# How can I check whether an optional parameter was provided?

60

2

Given a function with optional parameters:

```
function DoSomething(a, b?) {
    /** Implementation */
}
```

How can I determine whether an optional parameter was provided from within the function body? Currently, the best way to do this that I can think of is:

```
typeof b === 'undefined'
```

But this is kind of messy and not straight-forward to read. Since TypeScript provides optional parameter support, I'm hoping it also has an intuitive way to check if a parameter was provided.

As the above example shows, I don't mind whether the optional parameter was explicitly set to `undefined` or not supplied at all.

### Edit

Unfortunately, this question wasn't as clear as it should have been, particularly if it's skim-read. It was meant to be about how to *cleanly* check if an optional parameter is *completely omitted*, as in:

```
DoSomething("some value");
```

I've accepted Evan's answer since his solution ( `b === undefined` ) is cleaner than the one in my question ( `typeof b === 'undefined'` ) while still having the same behaviour.

The other answers are definitely useful, and which answer is correct for you depends on your use case.

typescript

1 Depends on what you want to check, someone could have called your method `foo(1, undefined)`. You could check `arguments.length`. – Evan Trimboli Jan 5 '14 at 23:32

@EvanTrimboli, thanks; I think you're right. I've updated the question to clarify that either possibility is acceptable. – Sam Jan 5 '14 at 23:42

I'm voting to close this question as unclear since it has run its course and ended up having multiple "correct" answers depending on how it's interpreted. – Sam Apr 25 at 2:17

## 5 Answers

3

You can just check the value to see if it's undefined:

```
var fn = function(a, b) {
    console.log(a, b === undefined);
};

fn(1);
fn(2, undefined);
fn(3, null);
```

answered Jan 6 '14 at 0:17

Evan Trimboli
**27.2k** 5 36 55

Yes, that generally seems to work in the same way that my example solution does. However, since this is a general question, I think it's safer to use the `typeof` check instead. – Sam Jan 6 '14 at 3:00

1 I don't really understand what you mean by "general" question. Typically the only time you want to use `typeof` is if the variable isn't defined. eg `typeof somevar === 'undefined'`. – Evan Trimboli Jan 6 '14 at 3:07

By "general question", I mean this question is not really providing any context, so I think it's best that the answer is also appropriate for all contexts. – Sam Jan 6 '14 at 3:18

3 The reason I believe the `typeof` check is safer is `undefined` can be redefined (either intentionally or accidentally) in some browsers. – Sam Jan 6 '14 at 3:20

38

After googling "typescript check for undefined", I saw this question at the top of the results, but the answer given by Evan Trimboli did not solve my problem.

Here is the answer that ultimately solved my problem. The following code is what I settled on. It should work in cases where the value equals `null` or `undefined`:

```
function DoSomething(a, b?) {
    if (b == null) doSomething();
}
```

edited May 23 '17 at 11:47
Community ♦
1 1

answered May 1 '15 at 15:37

Tod Birdsall
9,129 2 29 36

2 I wonder about this. github.com/Microsoft/TypeScript/wiki/… says "don't use null". basarat.gitbooks.io/typescript/content/docs/tips/null.html says "Null is bad". I'd be interested to find out more about the potential pitfalls of using `null` instead of `undefined`. – Sam Finnigan Nov 1 '16 at 11:32

1 It says "NOTE: These are Coding Guidelines for TypeScript Contributors." – throws_exceptions_at_you Feb 14 '17 at 13:24

1 If you want to check if it has been set is it safe to check `if (b != null) doSomethingWith(b);` ? – TheJKFever Aug 4 '17 at 21:45

`null` and `undefined` are different values. In this example `DoSomething('smth', null)` we DO provide optional argument and `doSomething()` will be executed. If optional argument is not passed, then it will be `undefined`. – Vladimir Prudnikov Jan 6 '18 at 17:51

@TheJKFever yes `if (b != null) ...` does work as well. – reads0520 Aug 7 '18 at 19:38

10

TypeScript's `util` module has function `isUndefined()`. You can use it like this.

```
import {isUndefined} from "util";

class A {
    test(b?: string): string {
        if (isUndefined(b)) {
```

answered Jul 31 '17 at 11:41

Vladimir Prudnikov
**4,330** 3 30 46

The `isUndefined` util function has been deprecated since v4. Use `value === undefined` instead. – Aaron Greenlee Feb 2 at 12:19

You are right. The reason they (util.is* functions) were deprecated is because there were some issues, and fixing them will result in a lot of broken code. They decided to deprecate. I just created my own utils module and defined these functions. – Vladimir Prudnikov Feb 3 at 17:27

As a side note: isUndefined is deprecated since v4.0.0 according to index.d.ts – Bas van Dijk Feb 20 at 13:12

7

you could simple add an optional parameter with an default value like this

```
function DoSomething(a, b: boolean=null) {
    if(b == null)
    {
      //parameter was not set
    }
    else
    {
    //parameter is true or false...no other option available
    }
}
```

answered Jul 19 '15 at 7:24

Tobias Koller
**973** 3 14 36

0

From here works without problmes:

```
function getSchool(name: string, address?: string, pinCode?: string): string {
```

```
    //...
}
```

answered Oct 15 '18 at 8:52

Pavel_K

**2,836** 4 24 79