (req.readyState === 4) {
            var buffer = req.response;
            var dataView = new DataView(buffer);
            var floatArr = new Float64Array(buffer.byteLength / 4);
            var element = floatArr.getFloat64(0);
        }
    }
```

Requirements

Supported in the following document modes: Internet Explorer 10 standards and Internet Explorer 11 standards.
Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards.

set Method (Float64Array)

10/18/2017 • 1 min to read • Edit Online

Sets a value or an array of values.

Syntax

```
float64Array.set(index, value);
float64Array.set(array, offset);
```

Parameters

`index`
The index of the location to set.

`value`
The value to set.

`array`
A typed or untyped array of values to set.

`offset`
The index in the current array at which the values are to be written.

Remarks

If the input array is a TypedArray, the two arrays may use the same underlying ArrayBuffer. In this situation, setting the values takes place as if all the data is first copied into a temporary buffer that does not overlap either of the arrays, and then the data from the temporary buffer is copied into the current array.

If the offset plus the length of the given array is out of range for the current TypedArray, an exception is raised.

Example

The following example shows how to set the first element of the array.

```
var req = new XMLHttpRequest();
    req.open('GET', "http://www.example.com");
    req.responseType = "arraybuffer";
    req.send();

    req.onreadystatechange = function () {
        if (req.readyState === 4) {
            var buffer = req.response;
            var dataView = new DataView(buffer);
            var floatArr = new Float64Array(buffer.byteLength / 8);
            floatArr.set(0, 9.1);
        }
    }
```

Requirements

Supported in the following document modes: Internet Explorer 10 standards and Internet Explorer 11 standards.
Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards.

subarray Method (Float64Array)

10/18/2017 • 1 min to read • Edit Online

Gets a new Float64Array view of the ArrayBuffer Object store for this array, specifying the first and last members of the subarray.

Syntax

```
var newFloat64Array = float64Array.subarray(begin, end);
```

Parameters

`newFloat64Array`
The subarray returned by this method.

`begin`
The index of the beginning of the array.

`end`
The index of the end of the array.

Remarks

If either `begin` or `end` is negative, it refers to an index from the end of the array, as opposed to from the beginning. If `end` is unspecified, the subarray contains all elements from `begin` to the end of the typed array. The range specified by the `begin` and `end` values is clamped to the valid index range for the current array. If the computed length of the new typed array would be negative, it is clamped to zero. The returned array is of the same type as the array on which this method is invoked.

Example

The following example shows how to get a subarray three elements long, starting with the first element of the array.

```
var req = new XMLHttpRequest();
    req.open('GET', "http://www.example.com");
    req.responseType = "arraybuffer";
    req.send();

    req.onreadystatechange = function () {
        if (req.readyState === 4) {
            var buffer = req.response;
            var floatArr = new Float64Array(buffer.byteLength / 8);
            var subArr = floatArr.subarray(0, 2);
        }
    }
```

Requirements

Supported in the following document modes: Internet Explorer 10 standards and Internet Explorer 11 standards.

Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards.

Function Object (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Creates a new function.

Syntax

```
function functionName([argname1  [, ...[, argnameN]]])
{
   body
}
```

Syntax

```
functionName = new Function( [argname1,  [... argnameN,]] body );
```

Parameters

`functionName`
Required. The name of the newly created function

`argname1...argnameN`
Optional. A list of arguments the function accepts.

`body`
Optional. A string that contains the block of JavaScript code to be executed when the function is called.

Remarks

The function is a basic data type in JavaScript. Syntax 1 creates a function value that JavaScript converts into a `Function` object when necessary. JavaScript converts `Function` objects created by Syntax 2 into function values at the time the function is called.

Syntax 1 is the standard way to create new functions in JavaScript. Syntax 2 is an alternative form used to create function objects explicitly.

For example, to declare a function that adds the two arguments passed to it, you can do it in one of two ways:

Example 1

```
function add(x, y)
{
   return(x + y);
}
```

Example 2

```
var add = function(x, y) {
     return(x+y);
}
```

In either case, you call the function with a line of code similar to the following:

```
add(2, 3);
```

> **NOTE**
>
> When you call a function, make sure that you always include the parentheses and any required arguments. Calling a function without parentheses causes the function itself to be returned, instead of the return value of the function.

Properties

0...n Properties |arguments Property | callee Property | caller Property | constructor Property | length Property (Function) | prototype Property

Methods

apply Method | bind Method | call Method | toString Method | valueOf Method

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

See Also

function Statement
new Operator
var Statement

arguments Property (Function) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Gets the arguments for the currently executing `Function` object.

Syntax

```
function.arguments
```

Remarks

The `function` argument is the name of the currently executing function, and can be omitted.

This property allows a function to handle a variable number of arguments. The **length** property of the `arguments` object contains the number of arguments passed to the function. The individual arguments contained in the `arguments` object can be accessed in the same way array elements are accessed.

Example

The following example illustrates the use of the `arguments` property:

```
function ArgTest(arg1, arg2){
   var s = "";
   s += "The individual arguments are: "
   for (n = 0; n < arguments.length; n++){
      s += ArgTest.arguments[n];
      s += " ";
   }
   return(s);
}
document.write(ArgTest(1, 2, "hello"));

//Output: function ArgTest(arg1, arg2){
   var s = "";
   s += "The individual arguments are: "
   for (n = 0; n < arguments.length; n++){
      s += ArgTest.arguments[n];
      s += " ";
   }
   return(s);
}
document.write(ArgTest(1, 2, "hello"));

// Output: The individual arguments are: 1 2 hello
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

See Also

arguments Object
function Statement

caller Property (Function) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Gets the function that invoked the current function.

Syntax

```
functionName.caller
```

Remarks

The `functionName` object is the name of any executing function.

The `caller` property is defined for a function only while that function is executing. If the function is called from the top level of a JavaScript 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 following example illustrates the use of the `caller` property:

```
function CallLevel(){
   if (CallLevel.caller == null)
      return("CallLevel was called from the top level.");
   else
      return("CallLevel was called by another function.");
}

document.write(CallLevel());

// Output: CallLevel was called from the top level.
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

See Also

function Statement

length Property (Function) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Gets the number of arguments defined for a function.

Syntax

```
functionName.length
```

Remarks

The required *functionName* is the name of the function.

The **length** property of a function is initialized by the scripting engine to the number of arguments in the function's definition when an instance of the function is created.

What happens when a function is called with a number of arguments different from the value of its **length** property depends on the function.

Example

The following example illustrates the use of the **length** property:

```
function ArgTest(a, b){
    var s = "";

    s += "Expected Arguments: " + ArgTest.length;
    s += "<br />";
    s += "Passed Arguments: " + arguments.length;

    return s;
}

document.write(ArgTest(1, 2));

// Output:
// Expected Arguments: 2
// Passed Arguments: 2
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

See Also

arguments Property (Function)
length Property (Array)
length Property (String)

apply Method (Function) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Calls the function, substituting the specified object for the `this` value of the function, and the specified array for the arguments of the function.

Syntax

```
apply([thisObj[,argArray]])
```

Parameters

`thisObj`
Optional. The object to be used as the `this` object.

`argArray`
Optional. A set of arguments to be passed to the function.

Remarks

If `argArray` is not a valid object, then an "Object expected" error occurs.

If neither `argArray` nor `thisObj` are supplied, the original `this` object is used as `thisObj` and no arguments are passed.

Example

The following code shows how to use the apply method.

```
function callMe(arg1, arg2){
    var s = "";

    s += "this value: " + this;
    s += "<br />";
    for (i in callMe.arguments) {
        s += "arguments: " + callMe.arguments[i];
        s += "<br />";
    }
    return s;
}

document.write("Original function: <br/>");
document.write(callMe(1, 2));
document.write("<br/>");

document.write("Function called with apply: <br/>");
document.write(callMe.apply(3, [ 4, 5 ]));

// Output:
// Original function:
// this value: [object Window]
// arguments: 1
// arguments: 2

// Function called with apply:
// this value: 3
// arguments: 4
// arguments: 5
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

See Also

Function Object

bind Method (Function) (JavaScript)

10/18/2017 • 2 min to read • Edit Online

For a given function, creates a bound function that has the same body as the original function. In the bound function, the `this` object resolves to the passed in object. The bound function has the specified initial parameters.

Syntax

```
function.bind(thisArg[,arg1[,arg2[,argN]]])
```

Parameters

`function`
Required. A function object.

`thisArg`
Required. An object to which the `this` keyword can refer inside the new function.

`arg1` [, `arg2` [, `argN`]]]
Optional. A list of arguments to be passed to the new function.

Return Value

A new function that is the same as the `function` function, except for the `thisArg` object and the initial arguments.

Exceptions

If the specified `function` is not a function, a `TypeError` exception is thrown.

Example

The following code shows how to use the `bind` method.

```
// Define the original function.
var checkNumericRange = function (value) {
    if (typeof value !== 'number')
        return false;
    else
        return value >= this.minimum && value <= this.maximum;
}

// The range object will become the this value in the callback function.
var range = { minimum: 10, maximum: 20 };

// Bind the checkNumericRange function.
var boundCheckNumericRange = checkNumericRange.bind(range);

// Use the new function to check whether 12 is in the numeric range.
var result = boundCheckNumericRange (12);
document.write(result);

// Output: true
```

Example

In the following example, the `thisArg` object is different from the object that contains the original method.

```
// Create an object that contains the original function.
var originalObject = {
    minimum: 50,
    maximum: 100,
    checkNumericRange: function (value) {
        if (typeof value !== 'number')
            return false;
        else
            return value >= this.minimum && value <= this.maximum;
    }
}

// Check whether 10 is in the numeric range.
var result = originalObject.checkNumericRange(10);
document.write(result + " ");
// Output: false

// The range object supplies the range for the bound function.
var range = { minimum: 10, maximum: 20 };

// Create a new version of the checkNumericRange function that uses range.
var boundObjectWithRange = originalObject.checkNumericRange.bind(range);

// Check whether 10 is in the numeric range.
var result = boundObjectWithRange(10);
document.write(result);
// Output: true
```

Example

The following code shows how to use the `arg1[,arg2[,argN]]]` arguments. The bound function uses the parameters specified in the `bind` method as the first and second parameters. Any parameters specified when the bound function is called are used as the third, fourth (and so on) parameters.

```
// Define the original function with four parameters.
var displayArgs = function (val1, val2, val3, val4) {
    document.write(val1 + " " + val2 + " " + val3 + " " + val4);
}

var emptyObject = {};

// Create a new function that uses the 12 and "a" parameters
// as the first and second parameters.
var displayArgs2 = displayArgs.bind(emptyObject, 12, "a");

// Call the new function. The "b" and "c" parameters are used
// as the third and fourth parameters.
displayArgs2("b", "c");
// Output: 12 a b c
```

Requirements

Supported in the following document modes: Internet Explorer 9 standards, Internet Explorer 10 standards, and Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7

standards, Internet Explorer 8 standards.

See Also

Function Object
filter Method (Array)
Using the bind method
Hilo JavaScript sample app (Windows Store)

call Method (Function) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Calls a method of an object, substituting another object for the current object.

Syntax

```
call([thisObj[, arg1[, arg2[,  [, argN]]]]])
```

Parameters

`thisObj`
Optional. The object to be used as the current object.

`arg1, arg2, , argN`
Optional. A list of arguments to be passed to the method.

Remarks

The `call` method is used to call a method on behalf of another object. It allows you to change the `this` object of a function from the original context to the new object specified by `thisObj`.

If `thisObj` is not supplied, the `global` object is used as `thisObj`.

Example

The following code shows how to use the `call` method.

```
function callMe(arg1, arg2){
    var s = "";

    s += "this value: " + this;
    s += "<br />";
    for (i in callMe.arguments) {
        s += "arguments: " + callMe.arguments[i];
        s += "<br />";
    }
    return s;
}

document.write("Original function: <br/>");
document.write(callMe(1, 2));
document.write("<br/>");

document.write("Function called with call: <br/>");
document.write(callMe.call(3, 4, 5));

// Output:
// Original function:
// this value: [object Window]
// arguments: 1
// arguments: 2

// Function called with call:
// this value: 3
// arguments: 4
// arguments: 5
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

See Also

Function Object
apply Method (Function)

Global Object (JavaScript)

10/18/2017 • 1 min to read • Edit Online

An intrinsic object whose purpose is to collect global functions and constants into one object.

Remarks

The `Global` object is never used directly, and cannot be created using the `new` operator. It is created when the scripting engine is initialized, thus making its functions and constants available immediately.

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

Constants

Infinity Constant | NaN Constant | undefined Constant

Functions

decodeURI Function | decodeURIComponent Function | encodeURI Function | encodeURIComponent Function | escape Function | eval Function | GetObject Function | isFinite Function | isNaN Function | parseFloat Function | parseInt Function | ScriptEngine Function | ScriptEngineBuildVersion Function | ScriptEngineMajorVersion Function | ScriptEngineMinorVersion Function | unescape Function

See Also

Object Object

Infinity Constant (JavaScript)

10/18/2017 • 1 min to read • Edit Online

A number that is larger than the largest floating point number. Returns an initial value of `Number.POSITIVE_INFINITY`. Negative Infinity (-Infinity) is smaller than the smallest floating point number.

Syntax

```
Infinity
```

Remarks

The `Infinity` constant is a member of the `Global` object, and is made available when the scripting engine is initialized.

Requirements

The `Infinity` property was introduced in Internet Explorer before Internet Explorer 6 and was made read-only in Internet Explorer 9 standards mode.

Applies To: Global Object

See Also

Number Constants

NaN Constant (JavaScript)

10/18/2017 • 1 min to read • Edit Online

A special constant that specifies thatNot an expression is not a number.

Syntax

```
NaN
```

Remarks

The `NaN` constant (not a number) is a member of the `Global` object, and is made available when the scripting engine is initialized.

Requirements

The `NaN` property was introduced in Internet Explorer before Internet Explorer 6, and was made read-only in Internet Explorer 9 standards mode.

Applies To: Global Object

See Also

isNaN Function
Number Constants

null Constant (JavaScript)

10/18/2017 • 1 min to read • Edit Online

`null` is used to indicate that a variable does not refer to valid data. This is not the same thing as undefined Constant.

undefined Constant (JavaScript)

10/18/2017 • 1 min to read • Edit Online

A value that has never been defined, such as a variable that has not been initialized.

Syntax

```
undefined
```

Remarks

The `undefined` constant is a member of the `Global` object, and becomes available when the scripting engine is initialized. When a variable has been declared but not initialized, its value is **undefined**.

If a variable has not been declared, you cannot compare it to `undefined`, but you can compare the type of the variable to the string "undefined".

The `undefined` constant is useful when explicitly testing or setting a variable to undefined.

Example

The following example shows how to use the `undefined` constant.

```
// A variable that has not been initialized.
var declared;

if (declared == undefined)
    document.write("declared has not been given a value <br/>");
else
    document.write("declared has been given a value <br/>");

document.write("typeof declared is " + typeof(declared) + "<br/>");

// An undeclared variable cannot be compared to undefined,
// so the next line would generate an error.
// if (notDeclared == undefined);

document.write("typeof notDeclared is " + typeof(notDeclared));

// Output:
// declared has not been given a value
// typeof declared is undefined
// typeof notDeclared is undefined
```

Requirements

The `undefined` property was introduced in Internet Explorer before Internet Explorer 6, and was made read-only in Internet Explorer 9 standards mode.

Applies To: Global Object

See Also

typeof Operator

decodeURI Function (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Gets the unencoded version of an encoded Uniform Resource Identifier (URI).

Syntax

```
decodeURI(URIstring)
```

Remarks

The required `URIstring` argument is a value representing an encoded URI.

Use the `decodeURI` function instead of the deprecated `unescape` function.

The `decodeURI` function returns a string value.

If the `URIString` is not valid, a URIError occurs.

Applies To: Global Object

Example

The following code first encodes a URI component and then decodes it.

```
var uriEncode = encodeURIComponent ("www.Not a URL.com");
var uriDecode = decodeURIComponent(uriEncode);

document.write (uriEncode);
document.write ("<br/>");
document.write (uriDecode);

// Output:
// www.Not%20a%20URL.com
// www.Not a URL.com
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

See Also

decodeURIComponent Function
encodeURI Function
Global Object

decodeURIComponent Function (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Gets the unencoded version of an encoded component of a Uniform Resource Identifier (URI).

Syntax

```
decodeURIComponent(encodedURIString)
```

Remarks

The required `encodedURIString` argument is a value representing an encoded URI component.

A URIComponent is part of a complete URI.

If the `encodedURIString` is not valid, a URIError occurs.

Example

The following code first encodes and then decodes a URI.

```
var uriEncode = encodeURI ("http://www.Not a URL.com");
var uriDecode = decodeURIComponent(uriEncode);

document.write (uriEncode);
document.write("<br/>");
document.write (uriDecode);

// Output:
// http://www.Not%20a%20URL.com
// http://www.Not a URL.com
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

See Also

decodeURI Function
encodeURI Function

encodeURI Function (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Encodes a text string as a valid Uniform Resource Identifier (URI)

Syntax

```
encodeURI(
URIString
)
```

Remarks

The required `URIString` argument is a value representing an encoded URI.

The `encodeURI` function returns an encoded URI. If you pass the result to `decodeURI`, the original string is returned. The `encodeURI` function does not encode the following characters: ":", "/", ";", and "?". Use `encodeURIComponent` to encode these characters.

Example

The following code first encodes and then decodes a URI.

```
var uriEncode = encodeURI ("http://www.Not a URL.com");
var uriDecode = decodeURIComponent(uriEncode);

document.write(uriEncode);
document.write("<br/>");
document.write(uriDecode);

// Output:
// http://www.Not%20a%20URL.com
// http://www.Not a URL.com
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

See Also

decodeURI Function
decodeURIComponent Function

encodeURIComponent Function (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Encodes a text string as a valid component of a Uniform Resource Identifier (URI).

Syntax

```
encodeURIComponent(encodedURIString)
```

Remarks

The required `encodedURIString` argument is a value representing an encoded URI component.

The `encodeURIComponent` function returns an encoded URI. If you pass the result to `decodeURIComponent`, the original string is returned. Because the `encodeURIComponent` function encodes all characters, be careful if the string represents a path such as **/folder1/folder2/default.html**. The slash characters will be encoded and will not be valid if sent as a request to a web server. Use the `encodeURI` function if the string contains more than a single URI component.

Example

The following code first encodes a URI component and then decodes it.

```
var uriEncode = encodeURIComponent ("www.Not a URL.com");
var uriDecode = decodeURIComponent(uriEncode);

document.write(uriEncode);
document.write("<br/>");
document.write(uriDecode);

// Output:
// www.Not%20a%20URL.com
// www.Not a URL.com
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

See Also

decodeURI Function
decodeURIComponent Function

escape Function (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Encodes strings so they can be read on all computers. Deprecated.

Syntax

```
escape(charString)
```

Remarks

The required `charString` argument is any `String` object or literal to be encoded.

The **escape** function returns a string value (in Unicode format) that contains the contents of `charstring`. All spaces, punctuation, accented characters, and any other non-ASCII characters are replaced with `%`*xx* encoding, where *xx* is equivalent to the hexadecimal number representing the character. For example, a space is returned as "%20."

Characters with a value greater than 255 are stored using the **%u** *xxxx* format.

> **NOTE**
>
> The **escape** function should not be used to encode Uniform Resource Identifiers (URI). Use `encodeURI` and `encodeURIComponent` functions instead.

Applies To: Global Object

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

See Also

encodeURI Function
encodeURIComponent Function
String Object
unescape Function

eval Function (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Evaluates JavaScript code and executes it.

Syntax

```
eval(codeString)
```

Parameters

`codeString`
Required. A `String` value that contains valid JavaScript code.

Remarks

The `eval` function enables dynamic execution of JavaScript source code.

The `codeString` string is parsed by the JavaScript parser and executed.

The code passed to the `eval` function is executed in the same context as the call to the `eval` function.

Whenever possible, use the JSON.parse function to de-serialize JavaScript Object Notation (JSON) text. The `JSON.parse` function is more secure and executes faster than the `eval` function.

Example

The following code initializes the variable `myDate` to a test date.

```
var dateFn = "Date(1971,3,8)";
var myDate;
eval("myDate = new " + dateFn + ";");

document.write(myDate);

// Output: Thu Apr 8 00:00:00 PDT 1971
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

Applies To: Global Object

See Also

String Object

GetObject Function (JavaScript)

10/18/2017 • 2 min to read • Edit Online

Returns a reference to an Automation object from a file.

> **NOTE**
> This function is not supported in Internet Explorer 9 (standards mode) or later.

Syntax

```
GetObject([pathname] [, class])
```

Parameters

`pathname`
Optional. Full path and name of the file containing the object to retrieve. If `pathname` is omitted, `class` is required.

`class`
Optional. Class of the object.

The `class` argument uses the syntax `appname.objectype` and has these parts:

`appname`
Required. Name of the application providing the object.

`objectype`
Required. Type or class of object to create.

Remarks

The `GetObject` function is not supported in Internet Explorer 9 standards mode, Internet Explorer 10 standards mode, Internet Explorer 11 standards mode, and Windows Store apps or later.

Use the `GetObject` function to access an Automation object from a file. Assign the object returned by `GetObject` to the object variable. For example:

```
var CADObject;
CADObject = GetObject("C:\\CAD\\SCHEMA.CAD");
```

When this code is executed, the application associated with the specified `pathname` is started, and the object in the specified file is activated. If `pathname` is a zero-length string (""), `GetObject` returns a new object instance of the specified type. If the `pathname` argument is omitted, `GetObject` returns a currently active object of the specified type. If no object of the specified type exists, an error occurs.

Some applications allow you to activate part of a file. To do so, add an exclamation point (!) to the end of the file name and follow it with a string that identifies the part of the file you want to activate. For information on how to create this string, see the documentation for the application that created the object.

For example, in a drawing application you might have multiple layers to a drawing stored in a file. You could use

the following code to activate a layer within a drawing called `SCHEMA.CAD`:

```
var LayerObject = GetObject("C:\\CAD\\SCHEMA.CAD!Layer3");
```

If you do not specify the object's class, Automation determines which application to start and which object to activate, based on the file name you provide. Some files, however, may support more than one class of object. For example, a drawing might support three different types of objects: an Application object, a Drawing object, and a Toolbar object, all of which are part of the same file. To specify which object in a file you want to activate, use the optional `class` argument. For example:

```
var MyObject;
MyObject = GetObject("C:\\DRAWINGS\\SAMPLE.DRW", "FIGMENT.DRAWING");
```

In the preceding example, `FIGMENT` is the name of a drawing application and `DRAWING` is one of the object types it supports. Once an object is activated, you reference it in code using the object variable you defined. In the preceding example, you access properties and methods of the new object using the object variable `MyObject`. For example:

```
MyObject.Line(9, 90);
MyObject.InsertText(9, 100, "Hello, world.");
MyObject.SaveAs("C:\\DRAWINGS\\SAMPLE.DRW");
```

NOTE

Use the `GetObject` function when there is a current instance of the object, or if you want to create the object with a file already loaded. If there is no current instance, and you don't want the object started with a file loaded, use the `ActiveXObject` object.

If an object has registered itself as a single-instance object, only one instance of the object is created, no matter how many times `ActiveXObject` is executed. With a single-instance object, `GetObject` always returns the same instance when called with the zero-length string ("") syntax, and it causes an error if the `pathname` argument is omitted.

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, and Internet Explorer 8 standards. See Version Information.

See Also

ActiveXObject Object

isFinite Function (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Determines whether a supplied number is finite.

Syntax

```
isFinite(number)
```

Remarks

The required `number` argument is any numeric value.

The `isFinite` function returns `true` if `number` is any value other than `NaN`, negative infinity, or positive infinity. In those three cases, it returns **false**.

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

Applies To: Global Object

See Also

isNaN Function

isNaN Function (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Returns a Boolean value that indicates whether a value is the reserved value `NaN` (not a number).

Syntax

```
isNaN(numValue)
```

Return Value

`true` if the value converted to the `Number` type is the `NaN`, otherwise `false`.

Remarks

The required `numValue` is the value to be tested against `NaN`.

You typically use this method to test return values from the `parseInt` and `parseFloat` methods.

Alternatively, a variable that contains `NaN` or another value could be compared to itself. If it compares as unequal, it is `NaN`. This is because `NaN` is the only value that is not equal to itself.

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

Applies To: Global Object

Example

```
// Returns false.
isNaN(100);

// Returns false.
isNaN("100");

// Returns true.
isNaN("ABC");

// Returns true.
isNaN("10C");

// Returns true.
isNaN("abc123");

// Returns true.
isNaN(Math.sqrt(-1));
```

See Also

isFinite Function
NaN Constant
parseFloat Function
parseInt Function

parseFloat Function (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Converts a string to a floating-point number.

Syntax

```
parseFloat(numString)
```

Remarks

The required `numString` argument is a string that contains a floating-point number.

The `parseFloat` function returns a numerical value equal to the number contained in `numString`. If no prefix of `numString` can be successfully parsed into a floating-point number, `NaN` (not a number) is returned.

```
parseFloat("abc")      // Returns NaN.
parseFloat("1.2abc")   // Returns 1.2.
```

You can test for `NaN` using the `isNaN` function.

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

Applies To: Global Object

See Also

isNaN Function
parseInt Function
String Object

parseInt Function (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Converts a string to an integer.

Syntax

```
parseInt(numString, [radix])
```

Parameters

`numString`
Required. A string to convert into a number.

`radix`
Optional. A value between 2 and 36 that specifies the base of the number in `numString`. If this argument is not supplied, strings with a prefix of '0x' are considered hexadecimal. All other strings are considered decimal.

Remarks

The `parseInt` function returns an integer value equal to the number contained in `numString`. If no prefix of `numString` can be successfully parsed into an integer, `NaN` (not a number) is returned.

```
parseInt("abc");     // Returns NaN.
parseInt("12abc");   // Returns 12.
```

You can test for `NaN` using the `isNaN` function.

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

Applies To: Global Object

> **NOTE**
> Starting in Internet Explorer 9 standards mode, the `parseInt` function does not treat a string that has a prefix of '0' as an octal. When you are not using the `parseInt` function, however, strings with a prefix of '0' can still be interpreted as octals. See Data Types for information about octal integers.

See Also

isNaN Function
parseFloat Function
String Object
valueOf Method (Object)

ScriptEngine Function (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Gets the name of the scripting language in use.

Syntax

```
ScriptEngine()
```

Remarks

The `ScriptEngine` function returns "JScript", which indicates that JavaScript is the current scripting engine.

Example

The following example illustrates the use of the `ScriptEngine` function:

```
if (window.ScriptEngine) {
    console.log(window.ScriptEngine());
}

// Output: JScript
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

See Also

ScriptEngineBuildVersion Function
ScriptEngineMajorVersion Function
ScriptEngineMinorVersion Function

ScriptEngineBuildVersion Function (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Gets the build version number of the scripting engine in use.

Syntax

```
ScriptEngineBuildVersion()
```

Remarks

The return value corresponds directly to the version information contained in the dynamic-link library (DLL) for the scripting language in use.

Example

The following example illustrates the use of the `ScriptEngineBuildVersion` function:

```
if(window.ScriptEngineBuildVersion) {
    console.log(window.ScriptEngineBuildVersion());
}

// Output: <current build version>
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

See Also

ScriptEngine Function
ScriptEngineMajorVersion Function
ScriptEngineMinorVersion Function

ScriptEngineMajorVersion Function (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Gets the major version number of the scripting engine in use.

Syntax

```
ScriptEngineMajorVersion()
```

Remarks

The return value corresponds directly to the version information contained in the dynamic-link library (DLL) for the scripting language in use.

Example

The following example illustrates the use of the `ScriptEngineMajorVersion` function:

```
if (window.ScriptEngineMajorVersion) {
    console.log(window.ScriptEngine());
}

Output: <current major version>
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

See Also

ScriptEngine Function
ScriptEngineBuildVersion Function
ScriptEngineMinorVersion Function

ScriptEngineMinorVersion Function (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Gets the minor version number of the scripting engine in use.

Syntax

```
ScriptEngineMinorVersion()
```

Remarks

The return value corresponds directly to the version information contained in the dynamic-link library (DLL) for the scripting language in use.

Example

The following example illustrates the use of the `ScriptEngineMinorVersion` function.

```
if (window.ScriptEngineMinorVersion) {
    console.log(window.ScriptEngineMinorVersion());
}

//Output: <current minor version>
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

See Also

ScriptEngine Function
ScriptEngineBuildVersion Function
ScriptEngineMajorVersion Function

unescape Function (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Decodes `String` objects encoded with the `escape` function. Deprecated.

Syntax

```
unescape(charString)
```

Remarks

The required `charString` argument is a `String` object or literal to be decoded.

The `unescape` function returns a string value that contains the contents of `charstring`. All characters encoded with the %*xx* hexadecimal form are replaced by their ASCII character set equivalents.

Characters encoded in **%u** *xxxx* format (Unicode characters) are replaced with the Unicode character with hexadecimal encoding *xxxx*.

> **NOTE**
> The `unescape` function should not be used to decode Uniform Resource Identifiers (URI). Use `decodeURI` and `decodeURIComponent` functions instead.

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

Applies To: Global Object

See Also

decodeURI Function
decodeURIComponent Function
escape Function
String Object

Int8Array Object

10/18/2017 • 1 min to read • Edit Online

A typed array of 8-bit integer values. The contents are initialized to 0. If the requested number of bytes could not be allocated an exception is raised.

Syntax

```
      int8Array = new Int8Array( length );
intArray = new Int8Array( array );
intArray = new Int8Array( buffer, byteOffset, length);
```

Parameters

`int8Array`
Required. The variable name to which the `Int8Array` object is assigned.

`length`
Specifies the number of elements in the array.

`array`
The array (or typed array) that is contained in this array.. The contents are initialized to the contents of the given array or typed array, with each element converted to the Int8 type.

`buffer`
The ArrayBuffer that the Int8Array represents.

`byteOffset`
Optional. Specifies the offset in bytes from the start of the buffer at which the Int8Array should begin.

`length`
The number of elements in the array.

Constants

The following table lists the constants of the `Int8Array` object.

CONSTANT	DESCRIPTION
BYTES_PER_ELEMENT Constant	The size in bytes of each element in the array.

Properties

The following table lists the constants of the `Int8Array` object.

PROPERTY	DESCRIPTION
buffer Property	Read-only. Gets the ArrayBuffer that is referenced by this array.

PROPERTY	DESCRIPTION
byteLength Property	Read-only. The length of this array from the start of its ArrayBuffer, in bytes, as fixed at construction time.
byteOffset Property	Read-only. The offset of this array from the start of its ArrayBuffer, in bytes, as fixed at construction time.
length Property	The length of the array.

Methods

The following table lists the methods of the `Int8Array` object.

METHOD	DESCRIPTION
set Method (Int8Array)	Sets a value or an array of values.
subarray Method (Int8Array)	Gets a new Int8Array view of the ArrayBuffer store for this array.

Example

The following example shows how to use an Int8Array object to process the binary data acquired from an XmlHttpRequest:

```
var req = new XMLHttpRequest();
    req.open('GET', "http://www.example.com");
    req.responseType = "arraybuffer";
    req.send();

    req.onreadystatechange = function () {
        if (req.readyState === 4) {
            var buffer = req.response;
            var dataview = new DataView(buffer);
            var ints = new Int8Array(buffer.byteLength);
            for (var i = 0; i < ints.length; i++) {
                ints[i] = dataview.getInt8(i);
            }
        alert(ints[10]);
        }
    }
```

Requirements

Supported in the following document modes: Internet Explorer 10 standards and Internet Explorer 11 standards.
Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards.

BYTES_PER_ELEMENT Constant (Int8Array)

10/18/2017 • 1 min to read • Edit Online

The size in bytes of each element in the array.

Syntax

```
var arraySize = int8Array.BYTES_PER_ELEMENT;
```

Example

The following example shows how to get the size of the array elements.

```
var req = new XMLHttpRequest();
    req.open('GET', "http://www.example.com");
    req.responseType = "arraybuffer";
    req.send();

    req.onreadystatechange = function () {
        if (req.readyState === 4) {
            var buffer = req.response;
            var dataView = new DataView(buffer);
            var intArr = new Int8Array(buffer.byteLength);
            alert(intArr.BYTES_PER_ELEMENT);
        }
    }
```

Requirements

Supported in the following document modes: Internet Explorer 10 standards and Internet Explorer 11 standards.
Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards.

buffer Property (Int8Array)

10/18/2017 • 1 min to read • Edit Online

Read-only. Gets the ArrayBuffer that is referenced by this array.

Syntax

```
var arrayBuffer = int8Array.buffer;
```

Example

The following example shows how to get the ArrayBuffer of the array.

```
var req = new XMLHttpRequest();
    req.open('GET', "http://www.example.com");
    req.responseType = "arraybuffer";
    req.send();

    req.onreadystatechange = function () {
        if (req.readyState === 4) {
            var buffer = req.response;
            var dataView = new DataView(buffer);
            var intArr = new Int8Array(buffer.byteLength);
            alert(intArr.buffer.byteLength);
        }
    }
```

Requirements

Supported in the following document modes: Internet Explorer 10 standards and Internet Explorer 11 standards.
Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards.

byteLength Property (Int8Array)

10/18/2017 • 1 min to read • Edit Online

Read-only. The length of this array from the start of its ArrayBuffer, in bytes, as fixed at construction time.

Syntax

```
var arrayByteLength = int8Array.byteLength;
```

Example

The following example shows how to get the length of the array.

```
var req = new XMLHttpRequest();
    req.open('GET', "http://www.example.com");
    req.responseType = "arraybuffer";
    req.send();

    req.onreadystatechange = function () {
        if (req.readyState === 4) {
            var buffer = req.response;
            var dataView = new DataView(buffer);
            var intArr = new Int8Array(buffer.byteLength);
            alert(intArr.byteLength);
        }
    }
```

Requirements

Supported in the following document modes: Internet Explorer 10 standards and Internet Explorer 11 standards.
Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards.

byteOffset Property (Int8Array)

10/18/2017 • 1 min to read • Edit Online

Read-only. The offset of this array from the start of its ArrayBuffer, in bytes, as fixed at construction time.

Syntax

```
var arrayOffset = int8Array.byteOffset;
```

Example

The following example shows how to get the offset of the array.

```
var req = new XMLHttpRequest();
    req.open('GET', "http://www.example.com");
    req.responseType = "arraybuffer";
    req.send();

    req.onreadystatechange = function () {
        if (req.readyState === 4) {
            var buffer = req.response;
            var dataView = new DataView(buffer);
            var intArr = new Int8Array(buffer.byteLength);
            alert(intArr.byteOffset);
        }
    }
```

Requirements

Supported in the following document modes: Internet Explorer 10 standards and Internet Explorer 11 standards.
Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards.

length Property (Int8Array)

10/18/2017 • 1 min to read • Edit Online

The length of the array.

Syntax

```
var arrayLength = int8Array.length;
```

Example

The following example shows how to get the length of the array.

```
var req = new XMLHttpRequest();
    req.open('GET', "http://www.example.com");
    req.responseType = "arraybuffer";
    req.send();

    req.onreadystatechange = function () {
        if (req.readyState === 4) {
            var buffer = req.response;
            var dataView = new DataView(buffer);
            var intArr = new Int8Array(buffer.byteLength);
            alert(intArr.length);
        }
    }
```

Requirements

Supported in the following document modes: Internet Explorer 10 standards and Internet Explorer 11 standards.
Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards.

set Method (Int8Array)

10/18/2017 • 1 min to read • Edit Online

Sets a value or an array of values.

Syntax

```
int8Array.set(index, value);
int8Array.set(array, offset);
```

Parameters

`index`
The index of the location to set.

`value`
The value to set.

`array`
A typed or untyped array of values to set.

`offset`
The index in the current array at which the values are to be written.

Remarks

If the input array is a TypedArray, the two arrays may use the same underlying ArrayBuffer. In this situation, setting the values takes place as if all the data is first copied into a temporary buffer that does not overlap either of the arrays, and then the data from the temporary buffer is copied into the current array.

If the offset plus the length of the given array is out of range for the current TypedArray, an exception is raised.

Example

The following example shows how to set the first element of the array.

```
var req = new XMLHttpRequest();
    req.open('GET', "http://www.example.com");
    req.responseType = "arraybuffer";
    req.send();

    req.onreadystatechange = function () {
        if (req.readyState === 4) {
            var buffer = req.response;
            var dataView = new DataView(buffer);
            var intArr = new Int8Array(buffer.byteLength);
            intArr.set(0, 9);
        }
    }
```

Requirements

Supported in the following document modes: Internet Explorer 10 standards and Internet Explorer 11 standards.
Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards.

subarray Method (Int8Array)

10/18/2017 • 1 min to read • Edit Online

Gets a new Int8Array view of the ArrayBuffer store for this array, referencing the elements at begin, inclusive, up to end, exclusive.

Syntax

```
var newInt8Array = int8Array.subset(begin, end);
```

Parameters

`newInt8Array`
The subarray returned by this method.

`begin`
The index of the beginning of the array.

`end`
The index of the end of the array. This is non-inclusive.

Remarks

If either begin or end is negative, it refers to an index from the end of the array, as opposed to from the beginning. If end is unspecified, the subarray contains all elements from begin to the end of the TypedArray. The range specified by the begin and end values is clamped to the valid index range for the current array. If the computed length of the new TypedArray would be negative, it is clamped to zero. The returned TypedArray will be of the same type as the array on which this method is invoked.

Example

The following example shows how to get a subarray two elements long, starting with the first element of the array.

```
var req = new XMLHttpRequest();
    req.open('GET', "http://www.example.com");
    req.responseType = "arraybuffer";
    req.send();

    req.onreadystatechange = function () {
        if (req.readyState === 4) {
            var buffer = req.response;
            var dataView = new DataView(buffer);
            var intArr = new Int8Array(buffer.byteLength);
            var subArr = intArr.subarray(0, 2);
        }
    }
```

Requirements

Supported in the following document modes: Internet Explorer 10 standards and Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards.

Int16Array Object

10/18/2017 • 2 min to read • Edit Online

A typed array of 16-bit integer values. The contents are initialized to 0. If the requested number of bytes could not be allocated an exception is raised.

Syntax

```
      int16Array = new Int16Array( length );
int16Array = new Int16Array( array );
int16Array = new Int16Array( buffer, byteOffset, length);
```

Parameters

`int16Array`
Required. The variable name to which the **Int16Array** object is assigned.

`length`
Specifies the number of elements in the array.

`array`
The array (or typed array) that is contained in this array. The contents are initialized to the contents of the given array or typed array, with each element converted to the Int16 type.

`buffer`
The ArrayBuffer that the Int16Array represents.

`byteOffset`
Optional. Specifies the offset in bytes from the start of the buffer at which the Int16Array should begin.

`length`
The number of elements in the array.

Constants

The following table lists the constants of the `Int16Array` object.

CONSTANT	DESCRIPTION
BYTES_PER_ELEMENT Constant	The size in bytes of each element in the array.

Properties

The following table lists the constants of the `Int16Array` object.

PROPERTY	DESCRIPTION
buffer Property	Read-only. Gets the ArrayBuffer that is referenced by this array.

PROPERTY	DESCRIPTION
byteLength Property	Read-only. The length of this array from the start of its ArrayBuffer, in bytes, as fixed at construction time.
byteOffset Property	Read-only. The offset of this array from the start of its ArrayBuffer, in bytes, as fixed at construction time.
length Property	The length of the array.

Methods

The following table lists the methods of the `Int16Array` object.

METHOD	DESCRIPTION
set Method (Int16Array)	Sets a value or an array of values.
subarray Method (Int16Array)	Gets a new Int16Array view of the ArrayBuffer store for this array.

Example

The following example shows how to use an Int16Array object to process the binary data acquired from an XmlHttpRequest:

```
var req = new XMLHttpRequest();
    req.open('GET', "http://www.example.com");
    req.responseType = "arraybuffer";
    req.send();

    req.onreadystatechange = function () {
        if (req.readyState === 4) {
            var buffer = req.response;
            var dataview = new DataView(buffer);
            var ints = new Int16Array(buffer.byteLength / 2);
            for (var i = 0; i < ints.length; i++) {
                ints[i] = dataview.getInt16(i * 2);
            }
        alert(ints[10]);
        }
    }
```

Requirements

Supported in the following document modes: Internet Explorer 10 standards and Internet Explorer 11 standards.
Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards.

BYTES_PER_ELEMENT Constant (Int16Array)

10/18/2017 • 1 min to read • Edit Online

The size in bytes of each element in the array.

Syntax

```
var arraySize = int8Array.BYTES_PER_ELEMENT;
```

Example

The following example shows how to get the size of the array elements.

```
var req = new XMLHttpRequest();
    req.open('GET', "http://www.example.com");
    req.responseType = "arraybuffer";
    req.send();

    req.onreadystatechange = function () {
        if (req.readyState === 4) {
            var buffer = req.response;
            var dataView = new DataView(buffer);
            var intArr = new Int16Array(buffer.byteLength / 2);
            alert(intArr.BYTES_PER_ELEMENT);
        }
    }
```

Requirements

Supported in the following document modes: Internet Explorer 10 standards and Internet Explorer 11 standards.
Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards.

buffer Property (Int16Array)

10/18/2017 • 1 min to read • Edit Online

Read-only. Gets the ArrayBuffer that is referenced by this array.

Syntax

```
var arrayBuffer = int16Array.buffer;
```

Example

The following example shows how to get the ArrayBuffer of the array.

```
var req = new XMLHttpRequest();
    req.open('GET', "http://www.example.com");
    req.responseType = "arraybuffer";
    req.send();

    req.onreadystatechange = function () {
        if (req.readyState === 4) {
            var buffer = req.response;
            var dataView = new DataView(buffer);
            var intArr = new Int16Array(buffer.byteLength / 2);
            alert(intArr.buffer.byteLength);
        }
    }
```

Requirements

Supported in the following document modes: Internet Explorer 10 standards and Internet Explorer 11 standards.
Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards.

byteLength Property (Int16Array)

10/18/2017 • 1 min to read • Edit Online

Read-only. The length of this array from the start of its ArrayBuffer, in bytes, as fixed at construction time.

Syntax

```
var arrayByteLength = int16Array.byteLength;
```

Example

The following example shows how to get the length of the array.

```
var req = new XMLHttpRequest();
    req.open('GET', "http://www.example.com");
    req.responseType = "arraybuffer";
    req.send();

    req.onreadystatechange = function () {
        if (req.readyState === 4) {
            var buffer = req.response;
            var dataView = new DataView(buffer);
            var intArr = new Int16Array(buffer.byteLength / 2);
            alert(intArr.byteLength);
        }
    }
```

Requirements

Supported in the following document modes: Internet Explorer 10 standards and Internet Explorer 11 standards.
Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards.

byteOffset Property (Int16Array)

10/18/2017 • 1 min to read • Edit Online

Read-only. The offset of this array from the start of its ArrayBuffer, in bytes, as fixed at construction time.

Syntax

```
var arrayOffset = int16Array.byteOffset;
```

Example

The following example shows how to get the offset of the array.

```
var req = new XMLHttpRequest();
    req.open('GET', "http://www.example.com");
    req.responseType = "arraybuffer";
    req.send();

    req.onreadystatechange = function () {
        if (req.readyState === 4) {
            var buffer = req.response;
            var dataView = new DataView(buffer);
            var intArr = new Int16Array(buffer.byteLength / 2);
            alert(intArr.byteOffset);
        }
    }
```

Requirements

Supported in the following document modes: Internet Explorer 10 standards and Internet Explorer 11 standards.
Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards.

length Property (Int16Array)

10/18/2017 • 1 min to read • Edit Online

The length of the array.

Syntax

```
var arrayLength = int16Array.length;
```

Example

The following example shows how to get the length of the array.

```
var req = new XMLHttpRequest();
    req.open('GET', "http://www.example.com");
    req.responseType = "arraybuffer";
    req.send();

    req.onreadystatechange = function () {
        if (req.readyState === 4) {
            var buffer = req.response;
            var dataView = new DataView(buffer);
            var intArr = new Int16Array(buffer.byteLength / 2);
            alert(intArr.length);
        }
    }
```

Requirements

Supported in the following document modes: Internet Explorer 10 standards and Internet Explorer 11 standards.
Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards.

set Method (Int16Array)

10/18/2017 • 1 min to read • Edit Online

Sets a value or an array of values.

Syntax

```
int16Array.set(index, value);
int16Array.set(array, offset);
```

Parameters

`index`
The index of the location to set.

`value`
The value to set.

`array`
A typed or untyped array of values to set.

`offset`
The index in the current array at which the values are to be written.

Remarks

If the input array is a TypedArray, the two arrays may use the same underlying ArrayBuffer. In this situation, setting the values takes place as if all the data is first copied into a temporary buffer that does not overlap either of the arrays, and then the data from the temporary buffer is copied into the current array.

If the offset plus the length of the given array is out of range for the current TypedArray, an exception is raised.

Example

The following example shows how to set the first element of the array.

```
var req = new XMLHttpRequest();
    req.open('GET', "http://www.example.com");
    req.responseType = "arraybuffer";
    req.send();

    req.onreadystatechange = function () {
        if (req.readyState === 4) {
            var buffer = req.response;
            var dataView = new DataView(buffer);
            var intArr = new Int16Array(buffer.byteLength / 2);
            intArr.set(0, 9);
        }
    }
```

Requirements

Supported in the following document modes: Internet Explorer 10 standards and Internet Explorer 11 standards.
Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards.

subarray Method (Int16Array)

10/18/2017 • 1 min to read • Edit Online

Gets a new Int16Array view of the ArrayBuffer Object store for this array, specifying the first and last members of the subarray.

Syntax

```
var newInt16Array = int16Array.subarray(begin, end);
```

Parameters

`newInt16Array`
The subarray returned by this method.

`begin`
The index of the beginning of the array.

`end`
The index of the end of the array. This is non-inclusive.

Remarks

If either `begin` or `end` is negative, it refers to an index from the end of the array, as opposed to from the beginning. If `end` is unspecified, the subarray contains all elements from begin to the end of the typed array. The range specified by the `begin` and `end` values is clamped to the valid index range for the current array. If the computed length of the new typed array would be negative, it is clamped to zero. The returned array is of the same type as the array on which this method is invoked.

Example

The following example shows how to get a subarray two elements long, starting with the first element of the array.

```
var req = new XMLHttpRequest();
    req.open('GET', "http://www.example.com");
    req.responseType = "arraybuffer";
    req.send();

    req.onreadystatechange = function () {
        if (req.readyState === 4) {
            var buffer = req.response;
            var intArr = new Int16Array(buffer.byteLength / 2);
            var subArr = intArr.subarray(0, 2);
        }
    }
```

Requirements

Supported in the following document modes: Internet Explorer 10 standards and Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards.

Int32Array Object

10/18/2017 • 2 min to read • Edit Online

A typed array of 32-bit integer values. The contents are initialized to 0. If the requested number of bytes could not be allocated an exception is raised.

Syntax

```
      int32Array = new Int32Array( length );
int32Array = new Int32Array( array );
int32Array = new Int32Array( buffer, byteOffset, length);
```

Parameters

`int32Array`
Required. The variable name to which the **Int32Array** object is assigned.

`length`
Specifies the number of elements in the array.

`array`
The array (or typed array) that is contained in this array.. The contents are initialized to the contents of the given array or typed array, with each element converted to the Int32 type.

`buffer`
The ArrayBuffer that the Int32Array represents.

`byteOffset`
Optional. Specifies the offset in bytes from the start of the buffer at which the Int32Array should begin.

`length`
The number of elements in the array.

Constants

The following table lists the constants of the `Int32Array` object.

CONSTANT	DESCRIPTION
BYTES_PER_ELEMENT Constant	The size in bytes of each element in the array.

Properties

The following table lists the constants of the `Int32Array` object.

PROPERTY	DESCRIPTION
buffer Property	Read-only. Gets the ArrayBuffer that is referenced by this array.

PROPERTY	DESCRIPTION
byteLength Property	Read-only. The length of this array from the start of its ArrayBuffer, in bytes, as fixed at construction time.
byteOffset Property	Read-only. The offset of this array from the start of its ArrayBuffer, in bytes, as fixed at construction time.
length Property	The length of the array.

Methods

The following table lists the methods of the `Int32Array` object.

METHOD	DESCRIPTION
set Method (Int32Array)	Sets a value or an array of values.
subarray Method (Int32Array)	Gets a new Int32Array view of the ArrayBuffer store for this array.

Example

The following example shows how to use an Int32Array object to process the binary data acquired from an XmlHttpRequest:

```
var req = new XMLHttpRequest();
    req.open('GET', "http://www.example.com");
    req.responseType = "arraybuffer";
    req.send();

    req.onreadystatechange = function () {
        if (req.readyState === 4) {
            var buffer = req.response;
            var dataview = new DataView(buffer);
            var ints = new Int32Array(buffer.byteLength / 4);
            for (var i = 0; i < ints.length; i++) {
                ints[i] = dataview.getInt32(i * 4);
            }
        alert(ints[10]);
        }
    }
```

Requirements

Supported in the following document modes: Internet Explorer 10 standards and Internet Explorer 11 standards.
Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards.

BYTES_PER_ELEMENT Constant (Int32Array)

10/18/2017 • 1 min to read • Edit Online

The size in bytes of each element in the array.

Syntax

```
var arraySize = int32Array.BYTES_PER_ELEMENT;
```

Example

The following example shows how to get the size of the array elements.

```
var req = new XMLHttpRequest();
    req.open('GET', "http://www.example.com");
    req.responseType = "arraybuffer";
    req.send();

    req.onreadystatechange = function () {
        if (req.readyState === 4) {
            var buffer = req.response;
            var dataView = new DataView(buffer);
            var intArr = new Int32Array(buffer.byteLength / 4);
            alert(intArr.BYTES_PER_ELEMENT);
        }
    }
```

Requirements

Supported in the following document modes: Internet Explorer 10 standards and Internet Explorer 11 standards.
Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards.

buffer Property (Int32Array)

10/18/2017 • 1 min to read • Edit Online

Read-only. Gets the ArrayBuffer that is referenced by this array.

Syntax

```
var arrayBuffer = int32Array.buffer;
```

Example

The following example shows how to get the ArrayBuffer of the array.

```
var req = new XMLHttpRequest();
    req.open('GET', "http://www.example.com");
    req.responseType = "arraybuffer";
    req.send();

    req.onreadystatechange = function () {
        if (req.readyState === 4) {
            var buffer = req.response;
            var dataView = new DataView(buffer);
            var intArr = new Int32Array(buffer.byteLength / 4);
            alert(intArr.buffer.byteLength);
        }
    }
```

Requirements

Supported in the following document modes: Internet Explorer 10 standards and Internet Explorer 11 standards.
Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards.

byteLength Property (Int32Array)

10/18/2017 • 1 min to read • Edit Online

Read-only. The length of this array from the start of its ArrayBuffer, in bytes, as fixed at construction time.

Syntax

```
var arrayByteLength = int32Array.byteLength;
```

Example

The following example shows how to get the length of the array.

```
var req = new XMLHttpRequest();
    req.open('GET', "http://www.example.com");
    req.responseType = "arraybuffer";
    req.send();

    req.onreadystatechange = function () {
        if (req.readyState === 4) {
            var buffer = req.response;
            var dataView = new DataView(buffer);
            var intArr = new Int32Array(buffer.byteLength / 4);
            alert(intArr.byteLength);
        }
    }
```

Requirements

Supported in the following document modes: Internet Explorer 10 standards and Internet Explorer 11 standards.
Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards.

byteOffset Property (Int32Array)

10/18/2017 • 1 min to read • Edit Online

Read-only. The offset of this array from the start of its ArrayBuffer, in bytes, as fixed at construction time.

Syntax

```
var arrayOffset = int32Array.byteOffset;
```

Example

The following example shows how to get the offset of the array.

```
var req = new XMLHttpRequest();
    req.open('GET', "http://www.example.com");
    req.responseType = "arraybuffer";
    req.send();

    req.onreadystatechange = function () {
        if (req.readyState === 4) {
            var buffer = req.response;
            var dataView = new DataView(buffer);
            var intArr = new Int32Array(buffer.byteLength / 4);
            alert(intArr.byteOffset);
        }
    }
```

Requirements

Supported in the following document modes: Internet Explorer 10 standards and Internet Explorer 11 standards.
Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards.

length Property (Int32Array)

10/18/2017 • 1 min to read • Edit Online

The length of the array.

Syntax

```
var arrayLength = int32Array.length;
```

Example

The following example shows how to get the length of the array.

```
var req = new XMLHttpRequest();
    req.open('GET', "http://www.example.com");
    req.responseType = "arraybuffer";
    req.send();

    req.onreadystatechange = function () {
        if (req.readyState === 4) {
            var buffer = req.response;
            var dataView = new DataView(buffer);
            var intArr = new Int32Array(buffer.byteLength / 4);
            alert(intArr.length);
        }
    }
```

Requirements

Supported in the following document modes: Internet Explorer 10 standards and Internet Explorer 11 standards.
Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards.

set Method (Int32Array)

10/18/2017 • 1 min to read • Edit Online

Sets a value or an array of values.

Syntax

```
Int32Array.set(index, value);
int32Array.set(array, offset);
```

Parameters

`index`
The index of the location to set.

`value`
The value to set.

`array`
A typed or untyped array of values to set.

`offset`
The index in the current array at which the values are to be written.

Remarks

If the input array is a TypedArray, the two arrays may use the same underlying ArrayBuffer. In this situation, setting the values takes place as if all the data is first copied into a temporary buffer that does not overlap either of the arrays, and then the data from the temporary buffer is copied into the current array.

If the offset plus the length of the given array is out of range for the current TypedArray, an exception is raised.

Example

The following example shows how to set the first element of the array.

```
var req = new XMLHttpRequest();
    req.open('GET', "http://www.example.com");
    req.responseType = "arraybuffer";
    req.send();

    req.onreadystatechange = function () {
        if (req.readyState === 4) {
            var buffer = req.response;
            var dataView = new DataView(buffer);
            var intArr = new Int32Array(buffer.byteLength / 4);
            intArr.set(0, 9);
        }
    }
```

Requirements

Supported in the following document modes: Internet Explorer 10 standards and Internet Explorer 11 standards.
Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards.

subarray Method (Int32Array)

10/18/2017 • 1 min to read • Edit Online

Gets a new Int32Array view of the ArrayBuffer Object store for this array, specifying the first and last members of the subarray.

Syntax

```
var newInt32Array = int32Array.subarray(begin, end);
```

Parameters

`newInt32Array`
The subarray returned by this method.

`begin`
The index of the beginning of the array.

`end`
The index of the end of the array. This is non-inclusive.

Remarks

If either `begin` or `end` is negative, it refers to an index from the end of the array, as opposed to from the beginning. If `end` is unspecified, the subarray contains all elements from `begin` to the end of the typed array. The range specified by the `begin` and `end` values is clamped to the valid index range for the current array. If the computed length of the new typed array would be negative, it is clamped to zero. The returned array is of the same type as the array on which this method is invoked.

Example

The following example shows how to get a subarray two elements long, starting with the first element of the array.

