# How to pass optional parameters in TypeScript while omitting some other optional parameters?

203

Given the following signature:

```
export interface INotificationService {
    error(message: string, title?: string, autoHideAfter?: number);
}
```

24

How can I call the function `error()` *not* specifying the `title` parameter, but setting `autoHideAfter` to say `1000` ?

typescript

edited Apr 12 at 5:03

asked Jun 9 '15 at 14:09

g.pickardou

**10k** 18 67 137

## 9 Answers

235

As specified in the documentation, use `undefined` :

```
export interface INotificationService {
    error(message: string, title?: string, autoHideAfter? : number);
}

class X {
    error(message: string, title?: string, autoHideAfter?: number) {
        console.log(message, title, autoHideAfter);
    }
```

edited Oct 4 '16 at 0:21

mrm
**2,692** 19 22

answered Jun 9 '15 at 14:20

Thomas
**118k** 35 258 370

6 @BBi7 I think you misunderstood the documentation. The `?` is added in the function *definition*, but the question is about actually *calling* the function. – Thomas Aug 1 '18 at 7:58

Hi @Thomas I totally understand the documentation and know you must add the ? after the parameter in the function definition. But, related to calling a function with optional parameter(s) I'd assume passing undefined if not applicable. I generally I try to find ways to make optional parameter(s) as the end parameter(s) so I can just not pass vs. undefined. But obviously if you have many then you'd need to pass undefined or anything non-truthy. I'm not sure what I was referring to back when I originally posted. Meaning I don't know if it was edited but what I see now fine correct. – BBi7 Aug 15 '18 at 14:56

1 @BBi7 There were no recent edits. Okay, never mind then :) (Note that you actually must pass `undefined` to get the same behaviour as leaving out the argument entirely. Just "anything non-truthy" will not work, because TypeScript actually compares to `void 0` which is a safer way of writing `undefined`.) – Thomas Aug 16 '18 at 10:04

I completely agree with you last message! – BBi7 Aug 20 '18 at 16:55

1 @prime That's exactly what this very question was asking, and what I answered above. You pass `undefined`. – Thomas Oct 17 '18 at 7:14

56

Unfortunately there is nothing like this in TypeScript (more details here: https://github.com/Microsoft/TypeScript/issues/467)

But to get around this you can change your params to be an interface:

```
export interface IErrorParams {
  message: string;
  title?: string;
  autoHideAfter?: number;
}

export interface INotificationService {
  error(params: IErrorParams);
}

//then to call it:
```

Brocco, thx for the creative idea using optional properties. I think we loose more than we win: now it is *mandatory* to name the "parameters", like message: 'msg'. When we have more mandatory parameters this is a big impact. Also thx for the appropriate link to the issue 467. and well... it is 42. – g.pickardou Jun 9 '15 at 15:06

No problem, what it boils down to is that there's more than 1 way to skin a cat. It is just a matter of choosing the solution that works best for you and your team. – Brocco Jun 9 '15 at 15:18

2 I complicated parameters (jQuery Ajax for example) this is a fantastic solution! – Erik Philips Jun 9 '16 at 16:19

32

you can use optional variable by `?` or if you have multiple optional variable by `...` , example:

```
function details(name: string, country="CA", address?: string, ...hobbies: string) {
    // ...
}
```

In the above:

- `name` is required
- `country` is required and has a default value
- `address` is optional
- `hobbies` is an array of optional params

edited Apr 18 '17 at 12:10
martinjbaker
945 10 19

answered Jan 18 '17 at 10:50

Hasan A Yousef
6,659 4 45 80

3 This should be the selected answer. The optional rest parameter is extremely useful for defining abstract base class methods that are expected to be extended. – Evan Plaice Feb 27 '18 at 19:21

1 Shouldn't hobbies be typed as an array ? – ProgrammerPer Nov 7 '18 at 9:16

There's a useful info in this answer, but it doesn't answer the question. The question is, as I see it, how one can bypass/skip several optional parameters

params object optional).

