

# TypeScript 1.4

16 Jan 2015 10:30 AM

This is a standalone, power tool release of TypeScript 1.4 for Visual Studio 2013. It includes both the TypeScript experience for Visual Studio and a standalone compiler that integrates with the Visual Studio experience that can be used from the command line.

<http://blogs.msdn.com/b/typescript/archive/2015/01/16/announcing-typescript-1-4.aspx>

Today we're happy to announce TypeScript 1.4. With TypeScript 1.4, we've continued to build new features that help you work with more JavaScript patterns, create richer typings, and use new ES6 features.

You can try these improvements out as part of [Visual Studio 2015 CTP5](#), [Visual Studio 2013](#), [NPM](#), and as [source](#).

## Type System Improvements

### *Union Types*

JavaScript functions may take a number of possible argument types. Up to now, we've supported this using function overloads. Starting with TypeScript 1.4, we've generalized this capability and now allow you to specify that that a value is one of a number of different types using a union type:

```
function f(x: number | number[]) {  
    if (typeof x === "number") {  
        return x + 10;  
    }  
    else {  
        // return sum of numbers  
    }  
}
```

Once you have a value of a union type, you can use a `typeof` and `instanceof` checks to use the value in a type-safe way. You'll notice we use this in the above example and can treat `x` as a `number` type inside of the `if`-block.

Union types are a new kind of type and work any place you specify a type.

### *Type Aliases*

You can now define an alias for a type using the `type` keyword:

```
type PrimitiveArray = Array<string|number|boolean>;  
type MyNumber = number;  
type NgScope = ng.IScope;  
type Callback = () => void;
```

Type aliases are exactly the same as their original types; they are simply alternative names. You can use these aliases to better document your code and aid readability.

## ***Const Enums***

For heavy uses of enums, it's helpful to have an even more restricted form that we know is safe to always inline. This helps with performance, code size, and with working with cases where the enum aliases themselves may not be exported. Starting with TypeScript 1.4, you'll be able to make const enums.

```
const enum Color { Blue, Green, Red };  
var c = Color.Blue;
```

When compiled, the const enum is removed, leaving 'var c = 0 /\* Blue \*/'.

## ***And more...***

You can get also more information and examples about updates to the type system on our [TypeScript 1.4 sneak peek blog post](#).

## **ECMAScript 6 Support**

In addition to the type system improvements, one of the main goals for the upcoming TypeScript 2.0 release is to fully support the ECMAScript 6 standard. With TypeScript 1.4, we take another step towards this goal. In this release, we've added a new ES6 output mode, support for let and const, and support for ES6 template strings.

## ***ES6 output mode***

In previous TypeScript releases, we included a '--target' commandline option that allows you to choose between ES3 and ES5 output modes. We've added a new 'ES6' option to the targets commandline option:

```
> tsc --target ES6 myfile.js
```

This option will be required for ES6 features that cannot be output to ES3 or ES5, just as ES5-only features, such as getters and setters, have required the ES5 target option.

We're in the process of updating the existing language features from ES6, such as lambdas and classes, to support ES6 mode by outputting their native forms. You may need to install an [additional .d.ts file](#) for this to work.

## ***Let/Const***

The ES6 'let' feature is similar to 'var', but it aims to simplify the mental model for the variable's scope. With 'let', you can scope variables to blocks of code rather than whole functions. For example:

```
function f() {
```

```

let total = 0;
let x = 5;
for (let x = 1; x < 10; x++) {
    total += x;
}
console.log(x);
}

f(); // outputs 5

```

Notice how the two 'let x' statements do not conflict. This is because the one used in the loop is in a different scope than the one outside of the loop. If we re-wrote this using vars, all 'x' vars effectively combine into one, leading to rather confusing output.

```

function f() {
    var total = 0;
    var x = 5;
    for (var x = 1; x < 10; x++) {
        total += x;
    }
    console.log(x);
}

f(); // outputs 10

```

TypeScript now supports using 'let' and 'const' in addition to 'var'. These currently require the ES6 output mode, but we're investigating relaxing this restriction in future versions.

## Template Strings

ECMAScript 6 has further improved on string interpolation in JavaScript by adding template strings. These special strings can freely mix in expressions, allowing a lighter syntax when pieces of a string depend on associated values:

```

var rectangle = { height: 20, width: 10 };
var areaMessage = `Rectangle area is ${rectangle.height * rectangle.width}`;

```

With the 1.4 release, TypeScript now supports ES6 template strings and can also compile them down to ES3/ES5 expressions.

## Looking Ahead

We're excited to bring new features into TypeScript, including more ECMAScript 6 features and our work on [async/await](#). As always, we'd love to hear your feedback. You can get involved by trying out the release, sending us a [bug](#), playing with the [source](#), and sending us a [pull request](#).