```
var req = new XMLHttpRequest();
    req.open('GET', "http://www.example.com");
    req.responseType = "arraybuffer";
    req.send();

    req.onreadystatechange = function () {
        if (req.readyState === 4) {
            var buffer = req.response;
            var intArr = new Int32Array(buffer.byteLength / 4);
            var subArr = intArr.subarray(0, 2);
        }
    }
```

Requirements

Supported in the following document modes: Internet Explorer 10 standards and Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards.

Intl.Collator Object (JavaScript)

10/18/2017 • 3 min to read • Edit Online

Provides locale-specific string comparisons.

Syntax

```
collatorObj = new Intl.Collator([locales][, options])
```

Parameters

`collatorObj`

Required. The variable name to assign the `Collator` object to.

`locales`

Optional. An array of locale strings that contain one or more language or locale tags. If you include more than one locale string, list them in descending order of priority so that the first entry is the preferred locale. If you omit this parameter, the default locale of the JavaScript runtime is used. See the Remarks section for more information.

`options`

Optional. An object that contains one or more properties that specify comparison options. See the Remarks section for details.

Remarks

The `locales` parameter must conform to BCP 47 language or locale tags such as "en-US" and "zh-Hans-CN". The tag may include language, region, country, and variant. For a list of languages, see the IANA language subtag registry. For examples of language tags, see Appendix A of BCP 47. For `Collator`, you may include the -u extension in the locale string to specify one or more of the following Unicode extensions:

- -co to specify variant collations (locale-specific): "*language-region*-u-co-*value*".
- -kn to specify a numeric comparison: "*language-region*-u-kn-true|false".
- -kf to specify whether to sort uppercase or lowercase characters first: "*language-region*-u-kf-upper|lower|false"). This extension is not currently supported.

  To specify a numeric comparison, you can set the -kn extension in the locale string or use the `numeric` property in the `options` parameter. If you're using the `numeric` property, the -kn value will not apply.

  The `options` parameter may include the following properties:

- `localeMatcher`. Specifies the locale-matching algorithm to use. The possible values are "lookup" and "best fit". The default value is "best fit".
- `usage`. Specifies whether the goal of comparison is sorting or searching. The possible values are "sort" and "search". The default value is "sort".
- `sensitivity`. Specifies the collator's sensitivity. The possible values are "base", "accent", "case", and "variant". The default value is `undefined`.
- `ignorePunctuation`. Specifies whether punctuation is ignored in the comparison. The possible values are "true" and "false". The default value is `false`.

- `numeric`. Specifies whether numeric sorting is used. The possible values are "true" and "false". The default value is `false`.
- `caseFirst`. Not currently supported.

Properties

The following table lists the properties of the `Collator` object.

Property	Description
compare	Returns a function that compares two strings by using the collator's sort order.
constructor	Specifies the function that creates a collator.
prototype	Returns a reference to the prototype for a collator.

Methods

The following table lists the methods of the `Collator` object.

Method	Description
resolvedOptions	Returns an object that contains the properties and values of the collator.

Example

The following example creates a `Collator` object and performs a comparison.

```
var co = new Intl.Collator(["de-DE"]);
co.compare("a", "b"); // Returns -1
```

Example

The following example uses `Collator` objects to sort an array. This example shows locale-specific differences.

```
var co1 = new Intl.Collator(["de-DE-u-co-phonebk"]);
var co2 = new Intl.Collator(["de-DE"]);
var co3 = new Intl.Collator(["en-US"]);

var arr = ["ä", "ad", "af", "a"];

if (console && console.log) {
    console.log(arr.sort(co1.compare));  // Returns a,ad,ä,af
    console.log(arr.sort(co2.compare));  // Returns a,ä,ad,af
    console.log(arr.sort(co3.compare));  // Returns a,ä,ad,af
}
```

Example

The following example uses a `Collator` object to search for a string and specifies comparison options.

```
// String to search
var arr = ["ä", "ad", "af", "a"];
// String searched for
var s = "af";

var co = new Intl.Collator("de-DE", { usage: "search" });
var matches = arr.filter(function (i) {
    return co.compare(i, s) === 0;
});

if (console && console.log) {
    console.log(matches);  // Returns af
}
```

Requirements

Supported in the Internet Explorer 11 standards document mode. Also supported in Store apps (Windows 8.1 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards. Not supported in Windows 8.

See Also

localeCompare Method (String)
Intl.DateTimeFormat Object
Intl.NumberFormat Object

compare Property (Intl.Collator)

10/18/2017 • 1 min to read • Edit Online

Returns a function that compares two strings by using the collator's sort order.

Syntax

```
collatorObj.compare
```

Parameters

`collatorObj`

Required. The name of the `Collator` object to use for the comparison.

Remarks

The function returned by the `compare` property takes two arguments, `x` and `y`, and returns the result of a locale-specific comparison of `x` and `y` by using the options specified in the `Collator` object.

The result of the comparison will be:

- -1 if `x` is before `y` in the sort order.
- 0 (zero) if `x` is equal to `y` in the sort order.
- 1 if `x` is after `y` in the sort order.

Example

The following example creates a `Collator` object and performs a comparison.

```
var co = new Intl.Collator(["de-DE-u-co-phonebk"]);

if (console && console.log) {
    console.log(co.compare("a", "b")); // Returns -1
}
```

Example

The following example uses `Collator` objects to sort an array. This example shows locale-specific differences.

```
var co1 = new Intl.Collator(["de-DE-u-co-phonebk"]);
var co2 = new Intl.Collator(["de-DE"]);
var co3 = new Intl.Collator(["en-US"]);

var arr = ["ä", "ad", "af", "a"];

if (console && console.log) {
    console.log(arr.sort(co1.compare));  // Returns a,ad,ä,af
    console.log(arr.sort(co2.compare));  // Returns a,ä,ad,af
    console.log(arr.sort(co3.compare));  // Returns a,ä,ad,af
}
```

Example

The following example uses a `Collator` object to search for a string.

```
// String to search
var arr = ["ä", "ad", "af", "a"];
// String searched for
var s = "af";

var co = new Intl.Collator("de-DE", { usage: "search" });
var matches = arr.filter(function (i) {
    return co.compare(i, s) === 0;
});

if (console && console.log) {
    console.log(matches);  // Returns af
}
```

Requirements

Supported in the Internet Explorer 11 standards document mode. Also supported in Store apps (Windows 8.1 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards. Not supported in Windows 8.

See Also

Intl.Collator Object

constructor Property (Intl.Collator)

10/18/2017 • 1 min to read • Edit Online

Specifies the function that creates a collator.

Syntax

```
collator.constructor
```

Remarks

The required `collator` is the name of the collator.

The `constructor` property is a member of the prototype of every object that has a prototype. This includes all intrinsic JavaScript objects except the `Global` and `Math` objects. The `constructor` property contains a reference to the function that constructs instances of that particular object.

Requirements

Supported in the Internet Explorer 11 standards document mode. Also supported in Store apps (Windows 8.1 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards. Not supported in Windows 8.

prototype Property (Intl.Collator)

10/18/2017 • 1 min to read • Edit Online

Returns a reference to the prototype for a collator.

Syntax

```
collator.prototype
```

Remarks

The `collator` argument is the name of a collator.

Use the `prototype` property to provide a base set of functionality to a class of objects. New instances of an object "inherit" the behavior of the prototype assigned to that object.

For example, to add a method to the `Intl.Collator` object that returns the value of the largest element of the set, declare the function, add it to `Intl.Collator.prototype`, and then use it.

Requirements

Supported in the Internet Explorer 11 standards document mode. Also supported in Store apps (Windows 8.1 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards. Not supported in Windows 8.

resolvedOptions Method (Intl.Collator)

10/18/2017 • 1 min to read • Edit Online

Returns an object containing the properties and values of the collator object.

Syntax

```
collatorObj.resolvedOptions()
```

Parameters

`collatorObj`

The `Collator` object to retrieve information from.

Remarks

The properties of the returned object correspond to the computed properties of the `Collator` object.

Requirements

Supported in the Internet Explorer 11 standards document mode. Also supported in Store apps (Windows 8.1 and Windows Phone 8.1). See Version Information.

Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards. Not supported in Windows 8.

See Also

Intl.Collator Object

Intl.DateTimeFormat Object (JavaScript)

10/18/2017 • 3 min to read • Edit Online

Provides locale-specific date and time formatting.

Syntax

```
dateTimeFormatObj = new Intl.DateTimeFormat([locales][, options])
```

Parameters

`dateTimeFormatObj`
Required. The variable name to assign the `DateTimeFormat` object to.

`locales`
Optional. An array of locale strings that contain one or more language or locale tags. If you include more than one locale string, list them in descending order of priority so that the first entry is the preferred locale. If you omit this parameter, the default locale of the JavaScript runtime is used. See the Remarks section for more information.

`options`
Optional. An object that contains one or more properties that specify formatting options for the date and time. See the Remarks section for details.

Remarks

The `locales` parameter must conform to BCP 47 language or locale tags such as "en-US" and "zh-CN". The tag may include language, region, country, and variant. For examples of language tags, see Appendix A of BCP 47. For `DateTimeFormat`, you may add the **-u** subtag in the locale string to include one or both of the following Unicode extensions:

- -nu to specify a numbering system extension: *language-region*-u-nu-*numberingsystem*

  where *numberingsystem* may be one of the following: arab, arabext, bali, beng, deva, fullwide, gujr, guru, hanidec, khmr, knda, laoo, latn, limb, mylm, mong, mymr, orya, tamldec, telu, thai, tibt.

- -ca to specify a calendar: *language-region*-u-ca-*calendar*

  where *calendar* may be one of the following: buddhist, chinese, gregory, islamic, islamicc, japanese.

  The `options` parameter may include the following properties:

PROPERTY	DESCRIPTION	POSSIBLE VALUES	DEFAULT VALUE
`localeMatcher`	Specifies the locale-matching algorithm to use.	"lookup","best fit"	"best fit"
`formatMatcher`	Specifies the format-matching algorithm to use.	"basic", "best fit"	"best fit"

PROPERTY	DESCRIPTION	POSSIBLE VALUES	DEFAULT VALUE
`hour12`	Specifies whether to use a 12-hour format for hours.	`true` (for 12-hour format), `false` (for 24-hour format)	
`timeZone`	Specifies the time zone. At minimum, "UTC" is always supported.	A timezone value such as "UTC".	"UTC"
`weekday`	Specifies formatting of the weekday.	"narrow", "short", "long".	undefined
`era`	Specifies formatting of the era.	"narrow", "short", "long"	undefined
`year`	Specifies formatting of the year.	"2-digit", "numeric"	undefined or "numeric"
`month`	Specifies formatting of the month.	"2-digit", "numeric", "narrow", "short", "long"	undefined or "numeric"
`day`	Specifies formatting of the day.	"2-digit", "numeric"	undefined or "numeric"
`hour`	Specifies formatting of the hour.	"2-digit", "numeric"	undefined
`minute`	Specifies formatting of the minute.	"2-digit", "numeric"	undefined
`second`	Specifies formatting of the second.	"2-digit", "numeric"	undefined
`timeZoneName`	Specifies formatting of the time zone. This property is not currently supported.	"short", "long".	This property is not currently supported.

The default values for `weekday`, `era`, `year`, `month`, `day`, `hour`, `minute`, and `second` are `undefined`. If you don't set these properties, "numeric" formatting is used for `year`, `month`, and `day`.

Each locale must support, at minimum, the following formats:

- weekday, year, month, day, hour, minute, second
- weekday, year, month, day
- year, month, day
- year, month
- month, day
- hour, minute, second
- hour, minute

Properties

The following table lists the properties of the `DateTimeFormat` object.

Property	Description
constructor	Specifies the function that creates a date/time formatter object.
format	Returns a function that formats a locale-specific date by using the date/time formatter settings.
prototype	Returns a reference to the prototype for a date/time formatter.

Methods

The following table lists the methods of the `DateTimeFormat` object.

Method	Description
resolvedOptions	Returns an object that contains the properties and values of the date/time formatter object.

Example

The following example shows the result of passing a date object to `DateTimeFormat` using different locales.

```
var date = new Date(Date.UTC(2013, 1, 1, 14, 0, 0));
var options = { weekday: "long", year: "numeric", month: "short",
    day: "numeric" };

if (console && console.log) {
    console.log(new Intl.DateTimeFormat("en-US").format(date));
    // Returns 2/1/2013
    console.log(new Intl.DateTimeFormat("ja-JP").format(date));
    // Returns 2013年2月1日
    console.log(new Intl.DateTimeFormat("ar-SA", options).format(date));
    // Returns الجمعة, ٢٠ ربيع الأول, ١٤٣٤
    console.log(new Intl.DateTimeFormat("hi-IN", options).format(date));
    // Returns शुक्रवार, 01 फरवरी 2013
}
```

Example

The following example creates a `DateTimeFormat` object that specifies the current weekday in long format using the Arabic (Saudi Arabia) locale, the Islamic calendar, and the Latin numbering system.

```
var dtf = new Intl.DateTimeFormat(["ar-SA-u-ca-islamic-nu-latn"], {
    weekday: "long",
    year: "numeric",
    day: "numeric",
    month: "long"
});

If (console && console.log) {
    console.log(dtf.format(new Date()));
    // Returns 1434 ,رمضان 19 ,الجمعة
}
```

Requirements

Supported in the Internet Explorer 11 standards document mode. Also supported in Store apps (Windows 8.1 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards. Not supported in Windows 8.

See Also

toLocaleString (Date)
toLocaleDateString Method (Date)
toLocaleTimeString Method (Date)
Intl.Collator Object
Intl.NumberFormat Object

constructor Property (Intl.DateTimeFormat)

10/18/2017 • 1 min to read • Edit Online

Specifies the function that creates a formatter.

Syntax

```
dateTimeFormatter.constructor
```

Remarks

The required `dateTimeFormatter` is the name of the formatter.

The `constructor` property is a member of the prototype of every object that has a prototype. This includes all intrinsic JavaScript objects except the `Global` and `Math` objects. The `constructor` property contains a reference to the function that constructs instances of that particular object.

Requirements

Supported in the Internet Explorer 11 standards document mode. Also supported in Store apps (Windows 8.1 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards. Not supported in Windows 8.

format Property (Intl.DateTimeFormat)

10/18/2017 • 1 min to read • Edit Online

Returns a function that formats a locale-specific date by using the specified date/time formatter settings.

Syntax

```
dateTimeFormatObj.format
```

Parameters

`dateTimeFormatObj`

Required. The name of the `DateTimeFormat` object to use as a formatter.

Remarks

The function returned by the `format` property takes a single argument, `date`, and returns a string that represents the localized date by using the options specified in the `DateTimeFormat` object. The `date` parameter of the returned function must be a number, date string, or a `Date` object. If `date` is not provided, the function uses Date.now as the default input value.

Example

The following example uses a `DateTimeFormat` object to localize the date "Dec 1, 2007" into German and reformat it.

```
var dtFormat = new Intl.DateTimeFormat(["de"], {
    year: "numeric",
    month: "long",
    day: "2-digit",
    hour: "numeric"
});

if (console && console.log) {
    console.log(dtFormat.format(new Date("Dec 1, 2007")));
    // Returns 01. Dezember 2007 00:00
}
```

Requirements

Supported in the Internet Explorer 11 standards document mode. Also supported in Store apps (Windows 8.1 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards. Not supported in Windows 8.

See Also

Intl.DateTimeFormat Object

prototype Property (Intl.DateTimeFormat)

10/18/2017 • 1 min to read • Edit Online

Returns a reference to the prototype for a formatter.

Syntax

```
dateTimeFormat.prototype
```

Remarks

The `dateTimeFormat` argument is the name of a formatter.

Use the `prototype` property to provide a base set of functionality to a class of objects. New instances of an object "inherit" the behavior of the prototype assigned to that object.

For example, to add a method to the `Intl.DateTimeFormat` object that returns the value of the largest element of the set, declare the function, add it to `Intl.DateTimeFormat.prototype`, and then use it.

Requirements

Supported in the Internet Explorer 11 standards document mode. Also supported in Store apps (Windows 8.1 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards. Not supported in Windows 8.

resolvedOptions Method (Intl.DateTimeFormat)

10/18/2017 • 1 min to read • Edit Online

Returns an object that contains the properties and values of the date/time formatter object.

Syntax

```
dateTimeFormatObj.resolvedOptions()
```

Parameters

`dateTimeFormatObj`

The `DateTimeFormat` object to retrieve information from.

Remarks

The properties of the returned object correspond to the computed properties of the `DateTimeFormat` object.

Requirements

Supported in the Internet Explorer 11 standards document mode. Also supported in Store apps (Windows 8.1 and Windows Phone 8.1). See Version Information.

Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards. Not supported in Windows 8.

See Also

Intl.DateTimeFormat Object

Intl.NumberFormat Object (JavaScript)

10/18/2017 • 3 min to read • Edit Online

Provides locale-specific number formatting.

Syntax

```
numberFormatObj = new Intl.NumberFormat([locales][, options])
```

Parameters

`numberFormatObj`
Required. The variable name to assign the `NumberFormat` object to.

`locales`
Optional. An array of locale strings that contain one or more language or locale tags. If you include more than one locale string, list them in descending order of priority so that the first entry is the preferred locale. If you omit this parameter, the default locale of the JavaScript runtime is used. See the Remarks section for more information.

`options`
Optional. An object that contains one or more properties that specify number formatting options. See the Remarks section for details.

Remarks

The `locales` parameter must conform to BCP 47 language or locale tags such as "en-US" and "zh-CN". The tag may include language, region, country, and variant. For examples of language tags, see Appendix A of BCP 47. For `NumberFormat`, you may add the **-u** subtag followed by -nu to specify a numbering system extension:

"*language-region*-u-nu-*numberingsystem*"

where *numberingsystem* may be one of the following: arab, arabext, bali, beng, deva, fullwide, gujr, guru, hanidec, khmr, knda, laoo, latn, limb, mylm, mong, mymr, orya, tamldec, telu, thai, tibt.

The `options` parameter may include the following properties:

PROPERTY	DESCRIPTION	POSSIBLE VALUES	DEFAULT VALUE
`localeMatcher`	Specifies the locale-matching algorithm to use.	"lookup", "best fit"	"best fit"
`style`	Specifies the number format style.	"decimal", "percent", "currency"	"decimal"
`currency`	Specifies the ISO 4217 currency value as an alphabetic code. If the `style` is set to "currency", this value is required.	See the ISO currency and funds code list.	undefined

PROPERTY	DESCRIPTION	POSSIBLE VALUES	DEFAULT VALUE
`currencyDisplay`	Specifies whether to display the currency as an ISO 4217 alphabetic currency code, a localized currency symbol, or a localized currency name. This value is used only if `style` is set to "currency".	"code", "symbol", "name"	"symbol"
`useGrouping`	Specifies whether a grouping separator should be used.	true, false	`true`.
`minimumIntegerDigits`	Specifies the minimum number of integral digits to use.	1 to 21.	21
`minimumFractionDigits`	. Specifies the minimum number of fractional digits to be used.	0 to 20.	0
`maximumFractionDigits`	Specifies the maximum number of fractional digits to be used.	This value can range from `minimumFractionDigits` to 20.	20.
`minimumSignificantDigits`	Specifies the minimum number of fractional digits to be shown.	This value can range from 1 to 21.	1
`maximumSignificantDigits`	Specifies the maximum number of fractional digits to be shown.	This value can range from `minimumSignificantDigits` to 21.	21

Properties

The following table lists the properties of the `NumberFormat` object.

Property	Description
constructor	Specifies the function that creates a number formatter object.
format	Returns a function that formats a number by using the number formatter settings.
prototype	Returns a reference to the prototype for a number formatter.

Methods

The following table lists the methods of the `NumberFormat` object.

Method	Description

resolvedOptions	Returns an object that contains the properties and values of the number formatter object.

Example

The following example creates a `NumberFormat` object for the en-US locale with the specified formatting options.

```
var nf = new Intl.NumberFormat(["en-US"], {
    style: "currency",
    currency: "CNY",
    currencyDisplay: "symbol",
    maximumFractionDigit: 1
});

if (console && console.log) {
    console.log(nf.format(100)); // Returns ¥100.00
}
```

Example

The following examples show the result from using several different locales and options.

```
var number = 123456789;
var options1 = { style: "percent" };
var options2 = { style: "currency", currency: "INR" };

if (console && console.log) {
    console.log(new Intl.NumberFormat("en-US").format(number));
    // Returns 123,456,789
    console.log(new Intl.NumberFormat("ja-JP").format(number));
    // Returns 123,456,789
    console.log(new Intl.NumberFormat("ar-SA", options1).format(number));
    // Returns ١٢,٣٤٥,٦٧٨,٩٠٠ %
    console.log(new Intl.NumberFormat("hi-IN", options2).format(number));
    // Returns ₹ 12,34,56,789.00
}
```

Requirements

Supported in the Internet Explorer 11 standards document mode. Also supported in Store apps (Windows 8.1 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards. Not supported in Windows 8.

See Also

toLocaleString (Number)
Intl.Collator Object
Intl.DateTimeFormat Object

constructor Property (Intl.NumberFormat)

10/18/2017 • 1 min to read • Edit Online

Specifies the function that creates a formatter.

Syntax

```
numberFormatter.constructor
```

Remarks

The required `numberFormatter` is the name of the formatter.

The `constructor` property is a member of the prototype of every object that has a prototype. This includes all intrinsic JavaScript objects except the `Global` and `Math` objects. The `constructor` property contains a reference to the function that constructs instances of that particular object.

Requirements

Supported in the Internet Explorer 11 standards document mode. Also supported in Store apps (Windows 8.1 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards. Not supported in Windows 8.

format Property (Intl.NumberFormat)

10/18/2017 • 1 min to read • Edit Online

Returns a function that formats a locale-specific number by using the specified number formatter settings.

Syntax

```
numberFormatObj.format
```

Parameters

`numberFormatObj`

Required. The name of the `NumberFormat` object to use as a formatter.

Remarks

The function returned by the `format` property takes a single argument, `value`, and returns a string that represents the localized number by using the options specified in the `NumberFormat` object. If `value` is not provided, the function returns `NaN` (not a number).

Example

The following example uses a `NumberFormat` object to create a localized number.

```
var nf = new Intl.NumberFormat(["en-US"], {
    style: "currency",
    currency: "CNY",
    currencyDisplay: "symbol",
    maximumFractionDigit: 1
})

if (console && console.log) {
    console.log(nf.format(100)); // "¥100.00"
}
```

Requirements

Supported in the Internet Explorer 11 standards document mode. Also supported in Store apps (Windows 8.1 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards. Not supported in Windows 8.

See Also

Intl.NumberFormat Object

prototype Property (Intl.NumberFormat)

10/18/2017 • 1 min to read • Edit Online

Returns a reference to the prototype for a formatter.

Syntax

```
numberFormat.prototype
```

Remarks

The `numberFormat` argument is the name of a formatter.

Use the `prototype` property to provide a base set of functionality to a class of objects. New instances of an object "inherit" the behavior of the prototype assigned to that object.

For example, to add a method to the `Intl.NumberFormat` object that returns the value of the largest element of the set, declare the function, add it to `Intl.NumberFormat.prototype`, and then use it.

Requirements

Supported in the Internet Explorer 11 standards document mode. Also supported in Store apps (Windows 8.1 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards. Not supported in Windows 8.

resolvedOptions Method (Intl.NumberFormat)

10/18/2017 • 1 min to read • Edit Online

Returns an object that contains the properties and values of the `NumberFormat` formatter object.

Syntax

```
numberFormatObj.resolvedOptions()
```

Parameters

`numberFormatObj`

The `NumberFormat` object to retrieve information from.

Remarks

The properties of the returned object correspond to the computed properties of the `NumberFormat` object.

Requirements

Supported in the Internet Explorer 11 standards document mode. Also supported in Store apps (Windows 8.1 and Windows Phone 8.1). See Version Information.

Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards. Not supported in Windows 8.

See Also

Intl.NumberFormat Object

JSON Object (JavaScript)

10/18/2017 • 1 min to read • Edit Online

An intrinsic object that provides functions to convert JavaScript values to and from the JavaScript Object Notation (JSON) format. The `JSON.stringify` function serializes a JavaScript value to JSON text. The `JSON.parse` function deserializes JSON text to produce a JavaScript value.

Syntax

```
JSON.[method]
```

Parameters

`Method`
Required. Name of one of the methods of the `JSON` object.

Remarks

You cannot create a `JSON` object by using the `new` operator. An error occurs if you try to do this. However, you can override or modify the `JSON` object.

The scripting engine creates the `JSON` object when the engine is loaded. Its methods are available to your script at all times.

To use the intrinsic `JSON` object, make sure that you do not override it with another `JSON` object that is defined in your script. You may need to modify existing script statements that detect the presence of a `JSON` object because those statements will evaluate differently. This is demonstrated in the following example.

```
if (!this.JSON) {
    // JSON object does not exist.
    }
```

In the previous example, `!this.JSON` evaluates to false in Internet Explorer 8 standards mode, Internet Explorer 9 standards mode, Internet Explorer 10 standards mode, Internet Explorer 11 standards mode, and win8_appname_long apps. Therefore, the code inside the `if` statement does not execute.

Functions

JSON.parse Function

JSON.stringify Function

Requirements

Supported in the following document modes: Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7

standards.

See Also

toJSON Method (Date)
JavaScript Objects
Hub template sample app (Windows Store)

JSON.parse Function (JavaScript)

1/12/2018 • 2 min to read • Edit Online

Converts a JavaScript Object Notation (JSON) string into an object.

Syntax

```
JSON.parse(text [, reviver])
```

Parameters

`text`
Required. A valid JSON string.

`reviver`
Optional. A function that transforms the results. This function is called for each member of the object. If a member contains nested objects, the nested objects are transformed before the parent object. For each member, the following occurs:

- If `reviver` returns a valid value, the member value is replaced with the transformed value.
- If `reviver` returns the same value it received, the member value is not modified.
- If `reviver` returns `null` or `undefined`, the member is deleted.

Return Value

An object or array.

Exceptions

If this function causes a JavaScript parser error (such as "SCRIPT1014: Invalid character"), the input text does not comply with JSON syntax. To correct the error, do one of the following:

- Modify the `text` argument to comply with JSON syntax. For more information, see the BNF syntax notation of JSON objects.

  For example, if the response is in JSONP format instead of pure JSON, try this code on the response object:

  ```
  var fixedResponse = response.responseText.replace(/\\'/g, "'");
  var jsonObj = JSON.parse(fixedResponse);
  ```

- Make sure that the `text` argument was serialized by a JSON-compliant implementation such as `JSON.stringify`.
- Run the `text` argument in a JSON validator such as JSLint or JSON to CSV to help identify syntax errors.

Example

The following example uses `JSON.parse` to convert a JSON string to an object.

```
var jsontext = '{"firstname":"Jesper","surname":"Aaberg","phone":["555-0100","555-0120"]}';
var contact = JSON.parse(jsontext);
document.write(contact.surname + ", " + contact.firstname);
document.write(contact.phone[1]);
// Output:
// Aaberg, Jesper
// 555-0100
```

Example

The following example converts an array to a JSON string by using `JSON.stringify`, and then converts the string back to an array by using `JSON.parse`.

```
var arr = ["a", "b", "c"];
var str = JSON.stringify(arr);
document.write(str);
document.write ("<br/>");

var newArr = JSON.parse(str);

while (newArr.length > 0) {
    document.write(newArr.pop() + "<br/>");
}

// Output:
// ["a","b","c"]
// c
// b
// a
```

Example

The `reviver` function is often used to transform JSON representation of International Organization for Standardization (ISO) date strings into Coordinated Universal Time (UTC) format `Date` objects. This example uses `JSON.parse` to deserialize an ISO-formatted date string. The `dateReviver` function returns `Date` objects for members that are formatted like ISO date strings.

```
var jsontext = '{ "hiredate": "2008-01-01T12:00:00Z", "birthdate": "2008-12-25T12:00:00Z" }';
var dates = JSON.parse(jsontext, dateReviver);
document.write(dates.birthdate.toUTCString());

function dateReviver(key, value) {
    var a;
    if (typeof value === 'string') {
        a = /^(\d{4})-(\d{2})-(\d{2})T(\d{2}):(\d{2}):(\d{2}(?:\.\d*)?)Z$/.exec(value);
        if (a) {
            return new Date(Date.UTC(+a[1], +a[2] - 1, +a[3], +a[4],
                            +a[5], +a[6]));
        }
    }
    return value;
};

// Output:
// Thu, 25 Dec 2008 12:00:00 UTC
```

Requirements

Supported in the following document modes: Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards.

See Also

JSON.stringify Function
toJSON Method (Date)
Hub template sample app (Windows Store)

JSON.stringify Function (JavaScript)

10/18/2017 • 3 min to read • Edit Online

Converts a JavaScript value to a JavaScript Object Notation (JSON) string.

Syntax

```
JSON.stringify(
value [, replacer] [, space])
```

Parameters

`value`
Required. A JavaScript value, usually an object or array, to be converted.

`replacer`
Optional. A function or array that transforms the results.

If `replacer` is a function, `JSON.stringify` calls the function, passing in the key and value of each member. The return value is used instead of the original value. If the function returns `undefined`, the member is excluded. The key for the root object is an empty string: "".

If `replacer` is an array, only members with key values in the array will be converted. The order in which the members are converted is the same as the order of the keys in the array. The `replacer` array is ignored when the `value` argument is also an array.

`space`
Optional. Adds indentation, white space, and line break characters to the return-value JSON text to make it easier to read.

If `space` is omitted, the return-value text is generated without any extra white space.

If `space` is a number, the return-value text is indented with the specified number of white spaces at each level. If `space` is greater than 10, text is indented 10 spaces.

If `space` is a non-empty string, such as '\t', the return-value text is indented with the characters in the string at each level.

If `space` is a string that is longer than 10 characters, the first 10 characters are used.

Return Value

A string that contains the JSON text.

Exceptions

EXCEPTION	CONDITION
Invalid replacer argument	The `replacer` argument is not a function or an array.

EXCEPTION	CONDITION
Circular reference in value argument not supported	The `value` argument contains a circular reference.

Remarks

If `value` has a `toJSON` method, the `JSON.stringify` function uses the return value of that method. If the return value of the `toJSON` method is `undefined`, the member is not converted. This enables an object to determine its own JSON representation.

Values that do not have JSON representations, such as `undefined`, will not be converted. In objects, they will be dropped. In arrays, they will be replaced with null.

String values begin and end with a quotation mark. All Unicode characters may be enclosed in the quotation marks except for the characters that must be escaped by using a backslash. The following characters must be preceded by a backslash:

- Quotation mark (")
- Backslash (\)
- Backspace (b)
- Formfeed (f)
- Newline (n)
- Carriage return (r)
- Horizontal tab (t)
- Four-hexadecimal-digits (uhhhh)

Order of Execution

During the serialization process, if a `toJSON` method exists for the `value` argument, `JSON.stringify` first calls the `toJSON` method. If it does not exist, the original value is used. Next, if a `replacer` argument is provided, the value (original value or `toJSON` return-value) is replaced with the return-value of the `replacer` argument. Finally, white spaces are added to the value based on the optional `space` argument to generate the final JSON text.

Example

This example uses `JSON.stringify` to convert the `contact` object to JSON text. The `memberfilter` array is defined so that only the `surname` and `phone` members are converted. The `firstname` member is omitted.

```
var contact = new Object();
contact.firstname = "Jesper";
contact.surname = "Aaberg";
contact.phone = ["555-0100", "555-0120"];

var memberfilter = new Array();
memberfilter[0] = "surname";
memberfilter[1] = "phone";
var jsonText = JSON.stringify(contact, memberfilter, "\t");
document.write(jsonText);
// Output:
// { "surname": "Aaberg", "phone": [ "555-0100", "555-0120" ] }
```

Example

This example uses `JSON.stringify` with an array. The `replaceToUpper` function converts every string in the array to uppercase.

```
var continents = new Array();
continents[0] = "Europe";
continents[1] = "Asia";
continents[2] = "Australia";
continents[3] = "Antarctica";
continents[4] = "North America";
continents[5] = "South America";
continents[6] = "Africa";

var jsonText = JSON.stringify(continents, replaceToUpper);

function replaceToUpper(key, value) {
    return value.toString().toUpperCase();
}

//Output:
// "EUROPE,ASIA,AUSTRALIA,ANTARCTICA,NORTH AMERICA,SOUTH AMERICA,AFRICA"
```

Example

This example uses the `toJSON` method to convert string values to uppercase.

```
var contact = new Object();
contact.firstname = "Jesper";
contact.surname = "Aaberg";
contact.phone = ["555-0100", "555-0120"];

contact.toJSON = function(key)
 {
    var replacement = new Object();
    for (var val in this)
    {
        if (typeof (this[val]) === 'string')
            replacement[val] = this[val].toUpperCase();
        else
            replacement[val] = this[val]
    }
    return replacement;
};

var jsonText = JSON.stringify(contact);
document.write(jsonText);

// Output:
{"firstname":"JESPER","surname":"AABERG","phone":["555-0100","555-0120"]}

'{"firstname":"JESPER","surname":"AABERG","phone":["555-0100","555-0120"]}'
*/
```

Requirements

Supported in the following document modes: Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7

standards.

See Also

JSON.parse Function
toJSON Method (Date)
Feed reader sample app (Windows Store)

Map Object (JavaScript)

10/18/2017 • 1 min to read • Edit Online

A collection of key/value pairs.

Syntax

```
mapObj = new Map()
```

Remarks

The keys and values in the collection may be of any type. If you add a value to the collection using an existing key, the new value will replace the old value.

Properties

The following table lists the properties of the `Map` object.

PROPERTY	DESCRIPTION
constructor	Specifies the function that creates a map.
prototype	Returns a reference to the prototype for a map.
size	Returns the number of elements in a map.

Methods

The following table lists the methods of the `Map` object.

METHOD	DESCRIPTION
clear	Removes all elements from a map.
delete	Removes a specified element from a map.
forEach	Performs the specified action for each element in a map.
get	Returns a specified element from a map.
has	Returns `true` if the map contains a specified element.
set	Adds a new element to a map.
toString	Returns a string representation of a map.
valueOf	Returns the primitive value of the specified object.

Example

The following example shows how to add members to a `Map` and then retrieve them.

```
var m = new Map();
m.set(1, "black");
m.set(2, "red");
m.set("colors", 2);
m.set({x:1}, 3);

m.forEach(function (item, key, mapObj) {
    document.write(item.toString() + "<br />");
});

document.write("<br />");
document.write(m.get(2));

// Output:
// black
// red
// 2
// 3
//
// red
```

Requirements

Supported in the Internet Explorer 11 standards document mode. Also supported in Store apps (Windows 8.1 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards. Not supported in Windows 8.

constructor Property (Map)

10/18/2017 • 1 min to read • Edit Online

Specifies the function that creates a map.

Syntax

```
map.constructor
```

Remarks

The required `map` is the name of the map.

The `constructor` property is a member of the prototype of every object that has a prototype. This includes all intrinsic JavaScript objects except the `Global` and `Math` objects. The `constructor` property contains a reference to the function that constructs instances of that particular object.

Requirements

Supported in the Internet Explorer 11 standards document mode. Also supported in Store apps (Windows 8.1 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards. Not supported in Windows 8.

prototype Property (Map)

10/18/2017 • 1 min to read • Edit Online

Returns a reference to the prototype for a map.

Syntax

```
map.prototype
```

Remarks

The `map` argument is the name of a map.

Use the `prototype` property to provide a base set of functionality to a class of objects. New instances of an object "inherit" the behavior of the prototype assigned to that object.

For example, to add a method to the `Map` object that returns the value of the largest element of the set, declare the function, add it to `Map.prototype`, and then use it.

Requirements

Supported in the Internet Explorer 11 standards document mode. Also supported in Store apps (Windows 8.1 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards. Not supported in Windows 8.

size Property (Map) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Returns the number of elements in a map.

Syntax

```
sizeVar = mapObj.size
```

Parameters

`sizeVar`

Required. Any number.

`mapObj`

Required. Any `Map` object.

Requirements

Supported in the Internet Explorer 11 standards document mode. Also supported in Store apps (Windows 8.1 and Windows Phone 8.1). See Version Information.

Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards. Not supported in Windows 8.

clear Method (Map) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Removes all elements from a map.

Syntax

```
mapObj.clear()
```

Parameters

`mapObj`

Required. The map to clear.

Requirements

Supported in the Internet Explorer 11 standards document mode. Also supported in Store apps (Windows 8.1 and Windows Phone 8.1). See Version Information.

Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards. Not supported in Windows 8.

delete Method (Map) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Removes the specified element from a map.

Syntax

```
mapObj.delete(key)
```

Parameters

`mapObj`

Required. A `Map` object.

`key`

Required. The key of the element to remove.

Property Value/Return Value

`true` if the element has been removed.

Example

The following example shows how to add members to a `Map` and then delete one of them.

```
var m = new Map();
m.set(1, "black");
m.set(2, "red");
m.set("colors", 2);
m.delete(1);

m.forEach(function (item) {
    document.write(item.toString() + "<br />");
});

// Output:
// red
// 2
```

Requirements

Supported in the Internet Explorer 11 standards document mode. Also supported in Store apps (Windows 8.1 and Windows Phone 8.1). See Version Information.

Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards. Not supported in Windows 8.

forEach Method (Map) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Performs the specified action for each element in a map.

Syntax

```
mapObj.forEach(callbackfn[, thisArg])
```

Parameters

`mapObj`
Required. A `Map` object.

`callbackfn`
Required. The function that `forEach` calls one time for each element in the map. `callbackfn` accepts up to three arguments. `forEach` calls the `callbackfn` function one time for each element in the map.

`thisArg`
Optional. An object that the `this` keyword can refer to in the `callbackfn` function. If `thisArg` is omitted, `undefined` is used as the `this` value.

Exceptions

If the `callbackfn` argument is not a function object, a `TypeError` exception is thrown.

Remarks

The syntax of the callback function is as follows:

`function callbackfn(value, key, mapObj)`

You can declare the callback function by using up to three parameters, as shown in the following table.

CALLBACK ARGUMENT	DEFINITION
`value`	A value contained in the map.
`key`	A key contained in the map.
`mapObj`	The `Map` object to traverse.

Example

The following example shows how to retrieve members of a `Map` using `forEach`.

```
var m = new Map();
m.set(1, "black");
m.set(2, "red");
m.set("colors", 2);
m.set({x:1}, 3);

m.forEach(function (item, key, mapObj) {
    document.write(item.toString() + "<br />");
});

document.write("<br />");
document.write(m.get(2));

// Output:
// black
// red
// 2
// 3
//
// red
```

Requirements

Supported in the Internet Explorer 11 standards document mode. Also supported in Store apps (Windows 8.1 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards. Not supported in Windows 8.

get Method (Map) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Returns a specified element from a map.

Syntax

```
mapObj.get(key)
```

Parameters

`mapObj`

Required. A `Map` object.

`key`

Required. The key of an element in the `Map`.

Property Value/Return Value

Returns the object associated with the key. If the `Map` does not contain the key, this method returns an `undefined` value.

Example

The following example shows how to retrieve an element from a `Map` object.

```
var m = new Map();
m.set(1, "black");
m.set(2, "red");
m.set("colors", 2);

document.write(m.get(2));

// Output:
// red
```

Requirements

Supported in the Internet Explorer 11 standards document mode. Also supported in Store apps (Windows 8.1 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards. Not supported in Windows 8.

has Method (Map) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Returns `true` if the map contains the specified element.

Syntax

```
mapObj.has(key)
```

Parameters

`mapObj`
Required. A `Map` object.

`key`
Required. The key of the element to test.

Property Value/Return Value

`true` if the map contains the specified element.

Example

The following example shows how to add a member to a `Map` and then check whether the map contains it.

```
var m = new Map();
m.set(2, "red");

document.write(m.has(2));

// Output:
// true
```

Requirements

Supported in the Internet Explorer 11 standards document mode. Also supported in Store apps (Windows 8.1 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards. Not supported in Windows 8.

set Method (Map) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Adds a new element to a map.

Syntax

```
mapObj.set(key, value)
```

Parameters

`mapObj`

Required. A `Map` object.

`key`

Required. The key for the new element.

`value`

Required. The value of the element to add.

Property Value/Return Value

Returns the `Map` object that contains the new key/value pair.

Remarks

If you add a value to the collection using an existing key, the new value will replace the old value.

Example

The following example shows how to add members to a `Map` and then retrieve them.

```
var m = new Map();
m.set(1, "black");
m.set(2, "red");
m.set("colors", 2);

m.forEach(function (item) {
    document.write(item.toString() + "<br />");
});

// Output:
// black
// red
// 2
```

Requirements

Supported in the Internet Explorer 11 standards document mode. Also supported in Store apps (Windows 8.1 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards. Not

supported in Windows 8.

toString Method (Map) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Returns a string representation of a map.

Syntax

```
mapObj.toString()
```

Parameters

`mapObj`

Required. A `Map` object.

Property Value/Return Value

The string representation of the map.

Exceptions

Requirements

Supported in the Internet Explorer 11 standards document mode. Also supported in Store apps (Windows 8.1 and Windows Phone 8.1). See Version Information.

Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards. Not supported in Windows 8.

valueOf Method (Map) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Returns the primitive value of the specified object.

Syntax

```
mapObj.valueOf()
```

Parameters

`mapObj`

Required. A `Map` object.

Property Value/Return Value

Returns the map instance.

Requirements

Supported in the Internet Explorer 11 standards document mode. Also supported in Store apps (Windows 8.1 and Windows Phone 8.1). See Version Information.

Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards. Not supported in Windows 8.

Math Object (JavaScript)

10/18/2017 • 2 min to read • Edit Online

An intrinsic object that provides basic mathematics functionality and constants.

Syntax

```
Math.[{property | method}]
```

Parameters

property
Required. Name of one of the properties of the **Math**. object.

method
Required. Name of one of the methods of the **Math**. object.

Remarks

The **Math** object cannot be created using the **new** operator, and gives an error if you attempt to do so. The scripting engine creates it when the engine is loaded. All of its methods and properties are available to your script at all times.

Requirements

The `Math` object was introduced in Internet Explorer before Internet Explorer 6.

Constants

The following table lists the constants of the `Math` object.

CONSTANT	DESCRIPTION
Math.E Constant	The mathematical constant e. This is Euler's number, the base of natural logarithms.
Math.LN2 Constant	The natural logarithm of 2.
Math.LN10 Constant	The natural logarithm of 10.
Math.LOG2E Constant	The base-2 logarithm of e.
Math.LOG10E Constant	The base-10 logarithm of e.
Math.PI Constant	Pi. This is the ratio of the circumference of a circle to its diameter.

CONSTANT	DESCRIPTION
Math.SQRT1_2 Constant	The square root of 0.5, or, equivalently, one divided by the square root of 2.
Math.SQRT2 Constant	The square root of 2.

Functions

The following table lists the functions of the `Math` object.

FUNCTION	DESCRIPTION
Math.abs Function	Returns the absolute value of a number.
Math.acos Function	Returns the arccosine of a number.
Math.acosh Function	Returns the hyperbolic arccosine (or inverse hyperbolic cosine) of a number.
Math.asin Function	Returns the arcsine of a number.
Math.asinh Function	Returns the inverse hyperbolic sine of a number.
Math.atan Function	Returns the arctangent of a number.
Math.atan2 Function	Returns the angle (in radians) from the X axis to a point represented by the supplied y and x coordinates.
Math.atanh Function	Returns the inverse hyperbolic tangent of a number.
Math.ceil Function	Returns the smallest integer that is greater than or equal to the supplied numeric expression.
Math.cos Function	Returns the cosine of a number.
Math.cosh Function	Returns the hyperbolic cosine of a number.
Math.exp Function	Returns *e* (the base of natural logarithms) raised to a power.
Math.expm1 Function	Returns the result of subtracting 1 from e (the base of the natural logarithms) raised to a power).
Math.floor Function	Returns the greatest integer that is less than or equal to the supplied numeric expression.
Math.hypot Function	Returns the square root of the sum of the squares of the arguments.
Math.imul Function	Returns the product of two numbers that are treated as 32-bit signed integers.

FUNCTION	DESCRIPTION
Math.log Function	Returns the natural logarithm of a number.
Math.log1p Function	Returns the natural logarithm of 1 + a number.
Math.log10 Function	Returns the base 10 logarithm of a number.
Math.log2 Function	Returns the base 2 logarithm of a number.
Math.max Function	Returns the greater of two supplied numeric expressions.
Math.min Function	Returns the lesser of two supplied numbers.
Math.pow Function	Returns the value of a base expression raised to a specified power.
Math.random Function	Returns a pseudorandom number between 0 and 1.
Math.round Function	Returns a specified numeric expression rounded to the nearest integer.
Math.sign Function	Returns the sign of a number, indicating whether the number is positive, negative, or 0.
Math.sin Function	Returns the sine of a number.
Math.sinh Function	Returns the inverse hyperbolic sine of a number.
Math.sqrt Function	Returns the square root of a number.
Math.tan Function	Returns the tangent of a number.
Math.tanh Function	Returns the hyperbolic tangent of a number.
Math.trunc Function	Returns the integer portion of a number, removing any fractional digits.

See Also

JavaScript Objects
Number Object

Math Constants (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Math constants return constant values that are properties of the `Math` object.

Math Object Constants

The following table lists constant values that are properties of the Math object.

CONSTANT	DESCRIPTION	APPROXIMATE VALUE
`Math.E`	The mathematical constant e. This is Euler's number, the base of natural logarithms.	2.718
`Math.LN2`	The natural logarithm of 2.	0.693
`Math.LN10`	The natural logarithm of 10.	2.302
`Math.LOG2E`	The base-2 logarithm of e.	1.443
`Math.LOG10E`	The base-10 logarithm of e.	0.434
`Math.PI`	Pi. This is the ratio of the circumference of a circle to its diameter.	3.14159
`Math.SQRT1_2`	The square root of 0.5, or, equivalently, one divided by the square root of 2.	0.707
`Math.SQRT2`	The square root of 2.	1.414

Example

The following example illustrates how to use the `Math.PI` constant.

```
var radius = 3;
var area = Math.PI * radius * radius;
document.write(area);

// Output: 28.274333882308138
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

Applies To: Math Object

See Also

Number Constants
JavaScript Constants

Math.abs Function (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Returns the absolute value of a number (the value without regard to whether it is positive or negative). For example, the absolute value of -5 is the same as the absolute value of 5.

Syntax

```
Math.abs(number)
```

Parameters

The required `number` argument is a numeric expression for which the absolute value is needed.

Return Value

The absolute value of the `number` argument.

Example

The following example illustrates the use of the `abs` function.

```
var s;
var v1 = Math.abs(6);
var v2 = Math.abs(-6);
if (v1 == v2) {
    document.write("Absolute values are the same.");
}
else {
document.write("Absolute values are different.");
}

// Output: Absolute values are the same.
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

Applies To: Math Object

See Also

Math Object

Math.acos Function (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Returns the arc cosine (or inverse cosine) of a number.

Syntax

```
Math.acos(number)
```

Parameters

The required `number` argument is a numeric expression.

Return Value

The arc cosine of the `number` argument, in radians.

Example

The following code shows how to use the `acos` function.

```
var v1 = Math.acos(-1.0);
var v2 = Math.cos(-1.0);

document.write(v1);
document.write("<br/>");
document.write(v2);

// Output:
// 3.141592653589793
// 0.5403023058681398
```

Remarks

Applies To: Math Object

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

See Also

Math.asin Function
Math.atan Function
Math.cos Function
Math.sin Function
Math.tan Function
Math Object

Math.acosh Function (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Returns the hyperbolic arccosine (or inverse hyperbolic cosine) of a number.

Syntax

```
Math.acosh(number)
```

Parameters

The required `number` argument is a numeric expression.

Return Value

The inverse hyperbolic cosine of the `number` argument, in radians.

Example

The following code shows how to use the `acosh` function.

```
var v1 = Math.acosh(3);
vary v2 = Math.acosh(-1);

document.write(v1);
document.write("</br>");
document.write(v2);

// Output:
// 1.762747174039086
// NaN
```

Remarks

Applies To: Math Object

Requirements

Supported in Microsoft Edge (Edge browser). Also supported in Store apps (Microsoft Edge on Windows 10). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Not supported in Windows 8.1.

See Also

Math.acos Function
Math.asin Function
Math.atan Function
Math.cos Function
Math.sin Function

Math.tan Function
Math Object

Math.asin Function (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Returns the arcsine of a number.

Syntax

```
Math.asin(number)
```

Remarks

The required `number` argument is a numeric expression for which the arcsine is needed.

The return value is the arcsine of the `number` argument, in radians.

Applies To: Math Object

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

See Also

Math.acos Function
Math.atan Function
Math.cos Function
Math.sin Function
Math.tan Function
Math Object

Math.asinh Function (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Returns the inverse hyperbolic sine of a number.

Syntax

```
Math.asinh(x)
```

Remarks

If `x` is `NaN`, the result is `NaN`.

Applies To: Math Object

Requirements

Supported in Microsoft Edge (Edge browser). Also supported in Store apps (Microsoft Edge on Windows 10). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Not supported in Windows 8.1.

See Also

Math.acos Function
Math.acosh Function
Math.asin Function
Math.atan Function
Math.atan2 Function
Math.cos Function
Math.cosh Function
Math.sin Function
Math.sinh Function
Math.tanh Function

Math.atan Function (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Returns the arctangent of a number.

Syntax

```
Math.atan(number)
```

Remarks

The required `number` argument is a numeric expression for which the arctangent is needed.

The return value is the arctangent of the `number` argument, in radians.

Applies To: Math Object

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

See Also

Math.acos Function
Math.asin Function
Math.atan2 Function
Math.cos Function
Math.sin Function
Math.tan Function
Math Object

Math.atan2 Function (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Returns the angle (in radians) from the X axis to a point (*y*,*x*).

Syntax

```
Math.atan2(y, x)
```

Parameters

x
Required. A numeric expression representing the cartesian x-coordinate.

y
Required. A numeric expression representing the cartesian y-coordinate.

Remarks

The return value is between *-pi* and *pi*. It represents the angle of the supplied (*y*,*x*) point, in radians.

Applies To: Math Object

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

See Also

Math.atan Function
Math.tan Function
Math Object

Math.atanh Function (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Returns the inverse hyperbolic tangent of a number.

Syntax

```
Math.atanh(x)
```

Remarks

If `x` is `NaN`, the result is `NaN`.

Applies To: Math Object

Requirements

Supported in Microsoft Edge (Edge browser). Also supported in Store apps (Microsoft Edge on Windows 10). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Not supported in Windows 8.1.

See Also

Math.acos Function
Math.acosh Function
Math.asin Function
Math.asinh Function
Math.atan Function
Math.atan2 Function
Math.cos Function
Math.cosh Function
Math.sin Function
Math.sinh Function
Math.tanh Function

Math.cbrt Function (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Returns the cube root of a number.

Syntax

```
Math.cbrt(
number
)
```

Remarks

If `number` is `NaN`, the result will be `NaN`.

Requirements

Supported in Microsoft Edge (Edge browser). Also supported in Store apps (Microsoft Edge on Windows 10). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Not supported in Windows 8.1.

Applies To: Math Object

Math.ceil Function (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Returns the smallest integer greater than or equal to its numeric argument.

Syntax

```
Math.ceil(number)
```

Remarks

The required `number` argument is a numeric expression.

The return value is an integer value equal to the smallest integer greater than or equal to its numeric argument.

Applies To: Math Object

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

See Also

Math.floor Function
Math Object

Math.clz32 Function (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Returns the number of leading zero bits in the 32-bit binary representation of a number.

Syntax

```
Math.clz32(
number
)
```

Remarks

If `number` is 0, the result will be 32. If the most significant bit of the 32-bit binary encoding of `number` is 1, the result will be 0.

Requirements

Supported in Microsoft Edge (Edge browser). Also supported in Store apps (Microsoft Edge on Windows 10). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Not supported in Windows 8.1.

Applies To: Math Object

Math.cos Function (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Returns the cosine of a number.

Syntax

```
Math.cos(radians)
```

Remarks

The required `radians` argument is a numeric expression that contains an angle measured in radians.

The return value is the cosine of the numeric argument of `radians`.

Applies To: Math Object

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

See Also

Math.acos Function
Math.asin Function
Math.atan Function
Math.sin Function
Math.tan Function

Math.cosh Function (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Returns the hyperbolic cosine of a number.

Syntax

```
Math.cosh(x)
```

Remarks

The return value is the same as `(exp(x) + exp(-x))/2`. If `x` is `NaN`, the result is `NaN`.

Applies To: Math Object

Requirements

Supported in Microsoft Edge (Edge browser). Also supported in Store apps (Microsoft Edge on Windows 10). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Not supported in Windows 8.1.

See Also

Math.acos Function
Math.acosh Function
Math.asin Function
Math.atan Function
Math.sin Function
Math.sinh Function
Math.tan Function

Math.exp Function (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Returns e (the base of natural logarithms) raised to a power.

Syntax

```
Math.exp(number)
```

Remarks

The required `number` argument is a numeric expression representing the power of *e*.

The return value is a number. The constant *e* is Euler's number, approximately equal to 2.71828 and `number` is the supplied argument.

Applies To: Math Object

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

See Also

Math Constants

Math.expm1 Function (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Returns the result of subtracting 1 from e (the base of the natural logarithms) raised to a power).

Syntax

```
Math.expm1(number)
```

Remarks

The required `number` argument is a numeric expression representing the power of e.

The return value is a number. The constant e is Euler's number, approximately equal to 2.71828 and `number` is the supplied argument.

Applies To: Math Object

Requirements

Supported in Microsoft Edge (Edge browser). Also supported in Store apps (Microsoft Edge on Windows 10). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Not supported in Windows 8.1.

See Also

Math Constants

Math.floor Function (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Returns the greatest integer less than or equal to its numeric argument.

Syntax

```
Math.floor(number)
```

Remarks

The required *number* argument is a numeric expression.

The return value is an integer value equal to the greatest integer less than or equal to its numeric argument.

Applies To: Math Object

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

See Also

Math.ceil Function

Math.fround Function (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Returns the nearest single-precision floating-point format of a number.

Syntax

```
Math.fround(
number
)
```

Remarks

If `number` is `NaN`, the result is `NaN`.

Requirements

Supported in Microsoft Edge (Edge browser). Also supported in Store apps (Microsoft Edge on Windows 10). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Not supported in Windows 8.1.

Applies To: Math Object

See Also

Math.round Function
Math.random Function

Math.hypot Function (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Returns the square root of the sum of the squares of the arguments.

Syntax

```
Math.hypot ( value1[, value2[, ...values] );
```

Parameters

`value1`

Required. The first number.

`value2`

Optional. The second number.

`values`

Optional. One or more numbers.

Remarks

If any argument is NaN, function returns NaN. If no arguments are provide, the function returns 0.

Example

The following example shows an example of using the `Math.hypot` function.

