

How are we doing? Please help us improve Stack Overflow. [Take our short survey](#)

get and set in TypeScript

Asked 7 years ago · Active 1 month ago · Viewed 467k times



I'm trying to create get and set method for a property:

546

```
private _name: string;

Name() {
  get:
  {
    return this._name;
  }
  set:
  {
    this._name = ???;
  }
}
```



76

What's the keyword to set a value?

typescript

edited Oct 10 '12 at 20:31

asked Oct 10 '12 at 19:52



MuriloKunze

5,815 13 44 73

10 The underscore and PascalCase conflicts with the Typescript coding guidelines: github.com/Microsoft/TypeScript/wiki/Coding-guidelines – Niels Steenbeek Jan 12 '17 at 8:48

1 Hi @NielsSteenbeek - following the TypeScript contributors guidelines with properties and backing fields you'd end up with a name conflict. What's the suggested approach? – Jordan May 18 '17 at 9:20

Perhaps: `typescript private name: string; getName() { get: { return this.name; } set: { this.name = ???; } }` – Jordan May 18 '17 at

By using our site, you acknowledge that you have read and understand our [Cookie Policy](#), [Privacy Policy](#), and our [Terms of Service](#).

5 @NielsSteenbeek: did you read that document? "This is NOT a prescriptive guideline for the TypeScript community" – [Jonathan Cast](#) Apr 12 '18 at 1:22

8 Answers



TypeScript uses getter/setter syntax that is like ActionScript3.

907



```
class foo {  
    private _bar: boolean = false;  
    get bar(): boolean {  
        return this._bar;  
    }  
    set bar(value: boolean) {  
        this._bar = value;  
    }  
}
```

That will produce this JavaScript, using the ECMAScript 5 Object.defineProperty() feature.

```
var foo = (function () {  
    function foo() {  
        this._bar = false;  
    }  
    Object.defineProperty(foo.prototype, "bar", {  
        get: function () {  
            return this._bar;  
        },  
        set: function (value) {  
            this._bar = value;  
        },  
        enumerable: true,  
        configurable: true  
    });  
    return foo;  
})();
```

So to use it,

By using our site, you acknowledge that you have read and understand our [Cookie Policy](#), [Privacy Policy](#), and our [Terms of Service](#).

```
myFoo.bar = false; // calls the setter and passes false
}
```

However, in order to use it at all, you must make sure the TypeScript compiler targets ECMAScript5. If you are running the command line compiler, use `--target` flag like this;

`tsc --target ES5`

If you are using Visual Studio, you must edit your project file to add the flag to the configuration for the TypeScriptCompile build tool. You can see that [here](#):

As @DanFromGermany suggests below, if your are simply reading and writing a local property like `foo.bar = true`, then having a setter and getter pair is overkill. You can always add them later if you need to do something, like logging, whenever the property is read or written.

edited Sep 3 at 13:12



wonea

2,171 15 65 125

answered Oct 12 '12 at 0:19



Ezward

10.4k 3 17 29

-
- 52 Nice answer. Also, note that, unlike in C#, properties are not currently virtualized in TypeScript (v0.9.5). When you implement "get bar()" in a derived class, you are replacing "get bar()" in the parent. Implications include not being able to call the base class accessor from the derived accessor. This is only true for properties - methods behave as you might expect. See answer by SteveFenton here: stackoverflow.com/questions/13121431/... – David Cuccia Jan 17 '14 at 19:29
-
- 13 I'm slightly confused about the underscore. Typescript convention says not to use underscores for private variables? But in this case, we have to use underscores - or we'll get a conflict between the private and public "bar" – Kokodoko Apr 7 '16 at 10:43
-
- 3 Use the the underscore is a personal preference for private properties. However, I believe you are right in that we want the property to have a different name than the getter/setter methods. – Ezward Apr 7 '16 at 16:24
-
- 3 Why do you use `myFoo.bar = true` instead of `myFoo.bar(true);` or `myFoo.setBar(true);` ?? – Daniel W. Feb 21 '17 at 10:30
-
- 5 @DanFromGermany A property is "syntactic sugar" for a pair of "get" and "set" methods. Microsoft originated the concept of a property with Visual Basic and carried it over to .NET languages such as C# and VB.NET. For example, see [Properties \(C# Programming Guide\)](#). Properties simplify accessing the state of an object and (in my opinion) eliminate the "noisiness" of having to deal with "get/set" method pairs. (Or sometimes only "get" methods where immutability is desired.) – DavidRR May 14 '17 at 14:53
-