## Angular 2: Built on TypeScript

5 Mar 2015 8:00 AM

<http://blogs.msdn.com/b/typescript/archive/2015/03/05/angular-2-0-built-on-typescript.aspx>

We're excited to unveil the result of a months-long partnership with the Angular team.

This partnership has been very productive and rewarding experience for us, and as part of this collaboration, we're happy to announce that Angular 2 will now be built with TypeScript. We're looking forward to seeing what people will be able to do with these new tools and continuing to work with the Angular team to improve the experience for Angular developers.

The first fruits of this collaboration will be in the upcoming TypeScript 1.5 release.

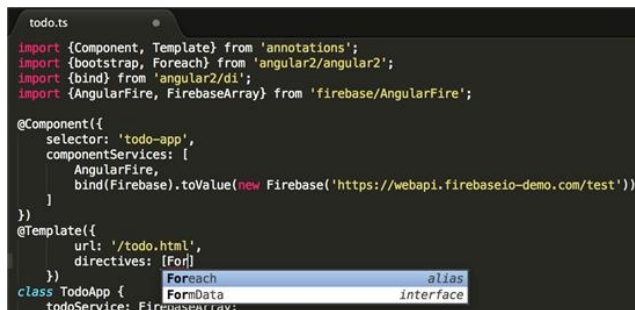
We have worked with the Angular team to design a set of new features that will help you develop cleaner code when working with dynamic libraries like Angular 2, including a new way to annotate class declarations with metadata. Library and application developers can use these metadata annotations to cleanly separate code from information about the code, such as configuration information or conditional compilation checks.

We've also added a way to retrieve type information at runtime. When enabled, this will enable developers to do a simple type introspection. To verify code correctness with additional runtime checks. It also enables libraries like Angular to use type information to set up dependency injection based on the types themselves.

## ***TodoMVC for Angular 2 in TypeScript***

At ng-conf, we are previewing this work by showing a TodoMVC example, based on [David East's Angular 2 TodoMVC](#). You can [try this example](#) out for yourself. If you're new to TypeScript, you can also learn TypeScript through our [interactive playground](#).

We'd love to hear your feedback.



```
todo.ts
import {Component, Template} from 'annotations';
import {bootstrap, Foreach} from 'angular2/angular2';
import {bind} from 'angular2/di';
import {AngularFire, FirebaseArray} from 'firebase/AngularFire';

@Component({
  selector: 'todo-app',
  componentServices: [
    AngularFire,
    bind(Firebase).toValue(new Firebase('https://webapi.firebaseio-demo.com/test'))
  ]
})
@Template({
  url: '/todo.html',
  directives: [Foreach]
})
class TodoApp {
  todoService: FirebaseArray;
}
```

*TypeScript autocomplete in Sublime 3 for Angular 2*

We're looking forward to releasing a beta of TypeScript 1.5 in the coming weeks, and along with it, growing TypeScript's tooling support to include more development styles and environments. We'd also like to give a huge thanks to Brad, Igor, Miško on the Angular team for being great partners. Special shout out to Yehuda Katz, who helped us design the annotation+decorator proposal which helped make this work possible.

## **TypeScript 1.5 Alpha**

27 Mar 2015 3:20 PM

<http://blogs.msdn.com/b/typescript/archive/2015/03/27/announcing-typescript-1-5-alpha.aspx>

Today we're announcing TypeScript 1.5 Alpha, the first preview of the TypeScript 1.5 release. This release shows off many of the features that will be in the final TypeScript 1.5 release. In the alpha release, you'll be able to use three new capabilities of the TypeScript tools: a richer ES6 experience, decorators, and a new [Sublime Text plugin](#).

You can try this alpha out today by installing the new compiler available on [npm](#).

## Richer ES6 experience

In TypeScript 1.5, we're adding a number of new ES6 features. These features work together with the TypeScript type system to give you helpful tooling when working with the new ES6 code patterns.

### Modules

The module syntax of ES6 is a powerful way of working with modules. You can interact with modules by importing whole modules or by working with individual exports directly.

```
import * as Math from "my/math";
import { add, subtract } from "my/math";
```

ES6 also supports a range of functionality in specifying exports. You can export declarations, such as classes or functions. You can also export a 'default' to import the module directly. For example:

```
// math.ts

export function add(x, y) { return x + y }
export function subtract(x, y) { return x - y }
export default function multiply(x, y) { return x * y }

// myFile.ts

import {add, subtract} from "math";
import times from "math";
var result = times(add(2, 3), subtract(5, 3));
```

If you've been using TypeScript, you may notice that this is similar to TypeScript external modules. This is no accident. When we created external modules for TypeScript, we were working on the same problems. The ES6 design takes the capabilities even further, showing a powerful, mature design. We will continue to support external modules, but we will begin encouraging developers to use the more capable ES6 module syntax.