```
Math.hypot(3, 4);
// Returns 5

Math.hypot(3, "4");
// Returns 5

Math.hypot(3, "four");
// Returns NaN

Math.hypot(3, 4, 10);
// Returns 11.180339887498949
```

Requirements

Supported in Microsoft Edge (Edge browser). Also supported in Store apps (Microsoft Edge on Windows 10). See Version Information.

Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Not supported in Windows 8.1.

Math.imul Function (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Returns the product of two numbers that are treated as 32-bit signed integers.

Syntax

```
Math.imul(x, y);
```

Parameters

`x`

Required. The first number.

`y`

Required. The second number.

Remarks

This function is used for compilers like Emscripten and Mandreel, which don't implement integer multiplication in the same way as JavaScript.

Example

The following code example shows how to multiply numbers using `Math.imul`.

```
var result1 = Math.imul(2, 5);
// result1 == 10

var result2 = Math.imul(Math.pow(2, 32) - 1, Math.pow(2, 32) - 2);
// result2 == 2
```

Requirements

Supported in Microsoft Edge (Edge browser). Also supported in Store apps (Microsoft Edge on Windows 10). See Version Information.

Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Not supported in Windows 8.1.

Math.log Function (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Returns the natural logarithm (base `e`) of a number.

Syntax

```
Math.log(number)
```

Parameters

number

A number.

Return Value

If `number` is positive, this function returns the natural logarithm of the number. If `number` is negative, this function returns `NaN`. If `number` is 0, this function returns `-Infinity`.

Example

The following code shows how to use this function.

```
var numArr = [ 45.3, 69.0, 557.04, 0.222 ];

for (i in numArr) {
    document.write(Math.log(numArr[i]));
    document.write("<br/>");
}

// Output:
// 3.8133070324889884
// 4.23410650459726
// 6.322637050634291
// -1.5050778971098575
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

Applies To: Math Object

See Also

Math.sqrt Function

Math.log1p Function (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Returns the natural logarithm of 1 + a number.

Syntax

```
Math.log1p(x)
```

Return Value

If `x` is `NaN`, this function returns `NaN`. If `x` is less than zero (0), this function returns `NaN`.

Requirements

Supported in Microsoft Edge (Edge browser). Also supported in Store apps (Microsoft Edge on Windows 10). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Not supported in Windows 8.1.

Applies To: Math Object

See Also

Math.log Function
Math.log10 Function
Math.log2 Function
Math.sqrt Function

Math.log10 Function (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Returns the base 10 logarithm of a number.

Syntax

```
Math.log10(x)
```

Return Value

If `x` is `NaN`, this function returns `NaN`. If `x` is less than zero (0), this function returns `NaN`.

Requirements

Supported in Microsoft Edge (Edge browser). Also supported in Store apps (Microsoft Edge on Windows 10). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Not supported in Windows 8.1.

Applies To: Math Object

See Also

Math.log Function
Math.sqrt Function

Math.log2 Function (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Returns the base 2 logarithm of a number.

Syntax

```
Math.log2(x)
```

Return Value

If `x` is `NaN`, this function returns `NaN`. If `x` is less than zero (0), this function returns `NaN`.

Requirements

Supported in Microsoft Edge (Edge browser). Also supported in Store apps (Microsoft Edge on Windows 10). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Not supported in Windows 8.1.

Applies To: Math Object

See Also

Math.log Function
Math.log10 Function
Math.sqrt Function

Math.max Function (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Returns the larger of a set of supplied numeric expressions.

Syntax

```
Math.max([number1[, number2[... [, numberN]]]])
```

Remarks

The optional `number1, number2, ..., numberN` arguments are numeric expressions to be evaluated.

If no arguments are provided, the return value is equal to Number.NEGATIVE_INFINITY. If any argument is `NaN`, the return value is also `NaN`.

Example

The following code shows how to get the larger of two expressions.

```
var x = Math.max(107 - 3,  48 * 90);
document.write(x);

// Output:
// 4320
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

See Also

Math.min Function

Math.min Function (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Returns the smaller of a set of numeric expressions.

Syntax

```
Math.min([number1[, number2[... [,numberN]]]])
```

Remarks

The optional `number1, number2, ..., numberN` arguments are numeric expressions to be evaluated.

If no arguments are provided, the return value is equal to Number.POSITIVE_INFINITY. If any argument is `NaN`, the return value is also `NaN`.

Example

The following code shows how to get the smaller of two expressions.

```
var x = Math.min(107 - 3, 48 * 90);
document.write(x);

// Output:
// 104
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

See Also

Math.max Function

Math.pow Function (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Returns the value of a base expression taken to a specified power.

Syntax

```
Math.pow(
base, exponent)
```

Parameters

base
Required. The base value of the expression.

exponent
Required. The exponent value of the expression.

Example

In the following example, a numeric expression equal to baseexponent returns 1000.

```
Math.pow(10,3);
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

Applies To: Math Object

See Also

Math Object

Math.random Function (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Returns a pseudorandom number between 0 and 1.

Syntax

```
Math.random( )
```

Remarks

The pseudorandom number generated is from 0 (inclusive) to 1 (exclusive), that is, the returned number can be zero, but it will always be less than one. The random number generator is seeded automatically when JavaScript is first loaded.

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

Applies To: Math Object

See Also

Math.pow Function

Math.round Function (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Returns a supplied numeric expression rounded to the nearest integer.

Syntax

```
Math.round(
number
)
```

Remarks

The required `number` argument is the value to be rounded to the nearest integer.

For positive numbers, if the decimal portion of `number` is 0.5 or greater, the return value is equal to the smallest integer greater than `number`. If the decimal portion is less than 0.5, the return value is the largest integer less than or equal to `number`.

For negative numbers, if the decimal portion is exactly -0.5, the return value is the smallest integer that is greater than the number.

For example, `Math.round(8.5)` returns 9, but `Math.round(-8.5)` returns -8.

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

Applies To: Math Object

See Also

Math.random Function

Math.sign Function (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Returns the sign of a number, indicating whether the number is positive, negative, or 0.

Syntax

```
Math.sign(number)
```

Remarks

The required `number` argument is a numeric expression for which the sign is needed.

The return value is one of the following:

- `NaN`, if `number` is `NaN`.
- -0, if `number` is -0.
- +0, if `number` is +0.
- -1, if `number` is negative and not -0.
- +1, if `number` is positive and not +0.

Requirements

Supported in Microsoft Edge (Edge browser). Also supported in Store apps (Microsoft Edge on Windows 10). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Not supported in Windows 8.1.

Math.sin Function (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Returns the sine of a number.

Syntax

```
Math.sin(radians)
```

Remarks

The `radians` argument is a numeric expression that contains an angle measured in radians.

The return value is the sine of the numeric argument.

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

Applies To: Math Object

See Also

Math.acos Function
Math.asin Function
Math.atan Function
Math.cos Function
Math.tan Function

Math.sinh Function (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Returns the inverse hyperbolic sine of a number.

Syntax

```
Math.sinh(x)
```

Remarks

The return value is the same as `(exp(x) - exp(-x))/2`. If x is `NaN`, the result is `NaN`.

Requirements

Supported in Microsoft Edge (Edge browser). Also supported in Store apps (Microsoft Edge on Windows 10). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Not supported in Windows 8.1.

Applies To: Math Object

See Also

Math.acos Function
Math.acosh Function
Math.asin Function
Math.atan Function
Math.cos Function
Math.tan Function

Math.sqrt Function (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Returns the square root of a number.

Syntax

```
Math.sqrt(
number
)
```

Remarks

The required `number` argument is a numeric expression.

If `number` is negative, the return value is `NaN`.

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

Applies To: Math Object

See Also

Math Constants

Math.tan Function (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Returns the tangent of a number.

Syntax

```
Math.tan(
radians
)
```

Remarks

The required `radians` argument is a numeric expression that contains an angle measured in radians.

The return value is the tangent of the numeric argument of `radians`.

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

Applies To: Math Object

See Also

Math.acos Function
Math.asin Function
Math.atan Function
Math.atan2 Function
Math.cos Function
Math.sin Function

Math.tanh Function (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Returns the hyperbolic tangent of a number.

Syntax

```
Math.tanh(
x
)
```

Remarks

The return value is the same as `(exp(x) - exp(-x))/(exp(x) + exp(-x))`. If `x` is `NaN`, the result is `NaN`.

Requirements

Supported in Microsoft Edge (Edge browser). Also supported in Store apps (Microsoft Edge on Windows 10). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Not supported in Windows 8.1.

Applies To: Math Object

See Also

Math.acos Function
Math.acosh Function
Math.asin Function
Math.atan Function
Math.atan2 Function
Math.cos Function
Math.cosh Function
Math.sin Function
Math.sinh Function

Math.trunc Function (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Returns the integer portion of a number, removing any fractional digits.

Syntax

```
Math.trunc(number)
```

Remarks

The required `number` argument is a numeric expression for which the truncated integer is needed.

If `number` is `NaN`, returns `NaN`.

Requirements

Supported in Microsoft Edge (Edge browser). Also supported in Store apps (Microsoft Edge on Windows 10). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Not supported in Windows 8.1.

Number Object (JavaScript)

10/18/2017 • 2 min to read • Edit Online

An object that represents a number of any kind. All JavaScript numbers are 64-bit floating-point numbers.

Syntax

```
numObj = new Number(value)
```

Parameters

`numObj`
Required. The variable name to which the `Number` object is assigned.

`value`
Required. The numeric value.

Remarks

JavaScript creates `Number` objects when a variable is set to a number value, for example `var num = 255.336;`. It is seldom necessary to create `Number` objects explicitly.

The `Number` object has its own properties and methods, in addition to the properties and methods inherited from `Object`. Numbers are converted into strings under certain circumstances, for example when a number is added or concatenated with a string, as well as by means of the `toString` method. For more information, see Addition Operator (+).

JavaScript has several number constants. For a complete list, see Number Constants.

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

Properties

The following table lists the properties of the `Number` object.

PROPERTY	DESCRIPTION
constructor Property	Specifies the function that creates an object.
prototype Property	Returns a reference to the prototype for a class of objects.

Functions

The following table lists the functions of the `Number` object.

FUNCTION	DESCRIPTION
Number.isFinite Function	Returns a Boolean value that indicates whether a value is finite.
Number.isInteger Function	Returns a Boolean value that indicates whether a value is an integer.
Number.isNaN Function	Returns a Boolean value that indicates whether a value is the reserved value `NaN` (not a number).
Number.isSafeInteger	Returns a Boolean value that indicates whether a value can be safely represented in JavaScript.

Methods

The following table lists the methods of the `Number` object.

METHOD	DESCRIPTION
hasOwnProperty Method	Returns a Boolean value that indicates whether an object has a property with the specified name.
isPrototypeOf Method	Returns a Boolean value that indicates whether an object exists in another object's prototype hierarchy.
propertyIsEnumerable Method	Returns a Boolean value that indicates whether a specified property is part of an object and whether it is enumerable.
toExponential method	Returns a string that contains a number represented in exponential notation.
toFixed method	Returns a string that represents a number in fixed-point notation.
toLocaleString Method	Returns an object converted to a string based on the current locale.
toPrecision method	Returns a string that contains a number that is represented in either exponential or fixed-point notation and that has a specified number of digits.
toString Method	Returns a string representation of an object.
valueOf Method	Returns the primitive value of the specified object.

See Also

JavaScript Objects
Math Object
new Operator

Number Constants (JavaScript)

10/18/2017 • 1 min to read • Edit Online

The following number constants are properties of the `Number` object.

Number Object Constants

You do not have to create `Number` object to access these constants.

CONSTANT	VALUE RETURNED
`Number.EPSILON`	The smallest number that can be represented in JavaScript. Equal to approximately 2.2204460492503130808472633361816E-16.
`Number.MAX_SAFE_INTEGER`	The largest number that can be safely represented in JavaScript. Equal to 9007199254740991.
`Number.MAX_VALUE`	The largest number that can be represented in JavaScript. Equal to approximately 1.79E+308.
`Number.MIN_SAFE_INTEGER`	The smallest number that can be safely represented in JavaScript. Equal to -9007199254740991.
`Number.MIN_VALUE`	The closest number to zero that can be represented in JavaScript. Equal to approximately 5.00E-324.
`Number.NaN`	A value that is not a number. In equality comparisons, `NaN` does not equal any value, including itself. To test whether a value is equivalent to `NaN`, use the isNaN function.
`Number.NEGATIVE_INFINITY`	A value that is less than the largest negative number that can be represented in JavaScript. JavaScript displays `NEGATIVE_INFINITY` values as `-infinity`.
`Number.POSITIVE_INFINITY`	A value greater than the largest number that can be represented in JavaScript. JavaScript displays `POSITIVE_INFINITY` values as `infinity`.

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

For `Number.EPSILON`, `Number.MAX_SAFE_INTEGER`, and `Number.MIN_SAFE_INTEGER`:

Supported in Microsoft Edge (Edge browser). Also supported in Store apps (Microsoft Edge on Windows 10). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Not supported in Windows 8.1.

Applies To: Number Object

See Also

Math Constants
JavaScript Constants
Infinity Constant
NaN Constant
isNaN Function

Number.isFinite Function (Number) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Returns a Boolean value that indicates whether a value is a finite number.

Syntax

```
Number.isFinite(numValue)
```

Return Value

`false` if the value is `NaN`, `+∞`, or `-∞`; otherwise `true`.

Remarks

Example

```
// Returns true
Number.isFinite(100)
Number.isFinite(-100)
Number.isFinite(100 / 3)

// Returns false
Number.isFinite(Number.NaN)
Number.isFinite(Infinity)
Number.isFinite("100")
```

Requirements

Supported in Microsoft Edge (Edge browser). Also supported in Store apps (Microsoft Edge on Windows 10). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Not supported in Windows 8.1.

Applies To: Number Object

Number.isInteger Function (Number) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Returns a Boolean value that indicates whether a value is an integer.

Syntax

```
Number.isInteger(numValue)
```

Return Value

`true` if the value is an integer, otherwise `false`.

Remarks

Requirements

Supported in Microsoft Edge (Edge browser). Also supported in Store apps (Microsoft Edge on Windows 10). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Not supported in Windows 8.1.

Applies To: Number Object

Example

```
// Returns true
Number.isInteger(100)
Number.isInteger(-100)

// Returns false
Number.isInteger(Number.NaN)
Number.isInteger(Infinity)
Number.isInteger(100 / 3)
Number.isInteger("100")
```

Number.isNaN Function (Number) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Returns a Boolean value that indicates whether a value is the reserved value `NaN` (not a number).

Syntax

```
Number.isNaN(numValue)
```

Parameters

`numValue`

Required. The value to be tested against `NaN`.

Return Value

`true` if the value converted to the `Number` type is the `NaN`, otherwise `false`.

Remarks

You typically use this method to test return values from the `parseInt` and `parseFloat` methods.

`Number.isNaN` corrects problems with the global `isNaN` function. `Number.isNaN` only converts its argument to the type Number after comparing it with `NaN`. As a result, it returns `true` if and only if the argument passed in is exactly the same value as `NaN`. The global `isNaN` function converts its argument to the type Number before the comparison, which can lead to non-number values returning `true`, whereas they might not return `true` for `Number.isNaN`.

Requirements

Supported in Microsoft Edge (Edge browser). Also supported in Store apps (Microsoft Edge on Windows 10). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Not supported in Windows 8.1.

Applies To: Number Object

Example

```
// Returns true.
Number.isNaN(NaN);
Number.isNaN(Number.NaN);
Number.isNaN(0 / 0);

// Returns false.
Number.isNaN(100);
// Returns false.
// Strings are converted to numbers and return false.
Number.isNaN("100");
Number.isNaN("ABC");
Number.isNaN("10C");
Number.isNaN("abc123");
```

Number.isSafeInteger (Number) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Returns a Boolean value that indicates whether a number can be safely represented in JavaScript.

Syntax

```
Number.isSafeInteger(numValue)
```

Return Value

`true` if the number is between `Number.MIN_SAFE_INTEGER` and `Number.MAX_SAFE_INTEGER`, inclusive; otherwise `false`.

Remarks

A safe integer in JavaScript is one that is an IEEE-754 double precisions number before any rounding has been applied.

Example

```
// Returns true
Number.isSafeInteger(-100)
Number.isSafeInteger(9007199254740991)

// Returns false
Number.isSafeInteger(Number.NaN)
Number.isSafeInteger(Infinity)
Number.isSafeInteger("100")
Number.isSafeInteger(9007199254740992);
```

Requirements

Supported in Microsoft Edge (Edge browser). Also supported in Store apps (Microsoft Edge on Windows 10). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Not supported in Windows 8.1.

Applies To: Number Object

constructor Property (Number)

10/18/2017 • 1 min to read • Edit Online

Specifies the function that creates a Number.

Syntax

```
number.constructor
```

Remarks

The required `number` is the name of a string.

The `constructor` property is a member of the prototype of every object that has a prototype. This includes all intrinsic JavaScript objects except the `Global` and `Math` objects. The `constructor` property contains a reference to the function that constructs instances of that particular object.

Example

The following example illustrates the use of the constructor property.

```
var num = new Number();

if (num.constructor == Number)
    document.write("Object is a Number.");
else
    document.write("Object is not a Number.");

// Output:
// Object is a Number.
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

prototype Property (Number)

10/18/2017 • 1 min to read • Edit Online

Returns a reference to the prototype for a class of number.

Syntax

```
number.prototype
```

Remarks

The `number` argument is the name of a number.

Use the `prototype` property to provide a base set of functionality to a class of objects. New instances of an object "inherit" the behavior of the prototype assigned to that object.

For example, to add a method to the `Number` object that returns the number of (integer) digits, declare the function, add it to `Number.prototype`, and then use it.

```
function number_digits() {
    var digits = 0;
    var num = this;
    while (num) >= 1) {
        digits++;
        num /= 10;
    }
    return digits;
}

Number.prototype.digits = number_digits;
var myNumber = new Number(3456.789);
document.write(myNumber.digits());
// Output:
// 4
```

All intrinsic JavaScript objects have a `prototype` property that is read-only. Properties and methods may be added to the prototype, but the object may not be assigned a different prototype. However, user-defined objects may be assigned a new prototype.

The method and property lists for each intrinsic object in this language reference indicate which ones are part of the object's prototype, and which are not.

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

toExponential Method (Number) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Represents a number in exponential notation.

Syntax

```
numObj. toExponential([fractionDigits])
```

Parameters

`numObj`
Required. A **Number** object.

`fractionDigits`
Optional. The number of digits after the decimal point. Must be in the range 0 - 20, inclusive.

Return Value

Returns a string representation of a number in exponential notation. The string contains one digit before the decimal point, and may contain `fractionDigits` digits after it.

If `fractionDigits` is not supplied, the `toExponential` method returns as many digits necessary to uniquely specify the number.

Example

```
var num = new Number(123);
var exp = num.toExponential();
document.write(exp);
document.write("<br/>");

num = new Number(123.456);
exp = num.toExponential(5);
document.write(exp);

// Output:
// 1.23e+2
// 1.23456e+2
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

Applies To: Number Object

See Also

toFixed Method (Number)
toPrecision Method (Number)

toFixed Method (Number) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Represents a number in fixed-point notation.

Syntax

```
numObj.toFixed([fractionDigits])
```

Parameters

`numObj`
Required A `Number` object.

`fractionDigits`
Optional. The number of digits after the decimal point. Must be in the range 0 - 20, inclusive.

Return Value

Returns a string representation of a number in fixed-point notation, containing `fractionDigits` digits after the decimal point.

If `fractionDigits` is not supplied or **undefined**, the default value is zero.

Example

The following code shows how to use `toFixed`.

```
var num = new Number(123);
var fix = num.toFixed();
document.write(fix);
document.write("<br/>");

num = new Number(123.456);
fix = num.toFixed(5);
document.write(fix);

// Output:
// 123
123.45600
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

Applies To: Number Object

See Also

toExponential Method (Number)
toPrecision Method (Number)

toLocaleString (Number)

10/18/2017 • 1 min to read • Edit Online

Converts a number to a string by using the current or specified locale.

Syntax

```
numberObj.toLocaleString([locales][, options])
```

Parameters

`numberObj`

Required. The `Number` object to convert.

`locales`

Optional. An array of locale strings that contain one or more language or locale tags. If you include more than one locale string, list them in descending order of priority so that the first entry is the preferred locale. If you omit this parameter, the default locale of the JavaScript runtime is used.

`options`

Optional. An object that contains one or more properties that specify comparison options.

Remarks

Starting in Internet Explorer 11, `toLocaleString` uses `Intl.NumberFormat` internally to make comparisons, which adds support for the `locales` and `options` parameters. For more information about these parameters, see Intl.NumberFormat.

> **IMPORTANT**
>
> The `locales` and `options` parameters are not supported in all document modes and browser versions. For more information, see the Requirements section.

> **NOTE**
>
> If you omit the `locales` parameter, use `toLocaleString` only to display results to a user; never use it to compute values within a script, because the returned result is machine-specific (the method returns the current locale).

Example

The following example shows how to use the `toLocaleString` method with no parameters.

```
var n, s;
n = new Number(100);
s = "Current locale value is: ";
s += n.toLocaleString();
document.write(s);

// Output:
// The value 100 as represented by the current locale.
```

Example

The following example shows how to use the `toLocaleString` method with a specified locale and comparison options.

```
var number = 123456789;
var options1 = { style: "percent" };
var options2 = { style: "currency", currency: "INR" };

document.write(number.toLocaleString("en-US"));
// 123,456,789
document.write(number.toLocaleString("ja-JP"));
// 123,456,789
document.write(number.toLocaleString("ar-SA", options1));
// ١٢,٣٤٥,٦٧٨,٩٠٠ %
document.write(number.toLocaleString("hi-IN", options2));
// ₹ 12,34,56,789.00
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

`locales` and `options` parameters:

Supported in the Internet Explorer 11 standards document mode. Also supported in Store apps (Windows 8.1 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards. Not supported in Windows 8.

See Also

toLocaleDateString Method (Date)

toPrecision Method (Number) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Represents a number either in exponential or fixed-point notation with a specified number of digits.

Syntax

```
numObj.toPrecision([precision])
```

Parameters

`numObj`
Required. A `Number` object.

`precision`
Optional. The number of significant digits. Must be in the range 1 - 21, inclusive.

Return Value

For numbers in exponential notation, `precision` - 1 digits are returned after the decimal point. For numbers in fixed notation, `precision` significant digits are returned.

If `precision` is not supplied or is **undefined**, the **toString** method is called instead.

Example

The following code shows how to use `toPrecision`.

```
var num = new Number(123);
var prec = num.toPrecision();
document.write(prec);
document.write("<br/>");

num = new Number(123.456);
prec = num.toPrecision(5);
document.write(prec);

// Output:
// 123
// 123.46
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

Applies To: Number Object

See Also

toFixed Method (Number)
toExponential Method (Number)

toString Method (Number)

10/18/2017 • 1 min to read • Edit Online

Returns a string representation of a number.

Syntax

```
number.toString()
```

Parameters

`number`

Required. The number to represent as a string.

Return Value

The string representation of the number.

Example

The following example illustrates the use of the **toString** method with a number.

```
var number = 234.567;
var s = number.toString();
document.write(s.length);

// Output: 7
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

valueOf Method (Number)

10/18/2017 • 1 min to read • Edit Online

Returns the primitive value of the specified number.

Syntax

```
number.valueOf()
```

Parameters

This method has no parameters.

Return Value

Returns the number.

Remarks

In the following example, the instantiated number object is the same as the return value of this method.

```
var num = 1234;
var s = num.valueOf();

if (num === s)
document.write("same");
else
document.write("different");

// Output:
// same
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

Object Object (JavaScript)

10/18/2017 • 2 min to read • Edit Online

Provides functionality common to all JavaScript objects.

Syntax

```
obj
 = new Object([value])
```

Parameters

`obj`
Required. The variable name to which the `Object` object is assigned.

value
Optional. Any one of the JavaScript primitive data types (Number, Boolean, or String). If value is an object, the object is returned unmodified. If *value* is `null`, **undefined**, or not supplied, an object with no content is created.

Remarks

The `Object` object is contained in all other JavaScript objects; all of its methods and properties are available in all other objects. The methods can be redefined in user-defined objects, and are called by JavaScript at appropriate times. The **toString** method is an example of a frequently redefined `Object` method.

In this language reference, the description of each `Object` method includes both default and object-specific implementation information for the intrinsic JavaScript objects.

Requirements

The `Object Object` was introduced in Internet Explorer before Internet Explorer 6. Some members in the following lists were introduced in later versions.

Properties

The following table lists properties of the `Object Object`.

PROPERTY	DESCRIPTION
*_proto_* Property	Specifies the prototype for an object.
constructor Property	Specifies the function that creates an object.
prototype Property	Returns a reference to the prototype for a class of objects.

Functions

The following table lists functions of the `Object Object`.

FUNCTION	DESCRIPTION
Object.assign Function	Copies the values from one or more source objects to a target object.
Object.create Function	Creates an object that has a specified prototype, and that optionally contains specified properties.
Object.defineProperties Function	Adds one or more properties to an object, and/or modifies attributes of existing properties.
Object.defineProperty Function	Adds a property to an object, or modifies attributes of an existing property.
Object.freeze Function	Prevents the modification of existing property attributes and values, and prevents the addition of new properties.
Object.getOwnPropertyDescriptor Function	Returns the definition of a data property or an accessor property.
Object.getOwnPropertyNames Function	Returns the names of the properties and methods of an object.
Object.getOwnPropertySymbols Function	Returns the symbol properties of an object.
Object.getPrototypeOf Function	Returns the prototype of an object.
Object.is Function	Returns a value that indicates whether two values are the same value.
Object.isExtensible Function	Returns a value that indicates whether new properties can be added to an object.
Object.isFrozen Function	Returns `true` if existing property attributes and values cannot be modified in an object and new properties cannot be added to the object.
Object.isSealed Function	Returns `true` if existing property attributes cannot be modified in an object and new properties cannot be added to the object.
Object.keys Function	Returns the names of the enumerable properties and methods of an object.
Object.preventExtensions Function	Prevents the addition of new properties to an object.
Object.seal Function	Prevents the modification of attributes of existing properties, and prevents the addition of new properties.
Object.setPrototypeOf Function	Sets the prototype of an object.

Methods

The following table lists methods of the `Object Object`.

METHOD	DESCRIPTION
hasOwnProperty method	Returns a Boolean value that indicates whether an object has a property with the specified name.
isPrototypeOf method	Returns a Boolean value that indicates whether an object exists in another object's prototype hierarchy.
propertyIsEnumerable method	Returns a Boolean value that indicates whether a specified property is part of an object and whether it is enumerable.
toLocaleString method	Returns an object converted to a string based on the current locale.
toString method	Returns a string representation of an object.
valueOf method	Returns the primitive value of the specified object.

See Also

JavaScript Objects

proto Property (Object) (JavaScript)

3/15/2018 • 1 min to read • Edit Online

Contains a reference to the internal prototype of the specified object.

WARNING

The `__proto__` property is a legacy feature. Use Object.getPrototypeOf instead.

Syntax

```
object.__proto__
```

Parameters

`object`

Required. The object on which to set the prototype.

Remarks

The `__proto__` property can be used to set the prototype for an object.

The object or function inherits all methods and properties of the new prototype, along with all methods and properties in the new prototype's prototype chain. An object can have only a single prototype (not including inherited prototypes in the prototype chain), so when you call the `__proto__` property, you replace the previous prototype.

You can set the prototype only on an extensible object. For more info, see Object.preventExtensions Function.

NOTE

The `__proto__` property name begins and ends with two underscores.

Example

The following code example shows how to set the prototype for an object.

```
function Rectangle() {
}

var rec = new Rectangle();

if (console && console.log) {
    console.log(rec.__proto__ === Rectangle.prototype);  // Returns true
    rec.__proto__ = Object.prototype;
    console.log(rec.__proto__ === Rectangle.prototype);  // Returns false
}
```

Example

The following code example shows how to add properties to an object by adding them to the prototype.

```
var proto = { y: 2 };

var obj = { x: 10 };
obj.__proto__ = proto;

proto.y = 20;
proto.z = 40;

if (console && console.log) {
    console.log(obj.x === 10);  // Returns true
    console.log(obj.y === 20);  // Returns true
    console.log(obj.z === 40);  // Returns true
}
```

Example

The following code example adds properties to the `String` object by setting a new prototype on it.

```
var stringProp = { desc: "description" };

String.__proto__ = stringProp;
var s1 = "333";
var s2 = new String("333");

if (console && console.log) {

    console.log(String.desc === "description"); // Returns true
    console.log(s1.desc === "description");     // Returns false
    console.log(s2.desc === "description");     // Returns false

    s1.__proto__ = String;  // Can't be set.
    s2.__proto__ = String;

    console.log(s1.desc === "description"); // Returns false
    console.log(s2.desc === "description"); // Returns true
}
```

Requirements

Supported in the Internet Explorer 11 standards document mode. Also supported in Store apps (Windows 8.1 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards. Not supported in Windows 8.

See Also

Prototypes and Prototype Inheritance

constructor Property (Object) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Specifies the function that creates an object.

Syntax

```
object.constructor
```

Remarks

The required `object` is the name of an object or function.

The `constructor` property is a member of the prototype of every object that has a prototype. This includes all intrinsic JavaScript objects except the `Global` and `Math` objects. The `constructor` property contains a reference to the function that constructs instances of that particular object.

Example

The following example illustrates the use of the constructor property.

```
// A constructor function.
function MyObj() {
    this.number = 1;
}

var x = new String("Hi");

if (x.constructor == String)
    document.write("Object is a String.");
document.write ("<br />");

var y = new MyObj;
if (y.constructor == MyObj)
    document.write("Object constructor is MyObj.");

// Output:
// Object is a String.
// Object constructor is MyObj.
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

See Also

prototype Property (Object)

prototype Property (Object) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Returns a reference to the prototype for a class of objects.

Syntax

```
objectName.prototype
```

Remarks

The `objectName` argument is the name of an object.

Use the `prototype` property to provide a base set of functionality to a class of objects. New instances of an object "inherit" the behavior of the prototype assigned to that object.

For example, to add a method to the `Array` object that returns the value of the largest element of the array, declare the function, add it to `Array.prototype`, and then use it.

```
function array_max( ){
    var i, max = this[0];
    for (i = 1; i < this.length; i++)
    {
    if (max < this[i])
    max = this[i];
    }
    return max;
}
Array.prototype.max = array_max;
var myArray = new Array(7, 1, 3, 11, 25, 9
);
document.write(myArray.max());

// Output:
// 25
```

All intrinsic JavaScript objects have a `prototype` property that is read-only. Properties and methods may be added to the prototype, but the object may not be assigned a different prototype. However, user-defined objects may be assigned a new prototype.

The method and property lists for each intrinsic object in this language reference indicate which ones are part of the object's prototype, and which are not.

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

See Also

constructor Property (Object)

Object.assign Function (Object) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Copies the values from one or more source objects to a target object.

Syntax

```
Object.assign(target, ...sources );
```

Parameters

`target`

Required. A object to which enumerable properties are copied.

`...sources`

Required. One or more objects from which enumerable properties are copied.

Exceptions

This function throws a `TypeError` if there is an assignment error, which terminates the copying operation. A `TypeError` will be thrown if a target property is not writable.

Remarks

This function returns the target object. Only *enumerable own* properties are copied from the source object to the target object. You can use this function to merge or clone objects.

`null` or `undefined` sources are treated like empty objects and contribute nothing to the target object.

Example

The following code example shows how to merge an object using `Object.assign`.

```
var first = { name: "Bob" };
var last = { lastName: "Smith" };

var person = Object.assign(first, last);
console.log(person);

// Output:
// { name: "Bob", lastName: "Smith" }
```

Example

The following example shows how to clone an object using `Object.assign`.

```
var obj = { person: "Bob Smith"};
var clone = Object.assign({}, obj);
```

Requirements

Supported in Microsoft Edge (Edge browser). Also supported in Store apps (Microsoft Edge on Windows 10). See Version Information.

Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Not supported in Windows 8.1.

Object.create Function (JavaScript)

10/18/2017 • 2 min to read • Edit Online

Creates an object that has the specified prototype, and that optionally contains specified properties.

Syntax

```
Object.create(prototype, descriptors)
```

Parameters

`prototype`

Required. The object to use as a prototype. May be `null`.

`descriptors`

Optional. A JavaScript object that contains one or more property descriptors.

A *data property* is a property that can get and set a value. A data property descriptor contains a `value` attribute, plus `writable`, `enumerable`, and `configurable` attributes. If the last three attributes are not specified, they default to `false`. An *accessor property* calls a user-provided function every time the value is retrieved or set. An accessor property descriptor contains a `set` attribute, a `get` attribute, or both. For more information, see Object.defineProperty Function.

Return Value

A new object that has the specified internal prototype and contains the specified properties, if any.

Exceptions

A `TypeError` exception is thrown if any of the following conditions is true:

- The `prototype` argument is not an object and is not `null`.
- A descriptor in the `descriptors` argument has a `value` or `writable` attribute, and has a `get` or `set` attribute.
- A descriptor in the `descriptors` argument has a `get` or `set` attribute that is not a function.

Remarks

You can use this function using a `null``prototype` parameter in order to stop the prototype chain. The object created will have no prototype.

Example

The following example creates an object using a `null` prototype and adds two enumerable properties.

```
var newObj = Object.create(null, {
            size: {
                value: "large",
                enumerable: true
            },
            shape: {
                value: "round",
                enumerable: true
            }
        });

document.write(newObj.size + "<br/>");
document.write(newObj.shape + "<br/>");
document.write(Object.getPrototypeOf(newObj));

// Output:
// large
// round
// null
```

Example

The following example creates an object that has the same internal prototype as the Object object. You can see that it has the same prototype as an object created by using an object literal. The Object.getPrototypeOf function gets the prototype of the original object. To get the object's property descriptor, you can use Object.getOwnPropertyDescriptor Function.

```
var firstLine = { x: undefined, y: undefined };

var secondLine = Object.create(Object.prototype, {
        x: {
                value: undefined,
                writable: true,
                configurable: true,
                enumerable: true
            },
            y: {
                value: undefined,
                writable: true,
                configurable: true,
                enumerable: true
            }
});

document.write("first line prototype = " + Object.getPrototypeOf(firstLine));
document.write("<br/>");
document.write("second line prototype = " + Object.getPrototypeOf(secondLine));

// Output:
// first line prototype = [object Object]
// second line prototype = [object Object]
```

Example

The following example creates an object that has the same internal prototype as the Shape object.

```
// Create the shape object.
var Shape = { twoDimensional: true, color: undefined, hasLineSegments: undefined };

var Square = Object.create(Object.getPrototypeOf(Shape));
```

Requirements

Supported in the following document modes: Internet Explorer 9 standards, Internet Explorer 10 standards, and Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards.

See Also

Object.getPrototypeOf Function
isPrototypeOf Method (Object)
Creating Objects

Object.defineProperties Function (JavaScript)

10/18/2017 • 2 min to read • Edit Online

Adds one or more properties to an object, and/or modifies attributes of existing properties.

Syntax

```
object.defineProperties(object, descriptors)
```

Parameters

`object`
Required. The object on which to add or modify the properties. This can be a native JavaScript object or a DOM object.

`descriptors`
Required. A JavaScript object that contains one or more descriptor objects. Each descriptor object describes a data property or an accessor property.

Return Value

The object that was passed to the function.

Remarks

The `descriptors` argument is an object that contains one or more descriptor objects.

A *data property* is a property that can store and retrieve a value. A data property descriptor contains a `value` attribute, a `writable` attribute, or both. For more information, see Data Properties and Accessor Properties.

An *accessor property* calls a user-provided function every time the property value is set or retrieved. An accessor property descriptor contains a `set` attribute, a `get` attribute, or both.

If the object already has a property that has the specified name, the property attributes are modified. For more information, see Object.defineProperty Function.

To create an object and add properties to the new object, you can use the Object.create Function.

Adding Properties

In the following example, the `Object.defineProperties` function adds a data property and an accessor property to a user-defined object.

The example uses an object literal to create the `descriptors` object with the `newDataProperty` and `newAccessorProperty` descriptor objects.

```
var newLine = "<br />";

var obj = {};
Object.defineProperties(obj, {
    newDataProperty: {
        value: 101,
        writable: true,
        enumerable: true,
        configurable: true
    },
    newAccessorProperty: {
        set: function (x) {
            document.write("in property set accessor" + newLine);
            this.newaccpropvalue = x;
        },
        get: function () {
            document.write("in property get accessor" + newLine);
            return this.newaccpropvalue;
        },
        enumerable: true,
        configurable: true
    }});

// Set the accessor property value.
obj.newAccessorProperty = 10;
document.write ("newAccessorProperty value: " + obj.newAccessorProperty + newLine);

// Output:
// in property set accessor
// in property get accessor
// newAccessorProperty value: 10
```

Like the earlier example, the following example adds properties dynamically instead of with an object literal.

```
var newLine = "<br />";

// Create the descriptors object.
var descriptors = new Object();

// Add a data property descriptor to the descriptors object.
descriptors.newDataProperty = new Object();
descriptors.newDataProperty.value = 101;
descriptors.newDataProperty.writable = true;
descriptors.newDataProperty.enumerable = true;
descriptors.newDataProperty.configurable = true;

// Add an accessor property descriptor to the descriptors object.
descriptors.newAccessorProperty = new Object();
descriptors.newAccessorProperty.set = function (x) {
    document.write("in property set accessor" + newLine);
    this.newaccpropvalue = x;
};
descriptors.newAccessorProperty.get = function () {
    document.write("in property get accessor" + newLine);
    return this.newaccpropvalue;
};
descriptors.newAccessorProperty.enumerable = true;
descriptors.newAccessorProperty.configurable = true;

// Call the Object.defineProperties function.
var obj = new Object();
Object.defineProperties(obj, descriptors);

// Set the accessor property value.
obj.newAccessorProperty = 10;
document.write ("newAccessorProperty value: " + obj.newAccessorProperty + newLine);

// Output:
// in property set accessor
// in property get accessor
// newAccessorProperty value: 10
```

Modifying Properties

To modify property attributes for the object, add the following code. The `Object.defineProperties` function modifies the `writable` attribute of `newDataProperty`, and modifies the `enumerable` attribute of `newAccessorProperty`. It adds `anotherDataProperty` to the object because that property name does not already exist.

```
Object.defineProperties(obj, {
    newDataProperty: { writable: false },
    newAccessorProperty: { enumerable: false },
    anotherDataProperty: { value: "abc" }
});
```

Requirements

Supported in Internet Explorer 9 standards, Internet Explorer 10 standards, and Windows 8.x Store apps.
Supported in Internet Explorer 8 for DOM objects only, otherwise not supported.

See Also

Object.getOwnPropertyDescriptor Function

Object.getOwnPropertyNames Function
Object.defineProperty Function
Object.create Function
Object Object

Object.defineProperty Function (JavaScript)

10/18/2017 • 4 min to read • Edit Online

Adds a property to an object, or modifies attributes of an existing property.

Syntax

```
Object.defineProperty(object, propertyname, descriptor)
```

Parameters

`object`
Required. The object on which to add or modify the property. This can be a native JavaScript object (that is, a user-defined object or a built in object) or a DOM object.

`propertyname`
Required. A string that contains the property name.

`descriptor`
Required. A descriptor for the property. It can be for a data property or an accessor property.

Return Value

The modified object.

Remarks

You can use the `Object.defineProperty` function to do the following:

- Add a new property to an object. This occurs when the object does not have the specified property name.
- Modify attributes of an existing property. This occurs when the object already has the specified property name.

  The property definition is provided in a descriptor object, which describes the attributes of a data property or an accessor property. The descriptor object is a parameter of the `Object.defineProperty` function.

  To add multiple properties to an object, or to modify multiple existing properties, you can use the Object.defineProperties Function.

Exceptions

A `TypeError` exception is thrown if any one of the following conditions is true:

- The `object` argument is not an object.
- The object is not extensible and the specified property name does not exist..
- The `descriptor` has a `value` or `writable` attribute, and has a `get` or `set` attribute.
- The `descriptor` has a `get` or `set` attribute that is not a function or undefined.

- The specified property name already exists, the existing property has a `configurable` attribute of `false`, and the `descriptor` contains one or more attributes that are different from those in the existing property. However, when the existing property has a `configurable` attribute of `false` and a `writable` attribute of `true`, it is permitted for the `value` or `writable` attribute to be different.

Adding a Data Property

In the following example, the `Object.defineProperty` function adds a data property to a user-defined object. To instead add the property to an existing DOM object, uncomment the `var = window.document` line.

```
var newLine = "<br />";

// Create a user-defined object.
var obj = {};

// Add a data property to the object.
Object.defineProperty(obj, "newDataProperty", {
    value: 101,
    writable: true,
    enumerable: true,
    configurable: true
});

// Set the property value.
obj.newDataProperty = 102;
document.write("Property value: " + obj.newDataProperty + newLine);

// Output:
// Property value: 102
```

To list the object properties, add the following code to this example.

```
var names = Object.getOwnPropertyNames(obj);
for (var i = 0; i < names.length; i++) {
    var prop = names[i];

    document.write(prop + ': ' + obj[prop]);
    document.write(newLine);
}

// Output:
//  newDataProperty: 102
```

Modifying a Data Property

To modify a property attribute for the object, add the following code to the `addDataProperty` function shown earlier. The `descriptor` parameter contains only a `writable` attribute. The other data property attributes remain the same.

```
// Modify the writable attribute of the property.
Object.defineProperty(obj, "newDataProperty", { writable: false });

// List the property attributes by using a descriptor.
// Get the descriptor with Object.getOwnPropertyDescriptor.
var descriptor = Object.getOwnPropertyDescriptor(obj, "newDataProperty");
for (var prop in descriptor) {
    document.write(prop + ': ' + descriptor[prop]);
    document.write(newLine);
}

// Output
// writable: false
// value: 102
// configurable: true
// enumerable: true
```

Adding an Accessor Property

In the following example, the `Object.defineProperty` function adds an accessor property to a user-defined object.

```
var newLine = "<br />";

// Create a user-defined object.
var obj = {};

// Add an accessor property to the object.
Object.defineProperty(obj, "newAccessorProperty", {
    set: function (x) {
        document.write("in property set accessor" + newLine);
        this.newaccpropvalue = x;
    },
    get: function () {
        document.write("in property get accessor" + newLine);
        return this.newaccpropvalue;
    },
    enumerable: true,
    configurable: true
});

// Set the property value.
obj.newAccessorProperty = 30;
document.write("Property value: " + obj.newAccessorProperty + newLine);

// Output:
// in property set accessor
// in property get accessor
// Property value: 30
```

To list the object properties, add the following code to this example.

```
var names = Object.getOwnPropertyNames(obj);
for (var i = 0; i < names.length; i++) {
    var prop = names[i];

    document.write(prop + ': ' + obj[prop]);
    document.write(newLine);
}
// Output:
// in property get accessor
// newAccessorProperty: 30
```

Modifying an Accessor Property

To modify a property attribute for the object, add the following code to the code shown earlier. The `descriptor` parameter contains only a `get` accessor definition. The other property attributes remain the same.

```
// Modify the get accessor.
Object.defineProperty(obj, "newAccessorProperty", {
    get: function () { return this.newaccpropvalue; }
});

// List the property attributes by using a descriptor.
// Get the descriptor with Object.getOwnPropertyDescriptor.
var descriptor = Object.getOwnPropertyDescriptor(obj, "newAccessorProperty");
for (var prop in descriptor) {
    document.write(prop + ': ' + descriptor[prop]);
    document.write(newLine);
}

// Output:
// get: function () { return this.newaccpropvalue; }
// set: function (x) { document.write("in property set accessor" + newLine); this.newaccpropvalue = x; }
// configurable: true
// enumerable: true
```

Modifying a Property on a DOM Element

The following example demonstrates how to customize built-in DOM properties by using the `Object.getOwnPropertyDescriptor` function to get and modify the property's property descriptor. For this example, there must by a DIV element with an ID of "div".

```
// Get the querySelector property descriptor.
var descriptor = Object.getOwnPropertyDescriptor(Element.prototype, "querySelector");

// Make the property read-only.
descriptor.value = "query";
descriptor.writable = false;
// Apply the changes to the Element prototype.
Object.defineProperty(Element.prototype, "querySelector", descriptor);

// Get a DOM element from the HTML body.
var elem = document.getElementById("div");

// Attempt to change the value. This causes the revised value attribute to be called.
elem.querySelector = "anotherQuery";
document.write(elem.querySelector);

// Output:
// query
```

Requirements

Internet Explorer 9 standards mode and Internet Explorer 10 standards mode, as well as Windows 8.x Store apps, support all features.

Internet Explorer 8 standards mode supports DOM objects but not user-defined objects. The `enumerable` and `configurable` attributes can be specified, but they are not used.

See Also

Document Object Model Prototypes, Part 2: Accessor (getter/setter) Support
Object.defineProperties Function
Object.create Function
Object.getOwnPropertyDescriptor Function
Object.getOwnPropertyNames Function

Object.freeze Function (JavaScript)

10/18/2017 • 2 min to read • Edit Online

Prevents the modification of existing property attributes and values, and prevents the addition of new properties.

Syntax

```
Object.freeze(object)
```

Parameters

`object`

Required. The object on which to lock the attributes.

Return Value

The object that is passed to the function.

Exceptions

If the `object` argument is not an object, a `TypeError` exception is thrown.

Remarks

The `Object.freeze` function does the following:

- Makes the object non-extensible, so that new properties cannot be added to it.
- Sets the `configurable` attribute to `false` for all properties of the object. When `configurable` is `false`, the property attributes cannot be changed and the property cannot be deleted.
- Sets the `writable` attribute to `false` for all data properties of the object. When `writable` is false, the data property value cannot be changed.

  For more information about how to set property attributes, see Object.defineProperty Function. To obtain the attributes of a property, you can use the Object.getOwnPropertyDescriptor Function.

Related Functions

The following related functions prevent the modification of object attributes.

FUNCTION	OBJECT IS MADE NON-EXTENSIBLE	`CONFIGURABLE` IS SET TO `FALSE` FOR EACH PROPERTY	`WRITABLE` IS SET TO `FALSE` FOR EACH PROPERTY
Object.preventExtensions	Yes	No	No
Object.seal	Yes	Yes	No
`Object.freeze`	Yes	Yes	Yes

The following functions return `true` if all of the conditions marked in the following table are true.

FUNCTION	OBJECT IS EXTENSIBLE?	`CONFIGURABLE` IS `FALSE` FOR ALL PROPERTIES?	`WRITABLE` IS `FALSE` FOR ALL DATA PROPERTIES?
Object.isExtensible	Yes	No	No
Object.isSealed	No	Yes	Yes
Object.isFrozen	No	Yes	Yes

Example

The following example illustrates the use of the `Object.freeze` function.

```
// Create an object that has two properties.
var obj = { pasta: "spaghetti", length: 10 };

// Freeze the object.
Object.freeze(obj);

// Try to add a new property, and then verify that it is not added.
    obj.newProp = 50;
    document.write(obj.newProp);
    document.write("<br/>");

// Try to delete a property, and then verify that it is still present.
delete obj.length;
document.write(obj.length);
document.write("<br/>");

// Try to change a property value, and then verify that it is not changed.
obj.pasta = "linguini";
document.write(obj.pasta);

// Output:
// undefined
// 10
// spaghetti
```

Requirements

Supported in the following document modes: Internet Explorer 9 standards, Internet Explorer 10 standards, and Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards.

See Also

Object.preventExtensions Function
Object.seal Function
Object.isExtensible Function
Object.isSealed Function
Object.isFrozen Function

Object.getOwnPropertyDescriptor Function (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Gets the own property descriptor of the specified object. An own property descriptor is one that is defined directly on the object and is not inherited from the object's prototype.

Syntax

```
Object.getOwnPropertyDescriptor(object, propertyname)
```

Parameters

`object`
Required. The object that contains the property.

`propertyname`
Required. The name of the property.

Return Value

The descriptor of the property.

Remarks

You can use the `Object.getOwnPropertyDescriptor` function to obtain a descriptor object that describes attributes of the property.

The Object.defineProperty Function is used to add or modify properties.

Data Property Example

The following example gets a data property descriptor and uses it to make the property read-only.

```
// Create a user-defined object.
var obj = {};

// Add a data property.
obj.newDataProperty = "abc";

// Get the property descriptor.
var descriptor = Object.getOwnPropertyDescriptor(obj, "newDataProperty");

// Change a property attribute.
descriptor.writable = false;
Object.defineProperty(obj, "newDataProperty", descriptor);
```

To list the property attributes, you can add the following code to this example.

```
// Get the descriptor from the object.
var desc2 = Object.getOwnPropertyDescriptor(obj, "newDataProperty");

// List the descriptor attributes.
for (var prop in desc2) {
    document.write(prop + ': ' + desc2[prop]);
    document.write("<br />");
}

// Output:
// value: abc
// writable: false
// enumerable: true
// configurable: true
```

Requirements

All features are supported in Internet Explorer 9 standards mode, Internet Explorer 10 standards mode, Internet Explorer 11 standards mode, and Windows Store apps.

Internet Explorer 8 standards mode supports DOM objects but not user-defined objects. The `enumerable` and `configurable` attributes can be specified, but they are not used.

See Also

Document Object Model Prototypes, Part 2: Accessor (getter/setter) Support
Object.defineProperty Function

Object.getOwnPropertyNames Function (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Returns the names of the own properties of an object. The own properties of an object are those that are defined directly on that object, and are not inherited from the object's prototype. The properties of an object include both fields (objects) and functions.

Syntax

```
Object.getOwnPropertyNames(object)
```

Parameters

PARAMETER	DEFINITION
`object`	Required. The object that contains the own properties.

Return Value

An array that contains the names of the own properties of the object.

Exceptions

If the value supplied for the `object` argument is not the name of an object, a `TypeError` exception is thrown.

Remarks

The `getOwnPropertyNames` method returns the names of both enumerable and non-enumerable properties and methods. To return only the names of enumerable properties and methods, you can use the Object.keys Function.

Example

The following example creates an object that has three properties and a method. It then uses the `getOwnPropertyNames` method to obtain the own properties (including the method) of the object.

```
function Pasta(grain, width, shape) {
    // Define properties.
    this.grain = grain;
    this.width = width;
    this.shape = shape;
    this.toString = function () {
        return (this.grain + ", " + this.width + ", " + this.shape);
    }
}

// Create an object.
var spaghetti = new Pasta("wheat", 0.2, "circle");

// Get the own property names.
var arr = Object.getOwnPropertyNames(spaghetti);
document.write (arr);

// Output:
//   grain,width,shape,toString
```

Example

The following example displays the names of properties that start with the letter 's' in a **spaghetti** object constructed with the **Pasta** constructor.

```
function Pasta(grain, size, shape) {
    this.grain = grain;
    this.size = size;
    this.shape = shape;
}

var spaghetti = new Pasta("wheat", 2, "circle");

var names = Object.getOwnPropertyNames(spaghetti).filter(CheckKey);
document.write(names);

// Check whether the first character of a string is 's'.
function CheckKey(value) {
    var firstChar = value.substr(0, 1);
    if (firstChar.toLowerCase() == 's')
        return true;
    else
         return false;
}
// Output:
// size,shape
```

Requirements

Supported in the following document modes: Internet Explorer 9 standards, Internet Explorer 10 standards, and Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards.

See Also

Object.keys Function

Object.getOwnPropertySymbols Function (JavaScript)

1/16/2018 • 1 min to read • Edit Online

Returns the own symbol properties of an object. The own symbol properties of an object are those that are defined directly on that object, and are not inherited from the object's prototype.

Syntax

```
Object.getOwnPropertySymbols(object);
```

Parameters

`object`

Required. The object that contains the own symbols.

Return Value

An array that contains the own symbols of the object.

Remarks

You need to use `Object.getOwnPropertySymbols` to get the symbol properties of an object. `Object.getOwnPropertyNames` will not return the symbol properties.

Example

The following code example shows how to get the symbol properties of an object.

```
var obj = {};
var key = Symbol('description');

obj[key] = 'data';

var symbols = Object.getOwnPropertySymbols(obj);

console.log(symbols[0].toString());

// Output:
// undefined
// Symbol(description)
```

Requirements

Supported in Microsoft Edge (Edge browser). Also supported in Store apps (Microsoft Edge on Windows 10). See Version Information.

Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Not supported in Windows 8.1.

Object.getPrototypeOf Function (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Returns the prototype of an object.

Syntax

```
Object.getPrototypeOf(object)
```

Parameters

`object`

Required. The object that references the prototype.

Return Value

The prototype of the `object` argument. The prototype is also an object.

Exceptions

If the `object` argument is not an object, a `TypeError` exception is thrown.

Example

The following example illustrates the use of the `Object.getPrototypeOf` function.

```
// Create a constructor function.
function Pasta(grain, width) {
    this.grain = grain;
    this.width = width;
}
// Create an object from the pasta constructor.
var spaghetti = new Pasta("wheat", 0.2);

// Obtain the prototype from the object.
var proto = Object.getPrototypeOf(spaghetti);

// Add a property to the prototype and validate that
// the original object has the property.
proto.foodgroup = "carbohydrates";
document.write(spaghetti.foodgroup + " ");

// Verify that the prototype obtained from the object
// is the same as the prototype of the constructor.
var result = (proto === Pasta.prototype);
document.write(result + " ");

// Verify that prototype obtained from the object
// is a prototype of the original object.
var result = proto.isPrototypeOf(spaghetti);
document.write(result);

// Output: carbohydrates true true
```

Example

The following example uses the `Object.getPrototypeOf` function to validate data types.

```
var reg = /a/;
var result = (Object.getPrototypeOf(reg) === RegExp.prototype);
document.write(result + " ");

var err = new Error("an error");
var result = (Object.getPrototypeOf(err) === Error.prototype);
document.write(result);

// Output: true true
```

Requirements

Supported in the following document modes: Internet Explorer 9 standards, Internet Explorer 10 standards, and Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards.

See Also

prototype Property (Object)
isPrototypeOf Method (Object)

Object.is Function (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Returns a value that indicates whether two values are the same value.

Syntax

```
Object.is(value1, value2)
```

Parameters

`value1`

Required. The first value to test.

`value2`

Required. The second value to test.

Return Value

`true` if the value is the same value; otherwise, `false`.

Remarks

Unlike the == operator, `Object.is` does not coerce any types when testing values.

The comparison applied by `Object.is` is similar to the comparison applied by the === operator, except that `Object.is` treats `Number.isNaN` as the same value as `NaN`. It also treats +0 and -0 as different values.

Requirements

Supported in Microsoft Edge (Edge browser). Also supported in Store apps (Microsoft Edge on Windows 10). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Not supported in Windows 8.1.

Object.isExtensible Function (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Returns a value that indicates whether new properties can be added to an object.

Syntax

```
Object.isExtensible(object)
```

Parameters

`object`

Required. The object to test.

Return Value

`true` if the object is extensible, which indicates that new properties can be added to the object; otherwise, `false`.

Exceptions

If the `object` argument is not an object, a `TypeError` exception is thrown.

Remarks

For information about how to set property attributes, see Object.defineProperty Function. To obtain the attributes of a property, you can use the Object.getOwnPropertyDescriptor Function.

Related Functions

The following related functions prevent the modification of object attributes.

FUNCTION	OBJECT IS MADE NON-EXTENSIBLE	`CONFIGURABLE` IS SET TO `FALSE` FOR EACH PROPERTY	`WRITABLE` IS SET TO `FALSE` FOR EACH PROPERTY
Object.preventExtensions	Yes	No	No
Object.seal	Yes	Yes	No
Object.freeze	Yes	Yes	Yes

The following functions return `true` if all of the conditions marked in the following table are true.

FUNCTION	OBJECT IS EXTENSIBLE?	`CONFIGURABLE` IS `FALSE` FOR ALL PROPERTIES?	`WRITABLE` IS `FALSE` FOR ALL DATA PROPERTIES?
`Object.isExtensible`	Yes	No	No
Object.isSealed	No	Yes	No

FUNCTION	OBJECT IS EXTENSIBLE?	`CONFIGURABLE` IS `FALSE` FOR ALL PROPERTIES?	`WRITABLE` IS `FALSE` FOR ALL DATA PROPERTIES?
Object.isFrozen	No	Yes	Yes

Example

The following example illustrates the use of the `Object.isExtensible` function.

