August 1st, 2025 9 reactions

Announcing TypeScript 5.9

Daniel Rosenwasser
Principal Product Manager

Table of contents

Read next

July 25, 2025

Announcing TypeScript 5.9 RC

 Daniel Rosenwasser

July 8, 2025

Announcing TypeScript 5.9 Beta

 Daniel Rosenwasser

Today we are excited to announce the release of TypeScript 5.9!

If you're not familiar with TypeScript, it's a language that builds on JavaScript by adding syntax for types. With types, TypeScript makes it possible to check your code to avoid bugs ahead of time. The TypeScript type-checker does all this, and is also the foundation of great tooling in your editor and elsewhere, making coding even easier. If you've written JavaScript in editors like Visual Studio and VS Code, TypeScript even powers features you might already be using like completions, go-to-definition, and more. You can learn more about TypeScript at our website.

But if you're already familiar, you can start using TypeScript 5.9 today!

Copy

```
npm install -D typescript
```

Let's take a look at what's new in TypeScript 5.9!

- Minimal and Updated `tsc --init`
- Support for `import defer`
- Support for `--module node20`
- Summary Descriptions in DOM APIs
- Expandable Hovers (Preview)
- Configurable Maximum Hover Length
- Optimizations
- Notable Behavioral Changes

What's New Since the Beta and RC?

There have been no changes to TypeScript 5.9 since the release candidate.

A few fixes for reported issues have been made since the 5.9 beta, including the restoration of `AbortSignal.abort()` to the DOM library. Additionally, we have added a section about Notable Behavioral Changes.

Minimal and Updated `tsc --init`

For a while, the TypeScript compiler has supported an `--init` flag that can create a `tsconfig.json` within the current directory. In the last few years, running `tsc --init` created a very "full" `tsconfig.json`, filled with commented-out settings and their descriptions. We designed this with the intent of making options discoverable and easy to toggle.

However, given external feedback (and our own experience), we found it's common to immediately delete most of the contents of these new `tsconfig.json` files. When users want to discover new options, we find they rely on auto-complete from their editor, or navigate to the tsconfig reference on our website (which the generated `tsconfig.json` links to!). What each setting does is also documented on that same page, and can be seen via editor hovers/tooltips/quick info. While surfacing some commented-out settings might be helpful, the generated `tsconfig.json` was often considered overkill.

We also felt that it was time that `tsc --init` initialized with a few more prescriptive settings than we already enable. We looked at some common pain points and papercuts users have when they create a new TypeScript project. For example, most users write in modules (not global scripts), and `--moduleDetection` can force TypeScript to treat every implementation file as a module. Developers also often want to use the latest ECMAScript features directly in their runtime, so `--target` can typically be set to `esnext`. JSX users often find that going back to set `--jsx` is needless friction, and its options are slightly confusing. And often, projects end up loading more declaration files from `node_modules/@types` than TypeScript actually needs; but specifying an empty `types` array can help limit this.

In TypeScript 5.9, a plain `tsc --init` with no other flags will generate the following `tsconfig.json`:

Copy

```
{
  // Visit https://aka.ms/tsconfig to read more about this file
  "compilerOptions": {
    // File Layout
    // "rootDir": "./src",
    // "outDir": "./dist",
```

```
    // Environment Settings
    // See also https://aka.ms/tsconfig_modules
    "module": "nodenext",
    "target": "esnext",
    "types": [],
    // For nodejs:
    // "lib": ["esnext"],
    // "types": ["node"],
    // and npm install -D @types/node

    // Other Outputs
    "sourceMap": true,
    "declaration": true,
    "declarationMap": true,

    // Stricter Typechecking Options
    "noUncheckedIndexedAccess": true,
    "exactOptionalPropertyTypes": true,

    // Style Options
    // "noImplicitReturns": true,
    // "noImplicitOverride": true,
    // "noUnusedLocals": true,
    // "noUnusedParameters": true,
    // "noFallthroughCasesInSwitch": true,
    // "noPropertyAccessFromIndexSignature": true,

    // Recommended Options
    "strict": true,
    "jsx": "react-jsx",
    "verbatimModuleSyntax": true,
    "isolatedModules": true,
    "noUncheckedSideEffectImports": true,
    "moduleDetection": "force",
    "skipLibCheck": true,
  }
}
```

For more details, see the implementing pull request and discussion issue.

Support for `import defer`

TypeScript 5.9 introduces support for ECMAScript's deferred module evaluation proposal using

the new `import defer` syntax. This feature allows you to import a module without immediately executing the module and its dependencies, providing better control over when work and side-effects occur.