Ezward has already provided a good answer, but I noticed that one of the comments asks how it is used. For people like me who

By using our site, you acknowledge that you have read and understand our [Cookie Policy](#), [Privacy Policy](#), and our [Terms of Service](#).

<http://www.typescriptlang.org/docs/handbook/classes.html>

In particular, for those not familiar with it, note that you don't incorporate the word 'get' into a call to a getter (and similarly for setters):

```
var myBar = myFoo.getBar(); // wrong
var myBar = myFoo.get('bar'); // wrong
```

You should simply do this:

```
var myBar = myFoo.bar; // correct (get)
myFoo.bar = true; // correct (set) (false is correct too obviously!)
```

given a class like:

```
class foo {
  private _bar:boolean = false;

  get bar():boolean {
    return this._bar;
  }
  set bar(theBar:boolean) {
    this._bar = theBar;
  }
}
```

then the 'bar' getter for the private '_bar' property will be called.

edited Mar 6 '18 at 22:36

answered Jan 15 '16 at 12:53



TornadoAli

1,089 1 11 10

If I was wanting to replace a public class-level var with a property, is it a straight drop-in replacement that I can put in place and not worry about it? In other words, if I regression test one accessor and one setter, can I deem it a success? Or are there cases where it won't work exactly the same as a var and i need to test all 100 places that use this var/prop? – Adam Plocher Nov 19 '18 at 11:51

I was wondering if there was a workaround for for using underscores to distinguish the property name from the getter or setter methods. In one course I was doing they said underscores were not preferred but didn't give an alternative. – cham May 19 at 2:53

By using our site, you acknowledge that you have read and understand our Cookie Policy, Privacy Policy, and our Terms of Service.

Here's a working example that should point you in the right direction:

51

```
class Foo {
  _name;

  get Name() {
    return this._name;
  }

  set Name(val) {
    this._name = val;
  }
}
```

Getters and setters in JavaScript are just normal functions. The setter is a function that takes a parameter whose value is the value being set.

edited Dec 6 '17 at 21:07



sandrozbinden

1,023 1 12 24

answered Oct 10 '12 at 20:07



Brian Terlson

6,092 1 16 18

29 To be clear, there's no need for the property, getter and setter to be `static` . – [Drew Noakes](#) Oct 17 '13 at 12:01

1 the variable references are still static though. `Foo._name` should be replaced with `this._name` – [Johannes](#) Oct 10 '17 at 21:22

You can write this

4

```
class Human {
  private firstName : string;
  private lastName : string;

  constructor (
    public FirstName?:string,
    public LastName?:string) {

  }
}
```

By using our site, you acknowledge that you have read and understand our [Cookie Policy](#), [Privacy Policy](#), and our [Terms of Service](#).

```

    set FirstName(value : string) {
        console.log("Set FirstName : ", value);
        this.firstName = value;
    }

    get LastName() : string {
        console.log("Get LastName : ", this.lastName);
        return this.lastName;
    }
    set LastName(value : string) {
        console.log("Set LastName : ", value);
        this.lastName = value;
    }
}

```

answered Oct 11 '12 at 18:32



k33g_org

368 3 5

1 Why the public in constructor? – [MuriloKunze](#) Oct 11 '12 at 18:33

15 Yes, can't have public in constructor in this code. `public` here defines duplicate members. – [orad](#) Jul 27 '13 at 15:51

1 You can write it but it wont compile – [Justin](#) Jun 3 '16 at 21:43

It is very similar to creating common methods, simply put the keyword reserved `get` or `set` at the beginning.