```
// Create an object that has two properties.
var obj = { pasta: "spaghetti", length: 10 };

// Make the object non-extensible.
Object.preventExtensions(obj);

// Try to add a new property, and then verify that it is not added.
obj.newProp = 50;
document.write(obj.newProp);

// Output:
undefined
```

Requirements

Supported in the following document modes: Internet Explorer 9 standards, Internet Explorer 10 standards, and Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards.

See Also

Object.preventExtensions Function
Object.seal Function
Object.freeze Function
Object.isSealed Function
Object.isFrozen Function

Object.isFrozen Function (JavaScript)

10/18/2017 • 2 min to read • Edit Online

Returns `true` if existing property attributes and values cannot be modified in an object, and new properties cannot be added to the object.

Syntax

```
Object.isFrozen(object)
```

Parameters

`object`

Required. The object to test.

Return Value

`true` if all of the following are true:

- The object is non-extensible, which indicates that new properties cannot be added to the object.
- The `configurable` attribute is `false` for all existing properties.
- The `writable` attribute is `false` for all existing data properties.

  If the object has no existing properties, the function returns `true` if the object is non-extensible.

Exceptions

If the `object` argument is not an object, a `TypeError` exception is thrown.

Remarks

When the `configurable` attribute of a property is `false`, the property attributes cannot be changed and the property cannot be deleted. When `writable` is `false`, the data property value cannot be changed. When `configurable` is `false` and `writable` is `true`, the `value` and `writable` attributes can be changed.

For information about how to set property attributes, see Object.defineProperty Function. To obtain the attributes of a property, you can use the Object.getOwnPropertyDescriptor Function.

Related Functions

The following related functions prevent the modification of object attributes.

FUNCTION	OBJECT IS MADE NON-EXTENSIBLE	`CONFIGURABLE` IS SET TO `FALSE` FOR EACH PROPERTY	`WRITABLE` IS SET TO `FALSE` FOR EACH PROPERTY
Object.preventExtensions	Yes	No	No
Object.seal	Yes	Yes	No
Object.freeze	Yes	Yes	Yes

FUNCTION	OBJECT IS MADE NON-EXTENSIBLE	`CONFIGURABLE` IS SET TO `FALSE` FOR EACH PROPERTY	`WRITABLE` IS SET TO `FALSE` FOR EACH PROPERTY

The following functions return `true` if all of the conditions marked in the following table are true.

FUNCTION	OBJECT IS EXTENSIBLE?	`CONFIGURABLE` IS `FALSE` FOR ALL PROPERTIES?	`WRITABLE` IS `FALSE` FOR ALL DATA PROPERTIES?
Object.isExtensible	Yes	No	No
Object.isSealed	No	Yes	No
`Object.isFrozen`	No	Yes	Yes

Example

The following example illustrates the use of the `Object.isFrozen` function.

```
// Create an object that has two properties.
var obj = { pasta: "spaghetti", length: 10 };

// Freeze the object, and verify that it is frozen.
Object.freeze(obj);
document.write(obj.isFrozen());

// Try to add a new property, and then verify that it is not added.
obj.newProp = 50;
document.write (obj.newProp);
document.write ("<br/>");

// Try to delete a property, and then verify that it is still present.
delete obj.length;
document.write (obj.length);
document.write ("<br/> ");

// Try to change a property value, and then verify that it is not changed.
obj.pasta = "linguini";
document.write (obj.pasta);

// Output:
// true
// undefined
// 10
// spaghetti
```

Requirements

Supported in the following document modes: Internet Explorer 9 standards, Internet Explorer 10 standards, and Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards.

See Also

Object.preventExtensions Function

Object.seal Function
Object.freeze Function
Object.isExtensible Function
Object.isSealed Function
Object.defineProperty Function
Object.getOwnPropertyDescriptor Function
Object.getOwnPropertyNames Function

Object.isSealed Function (JavaScript)

10/18/2017 • 2 min to read • Edit Online

Returns `true` if existing property attributes cannot be modified in an object and new properties cannot be added to the object.

Syntax

```
Object.isSealed(object)
```

Parameters

`object`

Required. The object to test.

Return Value

`true` if both of the following are true:

- The object is non-extensible, which indicates that new properties cannot be added to the object.
- The `configurable` attribute is `false` for all existing properties.

  If the object does not have any properties, the function returns `true` if the object is non-extensible.

Exceptions

If the `object` argument is not an object, a `TypeError` exception is thrown.

Remarks

When the `configurable` attribute of a property is `false`, the property attributes cannot be changed and the property cannot be deleted. When `writable` is `false`, the data property value cannot be changed. When `configurable` is `false` and `writable` is `true`, the `value` and `writable` attributes can be changed.

The `Object.isSealed` function does not use the `writable` attribute of properties to determine its return value.

For information about how to set property attributes, see Object.defineProperty Function. To obtain the attributes of a property, you can use the Object.getOwnPropertyDescriptor Function.

Related Functions

The following related functions prevent the modification of object attributes.

FUNCTION	OBJECT IS MADE NON-EXTENSIBLE	`CONFIGURABLE` IS SET TO `FALSE` FOR EACH PROPERTY	`WRITABLE` IS SET TO `FALSE` FOR EACH PROPERTY
Object.preventExtensions	Yes	No	No
Object.seal	Yes	Yes	No
Object.freeze	Yes	Yes	Yes

FUNCTION	OBJECT IS MADE NON-EXTENSIBLE	`CONFIGURABLE` IS SET TO `FALSE` FOR EACH PROPERTY	`WRITABLE` IS SET TO `FALSE` FOR EACH PROPERTY

The following functions return `true` if all of the conditions marked in the following table are true.

FUNCTION	OBJECT IS EXTENSIBLE?	`CONFIGURABLE` IS `FALSE` FOR ALL PROPERTIES?	`WRITABLE` IS `FALSE` FOR ALL DATA PROPERTIES?
Object.isExtensible	Yes	No	No
`Object.isSealed`	No	Yes	No
Object.isFrozen	No	Yes	Yes

Example

The following example illustrates the use of the `Object.isSealed` function.

```
// Create an object that has two properties.
var obj = { pasta: "spaghetti", length: 10 };

// Seal the object, and verify that it is sealed.
Object.seal(obj);
document.write(Object.isSealed(obj));
document.write("<br/>");

// Try to add a new property, and then verify that it is not added.
obj.newProp = 50;
document.write(obj.newProp);
document.write("<br/>");

// Try to delete a property, and then verify that it is still present.
delete obj.length;
document.write(obj.length);

// Output:
// true
// undefined
// 10
```

Requirements

Supported in the following document modes: Internet Explorer 9 standards, Internet Explorer 10 standards, and Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards.

See Also

Object.preventExtensions Function
Object.seal Function
Object.freeze Function
Object.isExtensible Function

Object.isFrozen Function
Object.defineProperty Function
Object.getOwnPropertyDescriptor Function

Object.keys Function (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Returns the names of the enumerable properties and methods of an object.

Syntax

```
Object.keys(object)
```

Parameters

PARAMETER	DEFINITION
`object`	Required. The object that contains the properties and methods. This can be an object that you created or an existing Document Object Model (DOM) object.

Return Value

An array that contains the names of the enumerable properties and methods of the object.

Exceptions

If the value supplied for the `object` argument is not the name of an object, a `TypeError` exception is thrown.

Remarks

The `keys` method returns only the names of enumerable properties and methods. To return the names of both enumerable and non-enumerable properties and methods, you can use Object.getOwnPropertyNames Function.

For information about the `enumerable` attribute of a property, see Object.defineProperty Function and Object.getOwnPropertyDescriptor Function.

Example

The following example creates an object that has three properties and a method. It then uses the `keys` method to get the properties and methods of the object.

```
// Create a constructor function.
function Pasta(grain, width, shape) {
    this.grain = grain;
    this.width = width;
    this.shape = shape;

    // Define a method.
    this.toString = function () {
        return (this.grain + ", " + this.width + ", " + this.shape);
    }
}

// Create an object.
var spaghetti = new Pasta("wheat", 0.2, "circle");

// Put the enumerable properties and methods of the object in an array.
var arr = Object.keys(spaghetti);
document.write (arr);

// Output:
// grain,width,shape,toString
```

Example

The following example displays the names of all enumerable properties that start with the letter "g" in the Pasta object.

```
// Create a constructor function.
function Pasta(grain, width, shape) {
    this.grain = grain;
    this.width = width;
    this.shape = shape;
}

var polenta = new Pasta("corn", 1, "mush");

var keys = Object.keys(polenta).filter(CheckKey);
document.write(keys);

// Check whether the first character of a string is "g".
function CheckKey(value) {
    var firstChar = value.substr(0, 1);
    if (firstChar.toLowerCase() == "g")
        return true;
    else
        return false;
}

// Output:
// grain
```

Requirements

Supported in the following document modes: Internet Explorer 9 standards, Internet Explorer 10 standards, and Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards.

See Also

Object.getOwnPropertyNames Function

Object.preventExtensions Function (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Prevents the addition of new properties to an object.

Syntax

```
Object.preventExtensions(object)
```

Parameters

`object`

Required. The object to make non-extensible.

Return Value

The object that is passed to the function.

Exceptions

If the `object` argument is not an object, a `TypeError` exception is thrown.

Remarks

The `Object.preventExtensions` function makes an object non-extensible, so that new named properties cannot be added to it. After an object is made non-extensible, it cannot be made extensible.

For information about how to set property attributes, see Object.defineProperty Function.

Related Functions

The following related functions prevent the modification of object attributes.

FUNCTION	OBJECT IS MADE NON-EXTENSIBLE	`CONFIGURABLE` IS SET TO `FALSE` FOR EACH PROPERTY	`WRITABLE` IS SET TO `FALSE` FOR EACH PROPERTY
`Object.preventExtensions`	Yes	No	No
Object.seal	Yes	Yes	No
Object.freeze	Yes	Yes	Yes

The following functions return `true` if all of the conditions marked in the following table are true.

FUNCTION	OBJECT IS EXTENSIBLE?	`CONFIGURABLE` IS `FALSE` FOR ALL PROPERTIES?	`WRITABLE` IS `FALSE` FOR ALL DATA PROPERTIES?
Object.isExtensible	Yes	No	No
Object.isSealed	No	Yes	No

FUNCTION	OBJECT IS EXTENSIBLE?	`CONFIGURABLE` IS `FALSE` FOR ALL PROPERTIES?	`WRITABLE` IS `FALSE` FOR ALL DATA PROPERTIES?
Object.isFrozen	No	Yes	Yes

Example

The following example illustrates the use of the `Object.preventExtensions` function.

```
// Create an object that has two properties.
var obj = { pasta: "spaghetti", length: 10 };

// Make the object non-extensible.
Object.preventExtensions(obj);
document.write(Object.isExtensible(obj));
document.write("<br/>");

// Try to add a new property, and then verify that it is not added.
obj.newProp = 50;
document.write(obj.newProp);

// Output:
// false
// undefined
```

Requirements

Supported in the following document modes: Internet Explorer 9 standards, Internet Explorer 10 standards, and Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards.

See Also

Object.seal Function
Object.freeze Function
Object.isExtensible Function
Object.isSealed Function
Object.isFrozen Function

Object.seal Function (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Prevents the modification of attributes of existing properties, and prevents the addition of new properties.

Syntax

```
Object.seal(object)
```

Parameters

`object`

Required. The object on which to lock the attributes.

Return Value

The object that is passed to the function.

Exceptions

If the `object` argument is not an object, a `TypeError` exception is thrown.

Remarks

The `Object.seal` function does both of the following:

- Makes the object non-extensible, so that new properties cannot be added to it.
- Sets the `configurable` attribute to `false` for all properties of the object.

  When the `configurable` attribute is `false`, property attributes cannot be changed and the property cannot be deleted. When `configurable` is `false` and `writable` is `true`, the `value` and `writable` attributes can be changed.

  The `Object.seal` function does not change the `writable` attribute.

  For more information about how to set property attributes, see Object.defineProperty Function. To get the attributes of a property, you can use the Object.getOwnPropertyDescriptor Function.

Related Functions

The following related functions prevent the modification of object attributes.

FUNCTION	OBJECT IS MADE NON-EXTENSIBLE	`CONFIGURABLE` IS SET TO `FALSE` FOR EACH PROPERTY	`WRITABLE` IS SET TO `FALSE` FOR EACH PROPERTY
Object.preventExtensions	Yes	No	No
`Object.seal`	Yes	Yes	No
Object.freeze	Yes	Yes	Yes

The following functions return `true` if all of the conditions marked in the following table are true.

FUNCTION	OBJECT IS EXTENSIBLE?	`CONFIGURABLE` IS `FALSE` FOR ALL PROPERTIES?	`WRITABLE` IS `FALSE` FOR ALL DATA PROPERTIES?
Object.isExtensible	Yes	No	No
Object.isSealed	No	Yes	No
Object.isFrozen	No	Yes	Yes

Example

The following example illustrates the use of the `Object.seal` function.

```
// Create an object that has two properties.
var obj = { pasta: "spaghetti", length: 10 };
// Seal the object.
Object.seal(obj);
document.write(Object.isSealed(obj));
document.write("<br/>");

// Try to add a new property, and then verify that it is not added.
obj.newProp = 50;
document.write(obj.newProp);
document.write("<br/>");

// Try to delete a property, and then verify that it is still present.
delete obj.length;
document.write(obj.length);

// Output:
// true
// undefined
// 10
```

Requirements

Supported in the following document modes: Internet Explorer 9 standards, Internet Explorer 10 standards, and Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards.

See Also

Object.preventExtensions Function
Object.freeze Function
Object.isExtensible Function
Object.isSealed Function
Object.isFrozen Function

Object.setPrototypeOf Function (JavaScript)

3/13/2018 • 1 min to read • Edit Online

Sets the prototype of an object.

Syntax

```
Object.setPrototypeOf(obj, proto);
```

Parameters

`obj`

Required. The object for which you are setting the prototype.

`proto`

Required. The new prototype object.

Remarks

WARNING

Setting the prototype may reduce performance on all JavaScript code that has access to an object whose prototype has been mutated.

Example

The following code example shows how to set the prototype for an object.

```
function Rectangle() {
}

var rec = new Rectangle();

if (console && console.log) {
    console.log(Object.getPrototypeOf(rec) === Rectangle.prototype);  // Returns true
    Object.setPrototypeOf(rec, Object.prototype);
    console.log(Object.getPrototypeOf(rec) === Rectangle.prototype);  // Returns false
}
```

Example

The following code example shows how to add properties to an object by adding them to the prototype.

```
var proto = { y: 2 };

var obj = { x: 10 };
Object.setPrototypeOf(obj, proto);

proto.y = 20;
proto.z = 40;

if (console && console.log) {
    console.log(obj.x === 10);  // Returns true
    console.log(obj.y === 20);  // Returns true
    console.log(obj.z === 40);  // Returns true
}
```

Example

The following code example adds properties to the `String` object by setting a new prototype on it.

```
var stringProp = { desc: "description" };

Object.setPrototypeOf(String, stringProp);
var s1 = "333";
var s2 = new String("333");

if (console && console.log) {

    console.log(String.desc === "description"); // Returns true
    console.log(s1.desc === "description");     // Returns false
    console.log(s2.desc === "description");     // Returns false

    Object.setPrototypeOf(s1, String); // Can't be set.
    Object.setPrototypeOf(s2, String);

    console.log(s1.desc === "description"); // Returns false
    console.log(s2.desc === "description"); // Returns true
}
```

Requirements

Supported in Microsoft Edge (Edge browser). Also supported in Store apps (Microsoft Edge on Windows 10). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Not supported in Windows 8.1.

hasOwnProperty Method (Object) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Determines whether an object has a property with the specified name.

Syntax

```
object.hasOwnProperty(proName)
```

Parameters

`object`
Required. Instance of an object.

`proName`
Required. String value of a property name.

Remarks

The `hasOwnProperty` method returns `true` if `object` has a property of the specified name, `false` if it does not. This method does not check the properties in the object's prototype chain; the property must be a member of the object itself.

This property is not supported on host objects for Internet Explorer 8 and below.

Example

In the following example, all `String` objects share a common split method. The following code will display **false** and **true**.

```
var s = new String("Sample");
document.write(s.hasOwnProperty("split"));
document.write("<br/>");
document.write(String.prototype.hasOwnProperty("split"));

// Output:
// false
// true
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

See Also

in Operator

isPrototypeOf Method (Object) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Determines whether an object exists in another object's prototype chain.

Syntax

```
prototype.isPrototypeOf(object)
```

Parameters

`prototype`
Required. An object prototype.

`object`
Required. Another object whose prototype chain is to be checked.

Remarks

The `isPrototypeOf` method returns `true` if `object` has `prototype` in its prototype chain. The prototype chain is used to share functionality between instances of the same object type. The `isPrototypeOf` method returns `false` when `object` is not an object or when `prototype` does not appear in the prototype chain of the `object`.

Example

The following example illustrates the use of the `isPrototypeOf` method.

```
function Rectangle() {
}

var rec = new Rectangle();

document.write(Rectangle.prototype.isPrototypeOf(rec));

// Output: true
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

See Also

prototype Property (Object)

propertyIsEnumerable Method (Object) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Determines whether a specified property is enumerable.

Syntax

```
object. propertyIsEnumerable(proName)
```

Parameters

`object`
Required. Instance of an object.

`proName`
Required. String value of a property name.

Remarks

The `propertyIsEnumerable` method returns `true` if `proName` exists in `object` and can be enumerated using a `For` loop. The `propertyIsEnumerable` method returns `false` if `object` does not have a property of the specified name or if the specified property is not enumerable. Typically, predefined properties are not enumerable, but user-defined properties are always enumerable.

The `propertyIsEnumerable` method does not consider objects in the prototype chain.

Example

```
var a = new Array("apple", "banana", "cactus");
document.write(a.propertyIsEnumerable(1));

// Output: true
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

See Also

prototype Property (Object)

toLocaleString Method (Object) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Returns a date converted to a string using the current locale.

Syntax

```
dateObj.toLocaleString()
```

Remarks

The required `dateObj` is any `Date` object.

The `toLocaleString` method returns a `String` object that contains the date written in the current locale's long default format.

- For dates between 1601 and 1999 A.D., the date is formatted according to the user's Control Panel Regional Settings.
- For dates outside this range, the default format of the **toString** method is used.

  For example, in the United States, `toLocaleString` returns "01/05/96 00:00:00" for January 5. In Europe, it returns "05/01/96 00:00:00" for the same date, as European convention puts the day before the month.

> **NOTE**
>
> `toLocaleString` should only be used to display results to a user; it should never be used as the basis for computation within a script as the returned result is machine-specific.

Example

The following example illustrates the use of the `toLocaleString` method.

```
function toLocaleStrDemo(){
   var d, s;                      //Declare variables.
   d = new Date();                //Create Date object.
   s = "Current setting is ";
   s += d.toLocaleString();       //Convert to current locale.
   return(s);                     //Return converted date
}
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

Applies To: Array Object| Date Object| Number Object| Object Object

See Also

toLocaleDateString Method (Date)

toString Method (Object) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Returns a string representation of an object.

Syntax

```
objectname.toString([radix])
```

Parameters

`objectname`
Required. An object for which a string representation is sought.

`radix`
Optional. Specifies a radix for converting numeric values to strings. This value is only used for numbers.

Remarks

The **toString** method is a member of all built-in JavaScript objects. How it behaves depends on the object type:

OBJECT	BEHAVIOR
Array	Elements of an `Array` are converted to strings. The resulting strings are concatenated, separated by commas.
Boolean	If the Boolean value is **true**, returns "`true`". Otherwise, returns "`false`".
Date	Returns the textual representation of the date.
Error	Returns a string containing the associated error message.
Function	Returns a string of the following form, where *functionname* is the name of the function whose **toString** method was called: `function functionname( ) { [native code] }`
Number	Returns the textual representation of the number.
String	Returns the value of the `String` object.
Default	Returns `"[object objectname]"`, where `objectname` is the name of the object type.

Example

The following example illustrates the use of the **toString** method with a radix argument. The return value of

function shown below is a Radix conversion table.

```
function CreateRadixTable (){
   var s = "";

   // Create table heading.
   s += "Hex  Dec  Bin \n";

   for (x = 0; x < 16; x++)
   {
      s += "   ";

      // Convert to hexidecimal.
      s += x.toString(16);
      s += "     ";
      if (x < 10) s += "  ";

      // Convert to decimal.
      s += x.toString(10);
      s += "     ";

      // Convert to binary.
      s += x.toString(2);
      s += "\n";
   }

   return(s);
}
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

Applies To: Array Object| Boolean Object| Date Object| Error Object| Function Object| Number Object| Object Object| String Object

See Also

function Statement

valueOf Method (Object) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Returns the primitive value of the specified object.

Syntax

```
object.valueOf( )
```

Remarks

The required `object` reference is any intrinsic JavaScript object.

The `valueOf` method is defined differently for each intrinsic JavaScript object.

OBJECT	RETURN VALUE
Array	Returns the array instance.
Boolean	The Boolean value.
Date	The stored time value in milliseconds since midnight, January 1, 1970 UTC.
Function	The function itself.
Number	The numeric value.
Object	The object itself. This is the default.
String	The string value.

The **Math** and `Error` objects do not have a `valueOf` method.

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

Applies To: Array Object| Boolean Object| Date Object| Function Object| Number Object| Object Object| String Object

See Also

toString Method (Object)

Promise Object (JavaScript)

10/18/2017 • 2 min to read • Edit Online

Provides a mechanism to schedule work to be done on a value that has not yet been computed. It is an abstraction for managing interactions with asynchronous APIs.

Syntax

```
var promise = new Promise(function(resolve, reject) { ... });
```

Parameters

`promise`

Required. The variable name to which the promise is assigned.

`resolve`

Required. A function that runs to indicate that the promise has completed successfully.

`reject`

Optional. A function that runs to indicate that the promise has been rejected with an error.

Remarks

A Promise must either be completed with a value, or it must be rejected with a reason. The `then` method of the Promise object runs when the promise is completed or rejected, whichever occurs first. If the promise is completed successfully, the fulfillment handler function of the `then` method runs. If the promise is rejected, the error handler function of the `then` method (or the `catch` method) runs.

Example

The following example shows how to call a function (`timeout`) that returns a promise. The fulfillment handler of the `then` method runs after the 5000ms timeout period expires.

```
function timeout(duration) {
    return new Promise(function(resolve, reject) {
        setTimeout(resolve, duration);
    });
}

// Note: This code uses arrow function syntax
var m = timeout(5000).then(() => {
    console.log("done!");
})

// Output (after 5 seconds):
// done!
```

Example

You can also chain calls to the `then` method as shown in the following code. Each completion handler must itself return a promise to support chaining. In this code, which calls the same `timeout` function, the first call to timeout returns after 1000 ms. The first completion handler calls `timeout` again, and this promise returns after 2000ms.

Its completion handler then throws an error. The error handler calls `Promise.all`, which returns only when both calls to timeout are completed or rejected.

```
var p = timeout(1000).then(() => {
    return timeout(2000);
}).then(() => {
    throw new Error("error");
}).catch(err => {
    return Promise.all([timeout(100), timeout(200)]);
})
```

Requirements

Supported in Microsoft Edge (Edge browser). Also supported in Store apps (Microsoft Edge on Windows 10). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Not supported in Windows 8.1.

Functions

The following table describes the functions of the `Promise` object.

FUNCTION	DESCRIPTION
Promise.all Function	Joins two or more promises and returns only when all the specified promises have completed or been rejected.
Promise.race Function	Creates a new promise that will resolve or reject with the same result value as the first promise to resolve or reject among the passed in arguments.
Promise.reject Function	Creates a new rejected promise with a result equal to the passed in argument.
Promise.resolve Function	Creates a new resolved promise with a result equal to its argument.

Methods

The following table describes the methods of the `Promise` object.

METHOD	DESCRIPTION
catch Method	Allows you to specify work to be done on the rejection of a promise.
then Method	Allows you to specify work to be done on the fulfillment of a promise.

Promise.all Function (Promise)

10/18/2017 • 1 min to read • Edit Online

Joins two or more promises and returns only when all the specified promises have completed or been rejected.

Syntax

```
Promise.all(func1, func2 [,funcN])
```

Parameters

`func1`

Required. A function that returns a promise.

`func2`

Required. A function that returns a promise.

`funcN`

Optional. One or more functions that return a promise.

Remarks

The result returned is an array of values returned by the completed promises. If one of the joined promises is rejected, `Promise.all` immediately returns with the reason for the rejected promise (all other returns values are discarded).

Example

In this code, the first call to timeout returns after 5000ms. The completion handler calls `Promise.all`, which returns only when both calls to timeout are completed or rejected.

```
function timeout(duration) {
    return new Promise(function(resolve, reject) {
        setTimeout(resolve, duration);
    });
}

var p = timeout(5000).then(() => {
    return Promise.all([timeout(100), timeout(200)]);
})
```

Requirements

Supported in Microsoft Edge (Edge browser). Also supported in Store apps (Microsoft Edge on Windows 10). See Version Information.

Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Not supported in Windows 8.1.

See Also

Promise Object

Promise.race Function (Promise)

1/12/2018 • 1 min to read • Edit Online

Creates a new promise that will resolve or reject with the same result value as the first promise to resolve or reject among the passed in arguments.

Syntax

```
Promise.race(iterable)
```

Parameters

`iterable`

Required. One or more promises.

Remarks

If one of the promises in `iterable` is already in a resolved or rejected state, `Promise.race` returns a promise resolved or rejected in the same way with the result value equal to the value used to resolve (or reject) that promise. If multiple promises in `iterable` are already resolved or rejected, `Promise.race` returns a promise resolved in the same way as the first promise iterated. If no promise in iterable resolves or rejects, the promise returned from `Promise.race` also does not resolve or reject.

Example

```
var p1 = new Promise(function(resolve, reject) {
    setTimeout(resolve, 0, 'success');
});
var p2 = new Promise(function(resolve, reject) { });
var p3 = new Promise(function(resolve, reject) { });

var race = Promise.race( [p1, p2, p3] );
race.then(function(result) {
    console.log(result);
});

// Output:
// success

var race = Promise.race( [Promise.reject('failure'),
    Promise.resolve('success')] );
race.catch(function(result) {
    console.log(result);
});

// Output:
// failure
```

Requirements

Supported in Microsoft Edge (Edge browser). Also supported in Store apps (Microsoft Edge on Windows 10). See Version Information.

Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7

standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Not supported in Windows 8.1.

See Also

Promise Object

Promise.reject Function (Promise)

10/18/2017 • 1 min to read • Edit Online

Creates a new rejected promise with a result equal to the passed in argument.

Syntax

```
Promise.reject(r);
```

Parameters

`r`

Required. The reason why the promise was rejected.

Remarks

The error handling function of the `then` or `catch` method runs when the rejected promise is returned.

Example

```
var p = Promise.reject('failure');
p.catch(function(result) {
    console.log(result);
});

// Output:
// failure
```

Requirements

Supported in Microsoft Edge (Edge browser). Also supported in Store apps (Microsoft Edge on Windows 10). See Version Information.

Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Not supported in Windows 8.1.

See Also

Promise Object

Promise.resolve Function (Promise)

10/18/2017 • 1 min to read • Edit Online

Creates a new resolved promise with a result equal to its argument.

Syntax

```
Promise.resolve(x)
```

Parameters

`x`

Required. The value returned with the completed promise.

Remarks

The fulfillment handling function of the `then` method runs when the completed promise object is returned.

Example

```
var p = Promise.resolve('success');
p.then(function(result) {
    console.log(result);
});

// Output:
// success
```

Requirements

Supported in Microsoft Edge (Edge browser). Also supported in Store apps (Microsoft Edge on Windows 10). See Version Information.

Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Not supported in Windows 8.1.

See Also

Promise Object

catch Method (Promise)

10/18/2017 • 1 min to read • Edit Online

Allows you to specify work to be done on the rejection of a promise.

Syntax

```
promise.catch(onRejected)
```

Parameters

`promise`

Required. The promise object.

`onRejected`

Required. The error handler function to run when a promise is rejected.

Remarks

Example

In the following code example, the first call to timeout returns after 5000ms. In this code, the promise is rejected, and the error handler function runs.

```
function timeout(duration) {
    return new Promise(function(resolve, reject) {
        setTimeout(reject, duration);
    });
}

var p = timeout(5000).then(() => {
    console.log("done!");
}).catch(err => {
    console.log("error!");
})

// Output (after five seconds):
// error!
```

Requirements

Supported in Microsoft Edge (Edge browser). Also supported in Store apps (Microsoft Edge on Windows 10). See Version Information.

Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Not supported in Windows 8.1.

then Method (Promise)

10/18/2017 • 1 min to read • Edit Online

Allows you to specify work to be done on the fulfillment of a promise.

Syntax

```
promise.then(onCompleted, onRejected);
```

Parameters

`promise`
Required. The promise object.

`onCompleted`
Required. The fulfillment handler function to run when the promise completes successfully.

`onRejected`
Optional. The error handler function to run when the promise is rejected.

Remarks

A Promise must either be completed with a value, or it must be rejected with a reason. The `then` method of the Promise object runs when the promise is completed or rejected, whichever occurs first. If the promise is completed successfully, the fulfillment handler function of the `then` method runs. If the promise is rejected, the error handler function of the `then` method (or the `catch` method) runs.

Example

The following example shows how to call a function (`timeout`) that returns a promise. The fulfillment handler of the `then` method runs after the 5000ms timeout period expires.

```
function timeout(duration) {
    return new Promise(function(resolve, reject) {
        setTimeout(resolve, duration);
    });
}

// Note: This code uses arrow function syntax
var m = timeout(5000).then(() => {
    console.log("done!");
})

// Output (after 5 seconds):
// done!
```

Requirements

Supported in Microsoft Edge (Edge browser). Also supported in Store apps (Microsoft Edge on Windows 10). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7

standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Not supported in Windows 8.1.

See Also

Promise Object

Proxy Object (JavaScript)

3/13/2018 • 2 min to read • Edit Online

Enables custom behavior for an object.

Syntax

```
proxyObj = new Proxy(target, handler)
```

Parameters

`target`

Required. An object or function to be virtualized by the proxy.

`handler`

Required. An object with methods (traps) that implement the custom behavior.

Remarks

A `Proxy` object is used to intercept internal low-level operations on another object. Proxy objects can be used for interception, object virtualization, logging/profiling, and other purposes.

If a trap for a specific operation has not been defined in the handler for the proxy, the operation is forwarded to the target.

The handler object defines the following methods (traps) to implement custom behavior. The examples here are not exhaustive. To support conditional default behavior in the handler method, use methods of Reflect Object.

HANDLER METHOD (TRAP) SYNTAX	EXAMPLES OF USAGE
`apply: function(target, thisArg, args)`	A trap for a function call.
`construct: function(target, args)`	A trap for a constructor.
`defineProperty: function(target, propertyName, descriptor)`	A trap for Object.defineProperty Function.
`deleteProperty: function(target, propertyName)`	A trap for the `delete` statement.
`enumerate: function(target)`	A trap for the for...in statement, Object.getOwnPropertySymbols, Object.keys function, and JSON.stringify.
`get: function(target, propertyName, receiver)`	A trap for any getter properties.
`getOwnPropertyDescriptor: function(target, propertyName)`	A trap for Object.getOwnPropertyDescriptor Function.
`getPrototypeOf: function(target)`	A trap for Object.getPrototypeOf Function.

HANDLER METHOD (TRAP) SYNTAX	EXAMPLES OF USAGE
`has: function(target, propertyName)`	A trap for the `in` operator, hasOwnProperty Method (Object), and other methods.
`isExtensible: function(target)`	A trap for Object.isExtensible Function.
`ownKeys: function(target)`	A trap for Object.getOwnPropertyNames Function.
`preventExtensions: function(target)`	A trap for Object.preventExtensions Function.
`set: function(target, propertyName, value, receiver)`	A trap for any setter properties.
`setPrototypeOf: function(target, prototype)`	A trap for Object.setPrototypeOf.

Example

The following code example shows how to create a proxy for an object literal using the `get` trap.

```
var target = {};
var handler = {
  get: function (target, property, receiver) {
    // This example includes a template string.
    return `Hello, ${property}!`;
  }
};

var p = new Proxy(target, handler);
console.log(p.world);

// Output:
// Hello, world!
```

Example

The following code example shows how to create a proxy for a function using the `apply` trap.

```
var target = function () { return 'I am the target'; };
var handler = {
  // This example includes a rest parameter.
  apply: function (receiver, ...args) {
    return 'I am the proxy';
  }
};

var p = new Proxy(target, handler);
console.log(target()):
console.log(p()):

// Output:
// I am the target
// I am the proxy
```

Requirements

Supported in Microsoft Edge (Edge browser). Also supported in Store apps (Microsoft Edge on Windows 10). See

Version Information.

Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Not supported in Windows 8.1.

Reflect Object (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Provides methods for use in operations that are intercepted.

Syntax

```
Reflect.[method]
```

Parameters

`method`

Required. Name of one of the methods of the `Reflect` object.

Remarks

The Reflect object cannot be instantiated with the `new` operator.

Reflect methods are often used with proxies because they allow you to delegate to default behavior without implementing the default behavior in your code.

Reflect provides a static method with the same name as each proxy trap. The descriptions in the table are not exhaustive.

METHOD	DESCRIPTION
`Reflect.apply(target, thisArg, args)`	Similar to the apply method of the Function object.
`Reflect.construct(target, args)`	A function equivalent for the `new` operator.
`Reflect.defineProperty(target, propertyName, descriptor)`	Similar to Object.defineProperty. Returns a boolean value indicating whether the call succeeded.
`Reflect.deleteProperty(target, propertyName)`	Similar to the `delete` statement. Returns a boolean value indicating whether the call succeeded.
`Reflect.enumerate(target)`	Similar to for...in statement, Object.getOwnPropertySymbols, Object.keys function, and JSON.stringify.
`Reflect.get(target, propertyName, receiver)`	A function equivalent for any getter properties.
`Reflect.getOwnPropertyDescriptor(target, propertyName)`	Similar to Object.getOwnPropertyDescriptor. Returns a Boolean value indicating whether the call succeeded.
`Reflect.getPrototypeOf(target)`	Similar to Object.getPrototypeOf.
`Reflect.has(target, propertyName)`	Similar to the `in` operator, hasOwnProperty Method (Object), and other methods. Returns a Boolean value indicating whether the call succeeded.

METHOD	DESCRIPTION
`Reflect.isExtensible(target)`	Similar to Object.isExtensible.
`Reflect.ownKeys(target)`	Similar to Object.getOwnPropertyNames.
`Reflect.preventExtensions(target)`	Similar to Object.preventExtensions. Returns a boolean value indicating whether the call succeeded.
`Reflect.set(target, propertyName, value, receiver)`	Similar to using any setter property. Returns a Boolean value indicating whether the call succeeded.
`Reflect.setPrototypeOf(target, prototype)`	Similar to Object.setPrototypeOf. Returns a boolean value indicating whether the call succeeded.

Example

The following code example shows how use `Reflect.get` to write a proxy that blocks get operations for properties that begin with an underscore.

```
var p = new Proxy({}, {
    get(k, t, r) {
        // return undefined if key begins with underscore
        if(k[0] === '_') return undefined;

       // otherwise do default behavior
       return Reflect.get(k, t, r);
    }
});

p._foo = 1;
console.log(p._foo);

p.foo = 1;
console.log(p.foo);

// Output:
// undefined
// 1
```

Requirements

Supported in Microsoft Edge (Edge browser). Also supported in Store apps (Microsoft Edge on Windows 10). See Version Information.

Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Not supported in Windows 8.1.

RegExp Object (JavaScript)

10/18/2017 • 1 min to read • Edit Online

An intrinsic global object that stores information about the results of regular expression pattern matches.

Syntax

```
RegExp.property
```

Remarks

The required *property* argument can be any one of the `RegExp` object properties.

The `RegExp` object cannot be created directly, but is always available for use. Until a successful regular expression search has been completed, the initial values of the various properties of the `RegExp` object are as follows:

PROPERTY	SHORTHAND	INITIAL VALUE
index		-1
input	$_	Empty string.
lastIndex		-1
lastMatch	$&	Empty string.
lastParen	$+	Empty string.
leftContext	$`	Empty string.
rightContext	$'	Empty string.
$1 - $9	$1 - $9	Empty string.

Its properties have undefined as their value until a successful regular expression search has been completed.

The global `RegExp` object should not be confused with the **Regular Expression** object. Even though they sound like the same thing, they are separate and distinct. The properties of the global `RegExp` object contain continually updated information about each match as it occurs, while the properties of the **Regular Expression** object contain only information about the matches that occur with that instance of the **Regular Expression**.

Example

The following example performs a regular expression search. It displays matches and submatches from the global `RegExp` object, and from the array that is returned by the `exec` method.

Properties

$1...$9 Properties | index Property | input Property | lastIndex Property | lastMatch Property | lastParen Property | leftContext Property | rightContext Property

Methods

The `RegExp` object has no methods.

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

See Also

Regular Expression Object
Regular Expression Syntax (JavaScript)
String Object

$1...$9 Properties (RegExp) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Return the nine most-recently memorized portions found during pattern matching. Read-only.

Syntax

```
RegExp.$n
```

Parameters

`RegExp`
Always the global `RegExp` object.

`n`
Any integer from 1 through 9.

Remarks

The values of the **$1...$9** properties are modified whenever a successful parenthesized match is made. Any number of parenthesized substrings may be specified in a regular expression pattern, but only the nine most recent can be stored.

Example

The following example performs a regular expression search. It displays matches and submatches from the global `RegExp` object. The submatches are successful parenthesized matches that are contained in the `$1...$9` properties. The example also displays matches and submatches from the array that is returned by the `exec` method.

```
var newLine = "<br />";

var re = /(\w+)@(\w+)\.(\w+)/g
var src = "Please send mail to george@contoso.com and someone@example.com. Thanks!"

var result;
var s = "";

// Get the first match.
result = re.exec(src);

while (result != null) {
    // Show the entire match.
    s += newLine;

    // Show the match and submatches from the RegExp global object.
    s += "RegExp.lastMatch: " + RegExp.lastMatch + newLine;
    s += "RegExp.$1: " + RegExp.$1 + newLine;
    s += "RegExp.$2: " + RegExp.$2 + newLine;
    s += "RegExp.$3: " + RegExp.$3 + newLine;

    // Show the match and submatches from the array that is returned
    // by the exec method.
    for (var index = 0; index < result.length; index++) {
        s +=  index + ": ";
        s += result[index];
        s += newLine;
    }

    // Get the next match.
    result = re.exec(src);
}
document.write(s);

// Output:
//  RegExp.lastMatch: george@contoso.com
//  RegExp.$1: george
//  RegExp.$2: contoso
//  RegExp.$3: com
//  0: george@contoso.com
//  1: george
//  2: contoso
//  3: com

//  RegExp.lastMatch: someone@example.com
//  RegExp.$1: someone
//  RegExp.$2: example
//  RegExp.$3: com
//  0: someone@example.com
//  1: someone
//  2: example
//  3: com
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

Applies To: RegExp Object

See Also

Regular Expression Syntax (JavaScript)

index Property (RegExp) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Returns the character position where the first successful match begins in a searched string. Read-only.

Syntax

```
RegExp.index
```

Remarks

The object associated with this property is always the global `RegExp` object.

The **index** property is zero-based. The initial value of the **index** property is -1. Its value changes whenever a successful match is made.

Example

The following example illustrates the use of the **index** property. This function iterates a search string and prints out the **index** and `lastIndex` values for each word in the string.

```
function RegExpTest()
{
   var ver = Number(ScriptEngineMajorVersion() + "." + ScriptEngineMinorVersion())
   if (ver < 5.5)
   {
      document.write("You need a newer version of JavaScript for this to work");
      return;
   }

   var src = "The quick brown fox jumps over the lazy dog.";

   // Create regular expression pattern with a global flag.
   var re = /\w+/g;

   // Get the next word, starting at the position of lastindex.
   var arr;
   while ((arr = re.exec(src)) != null)
      {
      // New line:
      document.write ("<br />");
      document.write (arr.index + "-" + arr.lastIndex + " ");
      document.write (arr);
      }
}
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

Applies To: RegExp Object

See Also

Regular Expression Syntax (JavaScript)

input Property ($_) (RegExp) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Returns the string against which a regular expression search was performed. Read-only.

Syntax

```
RegExp.input
```

Remarks

The object associated with this property is always the global `RegExp` object.

The value of **input** property is modified any time the searched string is changed.

The following example illustrates the use of the **input** property:

```
function inputDemo(){
   var s;
   var re = new RegExp("d(b+)(d)","ig");
   var str = "cdbBdbsbdbdz";
   var arr = re.exec(str);
   s = "The string used for the match was " + RegExp.input;
   return(s);
}
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

Applies To: RegExp Object

See Also

Regular Expression Syntax (JavaScript)

lastIndex Property (RegExp) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Returns the character position where the next match begins in a searched string.

Syntax

```
RegExp.lastIndex
```

Remarks

The object associated with this property is always the global `RegExp` object.

The `lastIndex` property is zero-based, that is, the index of the first character is zero. Its initial value is -1. Its value is modified whenever a successful match is made.

The `lastIndex` property is modified by the `exec` and **test** methods of the `RegExp` object, and the `match`, **replace**, and **split** methods of the `String` object.

The following rules apply to values of `lastIndex`:

- If there is no match, `lastIndex` is set to -1.
- If `lastIndex` is greater than the length of the string, **test** and `exec` fail and `lastIndex` is set to -1.
- If `lastIndex` is equal to the length of the string, the regular expression matches if the pattern matches the empty string. Otherwise, the match fails and `lastIndex` is reset to -1.
- Otherwise, `lastIndex` is set to the next position following the most recent match.

Example

The following example illustrates the use of the `lastIndex` property. This function iterates a search string and prints out the **index** and `lastIndex` values for each word in the string.

```
function RegExpTest()
{
   var ver = Number(ScriptEngineMajorVersion() + "." + ScriptEngineMinorVersion())
   if (ver < 5.5)
   {
      document.write("You need a newer version of JavaScript for this to work");
      return;
   }

   var src = "The quick brown fox jumps over the lazy dog.";

   // Create regular expression pattern with a global flag.
   var re = /\w+/g;

   // Get the next word, starting at the position of lastindex.
   var arr;
   while ((arr = re.exec(src)) != null)
      {
      // New line:
      document.write ("<br />");
      document.write (arr.index + "-" + re.lastIndex + " ");
      document.write (arr);
      }
}
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

Applies To: RegExp Object

See Also

Regular Expression Syntax (JavaScript)

lastMatch Property ($&) (RegExp) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Returns the last matched characters from any regular expression search. Read-only.

Syntax

```
RegExp.lastMatch
```

Remarks

The object associated with this property is always the global `RegExp` object.

The initial value of the `lastMatch` property is an empty string. The value of the `lastMatch` property changes whenever a successful match is made.

Example

The following example illustrates the use of the `lastMatch` property:

```
// Create the regular expression pattern.
var re = new RegExp("d(b+)(d)","ig");
var str = "cdbBdbsbdbdz";

// Perform the search.
var arr = re.exec(str);

// Print the output.
var s = ""
s += "$1: " + RegExp.$1 + "<br />";
s += "$2: " + RegExp.$2 + "<br />";
s += "$3: " + RegExp.$3 + "<br />";
s += "input: " + RegExp.input + "<br />";
s += "lastMatch: " + RegExp.lastMatch + "<br />";
s += "leftContext: " + RegExp.leftContext + "<br />";
s += "rightContext: " + RegExp.rightContext + "<br />";
s += "lastParen: " + RegExp.lastParen + "<br />";

document.write(s);
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

Applies To: RegExp Object

See Also

$1...$9 Properties (RegExp)

index Property (RegExp)
input Property ($_) (RegExp)
lastIndex Property (RegExp)
lastParen Property ($+) (RegExp)
leftContext Property ($`) (RegExp)
rightContext Property ($') (RegExp)

lastParen Property ($+) (RegExp) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Returns the last parenthesized submatch from any regular expression search, if any. Read-only.

Syntax

```
RegExp.lastParen
```

Remarks

The object associated with this property is always the global `RegExp` object.

The initial value of the `lastParen` property is an empty string. The value of the `lastParen` property changes whenever a successful match is made.

Example

The following example illustrates the use of the `lastParen` property:

```
// Create the regular expression pattern.
var re = new RegExp("d(b+)(d)","ig");
var str = "cdbBdbsbdbdz";

// Perform the search.
var arr = re.exec(str);

// Print the output.
var s = ""
s += "$1: " + RegExp.$1 + "<br />";
s += "$2: " + RegExp.$2 + "<br />";
s += "$3: " + RegExp.$3 + "<br />";
s += "input: " + RegExp.input + "<br />";
s += "lastMatch: " + RegExp.lastMatch + "<br />";
s += "leftContext: " + RegExp.leftContext + "<br />";
s += "rightContext: " + RegExp.rightContext + "<br />";
s += "lastParen: " + RegExp.lastParen + "<br />";

document.write(s);
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

Applies To: RegExp Object

See Also

$1...$9 Properties (RegExp)

index Property (RegExp)
input Property ($_) (RegExp)
lastIndex Property (RegExp)
lastMatch Property ($&) (RegExp)
leftContext Property ($`) (RegExp)
rightContext Property ($') (RegExp)

leftContext Property ($`) (RegExp) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Returns the characters from the beginning of a searched string up to the position before the beginning of the last match. Read-only.

Syntax

```
RegExp.leftContext
```

Remarks

The object associated with this property is always the global `RegExp` object.

The initial value of the `leftContext` property is an empty string. The value of the `leftContext` property changes whenever a successful match is made.

Example

The following example illustrates the use of the `leftContext` property:

```
// Create the regular expression pattern.
var re = new RegExp("d(b+)(d)","ig");
var str = "cdbBdbsbdbdz";

// Perform the search.
var arr = re.exec(str);

// Print the output.
var s = ""
s += "$1: " + RegExp.$1 + "<br />";
s += "$2: " + RegExp.$2 + "<br />";
s += "$3: " + RegExp.$3 + "<br />";
s += "input: " + RegExp.input + "<br />";
s += "lastMatch: " + RegExp.lastMatch + "<br />";
s += "leftContext: " + RegExp.leftContext + "<br />";
s += "rightContext: " + RegExp.rightContext + "<br />";
s += "lastParen: " + RegExp.lastParen + "<br />";

document.write(s);
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

Applies To: RegExp Object

See Also

$1...$9 Properties (RegExp)
index Property (RegExp)
input Property ($_) (RegExp)
lastIndex Property (RegExp)
lastMatch Property ($&) (RegExp)
lastParen Property ($+) (RegExp)
rightContext Property ($') (RegExp)

rightContext Property ($') (RegExp) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Returns the characters from the position following the last match to the end of the searched string. Read-only.

Syntax

```
RegExp.rightContext
```

Remarks

The object associated with this property is always the global `RegExp` object.

The initial value of the **rightContext** property is an empty string. The value of the **rightContext** property changes whenever a successful match is made.

Example

The following example illustrates the use of the **rightContext** property:

```
// Create the regular expression pattern.
var re = new RegExp("d(b+)(d)","ig");
var str = "cdbBdbsbdbdz";

// Perform the search.
var arr = re.exec(str);

// Print the output.
var s = ""
s += "$1: " + RegExp.$1 + "<br />";
s += "$2: " + RegExp.$2 + "<br />";
s += "$3: " + RegExp.$3 + "<br />";
s += "input: " + RegExp.input + "<br />";
s += "lastMatch: " + RegExp.lastMatch + "<br />";
s += "leftContext: " + RegExp.leftContext + "<br />";
s += "rightContext: " + RegExp.rightContext + "<br />";
s += "lastParen: " + RegExp.lastParen + "<br />";

document.write(s);
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

Applies To: RegExp Object

See Also

$1...$9 Properties (RegExp)

index Property (RegExp)
input Property ($_) (RegExp)
lastIndex Property (RegExp)
lastMatch Property ($&) (RegExp)
lastParen Property ($+) (RegExp)
leftContext Property ($`) (RegExp)

Regular Expression Object (JavaScript)

10/18/2017 • 3 min to read • Edit Online

An object that contains a regular expression pattern along with flags that identify how to apply the pattern.

Syntax

```
re = /pattern/[flags]
```

Syntax

```
re = new RegExp("pattern"[,"flags"])
```

Parameters

re
Required. The variable name to which the regular expression pattern is assigned.

pattern
Required. The regular expression pattern to use. If you use Syntax 1, delimit the pattern by "/" characters. If you use Syntax 2, enclose the pattern in quotation marks.

`flags`
Optional. Enclose flag in quotation marks if you use Syntax 2. Available flags, which may be combined, are:

- g (global search for all occurrences of *pattern*)
- i (ignore case)
- m (multiline search)
- u (unicode), enables EcmaScript 6 Unicode features. Supported only in Microsoft Edge.
- y (sticky match), searches for matches at the `lastIndex` property of the regular expression (and does not search at later indexes). Supported in Microsoft Edge with experimental JavaScript features enabled (about:flags).

Remarks

The **Regular Expression** object should not be confused with the global `RegExp` object. Even though they sound the same, they are separate and distinct. The properties of the **Regular Expression** object contain only information about one particular **Regular Expression** instance, while the properties of the global `RegExp` object contain continually updated information about each match as it occurs.

Regular Expression objects store patterns used when searching strings for character combinations. After the **Regular Expression** object is created, it is either passed to a string method, or a string is passed to one of the regular expression methods. Information about the most recent search performed is stored in the global `RegExp` object.

Use Syntax 1 when you know the search string ahead of time. Use Syntax 2 when the search string is changing

frequently, or is unknown, such as strings taken from user input.

The *pattern* argument is compiled into an internal format before use. For Syntax 1, *pattern* is compiled as the script is loaded. For Syntax 2, *pattern* is compiled just before use, or when the **compile** method is called.

Example

The following example illustrates the use of the **Regular Expression** object by creating an object (re) containing a regular expression pattern with its associated flags. In this case, the resulting **Regular Expression** object is then used by the `match` method:

```
var s = "through the pages of the book";

// Create regular expression pattern.
var re = new RegExp("the", "i");

// Attempt match on search string.
var r = s.match(re);

// Return first occurrence of "the".
if(console && console.log) {
    console.log(r);
}

// Output:
//
```

Example

The following example updates the regular expression pattern to search for multiple instances.

```
// Create regular expression pattern using the i and g flags.
var re = new RegExp("the", "ig");

// Attempt match on search string.
var r = s.match(re);

// Return the two occurrences of "the".
if(console && console.log) {
    console.log(r.length);
    console.log(r);
}

// Output:
// 2
// [object Array] ["the", "the"]
```

Example

When using the /y flag, if a match succeeds, it updates the `lastindex` to the index of next character after the last match. If the match fails, it resets the `lastindex` to 0.

The following example searches for a match at a specific index using the /y flag and the `lastIndex` property.

```
// Create regular expression pattern using the i and y flags.
var re = new RegExp("the", "iy");

// Set the lastIndex property and attempt a match
// at the specified index.
re.lastIndex = 20;
var r = s.match(re);

// No matches returned.
if(console && console.log) {
    console.log(r);
}
// Reset the lastIndex property and attempt a match.
re.lastIndex = 21;
var r = s.match(re);

// Return occurrence of "the" starting at index 21.
if(console && console.log) {
    console.log(r);
}

// Output:
// null
// [object Array] ["the"]
```

Properties

global Property | ignoreCase Property | multiline Property | source Property

Methods

compile Method | exec Method | test Method

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

The u flag is supported in Microsoft Edge.

The y flag is supported in Microsoft Edge with experimental JavaScript features enabled (about:flags).

See Also

RegExp Object
Regular Expression Syntax (JavaScript)
String Object

global Property (Regular Expression) (JavaScript)

10/18/2017 • 2 min to read • Edit Online

Returns a Boolean value indicating the state of the global flag (**g**) used with a regular expression. Default is **false**. Read-only.

Syntax

```
rgExp.global
```

Remarks

The required `rgExp` reference is an instance of a **Regular Expression** object.

The `global` property returns **true** if the global flag is set for a regular expression, and returns **false** if it is not.

The global flag, when used, indicates that a search should find all occurrences of the pattern within the searched string, not just the first one. This is also known as global matching.

Example

The following example illustrates the use of the `global` property. If you pass **g** in to the function shown below, all instances of the word "the" are replaced with the word "a". Note that the "The" at the beginning of the string is not replaced because the **i** (ignore case) flag is not passed to the function.

This function displays the condition of the properties associated with the allowable regular expression flags, which are **g**, **i**, and **m**. The function also displays the string with all replacements made.

```
function RegExpPropDemo(flag){
   // The flag parameter is a string that contains
   // g, i, or m.  The flags can be combined.

   // Check flags for validity.
   if (flag.match(/[^gim]/))
      {
      return ("Flag specified is not valid");
      }

   // Create the string on which to perform the replacement.
   var ss = "The batter hit the ball with the bat ";
   ss += "and the fielder caught the ball with the glove.";

   //Replace "the" with "a".
   var re = new RegExp("the", flag);
   var r = ss.replace(re, "a");

   // Output the resulting string and the flags.
   var s = "";
   s += "global: " + re.global.toString();
   s += "<br />";
   s += "ignoreCase: " + re.ignoreCase.toString();
   s += "<br />";
   s += "multiline: " + re.multiline.toString();
   s += "<br />";
   s += "Resulting String: " + r;

   return (s);
}

document.write(RegExpPropDemo("g"));
```

Example

Following is the resulting output.

```
global: true
ignoreCase: false
multiline: false
Resulting String: The batter hit a ball with a bat and a fielder caught a ball with a glove.
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

Applies To: Regular Expression Object

See Also

ignoreCase Property (Regular Expression)
multiline Property (Regular Expression)
Regular Expression Syntax (JavaScript)

ignoreCase Property (Regular Expression) (JavaScript)

10/18/2017 • 2 min to read • Edit Online

Returns a Boolean value indicating the state of the ignoreCase flag (**i**) used with a regular expression. Default is **false**. Read-only.

Syntax

```
rgExp.ignoreCase
```

Remarks

The required `rgExp` reference is an instance of the `RegExp` object.

The **ignoreCase** property returns **true** if the ignoreCase flag is set for a regular expression, and returns **false** if it is not.

The ignoreCase flag, when used, indicates that a search should ignore case sensitivity when matching the pattern within the searched string.

Example

The following example illustrates the use of the **ignoreCase** property. If you pass "gi" in to the function shown below, all instances of the word "the" are replaced with the word "a", including the initial "The". This is because with the ignoreCase flag set, the search ignores any case sensitivity. So "T" is the same as "t" for the purposes of matching.

This function returns the Boolean values that indicate the state of the allowable regular expression flags, which are **g**, **i**, and **m**. The function also returns the string with all replacements made.

