

# A Proposal For Type Syntax in JavaScript



Daniel Rosenwasser

March 9th, 2022 | 54 | 0

Today we're excited to announce our support and collaboration on [a new Stage 0 proposal](#) to bring optional and erasable type syntax to JavaScript. Because this new syntax wouldn't change how surrounding code runs, it would effectively act as *comments*. We think this has the potential to make TypeScript easier and faster to use for development at every scale. We'd like to talk about why we're pursuing this, and how this proposal works at a high level.



## Background

One recent trend our team has seen in the JavaScript world is a demand for faster iteration time and a reduction of build steps. In other words, "make it faster and make it simpler".

In some ways, this is already happening. Thanks to the success of evergreen browsers, developers can often avoid compiling newer versions of JavaScript to run on older runtimes. To some extent the same is also true of bundling – most browsers have built-in support for using modules, so bundling can be viewed as more of an optimization step than a necessity. This has increasingly been the case, so how is TypeScript keeping up?

If we go back to 2012 when TypeScript was first announced, the JavaScript world was drastically different! Some browsers shipped often, but not all. It was unclear how long we'd be stuck with ancient versions of Internet Explorer, and that led to tools like bundlers and compilers gaining adoption. TypeScript was able to really thrive in the age where adding a build step to JavaScript was a given – after all, if you need to compile your JavaScript anyway, why not compile away your types too? But if those trends we mentioned above continue, compiling away your types might be the only step between writing your TypeScript and running it, and we don't want to be the ones standing in the way of a good developer experience!

In some ways, our JavaScript support bridges the gap here, and maybe you've seen this if you use an editor like Visual Studio or Visual Studio Code. Today, you can create a `.js` file in your editor and start sprinkling in types in the form of JSDoc comments.

```
/**  
 * @param a {number}  
 * @param b {number}  
 */  
function add(a, b) {  
  return a + b;  
}
```

Because these are just comments, they don't change how your code runs at all – they're just a form of documentation, but TypeScript uses them to give you a better JavaScript editing experience through things like code completions, refactorings, and more. You can even [add type-checking](#) by adding a `// @ts-check` comment to the top of your file, or running those files through the TypeScript compiler with `checkJs`.



This feature makes it incredibly convenient to get some of the TypeScript experience without a build step, and you can use it for small scripts, basic web pages, server code in Node.js, etc.

Still, you'll notice that this is a little verbose – we love how lightweight the inner-loop is for writing JavaScript, but we're missing how convenient TypeScript makes it to just write types.

*So what if we had both?*

What if we could have something like TypeScript syntax which was totally ignored – sort of like comments – in JavaScript.

```
function add(a: number, b: number) {  
  return a + b;  
}
```

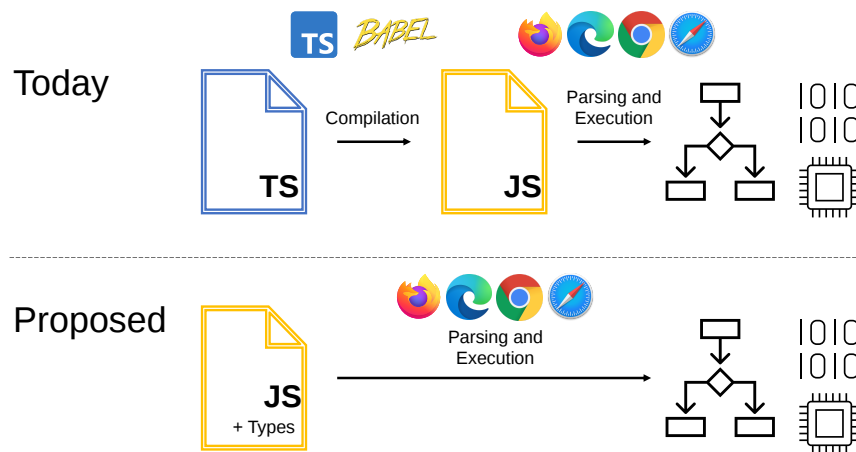
Our team believes there is *a lot* of potential here, and this month we're hoping to [bring it forward in a proposal](#) to TC39, the ECMAScript standards committee!

## How Would This Work?

When we've been asked "when are types coming to JavaScript?", we've had to hesitate to answer. Historically, the problem was that if you asked developers what they had in mind for types in JavaScript, you'd get many different answers. Some felt that types should be totally ignored, while others felt like they should have *some* meaning – possibly that they should enforce some sort of runtime validation, or that they should be introspectable, or that they should act as hints to the engine for optimization, and more! But in the last few years we've seen people converge more towards a design that works well with the direction TypeScript has moved towards – that types are totally ignored and erasable syntax at runtime. This convergence, alongside the broad use of TypeScript, made us feel more confident when several JavaScript and TypeScript developers outside of our core team approached us once more about a proposal called "types as comments".