### Destructuring

Destructuring is a handy new feature that comes as part of our ES6 support. With it, you can pull apart, or destructure, objects and arrays.

```
var [x, y] = [10, 20];
[x, y] = [y, x]; // a simple swap
```

You can also use destructuring to handle function parameters:

```
var myClient = {name: "Bob", height: 6};
function greetClient({name, height: howTall}) {
  console.log("Hello, " + name + ", who is " + howTall + " feet tall.");
}
greetClient(myClient);
```

In the above, `greetClient` takes in a single object that has a `name` and `height` property. Using the `'height: howTall'` syntax, we can rename the `height` property to `howTall` inside of `greetClient`.

## And more

We've also added for-of support for better iteration, let/const compiling to ES5, unicode support, an ES6 output mode, and better support for computed properties.

## Decorators

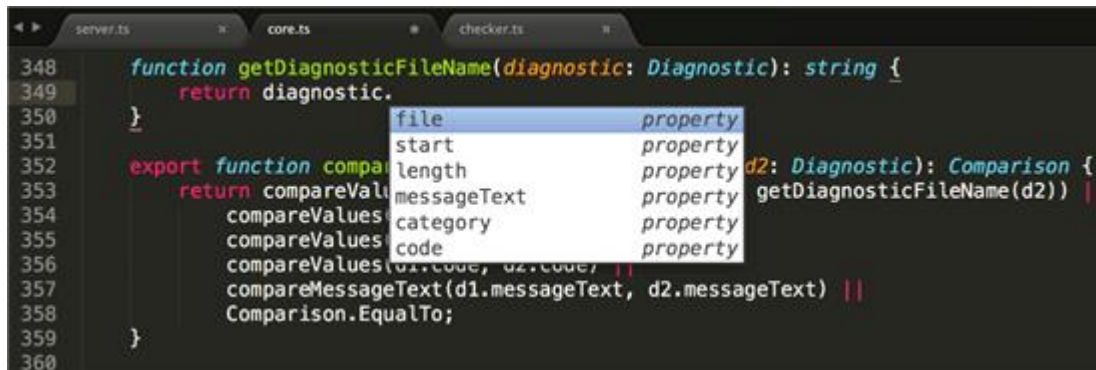
We've also worked with the [Angular](#), [Ember](#), and [Aurelia](#) (from the makers of Durandal) teams on a decorator proposal for ES7, which we're previewing in TypeScript 1.5 Alpha. Decorators allow you to create a clean separation of concerns. In this example, we see how the `@memoize` decorator could be used to note that a getter/setter pair can be memoized:

```
class Person {
  @memoize
  get name() { return `${this.first} ${this.last}` }

  set name(val) {
    let [first, last] = val.split(' ');
    this.first = first;
    this.last = last;
  }
}
```

Developers will be able to write new decorators and mix-and-match them to work with the type system.

## Sublime Text plugin



Along with TypeScript 1.5 Alpha, we're also releasing a [preview of the Sublime Text plugin for TypeScript](#) to enable more developers to get editor support when authoring TypeScript. This plugin works with both Sublime Text 2 and Sublime Text 3 and shows some of what is possible with the TypeScript type system. Sublime Text and the TypeScript plugin are available for OSX, Linux, and Windows.

Type	
TypeScript: Rename	^T, ^M
TypeScript: FindReferences	^T, ^R
TypeScript: FormatDocument	^T, ^F
TypeScript: FormatSelection	^T, ^F
TypeScript: SaveTmp	^T, ^S
TypeScript: GoToDefinition	F12
TypeScript: GoToType	
TypeScript: PasteAndFormat	⌘V
TypeScript: QuickInfoDocumentation	^T, ^Q
Set Syntax: TypeScript	

*TypeScript commands available in Sublime Text*

The Sublime Text plugin allows you to easily navigate, refactor, format, and investigate TypeScript code. Those who tried the Sublime plugin that came with the ng-conf demo may also notice that this updated plugin is snappier, especially with larger files.

We're excited to hear your feedback. If you'd like to leave a comment, you can fill out an [issue on the issue tracker](#). Also, feel free to jump in and send us your [pull requests](#) to help make the Sublime plugin even better.

## ***What's next***

This alpha release shows what will be possible in TypeScript 1.5 when it's released, and we want to hear from you. We're working hard on TypeScript 1.5, and you can help us make it a strong release by trying it out and [sending us any issues](#) you find.