

TypeScript 2.9 is now available. [Download \(/#download-links\)](#) our latest version today!

Documentation ▼

By Example

Introduction

The purpose of this guide is to teach you how to write a high-quality definition file. This guide is structured by showing documentation for some API, along with sample usage of that API, and explaining how to write the corresponding declaration.

These examples are ordered in approximately increasing order of complexity.

- Global Variables
- Global Functions
- Objects with Properties
- [Overloaded Function](#)
- Reusable Types (Interfaces)
- Reusable Types (Type Aliases)
- Organizing Types
- Classes

The Examples

Global Variables

Documentation

The global variable `foo` contains the number of widgets present.

Code

```
console.log("Half the number of widgets is " + (foo / 2));
```

Declaration

Use `declare var` to declare variables. If the variable is read-only, you can use `declare const`. You can also use `declare let` if the variable is block-scoped.

```
/** The number of widgets present */  
declare var foo: number;
```

Global Functions

Documentation

You can call the function `greet` with a string to show a greeting to the user.

Code

```
greet("hello, world");
```

Declaration

Use `declare function` to declare functions.

```
declare function greet(greeting: string): void;
```

Objects with Properties

Documentation

The global variable `myLib` has a function `makeGreeting` for creating greetings, and a property `numberOfGreetings` indicating the number of greetings made so far.

Code

```
let result = myLib.makeGreeting("hello, world");  
console.log("The computed greeting is:" + result);  
  
let count = myLib.numberOfGreetings;
```

Declaration

Use `declare namespace` to describe types or values accessed by dotted notation.

```
declare namespace myLib {  
    function makeGreeting(s: string): string;  
    let numberOfGreetings: number;  
}
```

Overloaded Functions

Documentation

The `getWidget` function accepts a number and returns a `Widget`, or accepts a string and returns a `Widget` array.

Code

```
let x: Widget = getWidget(43);  
  
let arr: Widget[] = getWidget("all of them");
```

Declaration

```
declare function getWidget(n: number): Widget;  
declare function getWidget(s: string): Widget[];
```

Reusable Types (Interfaces)

Documentation

When specifying a greeting, you must pass a `GreetingSettings` object. This object has the following properties:

- 1 - greeting: Mandatory string
- 2 - duration: Optional length of time (in milliseconds)
- 3 - color: Optional string, e.g. `'#ff00ff'`

Code

```
greet({  
  greeting: "hello world",  
  duration: 4000  
});
```

Declaration

Use an `interface` to define a type with properties.

```
interface GreetingSettings {  
  greeting: string;  
  duration?: number;  
  color?: string;  
}  
  
declare function greet(setting: GreetingSettings): void;
```

Reusable Types (Type Aliases)

Documentation

Anywhere a greeting is expected, you can provide a `string`, a function returning a `string`, or a `Greeter` instance.

Code

```
function getGreeting() {  
  return "howdy";  
}  
class MyGreeter extends Greeter { }  
  
greet("hello");  
greet(getGreeting);  
greet(new MyGreeter());
```

Declaration

You can use a type alias to make a shorthand for a type:

```
type GreetingLike = string | (() => string) | MyGreeter;  
  
declare function greet(g: GreetingLike): void;
```

Organizing Types

Documentation

The `greeter` object can log to a file or display an alert. You can provide `LogOptions` to `.log(...)` and alert options to `.alert(...)`

Code

```
const g = new Greeter("Hello");  
g.log({ verbose: true });  
g.alert({ modal: false, title: "Current Greeting" });
```

Declaration

Use namespaces to organize types.

```
declare namespace GreetingLib {  
    interface LogOptions {  
        verbose?: boolean;  
    }  
    interface AlertOptions {  
        modal: boolean;  
        title?: string;  
        color?: string;  
    }  
}
```

You can also create nested namespaces in one declaration:

```
declare namespace GreetingLib.Options {  
    // Refer to via GreetingLib.Options.Log  
    interface Log {  
        verbose?: boolean;  
    }  
    interface Alert {  
        modal: boolean;  
        title?: string;  
        color?: string;  
    }  
}
```

Classes

Documentation

You can create a greeter by instantiating the `Greeter` object, or create a customized greeter by extending from it.

Code

```
const myGreeter = new Greeter("hello, world");
myGreeter.greeting = "howdy";
myGreeter.showGreeting();

class SpecialGreeter extends Greeter {
  constructor() {
    super("Very special greetings");
  }
}
```


Declaration

Use `declare class` to describe a class or class-like object. Classes can have properties and methods as well as a constructor.

```
declare class Greeter {
  constructor(greeting: string);

  greeting: string;
  showGreeting(): void;
}
```

Made with ❤ in Redmond Follow @Typescriptlang (<https://twitter.com/typescriptlang>)

Privacy (<https://go.microsoft.com/fwlink/?LinkId=521839>) ©2012-2018 Microsoft  Microsoft