setter

The `set` syntax binds an object property to a function to be called when there is an attempt to set that property.

JavaScript Demo: Functions Setter

```
const language = {
  set current(name) {
    this.log.push(name);
  },
  log: []
}

language.current = 'EN';
language.current = 'FA';

console.log(language.log);
// expected output: Array ["EN", "FA"]

```

Run ›

Reset

Syntax

```
{set prop(val) { . . . }}
{set [expression](val) { . . . }}
```

Parameters

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

`val`
An alias for the variable that holds the value attempted to be assigned to *`prop`*.

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

Description

In JavaScript, a setter can be used to execute a function whenever a specified property is attempted to be changed. Setters are most often used in conjunction with getters to create a type of pseudo-property. It is not possible to simultaneously have a setter on a property that holds an actual value.

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

- It can have an identifier which is either a number or a string;
- It must have exactly one parameter (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 `set` or with a data entry for the same property.
  (`{ set x(v) { }, set x(v) { } }` and `{ x: ..., set x(v) { } }` are forbidden)

Examples

Defining a setter on new objects in object initializers

The following example define a pseudo-property `current` of object `language`. When `current` is assigned a value, it updates `log` with that value:

```
const language = {
  set current(name) {
    this.log.push(name);
  },
  log: []
}

language.current = 'EN';
console.log(language.log); // ['EN']

language.current = 'FA';
console.log(language.log); // ['EN', 'FA']
```

Note that `current` is not defined, and any attempts to access it will result in `undefined`.

Removing a setter with the `delete` operator

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

```
delete language.current;
```

Defining a setter on existing objects using `defineProperty`

To append a setter to an *existing* object, use `Object.defineProperty()`.

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

Object.defineProperty(o, 'b', {
  set: function(x) { this.a = x / 2; }
});

o.b = 10;
//  Runs the setter, which assigns 10 / 2 (5) to the 'a' property

console.log(o.a)
//  5
```

Using a computed property name

```
const expr = 'foo';

const obj = {
  baz: 'bar',
  set [expr](v) { this.baz = v; }
};

console.log(obj.baz);
//  "bar"

obj.foo = 'baz';
//  run the setter

console.log(obj.baz);
//  "baz"
```

Specifications

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

Browser compatibility

Update compatibility data on GitHub

`set`

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

- getter
- `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
- ▸ Statements & declarations
- ▾ Functions
  - Arrow function expressions
  - Default parameters
  - Method definitions
  - Rest parameters
  - The arguments object
  - getter
  - setter
- ▸ Classes

- ▸ Errors
- ▸ Misc