The syntax only permits namespace imports:

Copy

```
import defer * as feature from "./some-feature.js";
```

The key benefit of `import defer` is that the module is only evaluated when one of its exports is first accessed. Consider this example:

Copy

```
// ./some-feature.ts
initializationWithSideEffects();

function initializationWithSideEffects() {
  // ...
  specialConstant = 42;

  console.log("Side effects have occurred!");
}

export let specialConstant: number;
```

When using `import defer`, the `initializationWithSideEffects()` function will not be called until you actually access a property of the imported namespace:

Copy

```
import defer * as feature from "./some-feature.js";

// No side effects have occurred yet

// ...

// As soon as `specialConstant` is accessed, the contents of the `fea
// module are run and side effects have taken place.
console.log(feature.specialConstant); // 42
```

Because evaluation of the module is deferred until you access a member off of the module, you cannot use named imports or default imports with `import defer`:

Copy

```
// ❌ Not allowed
import defer { doSomething } from "some-module";

// ❌ Not allowed
import defer defaultExport from "some-module";

// ✅ Only this syntax is supported
import defer * as feature from "some-module";
```

Note that when you write `import defer`, the module and its dependencies are fully loaded and ready for execution. That means that the module will need to exist, and will be loaded from the file system or a network resource. The key difference between a regular `import` and `import defer` is that *the execution of statements and declarations* is deferred until you access a property of the imported namespace.

This feature is particularly useful for conditionally loading modules with expensive or platform-specific initialization. It can also improve startup performance by deferring module evaluation for app features until they are actually needed.

Note that `import defer` is not transformed or "downleveled" at all by TypeScript. It is intended to be used in runtimes that support the feature natively, or by tools such as bundlers that can apply the appropriate transformation. That means that `import defer` will only work under the `--module` modes `preserve` and `esnext`.

We'd like to extend our thanks to Nicolò Ribaudo who championed the proposal in TC39 and also provided the implementation for this feature.

Support for `--module node20`

TypeScript provides several `node*` options for the `--module` and `--moduleResolution`

settings. Most recently, `--module nodenext` has supported the ability to `require()` ECMAScript modules from CommonJS modules, and correctly rejects import assertions (in favor of the standards-bound import attributes).

TypeScript 5.9 brings a stable option for these settings called `node20`, intended to model the behavior of Node.js v20. This option is unlikely to have new behaviors in the future, unlike `--module nodenext` or `--moduleResolution nodenext`. Also unlike `nodenext`, specifying `--module node20` will imply `--target es2023` unless otherwise configured. `--module nodenext`, on the other hand, implies the floating `--target esnext`.

For more information, take a look at the implementation here.

Summary Descriptions in DOM APIs

Previously, many of the DOM APIs in TypeScript only linked to the MDN documentation for the API. These links were useful, but they didn't provide a quick summary of what the API does. Thanks to a few changes from Adam Naji, TypeScript now includes summary descriptions for many DOM APIs based on the MDN documentation. You can see more of these changes here and here.

Expandable Hovers (Preview)

Quick Info (also called "editor tooltips" and "hovers") can be very useful for peeking at variables to see their types, or at type aliases to see what they actually refer to. Still, it's common for people to want to *go deeper* and get details from whatever's displayed within the quick info tooltip. For example, if we hover our mouse over the parameter `options` in the following example:

Copy

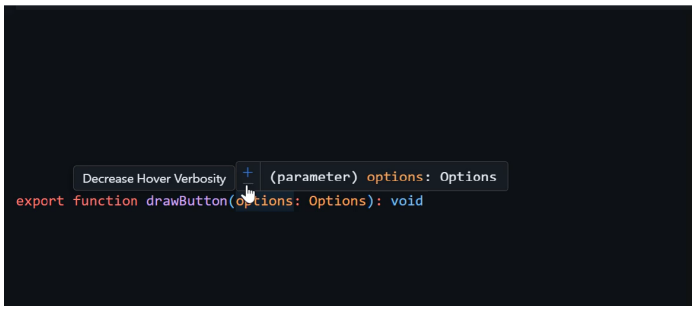
```
export function drawButton(options: Options): void
```

We're left with `(parameter) options: Options`.

```
                                    (parameter) options: Options
export function drawButton(options: Options): void
```

Do we really need to jump to the definition of the type `Options` just to see what members this value has?

Previously, that was actually the case. To help here, TypeScript 5.9 is now previewing a feature called *expandable hovers*, or "quick info verbosity". If you use an editor like VS Code, you'll now see a `+` and `-` button on the left of these hover tooltips. Clicking on the `+` button will expand out types more deeply, while clicking on the `-` button will collapse to the last view.



This feature is currently in preview, and we are seeking feedback for both TypeScript and our partners on Visual Studio Code. For more details, see the PR for this feature here.

Configurable Maximum Hover Length

Occasionally, quick info tooltips can become so long that TypeScript will truncate them to make them more readable. The downside here is that often the most important information will be omitted from the hover tooltip, which can be frustrating. To help with this, TypeScript 5.9's language server supports a configurable hover length, which can be configured in VS Code via the `js/ts.hover.maximumLength` setting.