```
function RegExpPropDemo(flag){
    // The flag parameter is a string that contains
    // g, i, or m. The flags can be combined.

    // Check flags for validity.
    if (flag.match(/[^gim]/))
        {
        return ("Flag specified is not valid");
        }

    // Create the string on which to perform the replacement.
    var orig = "The batter hit the ball with the bat ";
    orig += "and the fielder caught the ball with the glove.";

    // Replace "the" with "a".
    var re = new RegExp("the", flag);
    var r = orig.replace(re, "a");

    // Output the resulting string and the values of the flags.
    var s = "";
    s += "global: " + re.global.toString();
    s += "<br />";
    s += "ignoreCase: " + re.ignoreCase.toString();
    s += "<br />";
    s += "multiline: " + re.multiline.toString();
    s += "<br />";
    s += "Resulting String: " + r;
    s += "<br />";
    s += "<br />";
    return (s);
}

document.write(RegExpPropDemo("gi"));
document.write(RegExpPropDemo("g"));
```

Example

Following is the resulting output.

```
global: true
ignoreCase: true
multiline: false
Resulting String: a batter hit a ball with a bat and a fielder caught a ball with a glove.

global: true
ignoreCase: false
multiline: false
Resulting String: The batter hit a ball with a bat and a fielder caught a ball with a glove.
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

Applies To: Regular Expression Object

See Also

global Property (Regular Expression)

multiline Property (Regular Expression)
Regular Expression Syntax (JavaScript)

multiline Property (Regular Expression) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Returns a Boolean value indicating the state of the multiline flag (**m**) used with a regular expression. Default is **false**. Read-only.

Syntax

```
rgExp.multiline
```

Remarks

The required `rgExp` argument is an instance of the `RegExp` object

The **multiline** property returns **true** if the multiline flag is set for a regular expression, and returns **false** if it is not. The **multiline** property is **true** if the regular expression object was created with the **m** flag.

If **multiline** is **false**, "^" matches the position at the beginning of a string, and "$" matches the position at the end of a string. If **multiline** is **true**, "^" matches the position at the beginning of a string as well as the position following a "\n" or "\r", and "$" matches the position at the end of a string and the position preceding "\n" or "\r".

Example

The following example illustrates the behavior of the **multiline** property. If you pass "m" in to the function shown below, the word "while" is replaced with the word "and". This is because with the multiline flag is set and the word "while" occurs at the beginning of the line after a newline character. The multiline flag allows the search to be performed on multiline strings.

```
function RegExpMultilineDemo(flag){
   // The flag parameter is a string that contains
   // g, i, or m.  The flags can be combined.

   // Check flags for validity.
   if (flag.match(/[^gim]/))
      {
      return ("Flag specified is not valid");
      }

   // Create the string on which to perform the replacement.
   var ss = "The man hit the ball with the bat ";
   ss += "\nwhile the fielder caught the ball with the glove.";

   // Replace "while" with "and".
   var re = new RegExp("^while", flag);
   var r = ss.replace(re, "and");

   // Output the multiline flag and the resulting string.
   var s = "";
   s += "Result for multiline = " + re.multiline.toString();
   s += ": " + r;

   return(s);

}

sa = RegExpMultilineDemo("m");
sb = RegExpMultilineDemo("");
document.write (sa + "<br />" + sb);
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

Applies To: Regular Expression Object

See Also

global Property (Regular Expression)
ignoreCase Property (Regular Expression)
Regular Expression Syntax (JavaScript)

source Property (Regular Expression) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Returns a copy of the text of the regular expression pattern. Read-only. The `rgExp` argument is a **Regular expression** object. It can be a variable name or a literal.

Syntax

```
rgExp.source
```

Example

The following example illustrates the use of the **source** property:

```
function SourceDemo(re, s){
   var s1;
   // Test string for existence of regular expression.
   if (re.test(s))
      s1 = " contains ";
   else
      s1 = " does not contain ";
   // Get the text of the regular expression itself.
   return(s + s1 + re.source);
}
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

Applies To: Regular Expression Object

See Also

Regular Expression Object
Regular Expression Object
Regular Expression Syntax (JavaScript)

compile Method (Regular Expression) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Compiles a regular expression into an internal format for faster execution.

Syntax

```
rgExp.compile(pattern, [flags])
```

Parameters

`rgExp`
Required. An instance of a **Regular Expression** object. Can be a variable name or a literal.

pattern
Required. A string expression containing a regular expression pattern to be compiled

`flags`
Optional. Available flags, which may be combined, are:

- g (global search for all occurrences of *pattern*)
- i (ignore case)
- m (multiline search)

Remarks

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.

Example

The following example illustrates the use of the **compile** method:

```
function CompileDemo(){
   var rs;
   var s = "AaBbCcDdEeFfGgHhIiJjKkLlMmNnOoPp"
   // Create regular expression for uppercase only.
   var r = new RegExp("[A-Z]", "g");
   var a1 = s.match(r)              // Find matches.
   // Compile the regular expression for lowercase only.
   r.compile("[a-z]", "g");
// Find matches.
   var a2 = s.match(r)
   return(a1 + "\n" + a2);
}
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

Applies To: Regular Expression Object

See Also

Regular Expression Syntax (JavaScript)

exec Method (Regular Expression) (JavaScript)

10/18/2017 • 2 min to read • Edit Online

Executes a search on a string using a regular expression pattern, and returns an array containing the results of that search.

Syntax

```
rgExp.exec(str)
```

Parameters

`rgExp`
Required. An instance of a **Regular Expression** object containing the regular expression pattern and applicable flags.

`str`
Required. The `String` object or string literal on which to perform the search.

Remarks

If the `exec` method does not find a match, it returns `null`. If it finds a match, `exec` returns an array, and the properties of the global `RegExp` object are updated to reflect the results of the match. Element zero of the array contains the entire match, while elements 1 - *n* contain any submatches that have occurred within the match. This behavior is identical to the behavior of the `match` method without the global flag (**g**) set.

If the global flag is set for a regular expression, `exec` searches the string beginning at the position indicated by the value of `lastIndex`. If the global flag is not set, `exec` ignores the value of `lastIndex` and searches from the beginning of the string.

The array returned by the `exec` method has three properties, **input**, **index** and **lastIndex.** The **input** property contains the entire searched string. The **index** property contains the position of the matched substring within the complete searched string. The `lastIndex` property contains the position following the last character in the match.

Example

The following example illustrates the use of the `exec` method:

```
function RegExpTest()
{
   var ver = Number(ScriptEngineMajorVersion() + "." + ScriptEngineMinorVersion())
   if (ver < 5.5)
   {
      document.write("You need a newer version of JavaScript for this to work");
      return;
   }

   var src = "The quick brown fox jumps over the lazy dog.";

   // Create regular expression pattern with a global flag.
   var re = /\w+/g;

   // Get the next word, starting at the position of lastindex.
   var arr;
   while ((arr = re.exec(src)) != null)
      {
      // New line:
      document.write ("<br />");
      document.write (arr.index + "-" + arr.lastIndex + " ");
      document.write (arr[0]);
      }
}
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

Applies To: Regular Expression Object

See Also

match Method (String)
RegExp Object
Regular Expression Syntax (JavaScript)
search Method (String)
test Method (Regular Expression)
Regular Expression Programming (JavaScript)

test Method (Regular Expression) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Returns a Boolean value that indicates whether or not a pattern exists in a searched string.

Syntax

```
rgExp.test(str)
```

Parameters

`rgExp`
Required. An instance of a **Regular Expression** object containing the regular expression pattern and applicable flags.

`str`
Required. The string on which to perform the search.

Remarks

The **test** method checks to see if a pattern exists within a string and returns **true** if so, and **false** otherwise.

The properties of the global `RegExp` object are not modified by the **test** method.

Example

The following example illustrates the use of the **test** method. To use this example, pass the function a regular expression pattern and a string. The function will test for the occurrence of the regular expression pattern in the string and return a string indicating the results of that search:

```
function TestDemo(re, teststring)
{
   // Test string for existence of regular expression.
   var found = re.test(teststring)

   // Format the output.
   var s = "";
   s += "'" + teststring + "'"

   if (found)
      s += " contains ";
   else
      s += " does not contain ";

   s += "'" + re.source + "'"
   return(s);
}
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards,

Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

Applies To: Regular Expression Object

See Also

RegExp Object
Regular Expression Object
Regular Expression Syntax (JavaScript)

Set Object (JavaScript)

10/18/2017 • 1 min to read • Edit Online

A collection of unique values that may be of any type.

Syntax

```
setObj = new Set()
```

Remarks

If you try to add a non-unique value to a `Set`, the new value will not be added to the collection.

Properties

The following table lists the properties of the `Set` object.

PROPERTY	DESCRIPTION
constructor	Specifies the function that creates a set.
prototype	Returns a reference to the prototype for a set.
size	Returns the number of elements in a set.

Methods

The following table lists the methods of the `Set` object.

METHOD	DESCRIPTION
add	Adds an element to a set.
clear	Removes all elements from a set.
delete	Removes a specified element from a set.
forEach	Performs the specified action for each element in a set.
has	Returns `true` if the set contains a specified element.
toString	Returns a string representation of a set.
valueOf	Returns the primitive value of the specified object.

Example

The following example shows how to add members to a set and then retrieve them.

```
var s = new Set();
s.add("Thomas Jefferson");
s.add(1776);
s.add("founding father");

s.forEach(function (item) {
    document.write(item.toString() + ", ");
});

// Output:
// Thomas Jefferson, 1776, founding father,
```

Requirements

Supported in the Internet Explorer 11 standards document mode. Also supported in Store apps (Windows 8.1 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards. Not supported in Windows 8.

constructor Property (Set)

10/18/2017 • 1 min to read • Edit Online

Specifies the function that creates a set.

Syntax

```
set.constructor
```

Remarks

The required `set` is the name of the set.

The `constructor` property is a member of the prototype of every object that has a prototype. This includes all intrinsic JavaScript objects except the `Global` and `Math` objects. The `constructor` property contains a reference to the function that constructs instances of that particular object.

Requirements

Supported in the Internet Explorer 11 standards document mode. Also supported in Store apps (Windows 8.1 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards. Not supported in Windows 8.

prototype Property (Set)

10/18/2017 • 1 min to read • Edit Online

Returns a reference to the prototype for a set.

Syntax

```
set.prototype
```

Remarks

The `set` argument is the name of a set.

Use the `prototype` property to provide a base set of functionality to a class of objects. New instances of an object "inherit" the behavior of the prototype assigned to that object.

For example, to add a method to the `Set` object that returns the value of the largest element of the set, declare the function, add it to `Set.prototype`, and then use it.

Requirements

Supported in the Internet Explorer 11 standards document mode. Also supported in Store apps (Windows 8.1 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards. Not supported in Windows 8.

size Property (Set) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Returns the number of elements in a set.

Syntax

```
sizeVar = setObj.size
```

Parameters

`sizeVar`

Required. Any number.

`setObj`

Required. Any `Set` object.

Requirements

Supported in the Internet Explorer 11 standards document mode. Also supported in Store apps (Windows 8.1 and Windows Phone 8.1). See Version Information.

Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards. Not supported in Windows 8.

add Method (Set) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Adds a new element to a set.

Syntax

```
setObj.add(value)
```

Parameters

`setObj`

Required. A `Set` object.

`value`

Required. New element of the `Set`.

Remarks

The new element can be of any type and must be unique. If you add a non-unique element to a `Set`, the new element will not be added to the collection.

Example

The following example shows how to add members to a set and then retrieve them.

```
var s = new Set();
s.add("Thomas Jefferson");
s.add(1776);
s.add("founding father");

s.forEach(function (item) {
    document.write(item.toString() + ", ");
});

// Output:
// Thomas Jefferson, 1776, founding father,
```

Requirements

Supported in the Internet Explorer 11 standards document mode. Also supported in Store apps (Windows 8.1 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards. Not supported in Windows 8.

clear Method (Set) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Removes all elements from the set.

Syntax

```
setObj.clear()
```

Parameters

`setObj`

Required. The set to clear.

Requirements

Supported in the Internet Explorer 11 standards document mode. Also supported in Store apps (Windows 8.1 and Windows Phone 8.1). See Version Information.

Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards. Not supported in Windows 8.

delete Method (Set) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Removes the specified element from a set.

Syntax

```
setObj.delete(value)
```

Parameters

`setObj`

Required. A `Set` object.

`value`

Required. The element to remove.

Property Value/Return Value

`true` if the element has been removed.

Example

The following example shows how to add members to a `Set` and then delete one of them.

```
var s = new Set();
s.add("Thomas Jefferson");
s.add(1776);
s.add("founding father");
s.delete("founding father");

s.forEach(function (item) {
    document.write(item.toString() + ", ");
});

// Output:
// Thomas Jefferson, 1776,
```

Requirements

Supported in the Internet Explorer 11 standards document mode. Also supported in Store apps (Windows 8.1 and Windows Phone 8.1). See Version Information.

Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards. Not supported in Windows 8.

forEach Method (Set) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Performs the specified action for each element in a set.

Syntax

```
setObj.forEach(callbackfn[, thisArg])
```

Parameters

`setObj`
Required. A `Set` object.

`callbackfn`
Required. `callbackfn` accepts up to three arguments. The function that `forEach` calls one time for each element in the set.

`thisArg`
Optional. An object that the `this` keyword can refer to in the `callbackfn` function. If `thisArg` is omitted, `undefined` is used as the `this` value.

Exceptions

If the `callbackfn` argument is not a function object, a `TypeError` exception is thrown.

Remarks

The syntax of the callback function is as follows:

`function callbackfn(value, key, setObj)`

You can declare the callback function by using up to three parameters, as shown in the following table.

CALLBACK ARGUMENT	DEFINITION
`value`	A value contained in the set.
`key`	A value contained in the set. A set has no keys, so this value is the same as `value`.
`setObj`	The `Set` object to traverse.

Example

The following example shows how to use `forEach`. The `callbackfn` argument includes the code for the callback function.

```
var s = new Set();

s.add("scale");
s.add(10);
s.add("5");

s.forEach(function(item, sameItem, s) {
    document.write("Size of the set object is: " + s.size + "<br />");
    document.write("Deleting item: " + item + "<br />");
    s.delete(sameItem);
});

// Output:
// Size of the set object is: 3
// Deleting item: scale
// Size of the set object is: 2
// Deleting item: 10
// Size of the set object is: 1
// Deleting item: 5
```

Example

The following example shows that you can also retrieve members from a set by passing only a single parameter to the callback function.

```
var s = new Set();
s.add("Thomas Jefferson");
s.add(1776);
s.add("founding father");

s.forEach(function (item) {
    document.write(item.toString() + ", ");
});

// Output:
// Thomas Jefferson, 1776, founding father,
```

Requirements

Supported in the Internet Explorer 11 standards document mode. Also supported in Store apps (Windows 8.1 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards. Not supported in Windows 8.

has Method (Set) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Returns `true` if the set contains the specified element.

Syntax

```
setObj.has(value)
```

Parameters

`setObj`
Required. A `Set` object.

`value`
Required. The element to test.

Property Value/Return Value

`true` if the set contains the specified element.

Example

The following example shows how to add members to a `Set` and then check whether the set contains a specific member.

```
var s = new Set();
s.add("Thomas Jefferson");
s.add(1776);

document.write(s.has(1776));
document.write(s.has("1776"));

// Output:
// true
// false
```

Requirements

Supported in the Internet Explorer 11 standards document mode. Also supported in Store apps (Windows 8.1 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards. Not supported in Windows 8.

toString Method (Set) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Returns a string representation of a set.

Syntax

```
setObj.toString()
```

Parameters

`setObj`

Required. A `Set` object.

Property Value/Return Value

The string representation of the set.

Exceptions

Requirements

Supported in the Internet Explorer 11 standards document mode. Also supported in Store apps (Windows 8.1 and Windows Phone 8.1). See Version Information.

Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards. Not supported in Windows 8.

valueOf Method (Set) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Returns the primitive value of the specified object.

Syntax

```
setObj.valueOf()
```

Parameters

`setObj`

Required. A `Set` object.

Property Value/Return Value

Returns the set instance.

Requirements

Supported in the Internet Explorer 11 standards document mode. Also supported in Store apps (Windows 8.1 and Windows Phone 8.1). See Version Information.

Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards. Not supported in Windows 8.

String Object (JavaScript)

10/18/2017 • 5 min to read • Edit Online

Allows manipulation and formatting of text strings and determination and location of substrings within strings.

Syntax

```
newString = new String(["stringLiteral"])
```

Parameters

`newString`
Required. The variable name to which the `String` object is assigned.

`stringLiteral`
Optional. Any group of Unicode characters.

Remarks

JavaScript provides escape sequences that you can include in strings to create characters that you cannot type directly. For example, `\t` specifies a tab character. For more information, see Special Characters.

String Literals

A *string literal* is zero or more characters enclosed in single or double quotation marks. A string literal has a primary (primitive) data type of `string`. A *String object* is created by using the new Operator, and has a data type of `Object`.

The following example shows that the data type of a string literal is not the same as that of a `String` object.

```
var strLit = "This is a string literal.";
var strObj = new String("This is a string object.");

document.write(typeof strLit);
document.write("<br/>");
document.write(typeof strObj);
// Output:
// string
// object
```

Methods for String Literals

When you call a method on a string literal, it is temporarily converted to a string wrapper object. The string literal is treated as though the `new` operator were used to create it.

The following example applies the `toUpperCase` method to a string literal.

```
var strLit = "This is a string literal.";

var result1 = strLit.toUpperCase();

var result2 = (new String(strLit)).toUpperCase();

document.write(result1);
document.write("<br/>");
document.write(result2);
// Output:
// THIS IS A STRING LITERAL.
// THIS IS A STRING LITERAL.
```

Accessing an Individual Character

You can access an individual character of a string as a read-only array-indexed property. This feature was introduced in Internet Explorer 9 standards mode, Internet Explorer 10 standards mode, Internet Explorer 11 standards mode, and Windows Store apps. The following example accesses individual string characters.

```
var str = "abcd";
var result = str[2];
document.write (result);
// Output: c

var result = "the"[0];
document.write(result);
// Output: t
```

Requirements

The `String Object` was introduced in Internet Explorer before Internet Explorer 6. Some members in the following lists were introduced in later versions.

Properties

The following table lists the properties of the `String` object.

PROPERTY	DESCRIPTION
constructor Property	Specifies the function that creates an object.
length Property (String)	Returns the length of a `String` object.
prototype Property	Returns a reference to the prototype for a class of objects.

Functions

The following table lists the functions of the `String` object.

FUNCTION	DESCRIPTION
String.fromCharCode Function	Returns a string from a number of Unicode character values.

FUNCTION	DESCRIPTION
String.fromCodePoint Function	Returns the string associated with a Unicode UTF-16 code point.
String.raw Function	Returns the raw string form of a template string.

Methods

The following table lists the methods of the `String` object.

METHOD	DESCRIPTION
anchor Method	Places an HTML anchor that has a NAME attribute around text.
big Method	Places HTML <BIG> tags around text.
blink Method	Places HTML <BLINK> tags around text.
bold Method	Places HTML <B> tags around text.
charAt Method	Returns the character at the specified index.
charCodeAt Method	Returns the Unicode encoding of the specified character.
codePointAt Method	Returns the code point for a Unicode UTF-16 character.
concat Method (String)	Returns a string that contains the concatenation of two supplied strings.
endsWith Method	Returns a Boolean value that indicates whether a string or substring ends with the passed in string.
includes Method	Returns a Boolean value that indicates whether the passed in string is contained in the string object.
fixed Method	Places HTML <TT> tags around text.
fontcolor Method	Places HTML <FONT> tags with a COLOR attribute around text.
fontsize Method	Places HTML <FONT> tags with a SIZE attribute around text.
hasOwnProperty Method	Returns a Boolean value that indicates whether an object has a property with the specified name.
indexOf Method (String)	Returns the character position where the first occurrence of a substring occurs within a string.
isPrototypeOf Method	Returns a Boolean value that indicates whether an object exists in another object's prototype chain.

METHOD	DESCRIPTION
italics Method	Places HTML <I> tags around text.
lastIndexOf Method (String)	Returns the last occurrence of a substring within a string.
link Method	Places an HTML anchor that has an HREF attribute around text.
localeCompare Method	Returns a value that indicates whether two strings are equivalent in the current locale.
match Method	Searches a string by using a supplied **Regular Expression** object and returns the results as an array.
normalize Method	Returns the Unicode Normalization Form of a specified string.
propertyIsEnumerable Method	Returns a Boolean value that indicates whether a specified property is part of an object and whether it is enumerable.
repeat Method	Returns a new string object with a value equal to the original string repeated the specified number of times.
replace Method	Uses a regular expression to replace text in a string and returns the result.
search Method	Returns the position of the first substring match in a regular expression search.
slice Method (String)	Returns a section of a string.
small Method	Places HTML <SMALL> tags around text.
split Method	Returns the array of strings that results when a string is separated into substrings.
startsWith Method	Returns a Boolean value that indicates whether a string or substring starts with the passed in string.
strike Method	Places HTML <STRIKE> tags around text.
sub Method	Places HTML <SUB> tags around text.
substr Method	Returns a substring beginning at a specified location and having a specified length.
substring Method	Returns the substring at a specified location within a `String` object.
sup Method	Places HTML <SUP> tags around text.
toLocaleLowerCase Method	Returns a string in which all alphabetic characters are converted to lowercase, taking into account the host environment's current locale.

METHOD	DESCRIPTION
toLocaleString Method	Returns an object converted to a string, using the current locale.
toLocaleUpperCase Method	Returns a string in which all alphabetic characters are converted to uppercase, taking into account the host environment's current locale.
toLowerCase Method	Returns a string in which all alphabetic characters are converted to lowercase.
toString Method	Returns the string.
toUpperCase Method	Returns a string in which all alphabetic characters are converted to uppercase.
trim Method	Returns a string with leading and trailing white space and line terminator characters removed.
valueOf Method	Returns the string.

See Also

new Operator
Scrolling, panning, and zooming sample app

constructor Property (String)

10/18/2017 • 1 min to read • Edit Online

Specifies the function that creates a string.

Syntax

```
string.constructor
```

Remarks

The required `string` is the name of a string.

The `constructor` property is a member of the prototype of every object that has a prototype. This includes all intrinsic JavaScript objects except the `Global` and `Math` objects. The `constructor` property contains a reference to the function that constructs instances of that particular object.

Example

The following example illustrates the use of the constructor property.

```
var x = new String();

if (x.constructor == String)
    document.write("Object is a String.");
else
    document.write("Object is not a String.");

// Output:
// Object is a String.
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

length Property (String) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Returns the length of a `String` object.

> **WARNING**
>
> JavaScript strings are immutable, so the length of a string cannot be modified.

Syntax

```
      strVariable.length
"String Literal".length
```

Remarks

The `length` property contains an integer that indicates the number of characters in the `String` object. The last character in the `String` object has an index of i `length` - 1.

Example

The following code shows how to use `length`. JavaScript strings are immutable and cannot be modified in place. However, you can write the reversed string to an array and then call `join` with the empty character, which produces a string with no separator characters.

```
var str = "every good boy does fine";
        var start = 0;
        var end = str.length - 1;
        var tmp = "";
        var arr = new Array(end);

        while (end >= 0) {
            arr[start++] = str.charAt(end--);
        }

// Join the elements of the array with a
        var str2 = arr.join('');
        document.write(str2);

// Output: enif seod yob doog yreve
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

prototype Property (String)

10/18/2017 • 1 min to read • Edit Online

Returns a reference to the prototype for a class of string.

Syntax

```
string.prototype
```

Remarks

The `string` argument is the name of a string.

Use the `prototype` property to provide a base set of functionality to a class of objects. New instances of an object "inherit" the behavior of the prototype assigned to that object.

For example, to add a method to the `String` object that returns the value of the last element of the string, declare the function, add it to `String.prototype`, and then use it.

```
function string_last( ){
    return this.charAt(this.length - 1);
}
String.prototype.last = string_last;
var myString = new String("every good boy does fine");
document.write(myString.last());

// Output:
// e
```

All intrinsic JavaScript objects have a `prototype` property that is read-only. Properties and methods may be added to the prototype, but the object may not be assigned a different prototype. However, user-defined objects may be assigned a new prototype.

The method and property lists for each intrinsic object in this language reference indicate which ones are part of the object's prototype, and which are not.

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

String.fromCharCode Function (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Returns a string from a number of Unicode character values.

Syntax

```
String.fromCharCode([code1[, code2[, ...[, codeN]]]])
```

Parameters

`String`
Required. The `String` object.

`code1`, ..., `codeN`
Optional. A series of Unicode character values to convert to a string. If no arguments are supplied, the result is the empty string.

Remarks

You call this function on the `String` object rather than on a string instance.

The following example shows how to use this method:

```
var test = String.fromCharCode(112, 108, 97, 105, 110);
document.write(test);

// Output: plain
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

See Also

charCodeAt Method (String)

String.fromCodePoint Function (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Returns the string associated with a Unicode UTF-16 code point.

Syntax

```
String.fromCodePoint(...codePoints);
```

Parameters

`...codePoints`

Required. A rest parameter that specifies one or more UTF-16 code point values.

Remarks

This function throws a `RangeError` exception if `...codePoints` is not a valid UTF-16 code point.

Example

The following example shows how to use the `fromCodePoint` function.

```
var str1 = String.fromCodePoint(0x20BB7);
var str2 = String.fromCodePoint(98);
var str3 = String.fromCodePoint(97, 98, 99);

if(console && console.log) {
    console.log(str1);
    console.log(str2);
    console.log(str3);
}

// Output:
// 吉
// b
// abc
```

Requirements

Supported in Microsoft Edge (Edge browser). Also supported in Store apps (Microsoft Edge on Windows 10). See Version Information.

Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Not supported in Windows 8.1.

String.raw Function (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Returns the raw string form of a template string.

Syntax

```
String.raw`templateStr`;
String.raw(obj, ...substitutions);
```

Parameters

`templateStr`
Required. The template string.

`obj`
Required. A well-formed object specified using object literal notation, such as { raw: 'value' }.

`...substitutions`
Optional. An array (a rest parameter) consisting of one or more substitution values.

Remarks

The `String.raw` function is intended for use with template strings. The raw string will include any escape characters and backslashes that are present in the string.

An error is thrown if `obj` is not a well-formed object.

Example

```
function log(arg) {
    if(console && console.log) {
        console.log(arg);
    }
};

var name = "bob";

log(`hello \t${name}`);
log(String.raw`hello \t${name}`);
// The following usage for String.raw is supported but
// is not typical.
log(String.raw({ raw: 'fred'}, 'F', 'R', 'E'));

// Output:
// hello   bob
// hello \tbob
// fFrReEd
```

Requirements

Supported in Microsoft Edge (Edge browser). Also supported in Store apps (Microsoft Edge on Windows 10). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7

standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Not supported in Windows 8.1.

HTML Tag Methods (JavaScript)

10/18/2017 • 3 min to read • Edit Online

You can use HTML tag methods to place HTML elements around text in a `String` object.

Syntax

The following table lists the syntax for and a description of each HTML tag method.

In the Syntax column, `string1` is a `String` object or literal.

The Standard column indicates World Wide Web Consortium (W3C) recommendations for HTML 4. "Discouraged" indicates that the HTML element is discouraged in favor of style sheets.

SYNTAX	METHOD DESCRIPTION	PARAMETER DESCRIPTION	STANDARD
`string1`.anchor(`name`)	Places an HTML anchor that has a NAME attribute around the text.	The `name` parameter is text to put in the NAME attribute of the HTML anchor.	
`string1`.big()	Places HTML <BIG> tags around the text.		Discouraged
`string1`.blink()	Places HTML <BLINK> tags around the text. The <BLINK> tag is not supported in Internet Explorer.		Not in standard
`string1`.bold()	Places HTML <B> tags around the text.		Discouraged
`string1`.fixed()	Places HTML <TT> tags around the text.		Discouraged
`string1`.fontcolor(`color`)	Places HTML <FONT> tags with a COLOR attribute around the text.	The `color` parameter is a string value that contains the hexadecimal value or predefined name for a color. Valid predefined color names depend on the JavaScript host browser and its version.	Deprecated
`string1`.fontsize(`size`)	Places HTML <FONT> tags with a SIZE attribute around the text.	The `size` parameter is an integer value that specifies the size of the text. Valid integer values depend on the JavaScript host browser and its version.	Deprecated

SYNTAX	METHOD DESCRIPTION	PARAMETER DESCRIPTION	STANDARD
`string1`.italics()	Places HTML <I> tags around the text.		Discouraged
`string1`.link(`href`)	Places an HTML anchor that has an HREF attribute around the text.	The `href` parameter is text to put in the HREF attribute of the HTML anchor.	
`string1`.small()	Places HTML <SMALL> tags around the text.		Discouraged
`string1`.strike()	Places HTML <STRIKE> tags around the text.		Deprecated
`string1`.sub()	Places HTML <SUB> tags around the text.		
`string1`.sup()	Places HTML <SUP> tags around the text.		

Remarks

No checking is performed to determine whether the HTML tags have already been applied to the string.

Example

The following examples show how to use the HTML tag methods.

```
// anchor method.
var strVariable = "This is an anchor.";
document.write(strVariable.anchor("Anchor1"));
// Output: <A NAME="Anchor1">This is an anchor.</A>

// big method.
var strVariable = "This is a string.";
document.write(strVariable.big());
// Output: <BIG>This is a string.</BIG>

//  blink method.
var strVariable = "This is a string.";
document.write(strVariable.blink());
// Output: <BLINK>This is a string.</BLINK>

//  bold method.
var strVariable = "This is a string.";
document.write(strVariable.bold());
// Output: <B>This is a string.</B>

//  fixed method.
var strVariable = "This is a string.";
document.write(strVariable.fixed());
// Output: <TT>This is a string.</TT>

//  fontcolor method.
var strVariable = "This is a string.";
document.write(strVariable.fontcolor("red"));
// Output: <FONT COLOR="red">This is a string.</FONT>

//  fontsize method.
var strVariable = "This is a string.";
document.write(strVariable.fontsize(-1));
// Output: <FONT SIZE="-1">This is a string.</FONT>

//  italics method
var strVariable = "This is a string.";
document.write(strVariable.italics());
// Output: <I>This is a string.</I>

//  link method.
var strVariable = "This is a hyperlink.";
document.write(strVariable.link("http://www.microsoft.com"));
// Output: <A HREF="http://www.microsoft.com">This is a hyperlink.</A>

//  small method.
var strVariable = "This is a string.";
document.write(strVariable.small());
// Output: <SMALL>This is a string.</SMALL>

//  strike method.
var strVariable = "This is a string.";
document.write(strVariable.strike());
// Output: <STRIKE>This is a string.</STRIKE>

//  sub method.
var strVariable = "This is a string.";
document.write(strVariable.sub());
// Output: <SUB>This is a string.</SUB>

//  sup method.
var strVariable = "This is a string.";
document.write(strVariable.sup());
// Output: <SUP>This is a string.</SUP>
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

Applies To: String Object

See Also

String Object

charAt Method (String) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Returns the character at the specified index.

Syntax

```
strObj. charAt(index)
```

Parameters

`strObj`
Required. Any `String` object or string literal.

`index`
Required. The zero-based index of the desired character.

Remarks

The `charAt` method returns a character value equal to the character at the specified `index`. The first character in a string is at index 0, the second is at index 1, and so forth. Values of `index` that are out of range return an empty string.

Example

The following example illustrates the use of the `charAt` method:

```
var str = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
document.write(str.charAt(str.length - 1));

// Output: Z
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

Applies To: String Object

See Also

String Object

charCodeAt Method (String) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Returns the Unicode value of the character at the specified location.

Syntax

```
strObj. charCodeAt(index)
```

Parameters

`strObj`
Required. Any `String` object or string literal.

`index`
Required. The zero-based index of the desired character. If there is no character at the specified index, `NaN` is returned.

Remarks

Example

The following example illustrates the use of the `charCodeAt` method.

```
var str = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
document.write(str.charCodeAt(str.length - 1));

// Output: 90
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

See Also

String.fromCharCode Function

codePointAt Method (String) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Returns the code point for a Unicode UTF-16 character.

Syntax

```
stringObj.codePointAt(pos);
```

Parameters

`stringObj`
Required. The string object.

`pos`
Required. The position of the character.

Remarks

This method returns code point values, including astral code points (code points with more than four hexadecimal values), for all UTF-16 characters.

If `pos` is less than zero (0) or greater than the size of the string, the return value is `undefined`.

Example

The following example shows how to use the `codePointAt` method.

```
var cp1 = "吉".codePointAt(0);
var cp2 = 'abc'.codePointAt(1);

if(console && console.log) {
    console.log(cp1);
    console.log(cp2);}

// Output:
// 0x20BB7
// 98
```

Requirements

Supported in Microsoft Edge (Edge browser). Also supported in Store apps (Microsoft Edge on Windows 10). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Not supported in Windows 8.1.

concat Method (String) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Returns a string that contains the concatenation of two or more strings.

Syntax

```
string1. concat([string2[, string3[, . . . [, stringN]]]])
```

Parameters

`string1`
Required. The `String` object or string literal to which all other specified strings are concatenated.

`string2,. . ., stringN`
Optional. The strings to append to the end of `string1`.

Remarks

The result of the `concat` method is equivalent to: `result` = `string1` + `string2` + `string3` + `stringN`. A change of value in either a source or result string does not affect the value in the other string. If any of the arguments are not strings, they are first converted to strings before being concatenated to `string1`.

Example

The following example illustrates the use of the `concat` method when used with a string:

```
var str1 = "ABCD"
var str2 = "EFGH";
var str3 = "1234";
var str4 = "5678";
document.write(str1.concat(str2, str3, str4));

// Output: "ABCDEFGH12345678"
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

Applies To: String Object

See Also

Addition Operator (+)

endsWith Method (String) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Returns a value that indicates whether a string or substring ends with another specified string.

Syntax

```
stringObj.endsWith(str, [, position]);
```

Parameters

`stringObj`
Required. The string object to search against.

`str`
Required. The search string.

`position`
Optional. The position of the first character to search against in the string object, starting at 0.

Return Value

If the string beginning at `position` ends with the search string, the `endsWith` method returns `true`; otherwise, it returns `false`.

Exceptions

If `str` is a `RegExp`, a `TypeError` is thrown.

Remarks

Requirements

Supported in Microsoft Edge (Edge browser). Also supported in Store apps (Microsoft Edge on Windows 10). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Not supported in Windows 8.1.

includes Method (String) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Returns a Boolean value that indicates whether the passed in string is contained in the string object.

Syntax

```
stringObj.includes(substring [, position]);
```

Parameters

`stringObj`

Required. The string object to test against.

`substring`

Required. The string to test.

`position`

Optional. The position of the first character to test against in the string object, starting with 0.

Return Value

If the passed in string is contained in the string object, the `includes` method returns `true`; otherwise, it returns `false`.

Remarks

Example

```
// Returns true
"abcde".includes("cd")
"abcde".includes("cd", 2)

// Returns false
"abcde".includes("CD")
"abcde".includes("cd", 3)
```

Requirements

Supported in Microsoft Edge (Edge browser). Also supported in Store apps (Microsoft Edge on Windows 10). See Version Information.

Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Not supported in Windows 8.1.

indexOf Method (String) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Returns the position of the first occurrence of a substring.

Syntax

```
strObj. indexOf(subString[, startIndex])
```

Parameters

`strObj`
Required. A `String` object or string literal.

`subString`
Required. The substring to search for in the string

`startIndex`
Optional. The index at which to begin searching the `String` object. If omitted, search starts at the beginning of the string.

Remarks

The **indexOf** method returns the beginning of the substring in the `String` object. If the substring is not found, -1 is returned.

If `startindex` is negative, `startindex` is treated as zero. If it is greater than the highest index, it is treated as the highest index.

Searching is performed from left to right. Otherwise, this method is identical to **lastIndexOf**.

Example

The following example illustrates the use of the **indexOf** method.

```
var str = "original equipment manufacturer";

var s = "equip is at position " + str.indexOf("equip");
s += "<br />";
s += "abc is at position " + str.indexOf("abc");

document.write(s);

// Output:
// equip is at position 9
// abc is at position -1
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards,

Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

Applies To: String Object

See Also

lastIndexOf Method (String)
Scrolling, panning, and zooming sample app

lastIndexOf Method (String) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Returns the last occurrence of a substring in the string.

Syntax

```
strObj.lastIndexOf(substring[, startindex])
```

Parameters

`strObj`
Required. A `String` object or string literal.

`substring`
Required. The substring to search for.

`startindex`
Optional. The index at which to begin searching. If omitted, the search begins at the end of the string.

Remarks

The **lastIndexOf** method returns an integer value indicating the beginning of the substring within the `String` object. If the substring is not found, a -1 is returned.

If `startindex` is negative, `startindex` is treated as zero. If it is larger than the greatest character position index, it is treated as the largest possible index.

Searching is performed starting at the last character in the string. Otherwise, this method is identical to **indexOf**.

The following example illustrates the use of the **lastIndexOf** method.

```
var str = "time, time";

var s = "";
s += "time is at position " + str.lastIndexOf("time");
s += "<br />";
s += "abc is at position " + str.lastIndexOf("abc");

document.write(s);

// Output:
// time is at position 6
// abc is at position -1
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

Applies To: String Object

See Also

indexOf Method (String)

localeCompare Method (String) (JavaScript)

10/18/2017 • 2 min to read • Edit Online

Determines whether two strings are equivalent in the current locale.

Syntax

```
stringVar.localeCompare(stringExp[, locales][, options])
```

Parameters

`stringVar`
Required. The first string to compare.

`stringExp`
Required. The second string to compare.

`locales`
Optional. An array of locale strings that contain one or more language or locale tags. If you include more than one locale string, list them in descending order of priority so that the first entry is the preferred locale. If you omit this parameter, the default locale of the JavaScript runtime is used. This parameter must conform to BCP 47 standards; see the Intl.Collator object for details.

`options`
Optional. An object that contains one or more properties that specify comparison options. see the Intl.Collator object for details.

Remarks

For the comparison strings, you may specify either `String` objects or string literals.

Starting in Internet Explorer 11, `localeCompare` uses the `Intl.Collator` object internally to make comparisons, which adds support for the `locales` and `options` parameters. For more information about these parameters, see Intl.Collator and Intl.Collator.compare.

> **IMPORTANT**
> The `locales` and `options` parameters are not supported in all document modes and browser versions. For more information, see the Requirements section.

The `localeCompare` method performs a locale-sensitive string comparison of `stringVar` and `stringExp` and returns one of the following results, depending on the sort order of the system default locale:

- -1 if `stringVar` sorts before `stringExp`.
- +1 if `stringVar` sorts after `stringExp`.
- 0 (zero) if the two strings are equivalent.

Example

The following code shows how to use `localeCompare`.

```
var str1 = "def";
var str2 = "abc"

document.write(str1.localeCompare(str2) + "<br/>");

// Output: 1
var str3 = "ghi";

document.write(str1.localeCompare(str3)+ "<br/>");

// Output: -1
var str4 = "def";

document.write(str1.localeCompare(str4));

// Output: 0
```

Example

The following code shows how to use `localeCompare` with the German (Germany) locale.

```
var str1 = "a";
var str2 = "b";

document.write(str1.localeCompare(str2, "de-DE"));

// Output
// - 1
```

Example

The following example shows how to use `localeCompare` with the German (Germany) locale and a locale-specific extension that specifies the sort order for German phone books. This example shows locale-specific differences.

```
var arr = ["ä", "ad", "af", "a"];

document.write(arr[0].localeCompare(arr[1], "de-DE-u-co-phonebk"));  // Returns 1
document.write (arr[0].localeCompare(arr[2], "de-DE-u-co-phonebk"));  // Returns -1
document.write (arr[0].localeCompare(arr[3], "de-DE-u-co-phonebk"));  // Returns 1

document.write (arr[0].localeCompare(arr[1], "de-DE"));  // Returns -1
document.write (arr[0].localeCompare(arr[2], "de-DE"));  // Returns -1
document.write (arr[0].localeCompare(arr[3], "de-DE"));  // Returns 1
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

`locales` and `options` parameters:

Supported in the Internet Explorer 11 standards document mode. Also supported in Store apps (Windows 8.1 and Windows Phone 8.1). See Version Information.

Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards. Not supported in Windows 8.

See Also

toLocaleString Method (Object)

match Method (String) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Matches a string with a regular expression, and returns an array containing the results of that search.

Syntax

```
stringObj.match(rgExp)
```

Parameters

`stringObj`
Required. The `String` object or string literal on which to perform the search.

`rgExp`
Required. A regular expression object that contains the regular expression pattern and applicable flags. This can also be a variable name or string literal containing the regular expression pattern and flags.

Remarks

If the `match` method does not find a match, it returns `null`. If it finds a match, `match` returns an array, and the properties of the global `RegExp` object are updated to reflect the results of the match.

If the global flag (`g`) is not set, Element zero of the array contains the entire match, while elements 1 through *n* contain any submatches. This behavior is the same as the behavior of the exec Method (Regular Expression) when the global flag is not set. If the global flag is set, elements 0 through *n* contain all matches that occurred.

If the global flag is not set, the array returned by the `match` method has two properties, `input` and `index`. The `input` property contains the entire searched string. The `index` property contains the position of the matched substring within the complete searched string.

If the flag `i` is set, the search is not case-sensitive.

Example

The following example illustrates the use of the `match` method.

```
var src = "azcafAJAC";

var re = /[a-c]/;

var result = src.match(re);

// The entire match is in array element 0.
document.write(result[0] + "<br/>");

// Now try the same match with the global flag.
var reg = /[a-c]/g;
result = src.match(reg);

// The matches are in elements 0 through n.
for (var index = 0; index < result.length; index++)
{
    document.write ("submatch " + index + ": " +  result[index]);
    document.write("<br />");
}
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

Applies To: String Object

See Also

exec Method (Regular Expression)
RegExp Object
Regular Expression Object
replace Method (String)
search Method (String)
test Method (Regular Expression)
Regular Expression Programming (JavaScript)
Alternation and Subexpressions (JavaScript)
Backreferences (JavaScript)

normalize Method (String) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Returns the Unicode Normalization Form of a specified string.

Syntax

```
stringObj.normalize([form]);
```

Parameters

`stringObj`

Required. The string object to test against.

`form`

Optional. The Unicode Normalization Form value.

Return Value

The Unicode Normalization Form of a specified string.

Exceptions

If `form` is an unsupported value, a `RangeError` is thrown.

Remarks

If `stringObj` isn't a string, it will be converted to a string before the method attempts to return the Unicode Normalization Form of the string.

`form` must be a Unicode Normalization Form value, "NFC", "NFD", "NFKC", or "NFKD", corresponding to values specified for Unicode Standard Annex #15. The default value of `form` is "NFC".

Example

The following code examples show the use of the `normalize` method.

```
// ANGSTORM SIGN and LATIN CAPITAL A WITH RING ABOVE is canonically equivalent
"\u212b".normalize("NFC") === "\u00c5";

// Decomposed, ANGSTOM SIGN is LATIN CAPITAL A followed by COMBINING RING ABOVE
"\u212b".normalize("NFD") === "\u0041\u030a"

// Normalization Form C will combine the result back into the precombined character
"\u0041\u030a".normalize("NFC") === "\u00c5"

// LATIN SMALL LIGATURE FI is compatibility equivalent with LATIN SMALL LETTER F followed by
// LATIN SMALL LETTER I.
"\ufb01".normalize("NFKD") === "fi";

// Same mapping in NFKC
"\ufb01".normalize("NFKC") === "fi";

// NFKC will not recombine compatibility characters.
"fi".normalize("NFKC") === "fi";
```

Requirements

Supported in Microsoft Edge (Edge browser). Also supported in Store apps (Microsoft Edge on Windows 10). See Version Information.

Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Not supported in Windows 8.1.

repeat Method (String) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Returns a new string object with a value equal to the original string repeated the specified number of times.

Syntax

```
stringObj.repeat(count);
```

Parameters

`stringObj`

Required. The string object.

`count`

Required. The number of times to repeat the original string in the returned string.

Exceptions

This method throws a RangeError if and only if the argument is negative or +Infinity.

Remarks

The `repeat` method concatenates the original string to the new string the number of times specified by `count`.

This method throws an error if `count` is not a positive number less than `Infinity`.

Example

```
"abc".repeat(3); // Returns "abcabcabc"
"abc".repeat(0); // Returns an empty string.
```

Requirements

Supported in Microsoft Edge (Edge browser). Also supported in Store apps (Microsoft Edge on Windows 10). See Version Information.

Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Not supported in Windows 8.1.

replace Method (String) (JavaScript)

1/9/2018 • 3 min to read • Edit Online

Replaces text in a string, using a regular expression or search string.

Syntax

```
stringObj.replace(rgExp, replaceText)
```

Parameters

`stringObj`
Required. The `String` object or string literal on which to perform the replacement. This string is not modified by the **replace** method. Rather, the return value of this method is the string produced by the replacement.

`rgExp`
Required. An instance of a **Regular Expression** object containing the regular expression pattern and applicable flags. Can also be a `String` object or string literal that represents the regular expression. If `rgExp` is not an instance of a **Regular Expression** object, it is converted to a string, and an exact search is made for the results; no attempt is made to convert the string into a regular expression.

`replaceText`
Required. A `String` object or string literal containing the text to replace for every successful match of `rgExp` in `stringObj`. In Internet Explorer 5.5 or later, the `replaceText` argument can also be a function that returns the replacement text.

Return Value

The result of the **replace** method is a copy of `stringObj` after the specified replacements have been made.

Remarks

Any of the following match variables can be used to identify the most recent match and the string from which it came. The match variables can be used in text replacement where the replacement string has to be determined dynamically.

CHARACTERS	MEANING
$$	`$` (Internet Explorer 5.5 or later)
$&	Specifies that portion of `stringObj` that the entire pattern matched. (Internet Explorer 5.5 or later)
`` `$``	Specifies that portion of `stringObj` that precedes the match described by **$&**. (Internet Explorer 5.5 or later)
`$'`	Specifies that portion of `stringObj` that follows the match described by **$&**. (Internet Explorer 5.5 or later)

CHARACTERS	MEANING
`$` ***n***	The *n*th captured submatch, where *n* is a single decimal digit from 1 through 9. (Internet Explorer 5.5 or later)
`$` ***nn***	The *nn*th captured submatch, where *nn* is a two-digit decimal number from 01 through 99. (Internet Explorer 5.5 or later)

If `replaceText` is a function, for each matched substring the function is called with the following *m* + 3 arguments where *m* is the number of left capturing parentheses in the `rgExp`. The first argument is the substring that matched. The next *m* arguments are all of the captures that resulted from the search. Argument *m* + 2 is the offset within `stringObj` where the match occurred, and argument *m* + 3 is `stringObj`. The result is the string value that results from replacing each matched substring with the corresponding return value of the function call.

The **replace** method updates the properties of the global `RegExp` object.

Example

The following example illustrates the use of the **replace** method to replace all instances of "the" with "a."

```
var s = "the batter hit the ball with the bat";

// Replace "the" with "a".
var re = /the/g;
var result = s.replace(re, "a");
document.write(result);
// Output: a batter hit a ball with a bat
```

In addition, the **replace** method can also replace subexpressions in the pattern. The following example exchanges each pair of words in the string.

```
var s = "The quick brown fox jumped over the lazy dog.";
var re = /(\S+)(\s+)(\S+)/g;
// Exchange each pair of words.
var result = s.replace(re, "$3$2$1");
document.write(result);

// Output:  quick The fox brown over jumped lazy the dog.
```

The following example, which works in Internet Explorer 5.5 and later, shows how to use a function that returns the replacement text. It replaces any instance of a number followed by "F" with a Celsius conversion.

```
function f2c(s1) {
    // Initialize pattern.
    var test = /(\d+(\.\d*)?)F\b/g;

    // Use a function for the replacement.
    var s2 = s1.replace(test,
      function($0,$1,$2)
           {
           return((($1-32) * 5/9) + "C");
           }
        )
    return s2;
}
document.write(f2c("Water freezes at 32F and boils at 212F."));

// Output: Water freezes at 0C and boils at 100C.
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

Applies To: String Object

See Also

exec Method (Regular Expression)
match Method (String)
RegExp Object
search Method (String)
test Method (Regular Expression)
Regular Expression Programming (JavaScript)
Alternation and Subexpressions (JavaScript)
Backreferences (JavaScript)
HTML5 drag and drop sample app

search Method (String) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Finds the first substring match in a regular expression search.

Syntax

```
stringObj.search(rgExp)
```

Parameters

`stringObj`
Required. The `String` object or string literal on which to perform the search.

`rgExp`
Required. An instance of a **Regular Expression** object containing the regular expression pattern and applicable flags.

Return Value

If a match is found, the **search** method returns an integer value that indicates the offset from the beginning of the string where the first match occurred. If no match is found, it returns -1.

Remarks

You can also set the `i` flag that causes the search to be case-insensitive.

Example

The following example illustrates the use of the **search** method.

```
var src = "is but a Dream within a dream";
var re = /dream/;
var pos = src.search(re);
document.write(pos);
document.write("<br/>");

re = /dream/i;
pos = src.search(re);
document.write(pos);

// Output:
// 24
// 9
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

Applies To: String Object

See Also

exec Method (Regular Expression)
match Method (String)
Regular Expression Object
Regular Expression Syntax (JavaScript)
replace Method (String)
test Method (Regular Expression)
Regular Expression Programming (JavaScript)

slice Method (String) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Returns a section of a string.

Syntax

```
stringObj.slice(start, [end])
```

Parameters

`stringObj`
Required. A `String` object or string literal.

`start`
Required. The index to the beginning of the specified portion of `stringObj`.

`end`
Optional. The index to the end of the specified portion of `stringObj`. The substring includes the characters up to, but not including, the character indicated by `end`. If this value is not specified, the substring continues to the end of `stringObj`.

Remarks

The `slice` method returns a `String` object containing the specified portion of `stringObj`.

The `slice` method copies up to, but not including, the character indicated by `end`.

If `start` is negative, it is treated as *length* + `start` where *length* is the length of the string. If `end` is negative, it is treated as *length* + `end`. If `end` is omitted, copying continues to the end of `stringObj`. If `end` occurs before `start`, no characters are copied to the new string.

Example

In the first example, the `slice` method returns the entire string. In the second example, the `slice` method returns the entire string, except for the last character.

```
var str1 = "all good boys do fine";

var slice1 = str1.slice(0);
var slice2 = str1.slice(0,-1);
var slice3 = str1.slice(4);
var slice4 = str1.slice(4, 8);

document.write(slice1 + "<br/>");
document.write(slice2 + "<br/>");
document.write(slice3 + "<br/>");
document.write(slice4);

// Output:
// all good boys do fine
// all good boys do fin
// good boys do fine
// good
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

Applies To: String Object

See Also

slice Method (Array)

split Method (String) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Split a string into substrings using the specified separator and return them as an array.

Syntax

```
stringObj.split([separator[, limit]])
```

Parameters

`stringObj`
Required. The `String` object or string literal to be split. This object is not modified by the **split** method.

`separator`
Optional. A string or a **Regular Expression** object that identifies character or characters to use in separating the string. If omitted, a single-element array containing the entire string is returned.

`limit`
Optional. A value used to limit the number of elements returned in the array.

Return Value

The result of the **split** method is an array of strings split at each point where `separator` occurs in `stringObj`. The `separator` is not returned as part of any array element.

Example

The following example illustrates the use of the **split** method.

```
var s = "The quick brown fox jumps over the lazy dog.";
var ss = s.split(" ");
for (var i in ss) {
    document.write(ss[i]);
    document.write("<br/>");
}

// Output:
// The
// quick
// brown
// fox
// jumps
// over
// the
// lazy
// dog.
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards,

Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

Applies To: String Object

See Also

concat Method (String)
RegExp Object
Regular Expression Object
Regular Expression Syntax (JavaScript)
Scrolling, panning, and zooming sample app

startsWith Method (String) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Returns a value that indicates whether a string or substring starts with another specified string.

Syntax

```
stringObj.startsWith(str, [, position]);
```

Parameters

`stringObj`
Required. The string object to search against.

`str`
Required. The search string.

`position`
Optional. The position of the first character to search against in the string object, starting at 0.

Return Value

If the string beginning at `position` starts with the search string, the `startsWith` method returns `true`; otherwise, it returns `false`.

Exceptions

If `str` is a `RegExp`, a `TypeError` is thrown.

Remarks

Requirements

Supported in Microsoft Edge (Edge browser). Also supported in Store apps (Microsoft Edge on Windows 10). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Not supported in Windows 8.1.

substr Method (String) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Gets a substring beginning at the specified location and having the specified length.

Syntax

```
stringvar.substr(start [, length ])
```

Parameters

`stringvar`
Required. A string literal or `String` object from which the substring is extracted.

`start`
Required. The starting position of the desired substring. The index of the first character in the string is zero.

`length`
Optional. The number of characters to include in the returned substring.

Remarks

If `length` is zero or negative, an empty string is returned. If not specified, the substring continues to the end of `stringvar`.

Example

The following example illustrates the use of the `substr` method.

```
var s = "The quick brown fox jumps over the lazy dog.";
var ss = s.substr(10, 5);
document.write("[" + ss + "] <br>");

ss = s.substr(10);
document.write("[" + ss + "] <br>");

ss = s.substr(10, -5);
document.write("[" + ss + "] <br>");

// Output:
// [brown]
// [brown fox jumps over the lazy dog.]
// []
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

Applies To: String Object

See Also

substring Method (String)

substring Method (String) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Returns the substring at the specified location within a `String` object.

Syntax

```
      strVariable. substring(start [, end])
"String Literal".substring(start [, end])
```

Parameters

`start`
Required. The zero-based index integer indicating the beginning of the substring.

`end`
Optional. The zero-based index integer indicating the end of the substring. The substring includes the characters up to, but not including, the character indicated by `end`.

If `end` is omitted, the characters from `start` through the end of the original string are returned.

Remarks

The `substring` method returns a string containing the substring from `start` up to, but not including, `end`.

The **substring** method uses the lower value of `start` and `end` as the beginning point of the substring. For example, strvar.substring(0, 3) and strvar.substring(3, 0) return the same substring.

If either `start` or `end` is `NaN` or negative, it is replaced with zero.

The length of the substring is equal to the absolute value of the difference between `start` and `end`. For example, the length of the substring returned in strvar.substring(0, 3) and strvar.substring(3, 0) is three.

Example

The following example illustrates the use of the **substring** method.

