[Object.defineProperty及configurable、enumerable、value、writable、get、set 2](#_Toc370204096)

[Summary 2](#_Toc370204097)

[Syntax 2](#_Toc370204098)

[Parameters 2](#_Toc370204099)

[Description 3](#_Toc370204100)

[Creating a property 5](#_Toc370204101)

[Examples 6](#_Toc370204102)

[Modifying a property 7](#_Toc370204103)

[Writable attribute 7](#_Toc370204104)

[Example 7](#_Toc370204105)

[Enumerable attribute 8](#_Toc370204106)

[Example 8](#_Toc370204107)

[Configurable attribute 8](#_Toc370204108)

[Example 9](#_Toc370204109)

[Adding properties and default values 9](#_Toc370204110)

[Cross-browser concerns 10](#_Toc370204111)

[Redefining the length property of an Array object 10](#_Toc370204112)

[Internet Explorer 8 specific case 11](#_Toc370204113)

# Object.defineProperty及configurable、enumerable、value、writable、get、set

## Summary

Defines a new property directly on an object, or modifies an existing property on an object, and returns the object. If you want to see how to use the Object.defineProperty method with a binary-flags-like syntax, see [this article](https://developer.mozilla.org/en-US/docs/JavaScript/Reference/Global_Objects/Object/defineProperty/Additional_examples).

| **Method of**[**Object**](https://developer.mozilla.org/en-US/docs/JavaScript/Reference/Global_Objects/Object) | |
| --- | --- |
| Implemented in | JavaScript 1.8.5 |
| ECMAScript Edition | ECMAScript 5th Edition |

## Syntax

Object.defineProperty(obj, prop, descriptor)

### Parameters

**obj**

The object on which to define the property.

**prop**

The name of the property to be defined or modified.

**descriptor**

The descriptor for the property being defined or modified.

## Description

This method allows precise addition to or modification of a property on an object. Normal property addition through assignment creates properties which show up during property enumeration ([for...in loop](https://developer.mozilla.org/en-US/docs/JavaScript/Reference/Statements/for...in) or [Object.keys method](https://developer.mozilla.org/en-US/docs/JavaScript/Reference/Global_Objects/Object/keys)), whose values may be changed, and which may be [deleted](https://developer.mozilla.org/en-US/docs/JavaScript/Reference/Operators/delete). This method allows these extra details to be changed from their defaults.

Property descriptors present in objects come in two main flavors: data descriptors and accessor descriptors. A data descriptor is a property that has a value, which may or may not be writable. An accessor descriptor is a property described by a getter-setter pair of functions. A descriptor must be one of these two flavors; it cannot be both.

Both data and accessor descriptors are objects. They share the following optional keys:

**configurable**

true if and only if the type of this property descriptor may be changed and if the property may be deleted from the corresponding object. **Defaults to false.**

**enumerable**

true if and only if this property shows up during enumeration of the properties on the corresponding object. **Defaults to false.**

A data descriptor also has the following optional keys:

**value**

The value associated with the property. Can be any valid JavaScript value (number, object, function, etc) **Defaults to undefined.**

**writable**

True if and only if the value associated with the property may be changed with an [assignment operator](https://developer.mozilla.org/en-US/docs/JavaScript/Reference/Operators/Assignment_Operators).**Defaults to false.**

An accessor descriptor also has the following optional keys:

**get**

A function which serves as a getter for the property, or undefined if there is no getter. The function return will be used as the value of property. **Defaults toundefined**.

**set**

A function which serves as a setter for the property, or undefined if there is no setter. The function will receive as only argument the new value being assigned to the property. **Defaults to undefined**.

Bear in mind that these options are not necessarily own properties so, if inherited, will be considered too. In order to ensure these defaults are preserved you might freeze the Object.prototype upfront, specify all options explicitly, or point to null as \_\_proto\_\_ property.

// using \_\_proto\_\_

Object.defineProperty(obj, "key", {

\_\_proto\_\_: null, // no inherited properties

value: "static" // not enumerable

// not configurable

// not writable

// as defaults

});

// being explicit

Object.defineProperty(obj, "key", {

enumerable: false,

configurable: false,

writable: false,

value: "static"

});

// recycling same object

function withValue(value) {

var d = withValue.d || (

withValue.d = {

enumerable: false,

writable: false,

configurable: false,

value: null

}

);

d.value = value;

return d;

}

// ... and ...

Object.defineProperty(obj, "key", withValue("static"));

// if freeze is available, prevents the code to add

// value, get, set, enumerable, writable, configurable

// to the Object prototype

(Object.freeze||Object)(Object.prototype);

## Creating a property

When the property specified doesn't exist in the object, Object.defineProperty() creates a new property as described. Fields may be omitted from the descriptor, and default values for those fields are imputed. All of the Boolean-valued fields default to false. The value, get, and set fields default to undefined. A property which is defined without get/set/value/writable is called "generic" and is "typed" as a data descriptor.

### Examples

var o = {}; // Creates a new object

// Example of an object property added with defineProperty with a data property descriptor

Object.defineProperty(o, "a", {value : 37,

writable : true,

enumerable : true,

configurable : true});

// 'a' property exists in the o object and its value is 37

// Example of an object property added with defineProperty with an accessor property descriptor

var bValue;

Object.defineProperty(o, "b", {get : function(){ return bValue; },

set : function(newValue){ bValue = newValue; },

enumerable : true,

configurable : true});

o.b = 38;

// 'b' property exists in the o object and its value is 38

// The value of o.b is now always identical to bValue, unless o.b is redefined

// You cannot try to mix both :

Object.defineProperty(o, "conflict", { value: 0x9f91102,

get: function() { return 0xdeadbeef; } });

// throws a TypeError: value appears only in data descriptors, get appears only in accessor descriptors

## Modifying a property

When the property already exists, Object.defineProperty() attempts to modify the property according to the values in the descriptor and the object's current configuration. If the old descriptor had its configurable attribute set to false (the property is said to be "non-configurable"), then no attribute besides writable can be changed. In that case, it is also not possible to switch back and forth between the data and accessor property types.