Additionally, the new default hover length is substantially larger than the previous default. This means that in TypeScript 5.9, you should see more information in your hover tooltips by default. For more details, see the PR for this feature here and the corresponding change to Visual Studio Code here.

Optimizations

Cache Instantiations on Mappers

When TypeScript replaces type parameters with specific type arguments, it can end up instantiating many of the same intermediate types over and over again. In complex libraries like Zod and tRPC, this could lead to both performance issues and errors reported around excessive type instantiation depth. Thanks to a change from Mateusz Burzyński, TypeScript 5.9 is able to cache many intermediate instantiations when work has already begun on a specific type instantiation. This in turn avoids lots of unnecessary work and allocations.

Avoiding Closure Creation in `fileOrDirectoryExistsUsingSource`

In JavaScript, a function expression will typically allocate a new function object, even if the wrapper function is just passing through arguments to another function with no captured variables. In code paths around file existence checks, Vincent Bailly found examples of these pass-through function calls, even though the underlying functions only took single arguments. Given the number of existence checks that could take place in larger projects, he cited a speed-up of around 11%. See more on this change here.

Notable Behavioral Changes

`lib.d.ts` Changes

Types generated for the DOM may have an impact on type-checking your codebase.

Additionally, one notable change is that `ArrayBuffer` has been changed in such a way that it is no longer a supertype of several different `TypedArray` types. This also includes subtypes of `UInt8Array`, such as `Buffer` from Node.js. As a result, you'll see new error messages such as:

Copy

```
error TS2345: Argument of type 'ArrayBufferLike' is not assignable t
```

```
error TS2345: Argument of type 'ArrayBufferLike' is not assignable t
error TS2322: Type 'ArrayBufferLike' is not assignable to type 'Arra
error TS2322: Type 'Buffer' is not assignable to type 'Uint8Array<Ar
error TS2322: Type 'Buffer' is not assignable to type 'ArrayBuffer'.
error TS2345: Argument of type 'Buffer' is not assignable to paramet
```

If you encounter issues with `Buffer`, you may first want to check that you are using the latest version of the `@types/node` package. This might include running

Copy

```
npm update @types/node --save-dev
```

Much of the time, the solution is to specify a more specific underlying buffer type instead of using the default `ArrayBufferLike` (i.e. explicitly writing out `Uint8Array<ArrayBuffer>` rather than a plain `Uint8Array`). In instances where some `TypedArray` (like `Uint8Array`) is passed to a function expecting an `ArrayBuffer` or `SharedArrayBuffer`, you can also try accessing the `buffer` property of that `TypedArray` like in the following example:

Copy

```
  let data = new Uint8Array([0, 1, 2, 3, 4]);
- someFunc(data)
+ someFunc(data.buffer)
```

Type Argument Inference Changes

In an effort to fix "leaks" of type variables during inference, TypeScript 5.9 may introduce changes in types and possibly new errors in some codebases. These are hard to predict, but can often be fixed by adding type arguments to generic functions calls. See more details here.

What's Next?

Now that TypeScript 5.9 is out, you might be wondering what's in store for the next version:

TypeScript 6.0.

As you might have heard, much of our recent focus has been on the native port of TypeScript which will eventually be available as TypeScript 7.0. So what does that mean for TypeScript 6.0?

Our vision for TypeScript 6.0 is to act as a transition point for developers to adjust their codebases for TypeScript 7.0. While TypeScript 6.0 may still ship updates and features, most users should think of it as a readiness check for adopting TypeScript 7.0. This new version is meant to align with TypeScript 7.0, introducing deprecations around certain settings and possibly updating type-checking behavior in small ways. Luckily, we don't predict most projects will have too much trouble upgrading to TypeScript 6.0, and it will likely be entirely API compatible with TypeScript 5.9.

We'll have more details coming soon. That includes details on TypeScript 7.0 as well, where you can try it out in Visual Studio Code today and install it right in your project.

Otherwise, we hope that TypeScript 5.9 treats you well, and makes your day-to-day coding a joy.

Happy Hacking!

– Daniel Rosenwasser and the TypeScript Team


9


2


23

Category

TypeScript

Share

Author

Daniel Rosenwasser
Principal Product Manager

Daniel Rosenwasser is the product manager of the TypeScript team. He has a passion for programming languages, compilers, and great developer tooling.

2 comments

Join the discussion.

Sign in

Sort by : Newest

David Hanson August 2, 2025 • Edited 1

Nice work.

Do you have a proposed deprecation list for 6?

Log in to Vote or Reply

Akunyili Chukwuma August 1, 2025 0

Awesome, will TS 7.0 be released in 2026 or 2027?
The matching numbers would be epic.

Log in to Vote or Reply

Stay informed

Get notified when new posts are published.

Enter your email

Subscribe

Follow this blog