```
var s = "The quick brown fox jumps over the lazy dog.";
var ss = s.substring(10, 15);
document.write(ss);

// Output:
// brown
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

See Also

substr Method (String)

toLocaleLowerCase Method (String) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Converts all alphabetic characters to lowercase, taking into account the host environment's current locale.

Syntax

```
stringVar.toLocaleLowerCase( )
```

Remarks

The required `stringVar` reference is a `String` object or string literal.

The `toLocaleLowerCase` method converts the characters in a string, taking into account the host environment's current locale. In most cases, the result is the same as the result of the `toLowerCase` method. Results differ if the rules for a language conflict with the regular Unicode case mappings.

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

Applies To: String Object

See Also

toLocaleUpperCase Method (String)
toLowerCase Method

toLocaleUpperCase Method (String) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Returns a string where all alphabetic characters have been converted to uppercase, taking into account the host environment's current locale.

Syntax

```
stringVar.toLocaleUpperCase( )
```

Remarks

The required `stringVar` reference is a `String` object or string literal.

The `toLocaleUpperCase` method converts the characters in a string, taking into account the host environment's current locale. In most cases, the result is the same as the result the `toUpperCase` method. Results differ if the rules for a language conflict with the regular Unicode case mappings.

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

Applies To: String Object

See Also

toLocaleLowerCase Method (String)
toUpperCase Method (String)

toLowerCase Method (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Converts all the alphabetic characters in a string to lowercase.

Syntax

```
      strVariable.toLowerCase()
"String Literal".toLowerCase()
```

Remarks

The `toLowerCase` method has no effect on non-alphabetic characters.

The following example demonstrates the effects of the `toLowerCase` method:

```
var str1 = "This is a STRING.";
var str2 = str1. toLowerCase();
document.write(str2);

// Output: this is a string.
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

Applies To: String Object

See Also

toLocaleLowerCase Method (String)
toUpperCase Method (String)

toUpperCase Method (String) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Converts all the alphabetic characters in a string to uppercase.

Syntax

```
      strVariable.toUpperCase()
"String Literal".toUpperCase()
```

Remarks

The `toUpperCase` method has no effect on non-alphabetic characters.

Example

The following example demonstrates the effects of the `toUpperCase` method:

```
var str1 = "This is a STRING.";
var str2 = str1.toUpperCase();
document.write(str2);

// Output: THIS IS A STRING.
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

Applies To: String Object

See Also

toLocaleUpperCase Method (String)
toLowerCase Method

trim Method (String) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Removes the leading and trailing white space and line terminator characters from a string.

Syntax

```
stringObj.trim()
```

Parameters

`stringObj`
Required. A `String` object or string literal. This string is not modified by the `trim` method.

Return Value

The original string with leading and trailing white space and line terminator characters removed.

Remarks

The characters that are removed include space, tab, form feed, carriage return, and line feed. See Special Characters for a comprehensive list of white space and line terminator characters.

For an example that shows how to implement your own trim method, see Prototypes and Prototype Inheritance.

Example

The following example illustrates the use of the `trim` method.

```
var message = "    abc def     \r\n  ";

document.write("[" + message.trim() + "]");
document.write("<br/>");
document.write("length: " + message.trim().length);

// Output:
//  [abc def]
//  length: 7
```

Requirements

Supported in the following document modes: Internet Explorer 9 standards, Internet Explorer 10 standards, and Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards.

See Also

Special Characters
String Object
Scrolling, panning, and zooming sample app

toString Method (String) 1

10/18/2017 • 1 min to read • Edit Online

Returns a string representation of a string.

Syntax

```
string.toString()
```

Parameters

`string`

Required. The array to represent as a string.

Return Value

The string representation of the string.

Example

The following example illustrates the use of the **toString** method with a string.

```
var string = "this is a test";
var strStr = string.toString();
document.write(strStr);

// Output: this is a test
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

valueOf Method (String)

10/18/2017 • 1 min to read • Edit Online

Returns the string.

Syntax

```
string.valueOf()
```

Parameters

This method has no parameters.

Return Value

Returns the string value.

Remarks

In the following example, the string object is the same as the return value.

```
var str = "every good boy does fine";
var strStr = str.valueOf();

if (str === strStr)
document.write("same");
else
document.write("different");

// Output:
// same
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

Symbol Object (JavaScript)

10/18/2017 • 2 min to read • Edit Online

Allows you to create a unique identifier.

Syntax

```
obj = Symbol(desc)
```

Parameters

`desc`

Optional. The description of the symbol.

Remarks

Symbol objects allow properties to be added to existing objects with no possibility of interference with the existing object properties, with no unintended visibility, and with no other uncoordinated additions by other code.

Symbol is a primitive data type. Symbol objects cannot be created using the `new` operator.

To add symbol objects the global symbol registry, use the `Symbol.for` and `Symbol.keyFor` functions. If you serialize symbols as strings, use the global symbol registry to make sure that a particular string maps to the correct symbol for all lookups.

JavaScript includes the following built-in symbol values that are available in the global scope.

SYMBOL	DESCRIPTION
`Symbol.hasInstance`	This method determines whether a constructor object recognizes an object as one of the constructor's instances. It is used internally by the `instanceof` operator.
`Symbol.isConcatSpreadable`	This property returns a Boolean value that indicates whether an object should be flattened to its array elements by `Array.concat`.
`Symbol.iterator`	This method returns the default iterator for an object. It is used internally by the `for...of` statement.
`Symbol.toPrimitive`	This method converts an object to a corresponding primitive value. It is used internally by the `ToPrimitive` abstract operation.
`Symbol.toStringTag`	This property returns a string value that is used to help create the default string description of an object. It is used internally by the built-in method `Object.toString` method.
`Symbol.unscopables`	This property returns an object whose properties are excluded from the `with` environment bindings of the associated object.

Functions

The following table lists the functions of the `Symbol` object.

Property	Description
Symbol.for	Returns the symbol for a specified key or, if the key is not present, creates a new symbol object with the new key.
Symbol.keyFor	Returns the key for a specified symbol.

Example

```
(function() {

    // module-scoped symbol
    var key = Symbol("description");

    function MyClass(privateData) {
      this[key] = privateData;
    }

    MyClass.prototype = {
      someFunc: function() {
        return "data: " + this[key];
      }
    };

    var c = new MyClass("private data")
    console.log(key);
    console.log(c["key"]);
    console.log(c.someFunc());

})();

// Output:
// undefined
// undefined
// data: private data
```

Requirements

Supported in Microsoft Edge (Edge browser). Also supported in Store apps (Microsoft Edge on Windows 10). See Version Information.

Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Not supported in Windows 8.1.

Symbol.for Function (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Returns the symbol for a specified key or, if the key is not present, creates a new symbol object with the new key.

Syntax

```
Symbol.for(key);
```

Parameters

`key`

Required. The key for the symbol, which is also used as the description.

Remarks

This method searches for the symbol in the global symbol registry. If you serialize symbols as strings, use the global symbol registry to make sure that a particular string maps to the correct symbol for all lookups.

Example

```
var sym = Symbol.for("desc");

console.log(sym.toString());

// Two different object references.
console.log(Symbol("symbol") === Symbol.for("symbol");)
// Single object reference.
console.log(Symbol.for("symbol") === Symbol.for("symbol");)

// Output:
// Symbol(desc);
// false
// true
```

Requirements

Supported in Microsoft Edge (Edge browser). Also supported in Store apps (Microsoft Edge on Windows 10). See Version Information.

Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Not supported in Windows 8.1.

Symbol.keyFor Function (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Returns the key for a specified symbol.

Syntax

```
Symbol.keyFor(sym);
```

Parameters

`sym`

Required. The symbol object.

Remarks

This method searches for the symbol in the global symbol registry.

Example

```
// Local symbol
var sym1 = Symbol("desc");
// Global symbol
var sym2 = Symbol.for("desc");

console.log(Symbol.keyFor(sym1)):
console.log(Symbol.keyFor(sym2));

// Output:
// undefined
// desc
```

Requirements

Supported in Microsoft Edge (Edge browser). Also supported in Store apps (Microsoft Edge on Windows 10). See Version Information.

Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Not supported in Windows 8.1.

Uint8Array Object

10/18/2017 • 1 min to read • Edit Online

A typed array of 8-bit unsigned integer values. The contents are initialized to 0. If the requested number of bytes could not be allocated an exception is raised.

Syntax

```
      uint8Array = new Uint8Array( length );
uint8Array = new Uint8Array( array );
uint8Array = new Uint8Array( buffer, byteOffset, length);
```

Parameters

`uint8Array`
Required. The variable name to which the **Uint8Array** object is assigned.

`length`
Specifies the number of elements in the array.

`array`
The array (or typed array) that is contained in this array. The contents are initialized to the contents of the given array or typed array, with each element converted to the Uint8 type.

`buffer`
The ArrayBuffer that the Uint8Array represents.

`byteOffset`
Optional. Specifies the offset in bytes from the start of the buffer at which the Uint8Array should begin.

`length`
The number of elements in the array.

Constants

The following table lists the constants of the `Uint8Array` object.

CONSTANT	DESCRIPTION
BYTES_PER_ELEMENT Constant	The size in bytes of each element in the array.

Properties

The following table lists the constants of the `Uint8Array` object.

PROPERTY	DESCRIPTION
buffer Property	Read-only. Gets the ArrayBuffer that is referenced by this array.

PROPERTY	DESCRIPTION
byteLength Property	Read-only. The length of this array from the start of its ArrayBuffer, in bytes, as fixed at construction time.
byteOffset Property	Read-only. The offset of this array from the start of its ArrayBuffer, in bytes, as fixed at construction time.
length Property	The length of the array.

Methods

The following table lists the methods of the `Uint8Array` object.

METHOD	DESCRIPTION
set Method (Uint8Array)	Sets a value or an array of values.
subarray Method (Uint8Array)	Gets a new Uint8Array view of the ArrayBuffer store for this array.

Example

The following example shows how to use a Uint8Array object to process the binary data acquired from an XmlHttpRequest:

```
var req = new XMLHttpRequest();
    req.open('GET', "http://www.example.com");
    req.responseType = "arraybuffer";
    req.send();

    req.onreadystatechange = function () {
        if (req.readyState === 4) {
            var buffer = req.response;
            var dataview = new DataView(buffer);
            var ints = new Uint8Array(buffer.byteLength);
            for (var i = 0; i < ints.length; i++) {
                ints[i] = dataview.getUint8(i);
            }
        alert(ints[10]);
        }
    }
```

Requirements

Supported in the following document modes: Internet Explorer 10 standards and Internet Explorer 11 standards.
Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards.

BYTES_PER_ELEMENT Constant (Uint8Array)

10/18/2017 • 1 min to read • Edit Online

The size in bytes of each element in the array.

Syntax

```
var arraySize = int8Array.BYTES_PER_ELEMENT;
```

Example

The following example shows how to get the size of the array elements.

```
var req = new XMLHttpRequest();
    req.open('GET', "http://www.example.com");
    req.responseType = "arraybuffer";
    req.send();

    req.onreadystatechange = function () {
        if (req.readyState === 4) {
            var buffer = req.response;
            var dataView = new DataView(buffer);
            var intArr = new Uint8Array(buffer.byteLength);
            alert(intArr.BYTES_PER_ELEMENT);
        }
    }
```

Requirements

Supported in the following document modes: Internet Explorer 10 standards and Internet Explorer 11 standards.
Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards.

buffer Property (Uint8Array)

10/18/2017 • 1 min to read • Edit Online

Read-only. Gets the ArrayBuffer that is referenced by this array.

Syntax

```
var arrayBuffer = uint8Array.buffer;
```

Example

The following example shows how to get the ArrayBuffer of the array.

```
var req = new XMLHttpRequest();
    req.open('GET', "http://www.example.com");
    req.responseType = "arraybuffer";
    req.send();

    req.onreadystatechange = function () {
        if (req.readyState === 4) {
            var buffer = req.response;
            var dataView = new DataView(buffer);
            var intArr = new Uint8Array(buffer.byteLength);
            alert(intArr.buffer.byteLength);
        }
    }
```

Requirements

Supported in the following document modes: Internet Explorer 10 standards and Internet Explorer 11 standards.
Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards.

byteLength Property (Uint8Array)

10/18/2017 • 1 min to read • Edit Online

Read-only. The length of this array from the start of its ArrayBuffer, in bytes, as fixed at construction time.

Syntax

```
var arrayByteLength = uint8Array.byteLength;
```

Example

The following example shows how to get the length of the array.

```
var req = new XMLHttpRequest();
    req.open('GET', "http://www.example.com");
    req.responseType = "arraybuffer";
    req.send();

    req.onreadystatechange = function () {
        if (req.readyState === 4) {
            var buffer = req.response;
            var dataView = new DataView(buffer);
            var intArr = new Uint8Array(buffer.byteLength);
            alert(intArr.byteLength);
        }
    }
```

Requirements

Supported in the following document modes: Internet Explorer 10 standards and Internet Explorer 11 standards.
Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards.

byteOffset Property (Uint8Array)

10/18/2017 • 1 min to read • Edit Online

Read-only. The offset of this array from the start of its ArrayBuffer, in bytes, as fixed at construction time.

Syntax

```
var arrayOffset = uint8Array.byteOffset;
```

Example

The following example shows how to get the offset of the array.

```
var req = new XMLHttpRequest();
    req.open('GET', "http://www.example.com");
    req.responseType = "arraybuffer";
    req.send();

    req.onreadystatechange = function () {
        if (req.readyState === 4) {
            var buffer = req.response;
            var dataView = new DataView(buffer);
            var intArr = new Uint8Array(buffer.byteLength);
            alert(intArr.byteOffset);
        }
    }
```

Requirements

Supported in the following document modes: Internet Explorer 10 standards and Internet Explorer 11 standards.
Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards.

length Property (Uint8Array)

10/18/2017 • 1 min to read • Edit Online

The length of the array.

Syntax

```
var arrayLength = uint8Array.length;
```

Example

The following example shows how to get the length of the array.

```
var req = new XMLHttpRequest();
    req.open('GET', "http://www.example.com");
    req.responseType = "arraybuffer";
    req.send();

    req.onreadystatechange = function () {
        if (req.readyState === 4) {
            var buffer = req.response;
            var dataView = new DataView(buffer);
            var intArr = new Uint8Array(buffer.byteLength);
            alert(intArr.length);
        }
    }
```

Requirements

Supported in the following document modes: Internet Explorer 10 standards and Internet Explorer 11 standards.
Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards.

set Method (Uint8Array)

10/18/2017 • 1 min to read • Edit Online

Sets a value or an array of values.

Syntax

```
uint8Array.set(index, value);
uint8Array.set(array, offset);
```

Parameters

`index`
The index of the location to set.

`value`
The value to set.

`array`
A typed or untyped array of values to set.

`offset`
The index in the current array at which the values are to be written.

Remarks

If the input array is a TypedArray, the two arrays may use the same underlying ArrayBuffer. In this situation, setting the values takes place as if all the data is first copied into a temporary buffer that does not overlap either of the arrays, and then the data from the temporary buffer is copied into the current array.

If the offset plus the length of the given array is out of range for the current TypedArray, an exception is raised.

Example

The following example shows how to set the first element of the array.

```
var req = new XMLHttpRequest();
    req.open('GET', "http://www.example.com");
    req.responseType = "arraybuffer";
    req.send();

    req.onreadystatechange = function () {
        if (req.readyState === 4) {
            var buffer = req.response;
            var dataView = new DataView(buffer);
            var intArr = new Uint8Array(buffer.byteLength);
            intArr.set(0, 9);
        }
    }
```

Requirements

Supported in the following document modes: Internet Explorer 10 standards and Internet Explorer 11 standards.
Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards.

subarray Method (Uint8Array)

10/18/2017 • 1 min to read • Edit Online

Gets a new Uint8Array view of the ArrayBuffer Object store for this array, specifying the first and last members of the subarray.

Syntax

```
var newUint8Array = uint8Array.subarray(begin, end);
```

Parameters

`newUint8Array`
The subarray returned by this method.

`begin`
The index of the beginning of the array.

`end`
The index of the end of the array. This is non-inclusive.

Remarks

If either `begin` or `end` is negative, it refers to an index from the end of the array, as opposed to from the beginning. If `end` is unspecified, the subarray contains all elements from `begin` to the end of the typed array. The range specified by the `begin` and `end` values is clamped to the valid index range for the current array. If the computed length of the new typed array would be negative, it is clamped to zero. The returned array is of the same type as the array on which this method is invoked.

Example

The following example shows how to get a subarray two elements long, starting with the first element of the array.

```
var req = new XMLHttpRequest();
    req.open('GET', "http://www.example.com");
    req.responseType = "arraybuffer";
    req.send();

    req.onreadystatechange = function () {
        if (req.readyState === 4) {
            var buffer = req.response;
            var intArr = new Uint8Array(buffer.byteLength);
            var subArr = intArr.subarray(0, 2);
        }
    }
```

Requirements

Supported in the following document modes: Internet Explorer 10 standards and Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards.

Uint8ClampedArray Object (JavaScript)

10/18/2017 • 2 min to read • Edit Online

A typed array of 8-bit unsigned integers with values clamped to the range 0-255. The contents are initialized to 0. If the requested number of bytes cannot be allocated, an exception is thrown.

Syntax

```
      uint8ClampedArray = new Uint8ClampedArray( length );
uint8ClampedArray = new Uint8ClampedArray( array );
uint8ClampedArray = new Uint8ClampedArray( buffer, byteOffset, length);
```

Parameters

`uint8ClampedArray`
Required. The variable name to which the `Uint8ClampedArray` object is assigned.

`length`
Optional. The number of elements in the array.

`array`
Optional. The array (or typed array) that this array contains. The contents are initialized to the contents of the given array or typed array, with each element converted to the `Uint8` type.

`buffer`
Optional. The ArrayBuffer that the `Uint8ClampedArray` represents.

`byteOffset`
Optional. The offset, in bytes, from the start of the buffer at which the `Uint8ClampedArray` should begin.

`length`
Optional. The number of elements in the array.

Remarks

Values stored in a `Uint8ClampedArray` object are between 0 and 255. If you set a negative value for an array member, 0 is set for the value. If you set a value that is larger than 255, 255 is set as the value.

Values in a `Uint8ClampedArray` object are rounded to the nearest even value, which is called banker's rounding.

Constants

The following table lists the constants of the `Uint8ClampedArray` object.

CONSTANT	DESCRIPTION
BYTES_PER_ELEMENT Constant	The size, in bytes, of each element in the array.

Properties

The following table lists the constants of the `Uint8ClampedArray` object.

PROPERTY	DESCRIPTION
buffer Property	Read-only. Gets the ArrayBuffer that is referenced by this array.
byteLength Property	Read-only. The length of this array from the start of its ArrayBuffer, in bytes, as fixed at construction time.
byteOffset Property	Read-only. The offset of this array from the start of its ArrayBuffer, in bytes, as fixed at construction time.
length Property	The length of the array.

Methods

The following table lists the methods of the `Uint8ClampedArray` object.

METHOD	DESCRIPTION
set Method	Sets a value or an array of values.
subarray Method	Gets a new `Uint8ClampedArray` view of the ArrayBuffer store for this array, specifying the first and last elements of the subarray.

Example

The following example shows how to use a `Uint8ClampedArray` object to process the binary data acquired from an `XmlHttpRequest`:

```
var req = new XMLHttpRequest();
    req.open('GET', "http://www.example.com");
    req.responseType = "arraybuffer";
    req.send();

    req.onreadystatechange = function () {
        if (req.readyState === 4) {
            var buffer = req.response;
            var dataview = new DataView(buffer);
            var ints = new Uint8ClampedArray(buffer.byteLength);
            for (var i = 0; i < ints.length; i++) {
                ints[i] = dataview.getUint8(i);
            }
        alert(ints[10]);
        }
    }
```

Example

The following example shows how values are restricted in a `Uint8ClampedArray`.

```
var ints = new Uint8ClampedArray(2);
ints[0] = -1;  // 0 will be assigned.
ints[1] = 256; // 255 will be assigned.
```

Example

The following example shows how values are rounded in a `Uint8ClampedArray`.

```
var ints = new Uint8ClampedArray(4);
ints[0] = 11.3; // 11 will be assigned (same as Int8Array).
ints[1] = 11.8; // 12 will be assigned (same as Int8Array).
ints[2] = 10.5; // 10 will be assigned (rounded to the nearest
                // even value).
ints[3] = 11.5; // 12 will be assigned (rounded to the nearest
                // even value).
```

Requirements

Supported in the Internet Explorer 11 standards document mode. Also supported in Windows Store apps (Windows 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards. Not supported in Windows 8 or Windows Phone 8.1.

See Also

Uint8Array Object
ArrayBuffer Object

BYTES_PER_ELEMENT Constant (Uint8ClampedArray)

10/18/2017 • 1 min to read • Edit Online

The size, in bytes, of each element in the array.

Syntax

```
var arraySize = uint8ClampedArray.BYTES_PER_ELEMENT;
```

Example

The following example shows how to get the size of the array elements.

```
var req = new XMLHttpRequest();
    req.open('GET', "http://www.example.com");
    req.responseType = "arraybuffer";
    req.send();

    req.onreadystatechange = function () {
        if (req.readyState === 4) {
            var buffer = req.response;
            var dataView = new DataView(buffer);
            var intArr = new Uint8ClampedArray(buffer.byteLength);
            alert(intArr.BYTES_PER_ELEMENT);
        }
    }
```

Requirements

Supported in the Internet Explorer 11 standards document mode. Also supported in Windows Store apps (Windows 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards. Not supported in Windows 8 or Windows Phone 8.1.

See Also

Uint8ClampedArray Object

buffer Property (Uint8ClampedArray)

10/18/2017 • 1 min to read • Edit Online

Read-only. Gets the ArrayBuffer that is referenced by this array.

Syntax

```
var arrayBuffer = uint8ClampedArray.buffer;
```

Example

The following example shows how to get the ArrayBuffer of the array.

```
var req = new XMLHttpRequest();
    req.open('GET', "http://www.example.com");
    req.responseType = "arraybuffer";
    req.send();

    req.onreadystatechange = function () {
        if (req.readyState === 4) {
            var buffer = req.response;
            var dataView = new DataView(buffer);
            var intArr = new Uint8ClampedArray(buffer.byteLength);
            alert(intArr.buffer.byteLength);
        }
    }
```

Requirements

Supported in the Internet Explorer 11 standards document mode. Also supported in Windows Store apps (Windows 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards. Not supported in Windows 8 or Windows Phone 8.1.

See Also

ArrayBuffer Object
Uint8ClampedArray Object

byteLength Property (Uint8ClampedArray)

10/18/2017 • 1 min to read • Edit Online

Read-only. The length of this array from the start of its ArrayBuffer, in bytes, as fixed at construction time.

Syntax

```
var arrayByteLength = uint8ClampedArray.byteLength;
```

Example

The following example shows how to get the length of the array.

```
var req = new XMLHttpRequest();
    req.open('GET', "http://www.example.com");
    req.responseType = "arraybuffer";
    req.send();

    req.onreadystatechange = function () {
        if (req.readyState === 4) {
            var buffer = req.response;
            var dataView = new DataView(buffer);
            var intArr = new Uint8ClampedArray(buffer.byteLength);
            alert(intArr.byteLength);
        }
    }
```

Requirements

Supported in the Internet Explorer 11 standards document mode. Also supported in Windows Store apps (Windows 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards. Not supported in Windows 8 or Windows Phone 8.1.

See Also

ArrayBuffer Object
Uint8ClampedArray Object

byteOffset Property (Uint8ClampedArray)

10/18/2017 • 1 min to read • Edit Online

Read-only. The offset of this array from the start of its ArrayBuffer, in bytes, as fixed at construction time.

Syntax

```
var arrayOffset = uint8ClampedArray.byteOffset;
```

Example

The following example shows how to get the offset of the array.

```
var req = new XMLHttpRequest();
    req.open('GET', "http://www.example.com");
    req.responseType = "arraybuffer";
    req.send();

    req.onreadystatechange = function () {
        if (req.readyState === 4) {
            var buffer = req.response;
            var dataView = new DataView(buffer);
            var intArr = new Uint8ClampedArray(buffer.byteLength);
            alert(intArr.byteOffset);
        }
    }
```

Requirements

Supported in the Internet Explorer 11 standards document mode. Also supported in Windows Store apps (Windows 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards. Not supported in Windows 8 or Windows Phone 8.1.

See Also

ArrayBuffer Object
Uint8ClampedArray Object

length Property (Uint8ClampedArray)

10/18/2017 • 1 min to read • Edit Online

The length of the array.

Syntax

```
var arrayLength = uint8ClampedArray.length;
```

Example

The following example shows how to get the length of the array.

```
var req = new XMLHttpRequest();
    req.open('GET', "http://www.example.com");
    req.responseType = "arraybuffer";
    req.send();

    req.onreadystatechange = function () {
        if (req.readyState === 4) {
            var buffer = req.response;
            var dataView = new DataView(buffer);
            var intArr = new Uint8ClampedArray(buffer.byteLength);
            alert(intArr.length);
        }
    }
```

Requirements

Supported in the Internet Explorer 11 standards document mode. Also supported in Windows Store apps (Windows 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards. Not supported in Windows 8 or Windows Phone 8.1.

See Also

Uint8ClampedArray Object

set Method (Uint8ClampedArray)

10/18/2017 • 1 min to read • Edit Online

Sets a value or an array of values.

Syntax

```
uint8ClampedArray.set(index, value);
uint8ClampedArray.set(array, offset);
```

Parameters

`index`
The index of the location to set.

`value`
The value to set.

`array`
A typed or untyped array of values to set.

`offset`
The index in the current array at which the values are to be written.

Remarks

If the input array is a typed array, the two arrays may use the same underlying ArrayBuffer. In this situation, setting the values takes place as if all the data had first been copied into a temporary buffer that does not overlap either of the arrays, and then the data from the temporary buffer is copied into the current array.

If the offset plus the length of the given array is out of range for the current typed array, an exception is raised.

Example

The following example shows how to set the first element of the array.

```
var req = new XMLHttpRequest();
    req.open('GET', "http://www.example.com");
    req.responseType = "arraybuffer";
    req.send();

    req.onreadystatechange = function () {
        if (req.readyState === 4) {
            var buffer = req.response;
            var dataView = new DataView(buffer);
            var intArr = new Uint8ClampedArray(buffer.byteLength);
            intArr.set(0, 9);
        }
    }
```

Requirements

Supported in the Internet Explorer 11 standards document mode. Also supported in Windows Store apps (Windows 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards. Not supported in Windows 8 or Windows Phone 8.1.

See Also

ArrayBuffer Object
Uint8ClampedArray Object

subarray Method (Uint8ClampedArray)

10/18/2017 • 1 min to read • Edit Online

Gets a new Uint8ClampedArray view of the ArrayBuffer store for this array, specifying the first and last members of the subarray.

Syntax

```
var newUint8ClampedArray = uint8ClampedArray.subarray(begin, end);
```

Parameters

`newUint8ClampedArray`
Required. The subarray returned by this method.

`begin`
Optional. The index of the beginning of the array.

`end`
Optional. The index of the end of the array. This is non-inclusive.

Remarks

If either `begin` or `end` is negative, it refers to an index from the end of the array, as opposed to from the beginning. If `end` is unspecified, the subarray contains all elements from `begin` to the end of the typed array. The range specified by the `begin` and `end` values is clamped to the valid index range for the current array. If the computed length of the new typed array would be negative, it is clamped to zero. The returned array has the same type as the array on which this method is invoked.

Example

The following example shows how to get a subarray that is two elements long, starting with the first element of the array.

```
var req = new XMLHttpRequest();
    req.open('GET', "http://www.example.com");
    req.responseType = "arraybuffer";
    req.send();

    req.onreadystatechange = function () {
        if (req.readyState === 4) {
            var buffer = req.response;
            var intArr = new Uint8ClampedArray(buffer.byteLength);
            var subArr = intArr.subarray(0, 2);
        }
    }
```

Requirements

Supported in the Internet Explorer 11 standards document mode. Also supported in Windows Store apps

(Windows 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards. Not supported in Windows 8 or Windows Phone 8.1.

See Also

Uint8Array Object
ArrayBuffer Object
Uint8ClampedArray Object

Uint16Array Object

10/18/2017 • 2 min to read • Edit Online

A typed array of 16-bit unsigned integer values. The contents are initialized to 0. If the requested number of bytes could not be allocated an exception is raised.

Syntax

```
      uint16Array = new Uint16Array( length );
uint16Array = new Uint16Array( array );
uint16Array = new Uint16Array( buffer, byteOffset, length );
```

Parameters

`uint16Array`
Required. The variable name to which the **Uint16Array** object is assigned.

`length`
Specifies the number of elements in the array.

`array`
The array (or typed array) that is contained in this array.. The contents are initialized to the contents of the given array or typed array, with each element converted to the Uint16 type.

`buffer`
The ArrayBuffer that the Uint16Array represents.

`byteOffset`
Optional. Specifies the offset in bytes from the start of the buffer at which the Uint16Array should begin.

`length`
The number of elements in the array.

Constants

The following table lists the constants of the `Uint16Array` object.

CONSTANT	DESCRIPTION
BYTES_PER_ELEMENT Constant	The size in bytes of each element in the array.

Properties

The following table lists the constants of the `Uint16Array` object.

PROPERTY	DESCRIPTION
buffer Property	Read-only. Gets the ArrayBuffer that is referenced by this array.

PROPERTY	DESCRIPTION
byteLength Property	Read-only. The length of this array from the start of its ArrayBuffer, in bytes, as fixed at construction time.
byteOffset Property	Read-only. The offset of this array from the start of its ArrayBuffer, in bytes, as fixed at construction time.
length Property	The length of the array.

Methods

The following table lists the methods of the `Uint16Array` object.

METHOD	DESCRIPTION
set Method (Uint16Array)	Sets a value or an array of values.
subarray Method (Uint16Array)	Gets a new Uint16Array view of the ArrayBuffer store for this array.

Example

The following example shows how to use a Uint16Array object to process the binary data acquired from an XmlHttpRequest:

```
var req = new XMLHttpRequest();
    req.open('GET', "http://www.example.com");
    req.responseType = "arraybuffer";
    req.send();

    req.onreadystatechange = function () {
        if (req.readyState === 4) {
            var buffer = req.response;
            var dataview = new DataView(buffer);
            var ints = new Uint16Array(buffer.byteLength / 2);
            for (var i = 0; i < ints.length; i++) {
                ints[i] = dataview.getUint16(i * 2);
            }
        alert(ints[10]);
        }
    }
```

Requirements

Supported in the following document modes: Internet Explorer 10 standards and Internet Explorer 11 standards.
Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards.

BYTES_PER_ELEMENT Constant (Uint16Array)

10/18/2017 • 1 min to read • Edit Online

The size in bytes of each element in the array.

Syntax

```
var arraySize = uint16Array.BYTES_PER_ELEMENT;
```

Example

The following example shows how to get the size of the array elements.

```
var req = new XMLHttpRequest();
    req.open('GET', "http://www.example.com");
    req.responseType = "arraybuffer";
    req.send();

    req.onreadystatechange = function () {
        if (req.readyState === 4) {
            var buffer = req.response;
            var dataView = new DataView(buffer);
            var intArr = new Uint16Array(buffer.byteLength / 2);
            alert(intArr.BYTES_PER_ELEMENT);
        }
    }
```

Requirements

Supported in the following document modes: Internet Explorer 10 standards and Internet Explorer 11 standards.
Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards.

buffer Property (Uint16Array)

10/18/2017 • 1 min to read • Edit Online

Read-only. Gets the ArrayBuffer that is referenced by this array.

Syntax

```
var arrayBuffer = uint16Array.buffer;
```

Example

The following example shows how to get the ArrayBuffer of the array.

```
var req = new XMLHttpRequest();
    req.open('GET', "http://www.example.com");
    req.responseType = "arraybuffer";
    req.send();

    req.onreadystatechange = function () {
        if (req.readyState === 4) {
            var buffer = req.response;
            var dataView = new DataView(buffer);
            var intArr = new Uint16Array(buffer.byteLength / 2);
            alert(intArr.buffer.byteLength);
        }
    }
```

Requirements

Supported in the following document modes: Internet Explorer 10 standards and Internet Explorer 11 standards.
Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards.

byteLength Property (Uint16Array)

10/18/2017 • 1 min to read • Edit Online

Read-only. The length of this array from the start of its ArrayBuffer, in bytes, as fixed at construction time.

Syntax

```
var arrayByteLength = uint16Array.byteLength;
```

Example

The following example shows how to get the length of the array.

```
var req = new XMLHttpRequest();
    req.open('GET', "http://www.example.com");
    req.responseType = "arraybuffer";
    req.send();

    req.onreadystatechange = function () {
        if (req.readyState === 4) {
            var buffer = req.response;
            var dataView = new DataView(buffer);
            var intArr = new Uint16Array(buffer.byteLength / 2);
            alert(intArr.byteLength);
        }
    }
```

Requirements

Supported in the following document modes: Internet Explorer 10 standards and Internet Explorer 11 standards.
Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards.

byteOffset Property (Uint16Array)

10/18/2017 • 1 min to read • Edit Online

Read-only. The offset of this array from the start of its ArrayBuffer, in bytes, as fixed at construction time.

Syntax

```
var arrayOffset = uint16Array.byteOffset;
```

Example

The following example shows how to get the offset of the array.

```
var req = new XMLHttpRequest();
    req.open('GET', "http://www.example.com");
    req.responseType = "arraybuffer";
    req.send();

    req.onreadystatechange = function () {
        if (req.readyState === 4) {
            var buffer = req.response;
            var dataView = new DataView(buffer);
            var intArr = new Uint16Array(buffer.byteLength / 2);
            alert(intArr.byteOffset);
        }
    }
```

Requirements

Supported in the following document modes: Internet Explorer 10 standards and Internet Explorer 11 standards.
Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards.

length Property (UInt16Array)

10/18/2017 • 1 min to read • Edit Online

The length of the array.

Syntax

```
var arrayLength = uint16Array.length;
```

Example

The following example shows how to get the length of the array.

```
var req = new XMLHttpRequest();
    req.open('GET', "http://www.example.com");
    req.responseType = "arraybuffer";
    req.send();

    req.onreadystatechange = function () {
        if (req.readyState === 4) {
            var buffer = req.response;
            var dataView = new DataView(buffer);
            var intArr = new Uint16Array(buffer.byteLength / 2);
            alert(intArr.length);
        }
    }
```

Requirements

Supported in the following document modes: Internet Explorer 10 standards and Internet Explorer 11 standards.
Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards.

set Method (Uint16Array)

10/18/2017 • 1 min to read • Edit Online

Sets a value or an array of values.

Syntax

```
uint16Array.set(index, value);
uint16Array.set(array, offset);
```

Parameters

`index`
The index of the location to set.

`value`
The value to set.

`array`
A typed or untyped array of values to set.

`offset`
The index in the current array at which the values are to be written.

Remarks

If the input array is a TypedArray, the two arrays may use the same underlying ArrayBuffer. In this situation, setting the values takes place as if all the data is first copied into a temporary buffer that does not overlap either of the arrays, and then the data from the temporary buffer is copied into the current array.

If the offset plus the length of the given array is out of range for the current TypedArray, an exception is raised.

Example

The following example shows how to set the first element of the array.

```
var req = new XMLHttpRequest();
    req.open('GET', "http://www.example.com");
    req.responseType = "arraybuffer";
    req.send();

    req.onreadystatechange = function () {
        if (req.readyState === 4) {
            var buffer = req.response;
            var dataView = new DataView(buffer);
            var intArr = new Uint16Array(buffer.byteLength / 2);
            intArr.set(0, 9);
        }
    }
```

Requirements

Supported in the following document modes: Internet Explorer 10 standards and Internet Explorer 11 standards.
Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards.

subarray Method (Uint16Array)

10/18/2017 • 1 min to read • Edit Online

Gets a new Uint16Array view of the ArrayBuffer Object store for this array, specifying the first and last members of the subarray.

Syntax

```
var newUint16Array = uint16Array.subarray(begin, end);
```

Parameters

`newUint16Array`
The subarray returned by this method.

`begin`
The index of the beginning of the array.

`end`
The index of the end of the array. This is non-inclusive.

Remarks

If either `begin` or `end` is negative, it refers to an index from the end of the array, as opposed to from the beginning. If `end` is unspecified, the subarray contains all elements from `begin` to the end of the typed array. The range specified by the `begin` and `end` values is clamped to the valid index range for the current array. If the computed length of the new typed array would be negative, it is clamped to zero. The returned array is of the same type as the array on which this method is invoked.

Example

The following example shows how to get a subarray two elements long, starting with the first element of the array.

```
var req = new XMLHttpRequest();
    req.open('GET', "http://www.example.com");
    req.responseType = "arraybuffer";
    req.send();

    req.onreadystatechange = function () {
        if (req.readyState === 4) {
            var buffer = req.response;
            var intArr = new Uint16Array(buffer.byteLength / 2);
            var subArr = intArr.subarray(0, 2);
        }
    }
```

Requirements

Supported in the following document modes: Internet Explorer 10 standards and Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards.

Uint32Array Object

10/18/2017 • 2 min to read • Edit Online

A typed array of 32-bit unsigned integer values. The contents are initialized to 0. If the requested number of bytes could not be allocated an exception is raised.

Syntax

```
      uint32Array = new Uint32Array( length );
uint32Array = new Uint32Array( array );
uint32Array = new Uint32Array( buffer, byteOffset, length);
```

Parameters

`uint32Array`
Required. The variable name to which the **Uint32Array** object is assigned.

`length`
Specifies the number of elements in the array.

`array`
The array (or typed array) that is contained in this array. The contents are initialized to the contents of the given array or typed array, with each element converted to the Uint32 type.

`buffer`
The ArrayBuffer that the Uint32Array represents.

`byteOffset`
Optional. Specifies the offset in bytes from the start of the buffer at which the Uint32Array should begin.

`length`
The number of elements in the array.

Constants

The following table lists the constants of the `Uint32Array` object.

CONSTANT	DESCRIPTION
BYTES_PER_ELEMENT Constant	The size in bytes of each element in the array.

Properties

The following table lists the constants of the `Uint32Array` object.

PROPERTY	DESCRIPTION
buffer Property	Read-only. Gets the ArrayBuffer that is referenced by this array.

PROPERTY	DESCRIPTION
byteLength Property	Read-only. The length of this array from the start of its ArrayBuffer, in bytes, as fixed at construction time.
byteOffset Property	Read-only. The offset of this array from the start of its ArrayBuffer, in bytes, as fixed at construction time.
length Property	The length of the array.

Methods

The following table lists the methods of the `Uint32Array` object.

METHOD	DESCRIPTION
set Method (Uint32Array)	Sets a value or an array of values.
subarray Method (Uint32Array)	Gets a new Uint32Array view of the ArrayBuffer store for this array.

Example

The following example shows how to use a Uint32Array object to process the binary data acquired from an XMLHttpRequest:

```
var req = new XMLHttpRequest();
    req.open('GET', "http://www.example.com");
    req.responseType = "arraybuffer";
    req.send();

    req.onreadystatechange = function () {
        if (req.readyState === 4) {
            var buffer = req.response;
            var dataview = new DataView(buffer);
            var ints = new Uint32Array(buffer.byteLength / 4);
            for (var i = 0; i < ints.length; i++) {
                ints[i] = dataview.getUint32(i * 4);
            }
        alert(ints[10]);
        }
    }
```

Requirements

Supported in the following document modes: Internet Explorer 10 standards and Internet Explorer 11 standards.
Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards.

BYTES_PER_ELEMENT Constant (Uint32Array)

10/18/2017 • 1 min to read • Edit Online

The size in bytes of each element in the array.

Syntax

```
var arraySize = uint32Array.BYTES_PER_ELEMENT;
```

Example

The following example shows how to get the size of the array elements.

```
var req = new XMLHttpRequest();
    req.open('GET', "http://www.example.com");
    req.responseType = "arraybuffer";
    req.send();

    req.onreadystatechange = function () {
        if (req.readyState === 4) {
            var buffer = req.response;
            var dataView = new DataView(buffer);
            var intArr = new Uint32Array(buffer.byteLength / 4);
            alert(intArr.BYTES_PER_ELEMENT);
        }
    }
```

Requirements

Supported in the following document modes: Internet Explorer 10 standards and Internet Explorer 11 standards.
Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards.

buffer Property (Uint32Array)

10/18/2017 • 1 min to read • Edit Online

Read-only. Gets the ArrayBuffer that is referenced by this array.

Syntax

```
var arrayBuffer = uint32Array.buffer;
```

Example

The following example shows how to get the ArrayBuffer of the array.

```
var req = new XMLHttpRequest();
    req.open('GET', "http://www.example.com");
    req.responseType = "arraybuffer";
    req.send();

    req.onreadystatechange = function () {
        if (req.readyState === 4) {
            var buffer = req.response;
            var dataView = new DataView(buffer);
            var intArr = new Uint32Array(buffer.byteLength / 4);
            alert(intArr.buffer.byteLength);
        }
    }
```

Requirements

Supported in the following document modes: Internet Explorer 10 standards and Internet Explorer 11 standards.
Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards.

byteLength Property (UInt32Array)

10/18/2017 • 1 min to read • Edit Online

Read-only. The length of this array from the start of its ArrayBuffer, in bytes, as fixed at construction time.

Syntax

```
var arrayByteLength = uint32Array.byteLength;
```

Example

The following example shows how to get the length of the array.

```
var req = new XMLHttpRequest();
    req.open('GET', "http://www.example.com");
    req.responseType = "arraybuffer";
    req.send();

    req.onreadystatechange = function () {
        if (req.readyState === 4) {
            var buffer = req.response;
            var dataView = new DataView(buffer);
            var intArr = new Uint32Array(buffer.byteLength / 4);
            alert(intArr.byteLength);
        }
    }
```

Requirements

Supported in the following document modes: Internet Explorer 10 standards and Internet Explorer 11 standards.
Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards.

byteOffset Property (UInt32Array)

10/18/2017 • 1 min to read • Edit Online

Read-only. The offset of this array from the start of its ArrayBuffer, in bytes, as fixed at construction time.

Syntax

```
var arrayOffset = uint32Array.byteOffset;
```

Example

The following example shows how to get the offset of the array.

```
var req = new XMLHttpRequest();
    req.open('GET', "http://www.example.com");
    req.responseType = "arraybuffer";
    req.send();

    req.onreadystatechange = function () {
        if (req.readyState === 4) {
            var buffer = req.response;
            var dataView = new DataView(buffer);
            var intArr = new Uint32Array(buffer.byteLength / 4);
            alert(intArr.byteOffset);
        }
    }
```

Requirements

Supported in the following document modes: Internet Explorer 10 standards and Internet Explorer 11 standards.
Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards.

length Property (Uint32Array)

10/18/2017 • 1 min to read • Edit Online

The length of the array.

Syntax

```
var arrayLength = uint32Array.length;
```

Example

The following example shows how to get the length of the array.

```
var req = new XMLHttpRequest();
    req.open('GET', "http://www.example.com");
    req.responseType = "arraybuffer";
    req.send();

    req.onreadystatechange = function () {
        if (req.readyState === 4) {
            var buffer = req.response;
            var dataView = new DataView(buffer);
            var intArr = new Uint32Array(buffer.byteLength / 4);
            alert(intArr.length);
        }
    }
```

Requirements

Supported in the following document modes: Internet Explorer 10 standards and Internet Explorer 11 standards.
Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards.

set Method (Uint32Array)

10/18/2017 • 1 min to read • Edit Online

Sets a value or an array of values.

Syntax

```
uint32Array.set(index, value);
uint32Array.set(array, offset);
```

Parameters

`index`
The index of the location to set.

`value`
The value to set.

`array`
A typed or untyped array of values to set.

`offset`
The index in the current array at which the values are to be written.

Remarks

If the input array is a TypedArray, the two arrays may use the same underlying ArrayBuffer. In this situation, setting the values takes place as if all the data is first copied into a temporary buffer that does not overlap either of the arrays, and then the data from the temporary buffer is copied into the current array.

If the offset plus the length of the given array is out of range for the current TypedArray, an exception is raised.

Example

The following example shows how to set the first element of the array.

```
var req = new XMLHttpRequest();
    req.open('GET', "http://www.example.com");
    req.responseType = "arraybuffer";
    req.send();

    req.onreadystatechange = function () {
        if (req.readyState === 4) {
            var buffer = req.response;
            var dataView = new DataView(buffer);
            var intArr = new Uint32Array(buffer.byteLength / 4);
            intArr.set(0, 9);
        }
    }
```

Requirements

Supported in the following document modes: Internet Explorer 10 standards and Internet Explorer 11 standards.
Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards.

subarray Method (Uint32Array)

10/18/2017 • 1 min to read • Edit Online

Gets a new Uint32Array view of the ArrayBuffer Object store for this array, specifying the first and last members of the subarray.

Syntax

```
var newUint32Array = uint32Array.subarray(begin, end);
```

Parameters

`newUint32Array`
The subarray returned by this method.

`begin`
The index of the beginning of the array.

`end`
The index of the end of the array. This is non-inclusive.

Remarks

If either `begin` or `end` is negative, it refers to an index from the end of the array, as opposed to from the beginning. If `end` is unspecified, the subarray contains all elements from `begin` to the end of the typed array. The range specified by the `begin` and `end` values is clamped to the valid index range for the current array. If the computed length of the new typed array would be negative, it is clamped to zero. The returned array is of the same type as the array on which this method is invoked.

Example

The following example shows how to get a subarray that is two elements long, starting with the first element of the array.

```
var req = new XMLHttpRequest();
    req.open('GET', "http://www.example.com");
    req.responseType = "arraybuffer";
    req.send();

    req.onreadystatechange = function () {
        if (req.readyState === 4) {
            var buffer = req.response;
            var intArr = new Uint32Array(buffer.byteLength / 4);
            var subArr = intArr.subarray(0, 2);
        }
    }
```

Requirements

Supported in the following document modes: Internet Explorer 10 standards and Internet Explorer 11 standards.

Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards.

VBArray Object (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Provides access to Visual Basic safe arrays.

> **WARNING**
>
> This object is supported in Internet Explorer only, not in Windows 8.x Store apps.

Syntax

```
varName = new VBArray(safeArray)
```

Parameters

`varName`
Required. The variable name to which the VBArray is assigned.

safeArray
Required. A VBArray value.

Remarks

VBArrays are 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.

Example

The following example consists of three parts. The first part is VBScript code to create a Visual Basic safe array. The second part is JavaScript code that converts the Visual Basic safe array to a JavaScript array. Both of these parts go into the <HEAD> section of an HTML page. The third part is the JavaScript code that goes in the <BODY> section to run the other two parts.

```
<head>
<script type="text/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 type="text/javascript">
<!--
function VBArrayTest(vbarray){
   var a = new VBArray(vbarray);
   var b = a.toArray();
   var i;
   for (i = 0; i < 9; i++)
   {
      document.writeln(b[i]);
   }
}
-->
</script>
</head>

<body>
<script type="text/javascript">
<!--
   VBArrayTest(CreateVBArray());
-->
</script>
</body>
```

Properties

The `VBArray` object has no properties.

Methods

dimensions Method | getItem Method | lbound Method | toArray Method | ubound Method

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, and Internet Explorer 10 standards. Not supported in Windows 8.x Store apps. See Version Information.

See Also

Array Object

dimensions Method (VBArray) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Returns the number of dimensions in a VBArray.

Syntax

```
array.dimensions( )
```

Remarks

The required *array* is a VBArray object.

Example

The **dimensions** method provides a way to retrieve the number of dimensions in a specified VBArray.

The following example consists of three parts. The first part is VBScript code to create a Visual Basic safe array. The second part is JavaScript code that determines the number of dimensions in the safe array and the upper bound of each dimension. Both of these parts go into the <HEAD> section of an HTML page. The third part is the JavaScript code that goes in the <BODY> section to run the other two parts.

```
<head>
<script type="text/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
         k = k + 1
      Next
   Next
   CreateVBArray = a
End Function
-->
</script>

<script type="text/javascript">
<!--
function VBArrayTest(vba)
{
   var i;
   var a = new VBArray(vba);
   var s = "";
   for (i = 1; i <= a.dimensions(); i++)
   {
      s += "The upper bound of dimension ";
      s += i + " is ";
      s += a.ubound(i);
      s += ".<br />";
   }
   return(s);
}
-->
</script>
</head>

<body>
<script type="text/javascript">
   document.write(VBArrayTest(CreateVBArray()));
</script>
</body>
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, and Internet Explorer 10 standards. Not supported in Windows 8.x Store apps. See Version Information.

Applies To: VBArray Object

See Also

getItem Method (VBArray)
lbound Method (VBArray)
toArray Method (VBArray)
ubound Method (VBArray)

getItem Method (VBArray) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Returns the item at the specified location.

Syntax

```
safeArray.getItem(dimension1[, dimension2, ...], dimensionN)
```

Parameters

safeArray
Required. A VBArray object.

dimension1, ..., dimensionN
Specifies the exact location of the desired element of the VBArray. *n* equals the number of dimensions in the VBArray.

Example

The following example consists of three parts. The first part is VBScript code to create a Visual Basic safe array. The second part is JavaScript code that iterates the Visual Basic safe array and prints out the contents of each element. Both of these parts go into the <HEAD> section of an HTML page. The third part is the JavaScript code that goes in the <BODY> section to run the other two parts.

```
<head>
<script type="text/vbscript">
<!--
Function CreateVBArray()
   Dim i, j, k
   Dim a(2, 2)
   k = 1
   For i = 0 To 2
      For j = 0 To 2
         a(i, j) = k
         document.writeln(k)
         k = k + 1
      Next
      document.writeln("<BR>")
   Next
   CreateVBArray = a
End Function
-->
</script>
<script type="text/javascript">
<!--
function GetItemTest(vbarray)
{
   var i, j;
   var a = new VBArray(vbarray);
   for (i = 0; i <= 2; i++)
   {
      for (j =0; j <= 2; j++)
      {
         document.writeln(a.getItem(i, j));
      }
   }
}
-->
</script>
</head>
<body>
<script type="text/javascript">
<!--
   GetItemTest(CreateVBArray());
-->
</script>
</body>
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, and Internet Explorer 10 standards. Not supported in Windows 8.x Store apps. See Version Information.

Applies To: VBArray Object

See Also

dimensions Method (VBArray)
lbound Method (VBArray)
toArray Method (VBArray)
ubound Method (VBArray)

lbound Method (VBArray) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Returns the lowest index value used in the specified dimension of a VBArray.

Syntax

```
safeArray.lbound(dimension)
```

Parameters

safeArray
Required. A VBArray object.

`dimension`
Optional. The dimension of the VBArray for which the lower bound index is wanted. If omitted, `lbound` behaves as if a 1 was passed.

Remarks

If the VBArray is empty, the `lbound` method returns undefined. If `dimension` is greater than the number of dimensions in the VBArray, or is negative, the method generates a "Subscript out of range" error.

Example

The following example consists of three parts. The first part is VBScript code to create a Visual Basic safe array. The second part is JavaScript code that determines the number of dimensions in the safe array and the lower bound of each dimension. Since the safe array is created in VBScript rather than Visual Basic, the lower bound will always be zero. Both of these parts go into the <HEAD> section of an HTML page. The third part is the JavaScript code that goes in the <BODY> section to run the other two parts.

```
<head>
<script type="text/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
         k = k + 1
      Next
   Next
   CreateVBArray = a
End Function
-->
</script>

<script type="text/javascript">
<!--
function VBArrayTest(vba){
   var i;
   var a = new VBArray(vba);
   var s = "";
   for (i = 1; i <= a.dimensions(); i++)
   {
      s += "The lower bound of dimension ";
      s += i + " is ";
      s += a.lbound(i);
      s += ".<br />";
   }
   return (s);
}
-->
</script>
</head>

<body>
<script type="text/javascript">
   document.write(VBArrayTest(CreateVBArray()));
</script>
</body>
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, and Internet Explorer 10 standards. Not supported in Windows 8.x Store apps. See Version Information.

Applies To: VBArray Object

See Also

dimensions Method (VBArray)
getItem Method (VBArray)
toArray Method (VBArray)
ubound Method (VBArray)

toArray Method (VBArray) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Returns a standard JavaScript array converted from a VBArray.

Syntax

```
safeArray.toArray( )
```

Remarks

The required *safeArray* reference is a VBArray object.

The conversion translates the multidimensional VBArray into a single dimensional JavaScript array. Each successive dimension is appended to the end of the previous one. For example, a VBArray with three dimensions and three elements in each dimension is converted into a JavaScript array as follows:

Suppose the VBArray contains: (1, 2, 3), (4, 5, 6), (7, 8, 9). After translation, the JavaScript array contains: 1, 2, 3, 4, 5, 6, 7, 8, 9.

There is currently no way to convert a JavaScript array into a VBArray.

Example

The following example consists of three parts. The first part is VBScript code to create a Visual Basic safe array. The second part is JavaScript code that converts the Visual Basic safe array to a JavaScript array. Both of these parts go into the <HEAD> section of an HTML page. The third part is the JavaScript code that goes in the <BODY> section to run the other two parts.

```
<head>
<script type="text/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 type="text/javascript">
<!--
function VBArrayTest(vbarray)
{
   var a = new VBArray(vbarray);
   var b = a.toArray();
   var i;
   for (i = 0; i < 9; i++)
   {
      document.writeln(b[i]);
   }
}
-->
</script>
</head>

<body>
<script type="text/javascript">
<!--
   VBArrayTest(CreateVBArray());
-->
</script>
</body>
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, and Internet Explorer 10 standards. Not supported in Windows 8.x Store apps. See Version Information.

Applies To: VBArray Object

See Also

dimensions Method (VBArray)
getItem Method (VBArray)
lbound Method (VBArray)
ubound Method (VBArray)

ubound Method (VBArray) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Returns the highest index value used in the specified dimension of the VBArray.

Syntax

```
safeArray.ubound(dimension)
```

Parameters

safeArray
Required. A VBArray object.

`dimension`
Optional. The dimension of the VBArray for which the higher bound index is wanted. If omitted, `ubound` behaves as if a 1 was passed.

Remarks

If the VBArray is empty, the `ubound` method returns undefined. If `dim` is greater than the number of dimensions in the VBArray, or is negative, the method generates a "Subscript out of range" error.