9

It ends up being kind of like Python's **kwargs, but not exactly.

```
export interface IErrorParams {
  title?: string;
  autoHideAfter?: number;
}

export interface INotificationService {
  // make params optional so you don't have to pass in an empty object
  // in the case that you don't want any extra params
  error(message: string, params?: IErrorParams);
}

// all of these will work as expected
error('A message with some params but not others:', {autoHideAfter: 42});
error('Another message with some params but not others:', {title: 'StackOverflow'});
error('A message with all params:', {title: 'StackOverflow', autoHideAfter: 42});
error('A message with all params, in a different order:', {autoHideAfter: 42, title:
'StackOverflow'});
error('A message with no params at all:');
```

answered Jul 8 '16 at 16:19

Monkpit

**1,548** 15 28

---

Another approach is:

9

```
error(message: string, options?: {title?: string, autoHideAfter?: number});
```

So when you want to omit the title parameter, just send the data like that:

```
error('the message', { autoHideAfter: 1 })
```

906 1 9 20

This is one of the ideal solutions. I have seen this practice in couple of angular base librarie – Dibzmania May 14 at 23:40

5

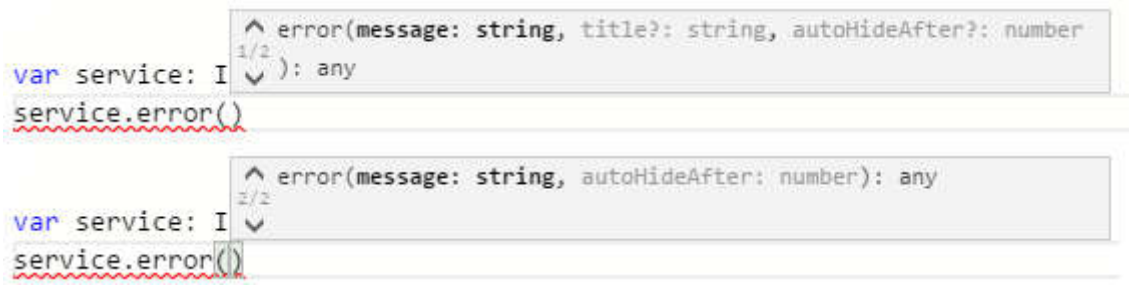
You can specify multiple method signatures on the interface then have multiple method overloads on the class method:

```
interface INotificationService {
    error(message: string, title?: string, autoHideAfter?: number);
    error(message: string, autoHideAfter: number);
}

class MyNotificationService implements INotificationService {
    error(message: string, title?: string, autoHideAfter?: number);
    error(message: string, autoHideAfter?: number);
    error(message: string, param1?: (string|number), param2?: number) {
        var autoHideAfter: number,
            title: string;

        // example of mapping the parameters
        if (param2 != null) {
            autoHideAfter = param2;
            title = <string> param1;
        }
        else if (param1 != null) {
            if (typeof param1 === "string") {
                title = param1;
            }
            else {
                autoHideAfter = param1;
            }
        }

        // use message, autoHideAfter, and title here
    }
}
```

Now all these will work:

```
service.error("My message", "My title");
service.error("My message", "My title", 1000);
```

...and the `error` method of `INotificationService` will have the following options:




Playground

edited Mar 8 '18 at 14:34

answered Jun 9 '15 at 17:40

David Sherret
**57.2k** 18 131 132

7 Just a note that I would recommend against this and instead pass in an object and put these parameters as properties on that object... it's a lot less work and the code will be more readable. – David Sherret Mar 1 '16 at 17:05

1

You can create a helper method that accept a one object parameter base on error arguments

```
error(message: string, title?: string, autoHideAfter?: number){}

getError(args: { message: string, title?: string, autoHideAfter?: number }) {
   return error(args.message, args.title, args.autoHideAfter);
}
```

answered Oct 17 '18 at 12:31

malbarmawi

0

```
class myClass{
  public error(message: string, title?: string, autoHideAfter? : number){
    //....
  }
}
```

use the `?` operator as an optional parameter.

answered Aug 30 '16 at 20:08

Rick
**6,591** 2 34 35

but this doesn't allow you any way to specify `message` and `autoHideAfter` – Simon_Weaver Nov 19 '16 at 2:27

2 you don't answer the question or you didn't read it. He wants to know how to call a method with multiple optional parameters, without having to specify the first optional if he only want to enter the second one. – Gregfr Mar 10 '18 at 5:43

0

you could try to set title to null.

This worked for me.

error('This is the ',null,1000)

answered Apr 6 '18 at 11:35

numpsi
**1** 1

2 This woulndt work since if the function parameter has a default value when you sent null to that function parameter will not set to it's default value – Okan SARICA Aug 1 '18 at 12:09