2

```

class Name{
    private _name: string;

    getMethod(): string{
        return this._name;
    }

    setMethod(value: string){
        this._name = value
    }
}

```

By using our site, you acknowledge that you have read and understand our [Cookie Policy](#), [Privacy Policy](#), and our [Terms of Service](#).

```

    set setMethod1(value: string){
        this._name = value
    }
}

class HelloWorld {

    public static main(){

        let test = new Name();

        test.setMethod('test.getMethod() --- need ( )');
        console.log(test.getMethod());

        test.setMethod1 = 'test.getMethod1 --- no need ( ), and used = for set ';
        console.log(test.getMethod1);
    }
}
HelloWorld.main();

```

In this case you can skip return type in `get getMethod1() {`

```

get getMethod1() {
    return this._name;
}

```

answered Mar 22 '16 at 14:13

Angel Angel

6,378 12 49 82



0



I think I probably get why is it so confusing. In your example, we wanted getters and setters for `_name` . But we achieve that by creating getters and setters for an unrelated class variable `Name` .

Consider this:

```

// ...

```

By using our site, you acknowledge that you have read and understand our [Cookie Policy](#), [Privacy Policy](#), and our [Terms of Service](#).

```

    }
    set yourCarTiresCount(count) {
        alert('You shouldn't change car tire count')
    }
}

```

Above code does following:

1. `get` and `set` create getter and setter for `yourCarTiresCount` (**not for** `tiresCount`).

The getter is :

```

function() {
    return this.tiresCount ;
}

```

and the setter is :

```

function(count) {
    alert('You shouldn't change car tire count');
}

```

Meaning, every time we do `new Car().yourCarTiresCount` , `getter` runs. And for every `new Car().yourCarTiresCount('7')` `setter` runs.

2. **Indirectly** create getter, but not the setter, for private `tireCount` .

edited Nov 25 '17 at 12:13



sohnryang

574 1 12 24

answered Oct 13 '17 at 19:11



dasfdsa

2,085 1 19 37



0



TS offers getters and setters which allow object properties to have more control of how they are accessed (getter) or updated (setter) **outside** of the object. Instead of directly accessing or updating the property a proxy function is called.

Example:

By using our site, you acknowledge that you have read and understand our [Cookie Policy](#), [Privacy Policy](#), and our [Terms of Service](#).


```

    },

    private _name: string;

    get name() {
        return this._name;
    }

    // first checks the length of the name and then updates the name.
    set name(name: string) {
        if (name.length > 10) {
            throw new Error("Name has a max length of 10");
        }

        this._name = name;
    }

    doStuff () {
        this._name = 'foofoooooooooooooo';
    }

}

const person = new Person('Willem');

// doesn't throw error, setter function not called within the object method when
// this._name is changed
person.doStuff();

// throws error because setter is called and name is longer than 10 characters
person.name = 'barbarbarbarbarbar';

```

answered Jul 11 at 14:53



Willem van der Veen

8,144 4 57 57



If you are working with TypeScript modules and are trying to add a getter that is exported, you can do something like this:

-5

```

// datastore.ts
export const myData: string = undefined; // just for typing support

```

By using our site, you acknowledge that you have read and understand our Cookie Policy, Privacy Policy, and our Terms of Service.

```
get.().string => {\n  if (_myData === undefined) {\n    _myData = "my data"; // pretend this took a long time\n  }\n\n  return _myData;\n},\n});
```

Then, in another file you have:

```
import * as datastore from "./dataStore"\nconsole.log(datastore.myData); // "my data"
```

answered Dec 28 '17 at 18:34



[cjbarth](#)

2,047 3 27 47

-
- 4 That's terrible advice. In particular, `this` must be undefined at the top level scope of a module. You could use `exports` instead but you should not do it at all as it is practically guaranteed to cause compatibility problems – [Aluan Haddad](#) Jan 12 '18 at 10:13
-

protected by [Josh Crozier](#) Oct 9 '17 at 20:12

Thank you for your interest in this question. Because it has attracted low-quality or spam answers that had to be removed, posting an answer now requires 10 [reputation](#) on this site (the [association bonus does not count](#)).

Would you like to answer one of these [unanswered questions](#) instead?