Example

The following example consists of three parts. The first part is VBScript code to create a Visual Basic safe array. The second part is JavaScript code that determines the number of dimensions in the safe array and the upper bound of each dimension. Both of these parts go into the <HEAD> section of an HTML page. The third part is the JavaScript code that goes in the <BODY> section to run the other two parts.

```
<head>
<script type="text/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
         k = k + 1
      Next
   Next
   CreateVBArray = a
End Function
-->
</script>

<script type="text/javascript">
<!--
function VBArrayTest(vba)
{
   var i;
   var a = new VBArray(vba);
   var s = "";
   for (i = 1; i <= a.dimensions(); i++)
   {
      s += "The upper bound of dimension ";
      s += i + " is ";
      s += a.ubound(i);
      s += ".<br />";
   }
   return (s);
}
-->
</script>
</head>

<body>
<script type="text/javascript">
   document.write(VBArrayTest(CreateVBArray()));
</script>
</body>
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, and Internet Explorer 10 standards. Not supported in Windows 8.x Store apps. See Version Information.

Applies To: VBArray Object

See Also

dimensions Method (VBArray)
getItem Method (VBArray)
lbound Method (VBArray)
toArray Method (VBArray)

WeakMap Object (JavaScript)

10/18/2017 • 1 min to read • Edit Online

A collection of key/value pairs in which each key is an object reference.

Syntax

```
weakmapObj = new WeakMap()
```

Remarks

A `WeakMap` object cannot be enumerated.

If you add a value to the collection using an existing key, the new value will replace the old value.

In a `WeakMap` object, references to key objects are held 'weakly'. This means that `WeakMap` will not prevent a garbage collection from happening on the key objects. When there are no references (other than `WeakMap`) to the key objects, the garbage collector may collect the key objects.

Properties

The following table lists the properties of the `WeakMap` object.

PROPERTY	DESCRIPTION
constructor	Specifies the function that creates a `WeakMap`.
prototype	Returns a reference to the prototype for a `WeakMap`.

Methods

The following table lists the methods of the `WeakMap` object.

METHOD	DESCRIPTION
clear	Removes all elements from a `WeakMap`.
delete	Removes a specified element from a `WeakMap`.
get	Returns a specified element from a `WeakMap`.
has	Returns `true` if the `WeakMap` contains a specified element.
set	Adds a new element to a `WeakMap`.
toString	Returns a string representation of a `WeakMap`.
valueOf	Returns the primitive value of the specified object.

METHOD	DESCRIPTION

Example

The following example shows how to add members to a `WeakMap` object and then retrieve them.

```
var dog = {
    breed: "yorkie"
}

var cat = {
    breed: "burmese"
}

var wm = new WeakMap();
wm.set(dog, "fido");
wm.set(cat, "pepper");

document.write(wm.get(dog) + ": ");
document.write(dog.breed);
document.write("<br />");
document.write(wm.get(cat) + ": ");
document.write(cat.breed);

// Output:
// fido: yorkie
// pepper: burmese
```

Requirements

Supported in the Internet Explorer 11 standards document mode. Also supported in Store apps (Windows 8.1 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards. Not supported in Windows 8.

constructor Property (WeakMap)

10/18/2017 • 1 min to read • Edit Online

Specifies the function that creates a `WeakMap` object.

Syntax

```
weakmap.constructor
```

Remarks

The required `weakmap` is the name of the `WeakMap` object.

The `constructor` property is a member of the prototype of every object that has a prototype. This includes all intrinsic JavaScript objects except the `Global` and `Math` objects. The `constructor` property contains a reference to the function that constructs instances of that particular object.

Requirements

Supported in the Internet Explorer 11 standards document mode. Also supported in Store apps (Windows 8.1 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards. Not supported in Windows 8.

prototype Property (WeakMap)

10/18/2017 • 1 min to read • Edit Online

Returns a reference to the prototype for a `WeakMap` object.

Syntax

```
weakmap.prototype
```

Remarks

The `weakmap` argument is the name of a `WeakMap` object.

Use the `prototype` property to provide a base set of functionality to a class of objects. New instances of an object "inherit" the behavior of the prototype assigned to that object.

For example, to add a method to the `WeakMap` object that returns the value of the largest element of the `WeakMap`, declare the function, add it to `WeakMap.prototype`, and then use it.

Requirements

Supported in the Internet Explorer 11 standards document mode. Also supported in Store apps (Windows 8.1 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards. Not supported in Windows 8.

clear Method (WeakMap) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Removes all elements from a `WeakMap` object.

Syntax

```
weakmapObj.clear()
```

Parameters

`weakmapObj`

Required. The `WeakMap` object to clear.

Requirements

Supported in the Internet Explorer 11 standards document mode. Also supported in Store apps (Windows 8.1 and Windows Phone 8.1). See Version Information.

Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards. Not supported in Windows 8.

delete Method (WeakMap) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Removes the specified element from a `WeakMap` object.

Syntax

```
weakmapObj.delete(key)
```

Parameters

`weakmapObj`
Required. A `WeakMap` object.

`key`
Required. The key of the element to remove.

Property Value/Return Value

`true` if the element has been removed.

Example

The following example shows how to add a member to a `WeakMap` and then delete it.

```
function Dog(breed) {
    this.breed = breed;
}

var dog = new Dog("yorkie");

var wm = new WeakMap();
wm.set(dog, "fido");
wm.delete(dog);
```

Requirements

Supported in the Internet Explorer 11 standards document mode. Also supported in Store apps (Windows 8.1 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards. Not supported in Windows 8.

get Method (WeakMap) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Returns a specified element from a `WeakMap` object.

Syntax

```
weakmapObj.get(key)
```

Parameters

`weakmapObj`
Required. A `WeakMap` object.

`key`
Required. The key of an element in the `WeakMap`.

Property Value/Return Value

Returns the object associated with the key. If the `WeakMap` does not contain the key, this method returns an `undefined` value.

Example

The following example shows how to retrieve members from a `WeakMap` object.

```
var dog = {
    breed: "yorkie"
}

var cat = {
    breed: "burmese"
}

var wm = new WeakMap();
wm.set(dog, "fido");
wm.set(cat, "pepper");

document.write(wm.get(dog) + ": ");
document.write(dog.breed);
document.write("<br />");
document.write(wm.get(cat) + ": ");
document.write(cat.breed);

// Output:
// fido: yorkie
// pepper: burmese
```

Requirements

Supported in the Internet Explorer 11 standards document mode. Also supported in Store apps (Windows 8.1 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7

standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards. Not supported in Windows 8.

has Method (WeakMap) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Returns `true` if the `WeakMap` object contains the specified element.

Syntax

```
weakmapObj.has(key)
```

Parameters

`weakmapObj`
Required. A `WeakMap` object.

`key`
Required. The key of the element to test.

Property Value/Return Value

`true` if the `WeakMap` contains the specified key.

Example

The following example shows how to add a member to a `WeakMap` and then use `has` to check whether it is present.

```
var dog = {
    breed: "yorkie"
}

var wm = new WeakMap();
wm.set(dog, "fido");

document.write(wm.has(dog));

// Output:
// true
```

Requirements

Supported in the Internet Explorer 11 standards document mode. Also supported in Store apps (Windows 8.1 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards. Not supported in Windows 8.

set Method (WeakMap) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Adds a new element to a `WeakMap` object.

Syntax

```
weakmapObj.set(key, value)
```

Parameters

`weakmapObj`
Required. A `WeakMap` object.

`key`
Required. An object representing the key of the element to add. This must be an object reference.

`value`
Required. The value of the element to add.

Property Value/Return Value

Returns the `WeakMap` object that contains the new key/value pair.

Remarks

If you add a value to the collection using an existing key, the new value will replace the old value.

Example

The following example shows how to add members to a `WeakMap` object.

```
var dog = {
    breed: "yorkie"
}

var cat = {
    breed: "burmese"
}

var wm = new WeakMap();
wm.set(dog, "fido");
wm.set(cat, "pepper");
```

Requirements

Supported in the Internet Explorer 11 standards document mode. Also supported in Store apps (Windows 8.1 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards. Not supported in Windows 8.

toString Method (WeakMap) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Returns a string representation of a `WeakMap` object.

Syntax

```
weakmapObj.toString()
```

Parameters

`weakmapObj`

Required. A `WeakMap` object.

Property Value/Return Value

The string representation of the `WeakMap`.

Exceptions

Requirements

Supported in the Internet Explorer 11 standards document mode. Also supported in Store apps (Windows 8.1 and Windows Phone 8.1). See Version Information.

Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards. Not supported in Windows 8.

valueOf Method (WeakMap) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Returns the primitive value of the specified object.

Syntax

```
weakmapObj.valueOf()
```

Parameters

`weakmapObj`

Required. A `WeakMap` object.

Property Value/Return Value

Returns the `WeakMap` instance.

Requirements

Supported in the Internet Explorer 11 standards document mode. Also supported in Store apps (Windows 8.1 and Windows Phone 8.1). See Version Information.

Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards. Not supported in Windows 8.

WeakSet Object (JavaScript)

10/18/2017 • 1 min to read • Edit Online

A collection of unique objects that may be of any type.

Syntax

```
setObj = new WeakSet()
```

Remarks

If you try to add a non-unique object to a `WeakSet`, the new object will not be added to the collection.

Unlike a `Set`, only objects may be added to the collection. Arbitrary values cannot be added to the collection.

In a `WeakSet` object, references to objects in the set are held 'weakly'. This means that `WeakSet` will not prevent a garbage collection from happening on the objects. When there are no references (other than `WeakSet`) to the objects, the garbage collector may collect the objects.

`WeakSet` (or `WeakMap`) may be helpful in some scenarios involving caching of objects or object metadata. For example, metadata for non-extensible objects may be stored in a `WeakSet`, or you may create a cache of user images using `WeakSet`.

Properties

The following table lists the properties of the `WeakSet` object.

PROPERTY	DESCRIPTION
constructor	Specifies the function that creates a set.
prototype	Returns a reference to the prototype for a set.

Methods

The following table lists the methods of the `WeakSet` object.

METHOD	DESCRIPTION
add	Adds an element to a set.
delete	Removes a specified element from a set.
has	Returns `true` if the set contains a specified element.

Example

The following example shows how to add members to a set and then verify that they have been added.

```
var ws = new WeakSet();

var str = new String("Thomas Jefferson");
var num = new Number(1776);

ws.add(str);
ws.add(num);

console.log(ws.has(str));
console.log(ws.has(num));

ws.delete(str);
console.log(ws.has(str));

// Output:
// true
// true
// false
```

Requirements

Supported in Microsoft Edge (Edge browser). Also supported in Store apps (Microsoft Edge on Windows 10). See Version Information.

Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Not supported in Windows 8.1.

constructor Property (WeakSet)

10/18/2017 • 1 min to read • Edit Online

Specifies the function that creates a `WeakSet`.

Syntax

```
weakset.constructor
```

Remarks

The required `weakset` is the name of the set.

The `constructor` property is a member of the prototype of every object that has a prototype. This includes all intrinsic JavaScript objects except the `Global` and `Math` objects. The `constructor` property contains a reference to the function that constructs instances of that particular object.

Requirements

Supported in Microsoft Edge (Edge browser). Also supported in Store apps (Microsoft Edge on Windows 10). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Not supported in Windows 8.1.

prototype Property (WeakSet)

10/18/2017 • 1 min to read • Edit Online

Returns a reference to the prototype for a `WeakSet`.

Syntax

```
weakset.prototype
```

Remarks

The `weakset` argument is the name of a set.

Use the `prototype` property to provide a base set of functionality to a class of objects. New instances of an object "inherit" the behavior of the prototype assigned to that object.

For example, to add a method to the `WeakSet` object that returns the value of the largest element of the set, declare the function, add it to `WeakSet.prototype`, and then use it.

Requirements

Supported in Microsoft Edge (Edge browser). Also supported in Store apps (Microsoft Edge on Windows 10). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Not supported in Windows 8.1.

add Method (WeakSet) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Adds a new element to a `WeakSet`.

Syntax

```
weaksetObj.add(obj)
```

Parameters

`weaksetObj`
Required. A `WeakSet` object.

`obj`
Required. New element of the `WeakSet`.

Remarks

The new element must be an object, rather than an arbitrary value, and must be unique. If you add a non-unique element to a `WeakSet`, the new element will not be added to the collection.

Example

The following example shows how to add members to a set and verify that they have been added.

```
var ws = new WeakSet();

var str = new String("Thomas Jefferson");
var num = new Number(1776);

ws.add(str);
ws.add(num);

console.log(ws.has(str));
console.log(ws.has(num));

ws.delete(str);
console.log(ws.has(str));

// Output:
// true
// true
// false
```

Requirements

Supported in Microsoft Edge (Edge browser). Also supported in Store apps (Microsoft Edge on Windows 10). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Not supported in Windows 8.1.

delete Method (WeakSet) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Removes the specified element from a `WeakSet`.

Syntax

```
weaksetObj.delete(obj)
```

Parameters

`weaksetObj`
Required. A `WeakSet` object.

`obj`
Required. The element to remove.

Property Value/Return Value

`true` if the element has been removed.

Example

The following example shows how to add and delete elements of a `WeakSet`.

```
var ws = new WeakSet();

var str = new String("Thomas Jefferson");
var num = new Number(1776);

ws.add(str);
ws.add(num);

console.log(ws.has(str));
console.log(ws.has(num));

ws.delete(str);
console.log(ws.has(str));

// Output:
// true
// true
// false
```

Requirements

Supported in Microsoft Edge (Edge browser). Also supported in Store apps (Microsoft Edge on Windows 10). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Not supported in Windows 8.1.

has Method (WeakSet) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Returns `true` if the `WeakSet` contains the specified element.

Syntax

```
setObj.has(obj)
```

Parameters

`setObj`
Required. A `WeakSet` object.

`obj`
Required. The element to test.

Property Value/Return Value

`true` if the set contains the specified element.

Example

The following example shows how to add members to a `WeakSet` and then check whether the set contains a specific member.

```
var ws = new WeakSet();

var str = new String("Thomas Jefferson");
var num = new Number(1776);

ws.add(str);
ws.add(num);

console.log(ws.has(str));
console.log(ws.has(num));

ws.delete(str);
console.log(ws.has(str));

// Output:
// true
// true
// false
```

Requirements

Supported in Microsoft Edge (Edge browser). Also supported in Store apps (Microsoft Edge on Windows 10). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Not supported in Windows 8.1.

WinRTError Object (JavaScript)

10/18/2017 • 1 min to read • Edit Online

When a Windows Runtime call returns an HRESULT that indicates a failure, JavaScript converts it to a special Windows Runtime error. It is available only in Windows 8.x Store apps, when the Windows Runtime is available, as part of the global JavaScript namespace.

Syntax

```
errorObj = new WinRTError();
```

Remarks

The WinRTError error type is used only for errors that originate in Windows Runtime APIs.

Example

The following example shows how a WinRTError is thrown and caught.

```
try {
            Windows.Storage.ApplicationData.localFolder.createFileAsync("sample.txt");
        } catch (err) {
            var n = err;
        }
```

Methods

The WinRTError object has no methods.

Properties

The WinRTError object has the same properties as the Error Object object.

Requirements

The WinRTError object is supported only in Windows 8.x Store apps, not in Internet Explorer.

See Also

Error Object

JavaScript Constants

10/18/2017 • 1 min to read • Edit Online

The following table lists predefined JavaScript constants that you can use in expressions.

Constants

CONSTANT	DESCRIPTION	JAVASCRIPT OBJECT
E	The mathematical constant e. This is Euler's number, the base of natural logarithms.	`Math`
Infinity	A value that is larger than the largest floating point number. Negative Infinity (-Infinity) is smaller than the smallest floating point number.	`Global`
LN2	The natural logarithm of 2.	`Math`
LN10	The natural logarithm of 10.	`Math`
LOG2E	The base-2 logarithm of e.	`Math`
LOG10E	The base-10 logarithm of e.	`Math`
MAX_VALUE	The largest number that can be represented in JavaScript.	`Number`
MIN_VALUE	The closest number to zero that can be represented in JavaScript.	`Number`
NaN	Indicates that an arithmetic expression returned a value that is not a number.	`Number`
NaN (Global)	A value that indicates that an expression is not a number.	`Global`
NEGATIVE_INFINITY	A value that is smaller than the smallest floating point number.	`Number`
null Constant (JavaScript)	The value of a variable that does not point to valid data.	`Global`
PI	Pi. This is the ratio of the circumference of a circle to its diameter.	`Math`
POSITIVE_INFINITY	A value that is larger than the largest floating point number.	`Number`

CONSTANT	DESCRIPTION	JAVASCRIPT OBJECT
SQRT1_2	The square root of 0.5, or, equivalently, one divided by the square root of 2.	`Math`
SQRT2	The square root of 2.	`Math`
undefined	The value of a variable that has been declared but not initialized.	`Global`

See Also

Global Object
Math Object
Number Object
JavaScript Reference

JavaScript Properties

10/18/2017 • 2 min to read • Edit Online

The following table listsJavaScript properties.

Properties

PROPERTY	DESCRIPTION	JAVASCRIPT OBJECT
*_proto_* Property	Sets the prototype for an object.	`Object`
0...n Properties	Returns the value of individual arguments from an **arguments** object.	`arguments`
$1...$9 Properties	Returns the nine most-recently memorized portions found during pattern matching.	`RegExp`
arguments Property	Returns an array containing each argument passed to the currently executing function.	`Function`
callee Property	Returns the `Function` object being executed.	`arguments`
caller Property	Returns a reference to the function that invoked the current function.	`Function`
constructor Property	Specifies the function that creates an object.	Multiple
description Property	Returns or sets the descriptive string associated with a specific error.	`Error`
global Property	Returns a Boolean value indicating the state of the global flag (**g**) used with a regular expression.	`Regular Expression`
ignoreCase Property	Returns a Boolean value indicating the state of the ignoreCase flag (**i**) used with a regular expression.	`Regular Expression`
index Property	Returns the character position where the first successful match begins in a searched string.	`RegExp`
input Property	Returns the string against which a search was performed.	`RegExp`
lastIndex Property	Returns the character position where the last successful match begins in a searched string.	`RegExp`

PROPERTY	DESCRIPTION	JAVASCRIPT OBJECT
lastMatch Property	Returns the last matched characters from any regular expression search.	`RegExp`
lastParen Property	Returns the last parenthesized submatch from any regular expression search, if any.	`RegExp`
leftContext Property	Returns the characters from the beginning of a searched string up to the position before the beginning of the last match.	`RegExp`
length Property (arguments)	Returns the actual number of arguments passed to a function by the caller.	`arguments`
length Property (Array)	Returns an integer value one higher than the highest element defined in an array.	`Array`
length Property (Function)	Returns the number of arguments defined for a function.	`Function`
length Property (String)	Returns the length of a `String` object.	`String`
message Property	Returns an error message string.	`Error`
multiline Property	Returns a Boolean value indicating the state of the multiline flag (**m**) used with a regular expression.	`Regular Expression`
name Property	Returns the name of an error.	`Error`
number Property	Returns or sets the numeric value associated with a specific error.	`Error`
prototype Property	Returns a reference to the prototype for a class of objects.	Multiple
rightContext Property	Returns the characters from the position following the last match to the end of the searched string.	`RegExp`
source Property	Returns a copy of the text of the regular expression pattern.	`Regular Expression`

See Also

JavaScript Objects
JavaScript Constants
JavaScript Methods
JavaScript Functions

JavaScript Functions

10/18/2017 • 4 min to read • Edit Online

The following table lists JavaScript functions.

Functions

FUNCTION	DESCRIPTION	JAVASCRIPT OBJECT
abs Function	Returns the absolute value of a number.	`Math`
acos Function	Returns the arccosine of a number.	`Math`
asin Function	Returns the arcsine of a number.	`Math`
atan Function	Returns the arctangent of a number.	`Math`
atan2 Function	Returns the angle (in radians) from the X axis to a point (y,x).	`Math`
ceil Function	Returns the smallest integer greater than or equal to its numeric argument.	`Math`
cos Function	Returns the cosine of a number.	`Math`
create Function	Creates an object that has a specified prototype, and that optionally contains specified properties.	`Object`
decodeURI Function	Returns the unencoded version of an encoded Uniform Resource Identifier.	`Global`
decodeURIComponent Function	Returns the unencoded version of an encoded component of a Uniform Resource Identifier.	`Global`
defineProperties Function	Adds one or more properties to an object, and/or modifies attributes of existing properties.	`Object`
defineProperty Function	Adds a property to an object, or modifies attributes of an existing property.	`Object`
encodeURI Function	Encodes a text string as a valid Uniform Resource Identifier.	`Global`
encodeURIComponent Function	Encodes a text string as a valid component of a Uniform Resource Identifier.	`Global`

FUNCTION	DESCRIPTION	JAVASCRIPT OBJECT
escape Function	Encodes `String` objects so they can be read on all computers.	`Global`
eval Function	Evaluates JavaScript code and executes it.	`Global`
exp Function	Returns *e* (the base of natural logarithms) raised to a power.	`Math`
floor Function	Returns the greatest integer less than or equal to its numeric argument.	`Math`
freeze Function	Prevents the modification of existing property attributes and values, and prevents the addition of new properties.	`Object`
fromCharCode Function	Returns a string from a number of Unicode character values.	`String`
GetObject Function	Returns a reference to an Automation object from a file.	`Global`
getOwnPropertyDescriptor Function	Returns the definition of a data property or an accessor property.	`Object`
getOwnPropertyNames Function	Returns the names of the properties and methods of an object.	`Object`
getPrototypeOf Function	Returns the prototype of an object.	`Object`
isArray Function	Returns a Boolean value that indicates whether an object is an array.	`Array`
isExtensible Function	Returns a value that indicates whether new properties can be added to an object.	`Object`
isFinite Function	Returns a Boolean value that indicates if a supplied number is finite.	`Global`
isFrozen Function	Returns `true` if existing property attributes and values cannot be modified in an object and new properties cannot be added to the object.	`Object`
isNaN Function	Returns a Boolean value that indicates whether a value is the reserved value `NaN` (not a number).	`Global`

FUNCTION	DESCRIPTION	JAVASCRIPT OBJECT
isSealed Function	Returns `true` if existing property attributes cannot be modified in an object and new properties cannot be added to the object.	`Object`
keys Function	Returns the names of the enumerable properties and methods of an object.	`Object`
log Function	Returns the natural logarithm of a number.	`Math`
max Function	Returns the greater of two supplied numeric expressions.	`Math`
min Function	Returns the lesser of two supplied numbers.	`Math`
now Function	Returns the number of milliseconds between January 1, 1970, and the current date and time.	`Date`
parse Function (Date)	Parses a string containing a date, and returns the number of milliseconds between that date and midnight, January 1, 1970.	`Date`
parse Function (JSON)	De-serializes JSON text to produce an in-memory object or array.	`JSON`
parseFloat Function	Returns a floating-point number converted from a string.	`Global`
parseInt Function	Returns an integer converted from a string.	`Global`
pow Function	Returns the value of a base expression raised to a specified power.	`Math`
preventExtensions Function	Prevents the addition of new properties to an object.	`Object`
random Function	Returns a pseudorandom number between 0 and 1.	`Math`
round Function	Returns a specified numeric expression rounded to the nearest integer.	`Math`
ScriptEngine Function	Returns a string representing the scripting language in use.	`Global`
ScriptEngineBuildVersion Function	Returns the build version number of the scripting engine in use.	`Global`

FUNCTION	DESCRIPTION	JAVASCRIPT OBJECT
ScriptEngineMajorVersion Function	Returns the major version number of the scripting engine in use.	`Global`
ScriptEngineMinorVersion Function	Returns the minor version number of the scripting engine in use.	`Global`
seal Function	Prevents the modification of attributes of existing properties, and prevents the addition of new properties.	`Object`
sin Function	Returns the sine of a number.	`Math`
sqrt Function	Returns the square root of a number.	`Math`
stringify Function	Serializes an object or array into JavaScript Object Notation (JSON) text.	`JSON`
tan Function	Returns the tangent of a number.	`Math`
unescape Function	Decodes `String` objects encoded with the **escape** method.	`Global`
UTC Function	Returns the number of milliseconds between midnight, January 1, 1970 Universal Coordinated Time (UTC) (or GMT) and the supplied date.	`Date`
write Function	Sends strings to the script debugger.	`Debug`
writeln Function	Sends strings to the script debugger, followed with a newline character.	`Debug`

See Also

JavaScript Objects
JavaScript Methods
JavaScript Properties
JavaScript Constants
Version Information
JavaScript Reference

JavaScript Methods

10/18/2017 • 8 min to read • Edit Online

The following table lists JavaScript methods.

Methods

METHOD	DESCRIPTION	JAVASCRIPT OBJECT
anchor method	Places an HTML anchor that has a NAME attribute around text.	`String`
apply method	Applies a method of an object, substituting another object for the current object.	`Function`
atEnd method	Returns a Boolean value indicating if the enumerator is at the end of the collection.	`Enumerator`
big method	Places HTML <BIG> tags around text.	`String`
bind method	Creates a function that is associated with a specified object, and that can have specific initial parameters.	`Function`
blink method	Places HTML <BLINK> tags around text.	`String`
bold method	Places HTML <B> tags around text.	`String`
call method	Calls a method of an object, substituting another object for the current object.	`Function`
charAt method	Returns the character at the specified index.	`String`
charCodeAt method	Returns the Unicode encoding of the specified character.	`String`
compile method	Compiles a regular expression into an internal format.	`Regular Expression`
concat method (Array)	Returns a new array consisting of a combination of two arrays.	`Array`
concat method (String)	Returns a `String` object containing the concatenation of two supplied strings.	`String`

METHOD	DESCRIPTION	JAVASCRIPT OBJECT
dimensions method	Returns the number of dimensions in a VBArray.	`VBArray`
every method	Checks whether a defined callback function returns `true` for all elements in an array.	`Array`
exec method	Executes a search for a match in a specified string.	`Regular Expression`
filter method	Calls a defined callback function on each element of an array, and returns an array of values for which the callback function returns `true`.	`Array`
fixed method	Places HTML <TT> tags around text.	`String`
fontcolor method	Places HTML <FONT> tags with a COLOR attribute around text.	`String`
fontsize method	Places HTML <FONT> tags with a SIZE attribute around text.	`String`
forEach method	Calls a defined callback function for each element in an array.	`Array`
getDate method	Returns the day-of-the-month value using local time.	`Date`
getDay method	Returns the day-of-the-week value using local time.	`Date`
getFullYear method	Returns the year value using local time.	`Date`
getHours method	Returns the hours value using local time.	`Date`
getItem method	Returns the item at the specified location.	`VBArray`
getMilliseconds method	Returns the milliseconds value using local time.	`Date`
getMinutes method	Returns the minutes value using local time.	`Date`
getMonth method	Returns the month value using local time.	`Date`
getSeconds method	Returns seconds value using local time.	`Date`

METHOD	DESCRIPTION	JAVASCRIPT OBJECT
getTime method	Returns the time value in a `Date` Object as the number of milliseconds since midnight January 1, 1970.	`Date`
getTimezoneOffset method	Returns the difference in minutes between the time on the host computer and Universal Coordinated Time (UTC).	`Date`
getUTCDate method	Returns the day-of-the-month value using UTC.	`Date`
getUTCDay method	Returns the day-of-the-week value using UTC.	`Date`
getUTCFullYear method	Returns the year value using UTC.	`Date`
getUTCHours method	Returns the hours value using UTC.	`Date`
getUTCMilliseconds method	Returns the milliseconds value using UTC.	`Date`
getUTCMinutes method	Returns the minutes value using UTC.	`Date`
getUTCMonth method	Returns the month value using UTC.	`Date`
getUTCSeconds method	Returns the seconds value using UTC.	`Date`
getVarDate method	Returns the VT_DATE value in a `Date` object.	`Date`
getYear method	Returns the year value .	`Date`
hasOwnProperty method	Returns a Boolean value that indicates whether an object has a property with the specified name.	Multiple
indexOf method (Array)	Returns the index of the first occurrence of a value in an array.	`Array`
indexOf method (String)	Returns the character position where the first occurrence of a substring occurs within a `String` object.	`String`
isPrototypeOf method	Returns a Boolean value that indicates whether an object exists in another object's prototype chain.	Multiple
italics method	Places HTML <I> tags around text.	`String`
item method	Returns the current item in the collection.	`Enumerator`

METHOD	DESCRIPTION	JAVASCRIPT OBJECT
join method	Returns a `String` object consisting of all the elements of an array concatenated together.	`Array`
lastIndexOf method (Array)	Returns the index of the last occurrence of a specified value in an array.	`Array`
lastIndexOf method (String)	Returns the last occurrence of a substring within a `String` object.	`String`
lbound method	Returns the lowest index value used in the specified dimension of a VBArray.	`VBArray`
link method	Places an HTML anchor that has an HREF attribute around text.	`String`
localeCompare method	Returns a value indicating whether two strings are equivalent in the current locale.	`String`
map method	Calls a defined callback function on each element of an array, and returns an array that contains the results.	`Array`
match method	Returns, as an array, the results of a search on a string using a supplied **Regular Expression** object.	`String`
moveFirst method	Resets the current item in the collection to the first item.	`Enumerator`
moveNext method	Moves the current item to the next item in the collection.	`Enumerator`
pop method	Removes the last element from an array and returns it.	`Array`
propertyIsEnumerable method	Returns a Boolean value that indicates whether a specified property is part of an object and whether it is enumerable.	Multiple
push method	Appends new elements to an array, and returns the new length of the array.	`Array`
reduce method	Accumulates a single result by calling a defined callback function for all elements in an array. The return value of the callback function is the accumulated result, and is provided as an argument in the next call to the callback function.	`Array`

METHOD	DESCRIPTION	JAVASCRIPT OBJECT
reduceRight method	Accumulates a single result by calling a defined callback function for all elements in an array, in descending order. The return value of the callback function is the accumulated result, and is provided as an argument in the next call to the callback function.	`Array`
replace method	Returns a copy of a string with text replaced using a regular expression.	`String`
reverse method	Returns an `Array` object with the elements reversed.	`Array`
search method	Returns the position of the first substring match in a regular expression search.	`String`
setDate method	Sets the numeric day of the month using local time.	`Date`
setFullYear method	Sets the year value using local time.	`Date`
setHours method	Sets the hours value using local time.	`Date`
setMilliseconds method	Sets the milliseconds value using local time.	`Date`
setMinutes method	Sets the minutes value using local time.	`Date`
setMonth method	Sets the month value using local time.	`Date`
setSeconds method	Sets the seconds value using local time.	`Date`
setTime method	Sets the date and time value in the `Date` object.	`Date`
setUTCDate method	Sets the numeric day of the month using UTC.	`Date`
setUTCFullYear method	Sets the year value using UTC.	`Date`
setUTCHours method	Sets the hours value using UTC.	`Date`
setUTCMilliseconds method	Sets the milliseconds value using UTC.	`Date`
setUTCMinutes method	Sets the minutes value using UTC.	`Date`
setUTCMonth method	Sets the month value using UTC.	`Date`
setUTCSeconds method	Sets the seconds value using UTC.	`Date`

METHOD	DESCRIPTION	JAVASCRIPT OBJECT
setYear method	Sets the year value using local time.	`Date`
shift method	Removes the first element from an array and returns it.	`Array`
slice method (Array)	Returns a section of an array.	`Array`
slice method (String)	Returns a section of a string.	`String`
small method	Places HTML <SMALL> tags around text.	`String`
some method	Checks whether a defined callback function returns `true` for any element of an array.	`Array`
sort method	Returns an `Array` object with the elements sorted.	`Array`
splice method	Removes elements from an array and, if necessary, inserts new elements in their place, returning the deleted elements.	`Array`
split method	Returns the array of strings that results when a string is separated into substrings.	`String`
strike method	Places HTML <STRIKE> tags around text.	`String`
sub method	Places HTML <SUB> tags around text.	`String`
substr method	Returns a substring beginning at a specified location and having a specified length.	`String`
substring method	Returns the substring at a specified location within a `String` object.	`String`
sup method	Places HTML <SUP> tags around text.	`String`
test method	Returns a Boolean value that indicates whether or not a pattern exists in a searched string.	`Regular Expression`
toArray method	Returns a standard JavaScript array converted from a VBArray.	`VBArray`
toDateString method	Returns a date as a string value.	`Date`
toExponential method	Returns a string containing a number represented in exponential notation.	`Number`

METHOD	DESCRIPTION	JAVASCRIPT OBJECT
toFixed method	Returns a string representing a number in fixed-point notation.	`Number`
toGMTString method	Returns a date converted to a string using Greenwich Mean Time (GMT).	`Date`
toISOString method	Returns a date as a string value in ISO format.	`Date`
toJSON method	Used to transform data of an object type before the JSON serialization.	`Date`
toLocaleDateString method	Returns a date as a string value appropriate to the host environment's current locale.	`Date`
toLocaleLowerCase method	Returns a string where all alphabetic characters have been converted to lowercase, taking into account the host environment's current locale.	`String`
toLocaleString method	Returns an object converted to a string using the current locale.	Multiple
toLocaleTimeString method	Returns a time as a string value appropriate to the host environment's current locale.	`Date`
toLocaleUpperCase method	Returns a string where all alphabetic characters have been converted to uppercase, taking into account the host environment's current locale.	`String`
toLowerCase method	Returns a string where all alphabetic characters have been converted to lowercase.	`String`
toPrecision method	Returns a string containing a number represented either in exponential or fixed-point notation with a specified number of digits.	`Number`
toString method	Returns a string representation of an object.	Multiple
toTimeString method	Returns a time as a string value.	`Date`
toUpperCase method	Returns a string where all alphabetic characters have been converted to uppercase.	`String`
toUTCString method	Returns a date converted to a string using UTC.	`Date`

METHOD	DESCRIPTION	JAVASCRIPT OBJECT
trim method	Returns a string with leading and trailing white space and line terminator characters removed.	`String`
ubound method	Returns the highest index value used in the specified dimension of the VBArray.	`VBArray`
unshift method	Inserts new elements at the start of an array.	`Array`
valueOf method	Returns the primitive value of the specified object.	Multiple

See Also

JavaScript Objects
JavaScript Functions
JavaScript Properties
JavaScript Constants
Version Information
JavaScript Reference

JavaScript Operators

3/2/2018 • 3 min to read • Edit Online

The following table lists JavaScript operators.

Operators

DESCRIPTION	LANGUAGE ELEMENT
Adds the value of an expression to the value of a variable and assigns the result to the variable.	Addition Assignment Operator (+=)
Sums two numbers or concatenates two strings.	Addition Operator (+)
Assigns a value to a variable.	Assignment Operator (=)
Performs a bitwise AND on the value of a variable and the value of an expression and assigns the result to the variable.	Bitwise AND Assignment Operator (&=)
Performs a bitwise AND on two expressions.	Bitwise AND Operator (&)
Shifts the bits of an expression to the left.	Bitwise Left Shift Operator (<<)
Performs a bitwise NOT (negation) on an expression.	Bitwise NOT Operator (~)
Performs a bitwise OR on the value of a variable and the value of an expression and assigns the result to the variable.	Bitwise OR Assignment Operator (|=)
Performs a bitwise OR on two expressions.	Bitwise OR Operator (|)
Shifts the bits of an expression to the right, maintaining sign.	Bitwise Right Shift Operator (>>)
Performs a bitwise exclusive OR on a variable and an expression and assigns the result to the variable.	Bitwise XOR Assignment Operator (^=)
Performs a bitwise exclusive OR on two expressions.	Bitwise XOR Operator (^)
Causes two expressions to be executed sequentially.	Comma Operator (,)
Returns a Boolean value indicating the result of the comparison.	Comparison Operators
List of compound assignment operators.	Compound Assignment Operators
Executes one of two expressions depending on a condition.	Conditional (ternary) Operator (?:)
Decrements a variable by one.	Decrement Operator (--)
Deletes a property from an object, or removes an element from an array.	delete Operator

DESCRIPTION	LANGUAGE ELEMENT
Divides the value of a variable by the value of an expression and assigns the result to the variable.	Division Assignment Operator (/=)
Divides two numbers and returns a numeric result.	Division Operator (/)
Tests for the existence of a property in an object.	in Operator
Compares two expressions to determine if they are equal.	Equality Operator (==)
Compares two expressions to determine if one is greater than the other.	Greater than Operator (>)
Compares two expressions to determine if one is greater than or equal to the other.	Greater than or equal to Operator (>=)
Compares two expressions to determine if they are equal in value and of the same data type.	Identity Operator (===)
Increments a variable by one.	Increment Operator (++)
Compares two expressions to determine if they are unequal.	Inequality Operator (!=)
Returns a Boolean value that indicates whether or not an object is an instance of a particular class.	instanceof Operator
Left shifts the value of a variable by the number of bits specified in the value of an expression and assigns the result to the variable.	Left Shift Assignment Operator (<<=)
Compares two expressions to determine if one is less than the other.	Less than Operator (<)
Compares two expressions to determine if one is less than or equal to the other.	Less than or equal to Operator (<=)
Performs a logical conjunction on two expressions.	Logical AND Operator (&&)
Performs logical negation on an expression.	Logical NOT Operator (!)
Performs a logical disjunction on two expressions.	Logical OR Operator (||)
Divides the value of a variable by the value of an expression, and assigns the remainder to the variable.	Remainder Assignment Operator (%=)
Divides two numbers and returns the remainder.	Remainder Operator (%)
Multiplies the value of a variable by the value of an expression and assigns the result to the variable.	Multiplication Assignment Operator (*=)
Multiplies two numbers.	Multiplication Operator (*)
Creates a new object.	new Operator

DESCRIPTION	LANGUAGE ELEMENT
Compares two expressions to determine that they are not equal in value or of the same data type.	Nonidentity Operator (!==)
Right shifts the value of a variable by the number of bits specified in the value of an expression, maintaining the sign, and assigns the result to the variable.	Right Shift Assignment Operator (>>=)
Allows an array to be initialized from multiple elements of an array literal, or allows an expression to be expanded to multiple arguments (in function calls).	Spread operator
Subtracts the value of an expression from the value of a variable and assigns the result to the variable.	Subtraction Assignment Operator (-=)
Performs subtraction of two expressions.	Subtraction Operator (-)
Returns a string that identifies the data type of an expression.	typeof Operator
Indicates the negative value of a numeric expression.	Unary Negation Operator (-)
Right shifts the value of a variable by the number of bits specified in the value of an expression, without maintaining sign, and assigns the result to the variable.	Unsigned Right Shift Assignment Operator (>>>=)
Performs an unsigned right shift of the bits in an expression.	Unsigned Right Shift Operator (>>>)
Prevents an expression from returning a value.	void Operator

See Also

Operator Precedence

Addition Assignment Operator (+=) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Adds the value of an expression to the value of a variable and assigns the result to the variable.

Syntax

```
result += expression
```

Parameters

`result`
Any variable.

`expression`
Any expression.

Remarks

Using this operator is exactly the same as specifying: `result = result + expression`.

The types of the two expressions determine the behavior of the `+=` operator.

IF	THEN
Both expressions are numeric or Boolean	Add
Both expressions are strings	Concatenate
One expression is numeric and the other is a string	Concatenate

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

See Also

Addition Operator (+)
Operator Precedence
Operator Summary (JavaScript)

Addition Operator (+) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Adds the value of one numeric expression to another, or concatenates two strings.

Syntax

```
result = expression1 + expression2
```

Parameters

`result`
Any variable.

`expression1`
Any expression.

`expression2`
Any expression.

Remarks

The types of the two expressions determine the behavior of the **+** operator.

IF	THEN
Both expressions are numeric or Boolean	Add
Both expressions are strings	Concatenate
One expression is numeric and the other is a string	Concatenate

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

See Also

Addition Assignment Operator (+=)
Operator Precedence
Operator Summary (JavaScript)

Assignment Operator (=) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Assigns a value to a variable.

Syntax

```
result = expression
```

Parameters

`result`
Any variable.

`expression`
Any numeric expression.

Remarks

The = operator behaves like other operators, so expressions that contain it have a value. This means that you can chain assignment operators as follows: `j = k = l = 0`. In this case `j`, `k`, and `l` equal zero.

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

See Also

Operator Precedence
Operator Summary (JavaScript)

Bitwise AND Assignment Operator (&=) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Sets the result of a bitwise AND operation on the value of a variable and the value of an expression. The variable and the expression are treated as 32-bit integers.

Syntax

```
result &= expression
```

Parameters

`result`
Any variable.

`expression`
Any expression.

Remarks

Using this operator is the same as specifying:

```
result = result & expression
```

The Bitwise AND Operator (&) looks at the binary representation of the values of `result` and `expression` and does a bitwise AND operation on them. The output of this operation behaves like this:

```
// 9 is 00000000000000000000000000001001
var expr1 = 9;

// 5 is 00000000000000000000000000000101
var expr2 = 5;

// 1 is 00000000000000000000000000000001
expr1 &= expr2;

document.write(expr1);
```

Any time both of the expressions have a 1 in a digit, the result has a 1 in that digit. Otherwise, the result has a 0 in that digit.

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

See Also

Bitwise AND Operator (&)
Operator Precedence
Operator Summary (JavaScript)

Bitwise AND Operator (&) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Performs a bitwise AND operation on two 32-bit expressions.

Syntax

```
result = expression1 & expression2
```

Parameters

`result`
The result of the operation.

`expression1`
Any expression.

`expression2`
Any expression.

Remarks

The `&` does a bitwise AND operation on the each of the bits of two 32-bit expressions. If both of the bits are 1, the result is 1. Otherwise, the result is 0.

BIT1	BIT2	ANDED VALUE
0	0	0
1	1	1
1	0	0
0	1	0

The following examples show how to use the `&` operator.

```
// 9 is 00000000000000000000000000001001
var expr1 = 9;

// 5 is 00000000000000000000000000000101
var expr2 = 5;

// 1 is 00000000000000000000000000000001
var result = expr1 & expr2;

document.write(result);
// Output: 1
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

See Also

Bitwise AND Assignment Operator (&=)
Operator Precedence
Operator Summary (JavaScript)

Bitwise Left Shift Operator (<<) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Left shifts the bits of an expression.

Syntax

```
result = expression1 << expression2
```

Parameters

result
Any variable.

expression1
Any expression.

expression2
Any expression.

Remarks

The **<<** operator shifts the bits of *expression1* left by the number of bits specified in *expression2*. For example:

```
var temp
temp = 14 << 2
```

The variable *temp* has a value of 56 because 14 (00001110 in binary) shifted left two bits equals 56 (00111000 in binary).

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

See Also

Left Shift Assignment Operator (<<=)
Bitwise Right Shift Operator (>>)
Unsigned Right Shift Operator (>>>)
Operator Precedence
Operator Summary (JavaScript)

Bitwise NOT Operator (~) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Performs a bitwise NOT (negation) on an expression.

Syntax

```
result = ~ expression
```

Parameters

result
Any variable.

expression
Any expression.

Remarks

All unary operators, such as the `~` operator, evaluate expressions as follows:

- If applied to undefined or `null` expressions, a run-time error is raised.
- Objects are converted to strings.
- Strings are converted to numbers if possible. If not, a run-time error is raised.
- Boolean values are treated as numbers (0 if false, 1 if true).

  The operator is applied to the resulting number.

  The `~` operator looks at the binary representation of the values of the expression and does a bitwise negation operation on it.

  Any digit that is a 1 in the expression becomes a 0 in the result. Any digit that is a 0 in the expression becomes a 1 in the result.

  The following example illustrates use of the bitwise NOT (~) operator.

```
var temp = ~5;
```

The resulting value is -6, as shown in the following table.

EXPRESSION	BINARY VALUE (TWO'S COMPLEMENT)	DECIMAL VALUE
5	00000000 00000000 00000000 00000101	5
~5	11111111 11111111 11111111 11111010	-6

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

See Also

Logical NOT Operator (!)
Operator Precedence
Operator Summary (JavaScript)

Bitwise OR Assignment Operator (|=) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Performs a bitwise OR on the value of a variable and the value of an expression and assigns the result to the variable.

Syntax

```
result |= expression
```

Parameters

result
Any variable.

expression
Any expression.

Remarks

Using this operator is exactly the same as specifying:

```
result = result | expression
```

The **|=** operator looks at the binary representation of the values of *result* and *expression* and does a bitwise OR operation on them. The result of this operation behaves like this:

```
0101    (result)
1100    (expression)
----
1101    (output)
```

Any time either of the expressions has a 1 in a digit, the result has a 1 in that digit. Otherwise, the result has a 0 in that digit.

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

See Also

Bitwise OR Operator (|)
Operator Precedence
Operator Summary (JavaScript)

Bitwise OR Operator (|) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Performs a bitwise OR on two expressions.

Syntax

```
result = expression1 | expression2
```

Parameters

result
Any variable.

expression1
Any expression.

expression2
Any expression.

Remarks

The **|** operator looks at the binary representation of the values of two expressions and does a bitwise OR operation on them. The result of this operation behaves as follows:

```
0101   (expression1)
1100   (expression2)
----
1101   (result)
```

Any time either of the expressions has a 1 in a digit, the result will have a 1 in that digit. Otherwise, the result will have a 0 in that digit.

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

See Also

Bitwise OR Assignment Operator (|=)
Operator Precedence
Operator Summary (JavaScript)

Bitwise Right Shift Operator (>>) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Right shifts the bits of an expression, maintaining sign.

Syntax

```
result = expression1 >> expression2
```

Parameters

result
Any variable.

expression1
Any expression.

expression2
Any expression.

Remarks

The >> operator shifts the bits of *expression1* right by the number of bits specified in *expression2*. The sign bit of *expression1* is used to fill the digits from the left. Digits shifted off the right are discarded. For example, after the following code is evaluated, *temp* has a value of -4: -14 (11110010 in two's complement binary) shifted right two bits equals -4 (11111100 in two's complement binary).

```
var temp
temp = -14 >> 2
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

See Also

Bitwise Left Shift Operator (<<)
Right Shift Assignment Operator (>>=)
Unsigned Right Shift Operator (>>>)
Operator Precedence
Operator Summary (JavaScript)

Bitwise XOR Assignment Operator (^=) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Performs a bitwise exclusive OR on a variable and an expression and assigns the result to the variable.

Syntax

```
result ^= expression
```

Parameters

result
Any variable.

expression
Any expression.

Remarks

Using the **^=** operator is exactly the same as specifying:

```
result = result ^ expression
```

The **^=** operator looks at the binary representation of the values of two expressions and does a bitwise exclusive OR operation on them. The result of this operation behaves as follows:

```
0101    (result)
1100    (expression)
----
1001    (result)
```

When one, and only one, of the expressions has a 1 in a digit, the result has a 1 in that digit. Otherwise, the result has a 0 in that digit.

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

See Also

Bitwise XOR Operator (^)
Operator Precedence
Operator Summary (JavaScript)

Bitwise XOR Operator (^) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Performs a bitwise exclusive OR on two expressions.

Syntax

```
result = expression1 ^ expression2
```

Parameters

result
Any variable.

expression1
Any expression.

expression2
Any expression.

Remarks

The **^** operator looks at the binary representation of the values of two expressions and does a bitwise exclusive OR operation on them. The result of this operation behaves as follows:

```
0101   (expression1)
1100   (expression2)
----
1001   (result)
```

When one, and only one, of the expressions has a 1 in a digit, the result has a 1 in that digit. Otherwise, the result has a 0 in that digit.

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

See Also

Bitwise XOR Assignment Operator (^=)
Operator Precedence
Operator Summary (JavaScript)

Comma Operator (,) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Causes two expressions to be executed sequentially.

Syntax

```
expression1, expression2
```

Parameters

`expression1`
Any expression.

`expression2`
Any expression.

Remarks

The `,` operator causes the expressions to be executed in left-to-right order. A common use for the `,` operator is in the increment expression of a `for` loop. For example:

```
j=25;
for (i = 0; i < 10; i++, j++)
{
   k = i + j;
}
```

The `for` statement allows only a single expression to be executed at the end of every pass through a loop. The `,` operator allows multiple expressions to be treated as a single expression, so both variables can be incremented.

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

See Also

for Statement
Operator Precedence
Operator Summary (JavaScript)

Comparison Operators (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Returns a Boolean value indicating the result of the comparison.

Syntax

```
expression1 comparisonoperator expression2
```

Parameters

`expression1`
Any expression.

`comparisonoperator`
Any comparison operator.

`expression2`
Any expression.

Remarks

When comparing strings, JavaScript uses the Unicode character value of the string expression.

The following describes how the different groups of operators behave depending on the types and values of `expression1` and `expression2`:

Relational operators: `<`, `>`, `<=`, `>=`

- Attempt to convert both `expression1` and `expression2` into numbers.
- If both expressions are strings, do a string comparison.
- If either expression is `NaN`, return `false`.
- Negative zero equals Positive zero.
- Negative Infinity is less than everything including itself.
- Positive Infinity is greater than everything including itself.

  Equality operators: `==`, `!=`

- If the types of the two expressions are different, attempt to convert them to a String, Number, or Boolean.
- `NaN` is not equal to anything including itself.
- Negative zero equals positive zero.
- `null` equals both `null` and `undefined`.
- Values are considered equal if they are identical strings, numerically equivalent numbers, the same object, identical Boolean values, or (if different types) they can be coerced into one of these

situations.

- Every other comparison is considered unequal.

  Identity operators: `===`, `!==`

  These operators behave the same as the equality operators, except that no type conversion is done. If the types of both expressions are not the same, these expressions always return `false`.

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

See Also

Operator Precedence
Operator Summary (JavaScript)

Compound Assignment Operators (JavaScript)

3/2/2018 • 1 min to read • Edit Online

The following table lists JavaScript assignment operators.

Assignment Operators

OPERATOR	SYMBOL
Addition	+=
Bitwise AND	&=
Bitwise Or	|=
Bitwise XOR	^=
Division	/=
Left Shift	<<=
Remainder	%=
Multiplication	*=
Right Shift	>>=
Subtraction	-=
Unsigned Right Shift	>>>=

See Also

Assignment Operator (=)

Conditional (Ternary) Operator (?:) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Returns one of two expressions depending on a condition.

Syntax

```
test ? expression1 : expression2
```

Parameters

`test`
Any Boolean expression.

`expression1`
An expression returned if `test` is `true`. May be a comma expression.

`expression2`
An expression returned if `test` is `false`. More than one expression may be a linked by a comma expression.

Remarks

The `?:` operator can be used as a shortcut for an `if...else` statement. It is typically used as part of a larger expression where an `if...else` statement would be awkward. For example:

```
var now = new Date();
var greeting = "Good" + ((now.getHours() > 17) ? " evening." : " day.");
```

The example creates a string containing "Good evening." if it is after 6pm. The equivalent code using an `if...else` statement would look as follows:

```
var now = new Date();
var greeting = "Good";
if (now.getHours() > 17)
   greeting += " evening.";
else
   greeting += " day.";
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

See Also

if...else Statement
Operator Precedence

Operator Summary (JavaScript)
Script Junkie configuration widget sample app

delete Operator (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Deletes a property from an object, or removes an element from an array.

Syntax

```
delete expression
```

Remarks

The `expression` argument is a valid JavaScript expression that usually results in a property name or array element.

If the result of `expression` is an object, the property specified in `expression` exists, and the object will not allow it to be deleted, `false` is returned.

In all other cases, `true` is returned.

Example

The following example shows how to remove an element from an array.

```
// Create an array.
var ar = new Array (10, 11, 12, 13, 14);

// Remove an element from the array.
delete ar[1];

// Print the results.
document.write ("element 1: " + ar[1]);
document.write ("<br />");
document.write ("array: " + ar);
// Output:
//  element 1: undefined
//  array: 10,,12,13,14
```

Example

The following example shows how to delete properties from an object.

```
// Create an object and add expando properties.
var myObj = new Object();
myObj.name = "Fred";
myObj.count = 42;

// Delete the properties from the object.
delete myObj.name;
delete myObj["count"];

// Print the results.
document.write ("name: " + myObj.name);
document.write ("<br />");
document.write ("count: " + myObj.count);
// Output:
//  name: undefined
//  count: undefined
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

See Also

Operator Precedence
Operator Summary (JavaScript)

Division Assignment Operator (-=) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Divides the value of a variable by the value of an expression and assigns the result to the variable.

Syntax

```
result /= expression
```

Parameters

`result`
Any numeric variable.

`expression`
Any numeric expression.

Remarks

Using the `/=` operator is the same as specifying: `result = result / expression`.

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

See Also

Division Operator (/)
Operator Precedence
Operator Summary (JavaScript)

Division Operator (-) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Divides the value of two expressions.

Syntax

```
result = number1 / number2
```

Parameters

`result`
Any numeric variable.

`number1`
Any numeric expression.

`number2`
Any numeric expression.

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

See Also

Division Assignment Operator (/=)
Operator Precedence
Operator Summary (JavaScript)

in Operator (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Tests for the existence of a property in an object.

Syntax

```
result = property in object
```

Parameters

`result`
Required. Any variable.

`property`
Required. An expression that evaluates to a string expression.

`object`
Required. Any object.

Remarks

The `in` operator determines whether an object has a property named `property`. It also determines whether the property is part of the object's prototype chain. For more information about object prototypes, see Prototypes and Prototype Inheritance.

Example

The following example shows how to use the `in` operator:

```
// Create an object that has some properties.
var myObject = new Object();
myObject.name = "James";
myObject.age = "22";
myObject.phone = "555 0234";

if ("phone" in myObject)
   document.write ("property is present");
else
   document.write ("property is not present");

// Output: property is present
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

See Also

Operator Precedence
Operator Summary (JavaScript)

Increment (++) and Decrement (--) Operators (JavaScript)

10/18/2017 • 1 min to read • Edit Online

The increment operator increments, and the decrement operator decrements the value of a variable by one.

Syntax

```
      result = ++variable
result = --variable
result = variable++
result = variable--
```

Parameters

`result`
Any variable.

`variable`
Any variable.

Remarks

If the operator appears before the variable, the value is modified before the expression is evaluated. If the operator appears after the variable, the value is modified after the expression is evaluated. In other words, given `j = ++k;`, the value of `j` is the original value of `k` plus one; given `j = k++;`, the value of `j` is the original value of `k`, which is incremented after its value is assigned to `j`.

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

See Also

Operator Precedence
Operator Summary (JavaScript)

instanceof Operator (JavaScript)

1/13/2018 • 1 min to read • Edit Online

Returns a Boolean value that indicates whether or not an object is an instance of a particular class.

Syntax

```
result = object instanceof class
```

Parameters

`result`
Required. Any variable.

`object`
Required. Any object expression.

`class`
Required. Any defined object class.

Remarks

The `instanceof` operator returns `true` if `object` is an instance of `class`. It returns `true` if `class` is present in the object's prototype chain. It returns `false` if `object` is not an instance of `class`, or if `object` is `null`.

Example

The following example shows how to use the `instanceof` operator.

```
function objTest(obj){
    var i, t, s = "";
    t = new Array();
    t["Date"] = Date;
    t["Object"] = Object;
    t["Array"] = Array;
        for (i in t){
            if (obj instanceof t[i]) {
                s += "obj is an instance of " + i + "<br/>";
            }
            else {
                s += "obj is not an instance of " + i + "<br/>";
        }
    }
    return(s);
}

var obj = new Date();
document.write(objTest(obj));

// Output:
// obj is an instance of Date
// obj is an instance of Object
// obj is not an instance of Array
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

See Also

Operator Precedence
Operator Summary (JavaScript)

Left Shift Assignment Operator (<<=) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Moves the specified number of bits to the left and assigns the result to `result`. The bits vacated by the operation are filled with 0.

Syntax

```
result <<= expression
```

Parameters

`result`
Any variable.

`expression`
The number of bits to move.

Remarks

Using the `<<=` operator is the same as specifying `result = result << expression`

The following example shows how to use the `<<=` operator.

