getter

English ▾

The `get` syntax binds an object property to a function that will be called when that property is looked up.

JavaScript Demo: Functions Getter

```
const obj = {
  log: ['a', 'b', 'c'],
  get latest() {
    if (this.log.length == 0) {
      return undefined;
    }
    return this.log[this.log.length - 1];
  }
}

console.log(obj.latest);
// expected output: "c"

```

Run ›

Reset

Syntax

```
{get prop() { ... } }
{get [expression]() { ... } }
```

Parameters

prop
The name of the property to bind to the given function.

expression
Starting with ECMAScript 2015, you can also use expressions for a computed property name to bind to the given function.

Description

Sometimes it is desirable to allow access to a property that returns a dynamically computed value, or you may want to reflect the status of an internal variable without requiring the use of explicit method calls. In JavaScript, this can be accomplished with the use of a *getter*.

It is not possible to simultaneously have a getter bound to a property and have that property actually hold a value, although it *is* possible to use a getter and a setter in conjunction to create a type of pseudo-property.

Note the following when working with the `get` syntax:

- It can have an identifier which is either a number or a string;
- It must have exactly zero parameters (see Incompatible ES5 change: literal getter and setter functions must now have exactly zero or one arguments for more information);

- It must not appear in an object literal with another `get` or with a data entry for the same property (`{ get x() { }, get x() { } }` and `{ x: ..., get x() { } }` are forbidden).

Examples

Defining a getter on new objects in object initializers

This will create a pseudo-property `latest` for object `obj`, which will return the last array item in `log`.

```
const obj = {
  log: ['example','test'],
  get latest() {
    if (this.log.length === 0) return undefined;
    return this.log[this.log.length - 1];
  }
}
console.log(obj.latest); // "test"
```

Note that attempting to assign a value to `latest` will not change it.

Deleting a getter using the `delete` operator

If you want to remove the getter, you can just `delete` it:

```
delete obj.latest;
```

Defining a getter on existing objects using `defineProperty`

To append a getter to an existing object later at any time, use `Object.defineProperty()`.

```
const o = {a: 0};

Object.defineProperty(o, 'b', { get: function() { return this.a + 1; } });

console.log(o.b) // Runs the getter, which yields a + 1 (which is 1)
```

Using a computed property name

```
const expr = 'foo';

const obj = {
  get [expr]() { return 'bar'; }
};

console.log(obj.foo); // "bar"
```

Smart / self-overwriting / lazy getters

Getters give you a way to *define* a property of an object, but they do not *calculate* the property's value until it is accessed. A getter defers the cost of calculating the value until the value is needed. If it is never needed, you never pay the cost.

An additional optimization technique to lazify or delay the calculation of a property value and cache it for later access are **smart (or "memoized") getters**. The value is calculated the first time the getter is called, and is then cached so subsequent accesses return the cached value without recalculating it. This is useful in the following situations:

- If the calculation of a property value is expensive (takes much RAM or CPU time, spawns worker threads, retrieves remote file, etc).
- If the value isn't needed just now. It will be used later, or in some case it's not used at all.
- If it's used, it will be accessed several times, and there is no need to re-calculate that value will never be changed or shouldn't be re-calculated.

> This means that you shouldn't use a lazy getter for a property whose value you expect to change, because the getter will not recalculate the value.

In the following example, the object has a getter as its own property. On getting the property, the property is removed from the object and re-added, but implicitly as a data property this time. Finally, the value gets returned.

```
1 get notifier() {
2   delete this.notifier;
3
```

```
    return this.notifier = document.getElementById('bookmarked-notification-anchor');
  },
```

For Firefox code, see also the `XPCOMUtils.jsm` code module, which defines the `defineLazyGetter()` function.

get vs. defineProperty

While using the `get` keyword and `Object.defineProperty()` have similar results, there is a subtle difference between the two when used on `classes`.

When using `get` the property will be defined on the instance's prototype, while using `Object.defineProperty()` the property will be defined on the instance it is applied to.

```
class Example {
  get hello() {
    return 'world';
  }
}

const obj = new Example();
console.log(obj.hello);
// "world"

console.log(Object.getOwnPropertyDescriptor(obj, 'hello'));
// undefined

console.log(
```

```
  Object.getOwnPropertyDescriptor(
    Object.getPrototypeOf(obj), 'hello'
  )
);
// { configurable: true, enumerable: false, get: function get hello() { return 'world'; }, set: undefi
```

Specifications

Specification
ECMAScript Latest Draft (ECMA-262) The definition of 'Method definitions' in that specification.

Browser compatibility

Update compatibility data on GitHub

get	
Chrome	1
Edge	12
Firefox	2

IE	9
Opera	9.5
Safari	3
WebView Android	1
Chrome Android	18
Firefox Android	4
Opera Android	14
Safari iOS	1
Samsung Internet Android	1.0
nodejs	Yes

Computed property names	
Chrome	46
Edge	12
Firefox	34
IE	No
Opera	47
Safari	No
WebView Android	46
Chrome Android	46
Firefox Android	34

Opera Android	33
Safari iOS	No
Samsung Internet Android	5.0
nodejs	Yes

What are we missing?

Full support

No support

See also

- setter
- `delete`
- `Object.defineProperty()`
- `__defineGetter__`
- `__defineSetter__`
- Defining Getters and Setters in JavaScript Guide

Last modified: Mar 19, 2020, by MDN contributors

Related Topics

JavaScript

Tutorials:

- Complete beginners
- JavaScript Guide
- Intermediate
- Advanced

References:

- Built-in objects
- Expressions & operators