If a property is non-configurable, its writable attribute can only be changed to false.

A [TypeError](https://developer.mozilla.org/en-US/docs/JavaScript/Reference/Global_Objects/TypeError) is thrown when attempts are made to change non-configurable property attributes (besides the writable attribute) unless the current and new values are the same.

## Writable attribute

When the writable property attribute is set to false, the property is said to be "non-writable". It cannot be reassigned.

### Example

var o = {}; // Creates a new object

Object.defineProperty(o, "a", { value : 37,

writable : false });

console.log(o.a); // logs 37

o.a = 25; // No error thrown (it would throw in strict mode, even if the value had been the same)

console.log(o.a); // logs 37. The assignment didn't work.

As seen in the example, trying to write into the non-writable property doesn't change it but doesn't throw an error either.

## Enumerable attribute

The enumerable property attribute defines whether the property shows up in a [for...in](https://developer.mozilla.org/en-US/docs/JavaScript/Reference/Statements/for...in) loop and Object.keys() or not.

### Example

var o = {};

Object.defineProperty(o, "a", { value : 1, enumerable:true });

Object.defineProperty(o, "b", { value : 2, enumerable:false });

Object.defineProperty(o, "c", { value : 3 }); // enumerable defaults to false

o.d = 4; // enumerable defaults to true when creating a property by setting it

for (var i in o) {

console.log(i);

}

// logs 'a' and 'd' (in undefined order)

Object.keys(o); // ["a", "d"]

o.propertyIsEnumerable('a'); // true

o.propertyIsEnumerable('b'); // false

o.propertyIsEnumerable('c'); // false

## Configurable attribute

The configurable attribute controls at the same time whether the property can be deleted from the object and whether its attributes (other than writable) can be changed.

### Example

var o = {};

Object.defineProperty(o, "a", { get : function(){return 1;},

configurable : false } );

Object.defineProperty(o, "a", {configurable : true}); // throws a TypeError

Object.defineProperty(o, "a", {enumerable : true}); // throws a TypeError

Object.defineProperty(o, "a", {set : function(){}}); // throws a TypeError (set was undefined previously)

Object.defineProperty(o, "a", {get : function(){return 1;}}); // throws a TypeError (even though the new get does exactly the same thing)

Object.defineProperty(o, "a", {value : 12}); // throws a TypeError

console.log(o.a); // logs 1

delete o.a; // Nothing happens

console.log(o.a); // logs 1

If the configurable attribute of o.a had been true, none of the errors would be thrown and the property would be deleted at the end.

## Adding properties and default values

It's important to consider the way default values of attributes are applied. There is often a difference between simply using dot notation to assign a value and usingObject.defineProperty(), as shown in the example below.

var o = {};

o.a = 1;

// is equivalent to :

Object.defineProperty(o, "a", {value : 1,

writable : true,

configurable : true,

enumerable : true});

// On the other hand,

Object.defineProperty(o, "a", {value : 1});

// is equivalent to :

Object.defineProperty(o, "a", {value : 1,

writable : false,

configurable : false,

enumerable : false});

## Cross-browser concerns

### Redefining the length property of an Array object

It is possible to redefine the length property of arrays, subject to the usual redefinition restrictions.  (The length property is initially non-configurable, non-enumerable, and writable.  Thus on an unaltered array it is possible to change the length property's value, or to make it non-writable.  It is not allowed to change its enumerability or configurability, or if it is non-writable to change its value or writability.)  However, not all browsers permit this redefinition.

Firefox 4 through 22 will throw a TypeError on any attempt whatsoever (whether permitted or not) to redefine the length property of an array.

Versions of Chrome which implement Object.defineProperty() in some circumstances ignore a length value different from the array's current length property.  In some circumstances changing writability seems to silently not work (and not throw an exception).  Also, relatedly, some array-mutating methods like[Array.prototype.push](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/push) don't respect a non-writable length.

Versions of Safari which implement Object.defineProperty() ignore a length value different from the array's current length property, and attempts to change writability execute without error but do not actually change the property's writability.

Only Internet Explorer 9 and later, and Firefox 23 and later, appear to fully and correctly implement redefinition of the length property of arrays.  For now, don't rely on redefining the length property of an array to either work, or to work in a particular manner.  And even when you can rely on it, [there's really no good reason to do so](http://whereswalden.com/2013/08/05/new-in-firefox-23-the-length-property-of-an-array-can-be-made-non-writable-but-you-shouldnt-do-it/).

### Internet Explorer 8 specific case

Internet Explorer 8 implemented a Object.defineProperty() method that could [only be used on DOM objects](http://msdn.microsoft.com/en-us/library/dd229916%28VS.85%29.aspx). A few things need to be noted:

* Trying to use Object.defineProperty() on native objects throws an error.
* Property attributes must be set to some values. true, true, true for data descriptor and true for configurable, false for enumerable for accessor descriptor.(?) Any attempt to provide other value(?) will result in an error being thrown.
* Reconfiguring a property requires first deleting the property. If the property isn't deleted, it stays as it was before the reconfiguration attempt.