```
// 14 is 00000000000000000000000000001110
var temp = 14;
temp <<= 2;
document.write(temp);
// 56 is 00000000000000000000000000111000
Output: 56
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

See Also

Bitwise Left Shift Operator (<<)
Bitwise Right Shift Operator (>>)
Unsigned Right Shift Operator (>>>)
Operator Precedence
Operator Summary (JavaScript)

Logical AND Operator (&&) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Performs a logical conjunction on two expressions.

Syntax

```
result = expression1 && expression2
```

Parameters

`result`
Any variable.

`expression1`
Any expression.

`expression2`
Any expression.

Remarks

If `expression1` evaluates to `false`, `result` is `expression1`. Otherwise, `result` is `expression2`. Consequently, the operation returns `true` if both operands are true; otherwise, it returns `false`.

JavaScript uses the following rules for converting non-Boolean values to Boolean values:

- All objects are considered to be `true`.
- Strings are considered to be `false` if they are empty.
- `null` and `undefined` are considered to be `false`.
- A Number is `false` if it is zero.

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

See Also

Operator Precedence
Operator Summary (JavaScript)

Logical NOT Operator (!) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Performs logical negation on an expression.

Syntax

```
result = !expression
```

Parameters

result
Any variable.

expression
Any expression.

Remarks

The following table illustrates how *result* is determined.

IF `EXPRESSION` IS	THEN `RESULT` IS
True	False
False	True

All unary operators, such as the **!** operator, evaluate expressions as follows:

- If applied to undefined or `null` expressions, a run-time error is raised.
- Objects are converted to strings.
- Strings are converted to numbers if possible. If not, a run-time error is raised.
- Boolean values are treated as numbers (0 if false, 1 if true).

  The operator is applied to the resulting number.

  For the **!** operator, if *expression* is nonzero, *result* is zero. If *expression* is zero, *result* is 1.

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

See Also

Bitwise NOT Operator (~)
Operator Precedence

Logical OR Operator (||) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Performs a logical disjunction on two expressions.

Syntax

```
result = expression1 || expression2
```

Parameters

result
Any variable.

expression1
Any expression.

expression2
Any expression.

Remarks

If either or both expressions evaluate to **True**, *result* is **True**. The following table illustrates how *result* is determined:

IF `EXPRESSION1` IS	AND `EXPRESSION2` IS	THE `RESULT` IS
True	True	True
True	False	True
False	True	True
False	False	False

JavaScript uses the following rules for converting non-Boolean values to Boolean values:

- All objects are considered true.
- Strings are considered false if and only if they are empty.
- `null` and undefined are considered false.
- Numbers are false if, and only if, they are 0.

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

See Also

Operator Precedence
Operator Summary (JavaScript)

Multiplication Assignment Operator (*=) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Multiplies the value of a variable by the value of an expression and assigns the result to the variable.

Syntax

```
result *= expression
```

Parameters

result
Any variable.

expression
Any expression.

Remarks

Using the ***=** operator is exactly the same as specifying:

```
result = result * expression
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

See Also

Multiplication Operator (*)
Operator Precedence
Operator Summary (JavaScript)

Multiplication Operator (*) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Multiplies the value of two expressions.

Syntax

```
result = number1*number2
```

Parameters

result
Any variable.

`number1`
Any expression.

`number2`
Any expression.

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

See Also

Multiplication Assignment Operator (*=)
Operator Precedence
Operator Summary (JavaScript)

new Operator (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Creates a new object.

Syntax

```
new constructor ([arguments])
```

Parameters

`constructor`
Required. The constructor of the object. The parentheses can be omitted if the constructor takes no arguments.

`arguments`
Optional. Any arguments to be passed to the new object's constructor.

Remarks

The `new` operator performs the following tasks:

- It creates an object with no members.
- It calls the constructor for that object, passing a pointer to the newly created object as the `this` pointer.
- The constructor then initializes the object according to the arguments passed to the constructor.

  These are examples of valid uses of the **new** operator.

```
my_object = new Object;
my_array = new Array();
my_date = new Date("Jan 5 1996");
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

See Also

function Statement

Remainder Assignment Operator (JavaScript)

3/2/2018 • 1 min to read • Edit Online

Divides the value of a variable by the value of an expression, and assigns the remainder to the variable.

Syntax

```
result %= expression
```

Arguments

`result`
Any variable.

`expression`
Any numeric expression.

Remarks

Using the `%=` operator is exactly the same as specifying:

```
result = result % expression
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

See Also

Remainder Operator
Operator Precedence
Operator Summary (JavaScript)

Remainder Operator (JavaScript)

3/2/2018 • 1 min to read • Edit Online

Divides the value of a numeric expression by the value of another numeric expression, and produces the remainder.

Syntax

```
result = expression1 % expression2
```

Arguments

`result`
Any variable.

`expression1`
Any numeric expression.

`expression2`
Any numeric expression.

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

See Also

Remainder Assignment Operator
Operator Precedence
Operator Summary (JavaScript)

Right Shift Assignment Operator (>>=) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Right shifts the value of a variable by the number of bits specified in the value of an expression, maintaining the sign, and assigns the result to the variable.

Syntax

```
result >>= expression
```

Parameters

result
Any variable.

expression
Any expression.

Remarks

Using the **>>=** operator is exactly the same as specifying:

```
result = result >> expression
```

The **>>=** operator shifts the bits of *result* right by the number of bits specified in *expression*. The sign bit of *result* is used to fill the digits from the left. Digits shifted off the right are discarded. For example, after the following code is evaluated, *temp* has a value of -4: 14 (11110010 in binary) shifted right two bits equals -4 (11111100 in binary).

```
var temp
temp = -14
temp >>= 2
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

See Also

Bitwise Left Shift Operator (<<)
Bitwise Right Shift Operator (>>)
Unsigned Right Shift Operator (>>>)
Operator Precedence
Operator Summary (JavaScript)

Spread Operator (...) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Allows parts of an array literal to be initialized from an iterable expression (such as another array literal), or allows an expression to be expanded to multiple arguments (in function calls).

Syntax

```
var array = [[arg0ToN ,] ...iterable [, arg0ToN]]
func([args ,] ...iterable [, args | ...iterable])
```

Parameters

`iterable`
Required. An iterable object.

`arg0ToN`
Optional. One or more elements of an array literal.

`args`
Optional. One or more arguments to a function.

Remarks

For more information on iterators, see Iterators and Generators. For more information on using the spread operator as a rest parameter, see Functions.

Example

In this following code example, the use of the spread operator is contrasted with the use of the `concat` method.

```
var a, b, c, d, e;
a = [1,2,3];
b = "dog";
c = [42, "cat"];

// Using the concat method.
d = a.concat(b, c);

// Using the spread operator.
e = [...a, b, ...c];

console.log(d);
console.log(e);

// Output:
// 1, 2, 3, "dog", 42, "cat"
// 1, 2, 3, "dog", 42, "cat"
```

Example

The following code example shows how to use the spread operator in a function call. In this example, two array

literals are passed to the function using the spread operator, and the arrays are expanded to multiple arguments.

```
function f(a, b, c, x, y, z) {
  return a + b + c + x + y + z;
}

var args = [1, 2, 3];
console.log(f(...args, 4, ...[5, 6]));

// Output:
// 21
```

Example

With spread operators, you can simplify code that previously required the use of `apply`.

```
function f(x, y, z) {
    return x + y + z;
}

var args = [1, 2, 3];

// Old method
f.apply(this, args);
// New method
f(...args);
```

Requirements

Supported in Microsoft Edge (Edge browser). Also supported in Store apps (Microsoft Edge on Windows 10). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Not supported in Windows 8.1.

See Also

Operator Precedence
Operator Summary (JavaScript)

Subtraction Assignment Operator (-=) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Subtracts the value of an expression from the value of a variable and assigns the result to the variable.

Syntax

```
result -= expression
```

Parameters

result
Any numeric variable.

expression
Any numeric expression.

Remarks

Using the `-=` operator is exactly the same as doing the following:

```
result = result - expression
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

See Also

Subtraction Operator (-)
Operator Precedence
Operator Summary (JavaScript)

Subtraction Operator (-) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Subtracts the value of one expression from another or provides unary negation of a single expression.

Syntax

```
result = number1 - number2;
```

Parameters

result
Any numeric variable.

`number`
Any numeric expression.

`number1`
Any numeric expression.

`number2`
Any numeric expression.

Remarks

In Syntax 1, the **-** operator is the arithmetic subtraction operator used to find the difference between two numbers. In Syntax 2, the **-** operator is used as the unary negation operator to indicate the negative value of an expression.

For Syntax 2, as for all unary operators, expressions are evaluated as follows:

- If applied to undefined or `null` expressions, a run-time error is raised.
- Objects are converted to strings.
- Strings are converted to numbers if possible. If not, a run-time error is raised.
- Boolean values are treated as numbers (0 if false, 1 if true).

  The operator is applied to the resulting number. In Syntax 2, if the resulting number is nonzero, *result* is equal to the resulting number with its sign reversed. If the resulting number is zero, *result* is zero.

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

See Also

Subtraction Assignment Operator (-=)
Operator Precedence
Operator Summary (JavaScript)

typeof Operator (JavaScript)

3/15/2018 • 1 min to read • Edit Online

Returns a string that identifies the data type of an expression.

Syntax

```
typeof[(]expression[)] ;
```

Remarks

The *expression* argument is any expression for which type information is sought.

The `typeof` operator returns type information as a string. There are seven possible values that `typeof` returns: "number," "string," "boolean," "object," "function," "undefined," and "unknown".

The parentheses are optional in the `typeof` syntax.

An object might return as an unknown type in an XMLHTTPRequest. A COM object with no analog in JavaScript may also return as an unknown type.

Example

The following example tests the data type of variables.

```
var index = 5;
var result = (typeof index === 'number');
// Output: true

var description = "abc";
var result = (typeof description === 'string');
// Output: true
```

Example

The following example tests for a data type of `undefined` for declared and undeclared variables.

```
var declared;
var result = (declared === undefined);
// Output: true

var result = (typeof declared === 'undefined');
// Output: true

var result = (typeof notDeclared === 'undefined')
// Output: true

var obj = {};
var result = (typeof obj.propNotDeclared === 'undefined');
// Output: true

// An undeclared variable cannot be used in a comparison without
// the typeof operator, so the next line generates an error.
//  var result = (notDeclared === undefined);
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

See Also

Array.isArray Function
Object.getPrototypeOf Function
undefined Constant
Comparison Operators
Data Types
Operator Precedence
Operator Summary (JavaScript)

Unsigned Right Shift Assignment Operator (>>>=) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Right shifts the value of a variable by the number of bits specified in the value of an expression, without maintaining sign, and assigns the result to the variable.

Syntax

```
result >>>= expression
```

Parameters

result
Any variable.

expression
Any expression.

Remarks

Using the >>>= operator is exactly the same as doing the following:

```
result = result >>> expression
```

The **>>>=** operator shifts the bits of *result* right by the number of bits specified in *expression*. Zeroes are filled in from the left. Digits shifted off the right are discarded. For example:

```
var temp
temp = -14
temp >>>= 2
```

The variable *temp* has an initial value of -14 (11111111 11111111 11111111 11110010 in two's complement binary). When shifted right two bits, the value equals 1073741820 (00111111 11111111 11111111 11111100 in binary).

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

See Also

Unsigned Right Shift Operator (>>>)
Bitwise Left Shift Operator (<<)
Bitwise Right Shift Operator (>>)

Operator Precedence
Operator Summary (JavaScript)

Unsigned Right Shift Operator (>>>) (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Right shifts the bits of an expression, without maintaining sign.

Syntax

```
result = expression1 >>> expression2
```

Parameters

result
Any variable.

expression1
Any expression.

expression2
Any expression.

Remarks

The **>>>** operator shifts the bits of *expression1* right by the number of bits specified in *expression2*. Zeroes are filled in from the left. Digits shifted off the right are discarded. For example:

```
var temp
temp = -14 >>> 2
```

The variable *temp* has an initial value -14 (11111111 11111111 11111111 11110010 in two's complement binary). When it is shifted right two bits, the value equals 1073741820 (00111111 11111111 11111111 11111100 in binary).

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

See Also

Unsigned Right Shift Assignment Operator (>>>=)
Bitwise Left Shift Operator (<<)
Bitwise Right Shift Operator (>>)
Operator Precedence
Operator Summary (JavaScript)

void Operator (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Prevents an expression from returning a value.

Syntax

```
void expression
```

Remarks

The `expression` argument is any valid JavaScript expression.

The `void` operator evaluates its expression and returns `undefined`. It is useful in situations where an expression should be evaluated but you do not want the results visible to the rest of the script.

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

See Also

Operator Precedence
Operator Summary (JavaScript)

JavaScript Statements

10/18/2017 • 1 min to read • Edit Online

The following table lists JavaScript statements.

Statements

DESCRIPTION	LANGUAGE ELEMENT
Terminates the current loop, or if in conjunction with a *label*, terminates the associated statement.	break Statement
Contains statements to execute when an error occurs in code within the **try** block.	catch Statement
Activates conditional compilation support.	@cc_on Statement
Declares a new class.	class Statement
Causes comments to be ignored by the JavaScript parser.	Comment Statements
Declares a block-scoped variable with a constant value.	const
Stops the current iteration of a loop, and starts a new iteration.	continue Statement
Starts the debugger.	debugger Statement
Executes a statement block once, and then repeats execution of the loop until a condition expression evaluates to **false**.	do...while Statement
Executes a block of statements for as long as a specified condition is **true**.	for Statement
Executes one or more statements for each element of an object or array.	for...in Statement
Executes one or more statements for each value of an object or each element of an array.	for...of Statement
Declares a new function.	function Statement
Conditionally executes a group of statements, depending on the value of an expression.	@if Statement
Conditionally executes a group of statements, depending on the value of an expression.	if...else Statement
Provides an identifier for a statement.	Labeled Statement

DESCRIPTION	LANGUAGE ELEMENT
Declares a block-scoped variable.	let
Exits from the current function and returns a value from that function.	return Statement
Creates variables used with conditional compilation statements.	@set Statement
Enables the execution of one or more statements when a specified expression's value matches a label.	switch Statement
Refers to the current object.	this Statement
Generates an error condition that can be handled by a **try...catch** statement.	throw Statement
Implements error handling for JavaScript.	try Statement
Declares a variable.	var Statement
Executes a statement until a specified condition is **false**.	while Statement
Establishes the default object for a statement.	with Statement

break Statement (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Terminates the current loop, or if in conjunction with a `label`, terminates the associated statement.

Syntax

```
break [label];
```

Remarks

The optional `label` argument specifies the label of the statement you are breaking from.

You typically use the `break` statement in `switch` statements and in `while`, `for`, `for...in`, or `do...while` loops. You most commonly use the `label` argument in `switch` statements, but it can be used in any statement, whether simple or compound.

Executing the `break` statement exits from the current loop or statement, and begins script execution with the statement immediately following.

Examples

In this example, the counter is set up to count from 1 to 99; however, the `break` statement terminates the loop after 14 counts.

```
for (var i = 1; i < 100; i++) {
    if (i == 15) {
        break;
    }
    document.write (i);
    document.write (" ");
}

// Output: 1234567891011121314
```

In the following code, the `break` statement refers to the `for` loop that is preceded by the `Inner:` statement. When `j` is equal to 24, the `break` statement causes the program flow to exit that loop. The numbers 21 through 23 print on each line.

```
Outer:
for (var i = 1; i <= 10; i++) {
    document.write ("<br />");
    document.write ("i: " + i);
    document.write (" j: ");
Inner:
    for (var j = 21; j <= 30; j++) {
        if (j == 24) {
            break Inner;
        }
        document.write (j + " ");
    }
}

// Output:
// i: 1 j: 21 22 23
// i: 2 j: 21 22 23
// i: 3 j: 21 22 23
// i: 4 j: 21 22 23
// i: 5 j: 21 22 23
// i: 6 j: 21 22 23
// i: 7 j: 21 22 23
// i: 8 j: 21 22 23
// i: 9 j: 21 22 23
// i: 10 j: 21 22 23
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

See Also

continue Statement
do...while Statement
for Statement
for...in Statement
Labeled Statement
while Statement

Statement (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Activates conditional compilation support within comments in a script.

> **WARNING**
> Conditional compilation is not supported in Internet Explorer 11 Standards mode and Windows 8.x Store apps. Conditional compilation is supported in Internet Explorer 10 Standards mode and in all earlier versions.

Syntax

```
@cc_on
```

Remarks

The `@cc_on` statement activates conditional compilation within comments in a script.

It is not common to use conditional compilation variables in scripts written for ASP or ASP.NET pages or command-line programs because the capabilities of the compilers can be determined by using other methods.

When you write a script for a Web page, always put conditional compilation code in comments. This enables hosts that do not support conditional compilation to ignore it.

It is strongly recommended that you use the `@cc_on` statement in a comment, so that browsers that do not support conditional compilation will accept your script as valid syntax:

An `@if` or `@set` statement outside of a comment also activates conditional compilation.

Example

The following example illustrates the use of the `@cc_on` statement.

```
/*@cc_on @*/
/*@
    document.write("JavaScript version: " + @_jscript_version + ".");
    document.write("<br />");
    @if (@_win32)
        document.write("Running on the 32-bit version of Windows.");
    @elif (@_win16)
        document.write("Running on the 16-bit version of Windows.");
    @else
        document.write("Running on a different operating system.");
    @end
@*/
```

Requirements

Supported in all versions of Internet Explorer, but not in Windows 8.x Store apps.

See Also

Conditional Compilation
Conditional Compilation Variables
@if Statement
@set Statement

class Statement (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Declares a new class.

Syntax

```
class classname () [extends object] {    [constructor([arg1 [,... [,argN]]]) {        statements    }]
[[static] method([arg1 [,... [,argN]]]) {        statements    }]}
```

Parameters

`classname`
Required. The name of the class.

`object`
Optional. An object or class from which the new class inherits properties and methods.

`constructor`
Optional. A constructor function that initializes the new class instance.

`arg1...argN`
Optional. An optional, comma-separated list of arguments the function understands.

`statements`
Optional. One or more JavaScript statements.

`static`
Optional. Specifies a static method.

`method`
Optional. One or more JavaScript instance or static methods that can be called on a class instance.

Remarks

A class allows you to create new objects using prototype-based inheritance, constructors, instance methods, and static methods. You can use the `super` object within a class constructor or class method to call the same constructor or method in the parent class or object. Optionally, use the `extends` statement after the class name to specify the class or object from which the new class inherits methods.

Example

```
class Spelunking extends EXPERIENCE.Outdoor {
  constructor(name, location) {
    super(name, location);

    this.minSkill = Spelunking.defaultSkill();
    //...
  }
  update(minSkill) {
    //...
    super.update(minSkill);
  }
  static defaultSkill() {
    return new EXPERIENCE.Level3();
  }
}
```

Example

You can also create computed property names for classes. The following code example creates a computed property name using `set` syntax.

```
var propName = "prop_42";

class Spelunking {
    set [propName](v) {
        this.value = v;
    }
};

var s = new Spelunking();
console.log(s.value);
s.prop_42 = 42;
console.log(s.value);

// Output:
// undefined
// 42
```

Example

The following code example creates a property name for a class dynamically using `get` syntax.

```
var propName = "prop_42";

class Spelunking {
    get [propName]() {
        return 777;
    }
}

var s = new Spelunking();
console.log(s.prop_42);

// Output:
// 777
```

Requirements

Supported in Microsoft Edge (Edge browser) with experimental JavaScript features enabled (about:flags). See

Version Information.

Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Not supported in Windows 8.1.

Comment Statements (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Causes comments to be ignored by the JavaScript parser.

Syntax

```
Single-line Comment:
// comment
```

```
Multiline Comment:
/*
comment
*/
```

```
Comments with conditional compilation:
//@CondStatement

/*@
condStatement
@*/
```

Remarks

The `comment` argument is the text of any comment you want to include in your script. The `condStatement` argument is to be used if conditional compilation is activated. If single-line comments are used, there can be no space between the "//" and "@" characters.

Use comments to keep parts of a script from being read by the JavaScript parser. You can use comments to include explanatory remarks in a program.

If single-line comments are used, the parser ignores any text between the comment marker and the end of the line. If multi-line comments are used, the parser ignores any text between the beginning and end markers.

Comments are used to support conditional compilation while retaining compatibility with browsers that do not support that feature. These browsers treat those forms of comments as single-line or multi-line comments respectively.

Example

The following example illustrates the most common uses of comments.

```
/* This is a multiline comment that
    can span as many lines as necessary.  */
function myfunction(arg1, arg2){
    var r;
    // This is a single line comment.
    r = arg1 + arg2
    return(r);
}
```

Example

The following example shows how to use conditional compilation. This example uses special comment delimiters that are used only if conditional compilation is activated by the `@cc_on` statement. Scripting engines that do not support conditional compilation see only the message that says conditional compilation is not supported.

```
/*@cc_on @*/
/*@if (@_jscript_version >= 4)
    alert("JavaScript version 4 or better");
    @else @*/
    alert("Conditional compilation not supported by this scripting engine.");
/*@end @*/
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

const Statement (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Declares a block-scoped variable with a constant value.

Syntax

```
const constant1 = value1
```

Parameters

`constant1`

The name of the variable being declared.

`value1`

The initial value assigned to the variable.

Remarks

Use the `const` statement to declare a variable with a constant value, the scope of which is restricted to the block in which it is declared. The value of the variable cannot be changed.

A variable declared using `const` must be initialized when it is declared.

Example

The following example illustrates the use of the `const` statement.

```
var c = 10;
{
    const c = 2;
   // At this point, c = 2.
}
// At this point, c = 10.

// Additional ways to declare a variable using const.
const name = "Thomas Jefferson";
const answer = 42, numpages = 10;
const myarray = new Array();
```

Requirements

Supported in the Internet Explorer 11 standards document mode. Also supported in Store apps (Windows 8.1 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards. Not supported in Windows 8.

See Also

let Statement
new Operator

Array Object
Variables

continue Statement (JavaScript)

10/18/2017 • 2 min to read • Edit Online

Stops the current iteration of a loop, and starts a new iteration.

Syntax

```
continue [label];
```

Remarks

The optional `label` argument specifies the statement to which `continue` applies.

You can use the `continue` statement only inside a `while`, `do...while`, **for**, or `for...in` loop. Executing the `continue` statement stops the current iteration of the loop and continues program flow with the beginning of the loop. This has the following effects on the different types of loops:

- `while` and `do...while` loops test their condition, and if true, execute the loop again.
- `for` loops execute their increment expression, and if the test expression is true, execute the loop again.
- `for...in` loops proceed to the next field of the specified variable and execute the loop again.

Examples

In this example, a loop iterates from 1 through 9. The statements between `continue` and the end of the `for` body are skipped because of the use of the `continue` statement together with the expression `(i < 5)`.

```
for (var i = 1; i < 10; i++) {
    if (i < 5) {
        continue;
    }
    document.write (i);
    document.write (" ");
}

// Output: 5 6 7 8 9
```

In the following code, the `continue` statement refers to the `for` loop that is preceded by the `Inner:` label. When `j` is 24, the `continue` statement causes that `for` loop to go to the next iteration. The numbers 21 through 23 and 25 through 30 print on each line.

```
Outer:
for (var i = 1; i <= 10; i++) {
    document.write ("<br />");
    document.write ("i: " + i);
    document.write (" j: ");

Inner:
    for (var j = 21; j <= 30; j++) {
        if (j == 24) {
             continue Inner;
        }
        document.write (j + " ");
    }
}

//Output:
//i: 1 j: 21 22 23 25 26 27 28 29 30
//i: 2 j: 21 22 23 25 26 27 28 29 30
//i: 3 j: 21 22 23 25 26 27 28 29 30
//i: 4 j: 21 22 23 25 26 27 28 29 30
//i: 5 j: 21 22 23 25 26 27 28 29 30
//i: 6 j: 21 22 23 25 26 27 28 29 30
//i: 7 j: 21 22 23 25 26 27 28 29 30
//i: 8 j: 21 22 23 25 26 27 28 29 30
//i: 9 j: 21 22 23 25 26 27 28 29 30
//i: 10 j: 21 22 23 25 26 27 28 29 30
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

See Also

break Statement
do...while Statement
for Statement
for...in Statement
Labeled Statement
while Statement

debugger Statement (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Suspends execution.

Syntax

```
debugger
```

Remarks

You can place `debugger` statements anywhere in procedures to suspend execution. Using the `debugger` statement is similar to setting a breakpoint in the code.

The `debugger` statement suspends execution, but it does not close any files or clear any variables.

> **NOTE**
>
> The `debugger` statement has no effect unless the script is being debugged.

Example

This example uses the `debugger` statement to suspend execution for each iteration through the `for` loop.

> **NOTE**
>
> To run this example, you must have a script debugger installed and the script must run in debug mode.
>
> Internet Explorer 8 includes the JavaScript debugger. If you are using an earlier version of Internet Explorer, see How to: Enable and Start Script Debugging from Internet Explorer.

```
for(i = 1; i<5; i++) {
   // Print i to the Output window.
   Debug.write("loop index is " + i);
   // Wait for user to resume.
   debugger
}
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

See Also

JavaScript Statements
Conditional Compilation

do...while Statement (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Executes a statement block once, and then repeats execution of the loop until a condition expression evaluates to `false`.

Syntax

```
do {
    statement
}
while (expression) ;
```

Parameters

`statement`
Required. The statement to be executed if `expression` is `true`. Can be a compound statement.

`expression`
Required. An expression that can be coerced to Boolean `true` or `false`. If `expression` is `true`, the loop is executed again. If `expression` is `false`, the loop is terminated.

Remarks

Unlike the `while` statement, a `do...while` loop is executed one time before the conditional expression is evaluated.

On any line in a `do...while` block, you can use the `break` statement to cause the program flow to exit the loop, or you can use the `continue` statement to go directly to the `while` expression.

Example

In the following example, the statements in the `do...while` loop continue to execute as long as the variable `i` is less than 10.

```
var i = 0;
do {
    document.write(i + " ");
    i++;
} while (i < 10);

// Output: 0 1 2 3 4 5 6 7 8 9
```

Example

In the following example, the statements inside the loop are executed once even though the condition is not met.

```
var i = 10;
do {
    document.write(i + " ");
    i++;
} while (i < 10);

// Output: 10
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

See Also

break Statement
continue Statement
for Statement
for...in Statement
while Statement
Labeled Statement

for Statement (JavaScript)

10/18/2017 • 2 min to read • Edit Online

Executes a block of statements for as long as a specified condition is true.

Syntax

```
for ([initialization]; [test]; [increment])
   statement
```

Parameters

`initialization`
Optional. An expression. This expression is executed only once, before the loop is executed.

`test`
Optional. A Boolean expression. If `test` is `true`, `statement` is executed. If `test` if `false`, the loop is terminated.

`increment`
Optional. An expression. The increment expression is executed at the end of every pass through the loop.

`statement`
Optional. One or more statements to be executed if `test` is **true**. Can be a compound statement.

Remarks

You usually use a `for` loop when the loop is to be executed a known number of times. A `for` loop is useful for iterating over arrays and for performing sequential processing.

The test of a conditional expression occurs before the execution of the loop, so a `for` statement executes zero or more times.

On any line in a `for` loop statement block, you can use the `break` statement to exit the loop, or you can use the `continue` statement to transfer control to the next iteration of the loop.

Example

In the following example, the `for` statement executes the enclosed statements as follows:

- First, the initial value of the variable `i` is evaluated.
- Then, as long as the value of `i` is less than or equal to 9, the `document.write` statements are executed and `i` is reevaluated.
- When `i` is greater than 9, the condition becomes false and control is transferred outside the loop.

```
// i is set to 0 at the start and is incremented by 1 at the
// end of each iteration.
// The loop terminates when i is not less than or equal to
// 9 before a loop iteration.
for (var i = 0; i <= 9; i++) {
   document.write (i);
   document.write (" ");
}

// Output: 0 1 2 3 4 5 6 7 8 9
```

Example

All of the expressions of the `for` statement are optional. In the following example, the `for` statements create an infinite loop, and a `break` statement is used to exit the loop.

```
var j = 0;
for (;;) {
    if (j >= 5) {
        break;
    }
    j++;
    document.write (j + " ");
}

// Output: 1 2 3 4 5
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

See Also

for...in Statement
while Statement

for...in Statement (JavaScript)

10/18/2017 • 2 min to read • Edit Online

Executes one or more statements for each property of an object, or each element of an array.

Syntax

```
for (variable in [object | array]) {
    statements
}
```

Parameters

`variable`
Required. A variable that can be any property name of `object` or any element index of an `array`.

`object`, `array`
Optional. An object or array over which to iterate.

`statements`
Optional. One or more statements to be executed for each property of `object` or each element of `array`. Can be a compound statement.

Remarks

At the beginning of each iteration of a loop, the value of `variable` is the next property name of `object` or the next element index of `array`. You can then use `variable` in any of the statements inside the loop to reference the property of `object` or the element of `array`.

The properties of an object are not assigned in a determinate manner. You cannot specify a particular property by its index, only by the name of the property.

Iterating through an array is performed in element order, that is, 0, 1, 2.

Example

The following example illustrates the use of the `for...in` statement with an object used as an associative array.

```
// Initialize object.
a = {"a" : "Athens" , "b" : "Belgrade", "c" : "Cairo"}

// Iterate over the properties.
var s = ""
for (var key in a) {
    s += key + ": " + a[key];
    s += "<br />";
    }
document.write (s);

// Output:
// a: Athens
// b: Belgrade
// c: Cairo
```

Example

This example illustrates the use of the `for ... in` statement to iterate though an `Array` object that has expando properties.

```
// Initialize the array.
var arr = new Array("zero","one","two");

// Add a few expando properties to the array.
arr["orange"] = "fruit";
arr["carrot"] = "vegetable";

// Iterate over the properties and elements.
var s = "";
for (var key in arr) {
    s += key + ": " + arr[key];
    s += "<br />";
}

document.write (s);

// Output:
//   0: zero
//   1: one
//   2: two
//   orange: fruit
//   carrot: vegetable
```

NOTE

Use the `Enumerator` object to iterate over the members of a collection.

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

See Also

for Statement

for...of Statement (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Executes one or more statements for each value of an iterator obtained from an iterable object.

Syntax

```
for (variable of object) {
    statements
}
```

Parameters

`variable`
Required. A variable that can be any property value of `object`.

`object`
Required. An iterable object such as an `Array`, `Map`, `Set`, or an object that implements the iterator interfaces.

`statements`
Optional. One or more statements to be executed for each value of an `object`. Can be a compound statement.

Remarks

At the beginning of each iteration of a loop, the value of `variable` is the next property value of `object`.

Example

The following example illustrates the use of the `for...of` statement on an array.

```
let arr = [ "fred", "tom", "bob" ];

for (let i of arr) {
    console.log(i);
}

// Output:
// fred
// tom
// bob
```

Example

The following example illustrates the use of the `for...of` statement on a `Map` object.

```
var m = new Map();
m.set(1, "black");
m.set(2, "red");

for (var n of m) {
  console.log(n);
}

// Output:
// 1,black
// 2,red
```

Requirements

Supported in Microsoft Edge (Edge browser). Also supported in Store apps (Microsoft Edge on Windows 10). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Not supported in Windows 8.1.

See Also

for...in Statement
for Statement
while Statement

function Statement (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Declares a new function.

Syntax

```
function functionname ([arg1 [, arg2 [,...[, argN]]]]) {
    statements
}
```

Parameters

`functionname`
Required. The name of the function.

`arg1...argN`
Optional. An optional, comma-separated list of arguments the function understands.

`statements`
Optional. One or moreJavaScript statements.

Remarks

Use the `function` statement to declare a function for later use. The code that is contained in `statements` is not executed until the function is called from elsewhere in the script.

The return statement is used to return a value from the function. You do not have to use a `return` statement; the program will return when it reaches the end of the function. If no `return` statement is executed in the function, or if the `return` statement has no expression, the function returns the value `undefined`.

> **NOTE**
> When you call a function, be sure to include the parentheses and any required arguments. Calling a function without parentheses returns a reference to the function, not the results of the function.

Example

The following example illustrates the use of the `function` statement.

```
function myfunction (arg1, arg2) {
    var r = arg1 * arg2;
    return(r);
}
```

Example

A function can be assigned to a variable. This is illustrated in the following example.

```
function AddFive(x) {
    return x + 5;
}

function AddTen(x) {
    return x + 10;
}

var condition = false;

var MyFunc;
if (condition) {
    MyFunc = AddFive;
}
else {
    MyFunc = AddTen;
}

var result = MyFunc(123);
// Output: 133
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

See Also

new Operator

Statement (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Conditionally executes a group of statements, depending on the value of an expression.

> **WARNING**
> Conditional compilation is not supported in Internet Explorer 11 Standards mode and Windows 8.x Store apps. Conditional compilation is supported in Internet Explorer 10 Standards mode and in all earlier versions.

Syntax

```
      @if (
   condition1
)
   text1
[@elif (
   condition2
)
   text2]
[@else
   text3]
@end
```

Parameters

condition1, condition2
Optional. An expression that can be coerced into a Boolean expression.

`text1`
Optional. Text to be parsed if `condition1` is **true**.

`text2`
Optional. Text to be parsed if `condition1` is **false** and `condition2` is **true**.

`text3`
Optional. Text to be parsed if both `condition1` and `condition2` are **false**.

Remarks

When you write an `@if` statement, you do not have to place each clause on a separate line. You can use multiple **@elif** clauses. However, all **@elif** clauses must come before an **@else** clause.

The `@if` statement is typically used to determine which text among several options should be used for text output.

It is not common to use conditional compilation variables in scripts written for ASP or ASP.NET pages or command-line programs. This is because the capabilities of the compilers can be determined by using other methods.

When you write a script for a Web page, always add conditional compilation code in comments. This enables hosts that do not support conditional compilation to ignore it.

Example

The following example illustrates the use of the **@if...@elif...@else...@end** statement.

```
/*@cc_on @*/
/*@
    document.write("JavaScript version: " + @_jscript_version + ".");
    document.write("<br />");
    @if (@_win32)
        document.write("Running on a 32-bit version of Windows.");
    @elif (@_win16)
        document.write("Running on a 16-bit version of Windows.");
    @else
        document.write("Running on a different operating system.");
    @end
@*/
```

Requirements

Supported in all versions of Internet Explorer, but not in Windows 8.x Store apps.

See Also

Conditional Compilation
Conditional Compilation Variables
@cc_on Statement
@set Statement

if...else Statement (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Conditionally executes a group of statements, depending on the value of an expression.

Syntax

```
if (condition1) {
    statement1
}
[else if (condition2) {
    statement2
}]
[else {
    statement3]
}]
```

Parameters

`condition1`
Required. A Boolean expression. If `condition1` is `null` or `undefined`, `condition1` is treated as `false`.

`statement1`
Optional. The statement to be executed if `condition1` is **true**. Can be a compound statement.

`condition2`
The condition to be evaluated.

`statement2`
Optional. The statement to be executed if `condition2` is `true`. Can be a compound statement.

`statement3`
If both `condition1` and `condition2` are `false`, this statement is executed.

Example

The following code shows how to use `if`, `if else`, and `else`.

It is good practice to enclose `statement1` and `statement2` in braces ({}) for clarity and to avoid inadvertent errors.

```
var z = 3;
if (x == 5) {
    z = 10;
}
else if (x == 10) {
    z = 15;
}
else {
    z = 20;
}
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

See Also

Conditional (Ternary) Operator (?:)

Labeled Statement (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Provides an identifier for a statement.

Syntax

```
   label :
statements
```

Parameters

label
Required. A unique identifier used when referring to the labeled statement.

`statements`
Optional. One or more statements associated with *label*.

Remarks

Labels are used by the **break** and **continue** statements to specify the statement to which the **break** and **continue** apply.

Example

In the following code, the **continue** statement refers to the **for** loop that is preceded by the `Inner:` statement. When `j` is 24, the **continue** statement causes that **for** loop to go to the next iteration. The numbers 21 through 23 and 25 through 30 print on each line.

```
Outer:
for (i = 1; i <= 10; i++) {
   document.write ("<br />");
   document.write ("i: " + i);
   document.write (" j: ");

Inner:
   for (j = 21; j <= 30; j++) {
      if (j == 24)
          {
          continue Inner;
      }
      document.write (j + " ");
  }
}
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

See Also

break Statement
continue Statement

let Statement (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Declares a block-scoped variable.

Syntax

```
let variable1 = value1
```

Parameters

`variable1`
The name of the variable being declared.

`value1`
The initial value assigned to the variable.

Remarks

Use the `let` statement to declare a variable, the scope of which is restricted to the block in which it is declared. You can assign values to the variables when you declare them or later in your script.

A variable declared using `let` cannot be used before its declaration or an error will result.

If you do not initialize your variable in the `let` statement, it is automatically assigned the JavaScript value `undefined`.

Example

The following example illustrates the use of the `let` statement.

```
var  l = 10;
{
    let l = 2;
   // At this point, l = 2.
}
// At this point, l = 10.

// Additional ways to declare a variable using let.
let index;
let name = "Thomas Jefferson";
let answer = 42, counter, numpages = 10;
let myarray = new Array();
```

Requirements

Supported in the Internet Explorer 11 standards document mode. Also supported in Store apps (Windows 8.1 and Windows Phone 8.1). See Version Information.
Not supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards. Not supported in Windows 8.

See Also

const Statement
new Operator
Array Object
Variables

return Statement (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Exits from the current function and returns a value from that function.

Syntax

```
return[(][expression][)];
```

Remarks

The optional *expression* argument is the value to be returned from the function. If omitted, the function does not return a value.

You use the `return` statement to stop execution of a function and return the value of *expression*. If *expression* is omitted, or no `return` statement is executed from within the function, the expression that called the current function is assigned the value undefined.

Example

The following example illustrates the use of the `return` statement.

```
function myfunction(arg1, arg2){
   var r;
   r = arg1 * arg2;
   return(r);
}
```

Example

The following example illustrates the use of the `return` statement to return a function.

```
function doWork() {
    return function calculate(y) { return y + 1; };
}

var func = doWork();
var x = func(5);
document.write(x);

// Output: 6
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

See Also

function Statement

Statement (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Creates variables used with conditional compilation statements.

WARNING

Conditional compilation is not supported in Internet Explorer 11 Standards mode and Windows 8.x Store apps. Conditional compilation is supported in Internet Explorer 10 Standards mode and in all earlier versions.

Syntax

```
@set @varname = term
```

Parameters

varname

Required. Valid JavaScript variable name. Must be preceded by an "@" character at all times.

`term`

Required. Zero or more unary operators followed by a constant, conditional compilation variable, or parenthesized expression.

Remarks

Numeric and Boolean variables are supported for conditional compilation. Strings are not. Variables created using `@set` are generally used in conditional compilation statements, but can be used anywhere in JavaScript code.

Examples of variable declarations look like this:

```
      @set @myvar1 = 12

@set @myvar2 = (@myvar1 * 20)

@set @myvar3 = @_jscript_version
```

The following operators are supported in parenthesized expressions:

- `! ~`
- `* / %`
- `+ -`
- `<< >> >>>`
- `< <= > >=`
- `== != === !==`

- `& ^ |`
- `&& | |`

  If a variable is used before it has been defined, its value is `NaN`. `NaN` can be checked for using the `@if` statement:

```
@if (@newVar != @newVar)
   ...
```

This works because `NaN` is the only value not equal to itself.

Requirements

Supported in all versions of Internet Explorer, but not in Windows 8.x Store apps.

See Also

Conditional Compilation
Conditional Compilation Variables
@cc_on Statement
@if Statement

switch Statement (JavaScript)

10/18/2017 • 2 min to read • Edit Online

Enables the execution of one or more statements when a specified expression's value matches a label.

Syntax

```
switch (expression) {
    case label :
        statementlist
    case label :
    default :
        statementlist
}
```

Parameters

`expression`
The expression to be evaluated.

`label`
An identifier to be matched against `expression`. If `label` is an `expression`, execution starts with the `statementlist` immediately after the colon, and continues until it encounters either a `break` statement, which is optional, or the end of the `switch` statement.

`statementlist`
One or more statements to be executed.

Remarks

Use the `default` clause to provide a statement to be executed if none of the label values matches `expression`. It can appear anywhere within the `switch` code block.

Zero or more `label` blocks may be specified. If no `label` matches the value of `expression`, and a `default` case is not supplied, no statements are executed.

Execution flows through a `switch` statement as follows:

- Evaluate `expression` and look at `label` in order until a match is found.
- If a `label` value equals `expression`, execute its accompanying `statementlist`.

  Continue execution until a `break` statement is encountered, or the `switch` statement ends. This means that multiple `label` blocks are executed if a `break` statement is not used.
- If no `label` equals `expression`, go to the `default` case. If there is no `default` case, go to last step.
- Continue execution at the statement following the end of the `switch` code block.

Example

The following example tests an object for its type.

```
function MyObjectType(obj) {
    switch (obj.constructor) {
        case Date:
            document.write("Object is a Date.");
            break;
        case Number:
            document.write("Object is a Number.");
            break;
        case String:
            document.write("Object is a String.");
            break;
        default:
            document.write("Object is unknown.");
    }
}

// Output when obj is a Date:
// Object is a Date.

// Output when obj is a Number:
// Object is a Number.

// Output when obj is a String:
// Object is a String.

// Output when obj is something other than a Date, Number, or String:
// Object is unknown.
```

Example

The following code shows what happens if you do not use a `break` statement.

```
function MyObjectType(obj) {
    switch (obj.constructor) {
        case Date:
            document.write("Object is a Date.");
        case Number:
            document.write("Object is a Number.");
        case String:
            document.write("Object is a String.");
        default:
            document.write("Object is unknown.");
    }
}

// Output when obj is a Date:
// Object is a Date.Object is a Number.Object is a String.Object is unknown.

// Output when obj is a Number:
// Object is a Number.Object is a String.Object is unknown.

// Output when obj is a String:
// Object is a String.Object is unknown.

// Output when obj is something other than a Date, Number, or String:
// Object is unknown.
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

See Also

break Statement
if...else Statement

this Statement (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Refers to the current object.

Syntax

```
this.property
```

Remarks

The required `property` argument is one of the current object's properties

The `this` keyword can be used in object constructors to refer to the current object.

Example

In the following example, **this** refers to the newly created Car object, and assigns values to three properties:

```
function Car(color, make, model){
   this.color = color;
   this.make = make;
   this.model = model;
}
```

The **this** keyword generally refers to the **window** object if used outside of the scope of any other object. However, inside event handlers `this` refers to the DOM element that raised the event.

In the following code (for Internet Explorer 9 and later), the event handler prints the string version of a button that has an ID of "clicker".

```
document.getElementById("clicker").addEventListener("click", eventHandler, false);

        function eventHandler(ev) {
            document.write(this.toString());
        }

// Output (when you click the button): [object HTMLButtonElement]
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

See Also

new Operator
Using the bind method

throw Statement (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Generates an error condition that can be handled by a `try...catch...finally` statement.

Syntax

```
throw exception
```

Remarks

The required `exception` argument can be any expression.

The following example throws an error inside a `try` block, and it is caught in the `catch` block.

```
try {
        throw new Error(200, "x equals zero");
}
catch (e) {
    document.write(e.message);
}

// Output: x equals zero.
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

See Also

try...catch...finally Statement
Error Object

try...catch...finally Statement (JavaScript)

10/18/2017 • 2 min to read • Edit Online

Sets up blocks of code in which errors that are thrown in one block are handled in another. Errors that are thrown inside the `try` block are caught in the `catch` block. JavaScript.

Syntax

```
try {
    tryStatements
}
catch(exception){
    catchStatements
}
finally {
    finallyStatements
}
```

Parameters

`tryStatements`
Required. Statements where an error can occur.

`exception`
Required. Any variable name. The initial value of `exception` is the value of the thrown error.

`catchStatements`
Optional. Statements to handle errors occurring in the associated `tryStatements`.

`finallyStatements`
Optional. Statements that are unconditionally executed after all other error processing has occurred.

Remarks

The `try...catch...finally` statement provides a way to handle some or all of the errors that may occur in a given block of code, while still running code. If errors occur that are not handled, JavaScript provides the normal error message.

The `try` block contains code that may provoke an error, while the `catch` block contains the code that handles some or all errors. If an error occurs in the `try` block, program control is passed to the `catch` block. The value of `exception` is the value of the error that occurred in the `try` block. If no error occurs, the code in the `catch` block is never executed.

You can pass the error up to the next level by using the `throw` statement to re-throw the error.

After all the statements in the `try` block have been executed and error handling has been done in the `catch` block, the statements in the `finally` block are executed, whether or not an error was handled. The code in the `finally` block is guaranteed to run unless an unhandled error occurs (for example, a run-time error inside the **catch** block).

Example

The following example causes a `ReferenceError` exception to be thrown and displays the name of the error and its message.

```
try {
    addalert("bad call");
}
catch(e) {
    document.write ("Error Message: " + e.message);
    document.write ("<br />");
    document.write ("Error Code: ");
    document.write (e.number & 0xFFFF);
    document.write ("<br />");
    document.write ("Error Name: " + e.name);
}

// Output:
Error Message: 'addalert' is undefined
Error Code: 5009
Error Name: ReferenceError
```

Example

The following example shows how to re-throw errors, as well as the execution of nested `try...catch` blocks. When the error is thrown from the nested `try` block, it passes to the nested `catch` block, which re-throws it. The nested `finally` block runs before the outer `catch` block handles the error, and at the end the outer `finally` block runs.

```
try {
    document.write("Outer try running...<br/>");

    try {
        document.write("Nested try running...<br/>");
        throw new Error(301, "an error");
    }
    catch (e) {
        document.write ("Nested catch caught " + e.message + "<br/>");
        throw e;
    }
    finally {
        document.write ("Nested finally is running...<br/>");
    }
}
catch (e) {
    document.write ("Outer catch caught " + e.message + "<br/>");
}
finally {
    document.write ("Outer finally running");
}

// Output:
// Outer try running...
// Nested try running...
// Nested catch caught error from nested try
// Nested finally is running...
// Outer catch caught error from nested try
// Outer finally running
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet

Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

> **NOTE**
>
> Starting with Internet Explorer 8 Standards mode, the **catch** block is no longer required for `finally` to run.

See Also

throw Statement
Script Junkie configuration wizard sample app

var Statement (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Declares a variable.

Syntax

```
var variable1 = value1
```

Parameters

`variable1`
The name of the variable being declared.

`value1`
The initial value assigned to the variable.

Remarks

Use the `var` statement to declare variables. You can assign values to the variables when you declare them or later in your script.

A variable is declared the first time it appears in your script.

You can declare a variable without using the `var` keyword and assign a value to it. This is known as an *implicit declaration*, and it is not recommended. An implicit declaration gives the variable global scope. When you declare a variable at the procedure level, though, you typically do not want it to have global scope. To avoid giving the variable global scope, you must use the `var` keyword in your variable declaration.

If you do not initialize your variable in the `var` statement, it is automatically assigned the JavaScript value `undefined`.

Example

The following examples illustrate the use of the `var` statement.

```
var index;
var name = "Thomas Jefferson";
var answer = 42, counter, numpages = 10;
var myarray = new Array();
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

See Also

function Statement
new Operator
Array Object
Variables

while Statement (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Executes a statement or series of statements until a specified condition is `false`.

Syntax

```
while (expression) {
    statements
}
```

Parameters

`expression`
Required. A Boolean expression that is checked before each iteration of the loop. If `expression` is `true`, the loop is executed. If `expression` is `false`, the loop is terminated.

`statements`
Optional. One or more statements to be executed if `expression` is `true`.

Remarks

The `while` statement checks `expression` before a loop is first executed. If `expression` is `false` at this time, the loop is never executed.

Example

The following example illustrates the use of the `while` statement.

```
var i = 0;
var j = 10;
while (i < 100) {
    if (i == j)
        break;
    i++;
}
document.write(i);

// Output: 10
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11 standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

See Also

break Statement

continue Statement
do...while Statement
for Statement
for...in Statement

with Statement (JavaScript)

10/18/2017 • 1 min to read • Edit Online

Establishes the default object for a statement.

Syntax

```
with (object) {
    statements
}
```

Parameters

`object`
The default object.

`statements`
One or more statements for which `object` is the default object.

Remarks

The `with` statement is commonly used to shorten the amount of code that you have to write in certain situations.

> **WARNING**
> The use of `with` is not allowed in strict mode. The use of `with` can make code harder to read and to debug and should generally be avoided.

Example

In this example, the `Math` object is used repeatedly:

```
x = Math.cos(3 * Math.PI) + Math.sin(Math.LN10)
y = Math.tan(14 * Math.E)
```

Example

If you rewrite the example to use the `with` statement, your code becomes more succinct:

```
with (Math){
   x = cos(3 * PI) + sin (LN10)
   y = tan(14 * E)
}
```

Requirements

Supported in the following document modes: Quirks, Internet Explorer 6 standards, Internet Explorer 7 standards, Internet Explorer 8 standards, Internet Explorer 9 standards, Internet Explorer 10 standards, Internet Explorer 11

standards. Also supported in Store apps (Windows 8 and Windows Phone 8.1). See Version Information.

See Also

this Statement

JavaScript Directives

10/18/2017 • 1 min to read • Edit Online

The following table lists JavaScript directives.

Directives

DESCRIPTION	LANGUAGE ELEMENT
Adds certain syntactic and semantic restrictions. Supported in Internet Explorer 10 and Windows 8.x Store apps.	use strict

use strict Directive

10/18/2017 • 1 min to read • Edit Online

Restricts the use of some features in JavaScript. Supported in Internet Explorer 10 and Windows 8.x Store apps only.

Syntax

```
use strict
```

Remarks

Example

The following code causes a syntax error because in strict mode all variables must be declared with `var`.

```
"use strict";
function testFunction(){
   var testvar = 4;
    return testvar;
}
intvar = 5;
```

See Also

Strict Mode

JavaScript Errors

10/18/2017 • 1 min to read • Edit Online

The following table lists the types of JavaScript errors.

Error Types

FOR MORE INFORMATION ABOUT	SEE
List of JavaScript run-time errors	Run-time Errors
List of JavaScript syntax errors	Syntax Errors

JavaScript Run-time Errors

10/18/2017 • 1 min to read • Edit Online

JavaScript run-time errors are errors that occur when your script attempts to perform an action that the system cannot execute. You may see run-time errors when variable expressions are being evaluated or memory is being allocated.

Windows Runtime Errors

If you are using Windows Runtime APIs in your Windows 8.x Store app, you may see JavaScript errors that have been converted from Windows Runtime HRESULTs. Windows Runtime HRESULTs in the range over 0x80070000 are converted to JavaScript errors by taking the hexadecimal value of the low bits and converting it to a decimal. For example, the HRESULT 0x80070032 is converted to the decimal value 50, and the JavaScript error is SCRIPT50. The HRESULT 0x80074005 is converted to the decimal value 16389, and the JavaScript error is SCRIPT16389.

Errors

ERROR NUMBER	DESCRIPTION
5	Access is denied
438	Object doesn't support this property or method
1001	Out of memory
5029	Array length must be a finite positive integer
5030	Array length must be assigned a finite positive number
5028	Array or arguments object expected
5010	Boolean expected
5003	Cannot assign to a function result
5000	Cannot assign to 'this'
5034	Circular reference in value argument not supported
5006	Date object expected
5015	Enumerator object expected
5022	Exception thrown and not caught
5020	Expected ')' in regular expression
5019	Expected ']' in regular expression

ERROR NUMBER	DESCRIPTION
5023	Function does not have a valid prototype object
5002	Function expected
5008	Illegal assignment
5021	Invalid range in character set
5035	Invalid replacer argument
5014	JavaScript object expected
5001	Number expected
5007	Object expected
5012	Object member expected
5016	Regular Expression object expected
5005	String expected
5017	Syntax error in regular expression
5026	The number of fractional digits is out of range
5027	The precision is out of range
5025	The URI to be decoded is not a valid encoding
5024	The URI to be encoded contains an invalid character
5009	Undefined identifier
5018	Unexpected quantifier
5013	VBArray expected

See Also

JavaScript Syntax Errors

JavaScript Syntax Errors

10/18/2017 • 1 min to read • Edit Online

JavaScript syntax errors occur when the structure of one of your JavaScript statements violates one or more of the syntactic rules.

Errors

ERROR NUMBER	DESCRIPTION
1019	Can't have 'break' outside of loop
1020	Can't have 'continue' outside of loop
1030	Conditional compilation is turned off
1027	'default' can only appear once in a 'switch' statement
1005	Expected '('
1006	Expected ')'
1012	Expected '/'
1003	Expected ':'
1004	Expected ';'
1032	Expected '@'
1029	Expected '@end'
1007	Expected ']'
1008	Expected '{'
1009	Expected '}'
1011	Expected '='
1033	Expected 'catch'
1031	Expected constant
1023	Expected hexadecimal digit
1010	Expected identifier
1028	Expected identifier, string or number

ERROR NUMBER	DESCRIPTION
1024	Expected 'while'
1014	Invalid character
1026	Label not found
1025	Label redefined
1018	'return' statement outside of function
1002	Syntax error
1035	Throw must be followed by an expression on the same source line
1016	Unterminated comment
1015	Unterminated string constant

Script Host Errors

The following errors are properly speaking errors pertaining to the script host, but you may see them occasionally.

ERROR	HRESULT	DESCRIPTION
SCRIPT_E_RECORDED	0x86664004	An error has been recorded to be passed between script engine and host. The host needs to pass the error code to the caller.
SCRIPT_E_REPORTED	0x80020101	Script engine has reported an unhandled exception to the host via IActiveScriptSite::OnScriptError. Host can ignore this error.
SCRIPT_E_PROPAGATE	0x8002010	A script error is being propagated to the caller which might be in a different thread. The host should pass the error code to the caller.

See Also

JavaScript Run-time Errors

JavaScript Reserved Words

10/18/2017 • 1 min to read • Edit Online

JavaScript has a number of reserved words that you cannot use as identifiers. Reserved words have a specific meaning to the JavaScript language, as they are part of the language syntax. Using a reserved word causes a compilation error when loading your script.

JavaScript also has a list of future reserved words. These words are not currently part of the JavaScript language, although they are reserved for future use. For more information about future reserved keywords in JavaScript, see JavaScript Future Reserved Words.

When choosing identifiers it is also important to avoid any words that are already the names of intrinsic JavaScript objects or functions, such as `String` or `parseInt`.

Reserved Keywords

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

JavaScript Future Reserved Words

10/18/2017 • 1 min to read • Edit Online

In JavaScript, future reserved keywords must not be used as identifiers, even though they have no special meaning in the current version.

For a list of reserved words in JavaScript, see JavaScript Reserved Words.

Future reserved words

The following are future reserved words:

1. class
2. enum
3. export
4. extends
5. import
6. super

Future reserved words in strict mode

The following are future reserved words only in strict mode. For more information about strict mode, see use strict Directive.

1. implements
2. interface
3. package
4. private
5. protected
6. public
7. static
8. yield