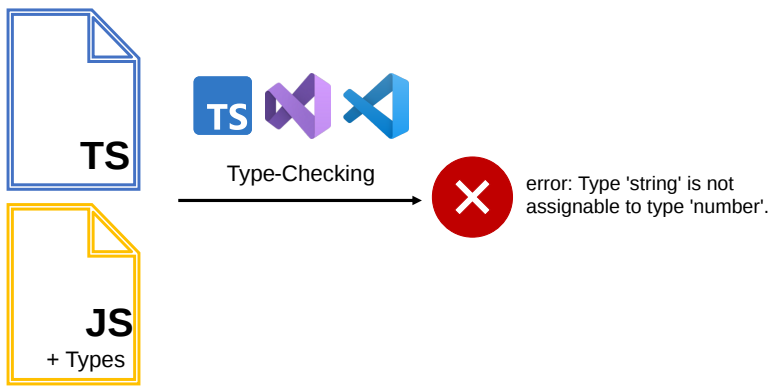
[The idea of this proposal](#) is that JavaScript could carve out a set of syntax for types that engines would *entirely ignore*, but which tools like TypeScript, Flow, and others could use. This allows us to keep the things you love about TypeScript – its type-checking and editing experience – while removing the need for a build step in development.

So when it comes to writing and *running* code, a developer's inner-loop would look a little different.



Meanwhile, writing code and *type-checking* would stay the same. A developer could get instant type-checking feedback in an editor with TypeScript support, run TypeScript on the command line, and add TypeScript as part of their CI tasks. The biggest difference is that because we would not need a build step, we would *dramatically* lower the barrier to entry for JavaScript devs to experience the power of types and great tooling.





To make this happen, JavaScript would minimally need to add syntax for things like type annotations on variables and functions, optionality modifiers (?) for parameters and class members, type declarations (**interfaces** and **type** aliases), and type assertion operators (**as** and **!**) – all of which would have no effect on how the surrounding code is run.

Things like visibility modifiers (e.g. **public**, **private**, and **protected**) might be in scope as well; however, enums, namespaces, and parameter properties would be out of scope for this proposal since they have observable runtime behavior. Those features could be proposed as separate ECMAScript features based on feedback, but our current goal is to support some large subset of TypeScript that we think could be a valuable addition to JavaScript.

With this carve out, we've left room for type-checkers to innovate in ways that require new syntax. That does mean that engines would happily run code with nonsensical types, but we believe type-checkers could (and should) be prescriptive and enforce stricter constraints than runtimes. Combined, this makes for a type syntax that could be customized across different checkers, or removable entirely if someone decides they're not happy with TypeScript or any other type-checker.

## What is this not?

It's worth mentioning what this proposal *isn't*.

Our team isn't proposing putting TypeScript's type-checking in every browser and JavaScript runtime – nor are we proposing any new type-checker to be put in the browser. We think doing that would cause problems for JavaScript and TypeScript users alike due to a range of issues, such as runtime performance, compatibility issues with existing TypeScript code, and the risk of halting innovation in the type-checking space.

Instead, we're just proposing syntax that is compatible with and motivated by TypeScript, which could be used by any type-checker, but which would be skipped over by JavaScript engines. We believe that this approach is the most promising for everyone, and would continue to allow TypeScript, Flow, and others to continue to innovate.

## What's next?

Given all this, we plan to present [this proposal](#) for [Stage 1](#) at the upcoming March 2022 plenary meeting of TC39. We'll be doing so with the support and guidance from our co-champions of this proposal, [Rob Palmer](#) at Bloomberg and [Romulo Cintra](#) at Igalia.

Reaching Stage 1 would mean that the standards committee believes that supporting type syntax is worth considering for ECMAScript. This isn't a sure-fire thing – there are many valuable perspectives within the committee, and we do expect some amount of skepticism. A proposal like this will receive a lot of feedback and appropriate scrutiny. It may involve lots design changes along the way, and may take years to yield results.

*But* if we pull this all off, we have the chance to make one of the most impactful improvements to the world of JavaScript. We're excited by that, and we hope you are too.



If you're interested in hearing more about the specifics and current direction, [head on over to the proposal repository](#). We look forward to hearing what you think!

And lastly, the TypeScript team and the champions group would like to recognize and extend our thanks to all those who worked on [prior art](#), along with the contributors who reached out to help with types as comments, and especially [Gil Tayar](#) who helped spearhead it. We're grateful to be part of such a passionate community!



**Daniel Rosenwasser** Senior Program Manager, TypeScript

Follow

## Read next



### Announcing TypeScript 4.7 Beta

Today we are excited to announce the beta release of TypeScript 4.7! To get started using the beta, you can use npm with the following command: You can also get ...

Daniel Rosenwasser

April 8, 2022

7 comments

### Announcing TypeScript 4.7 RC

Today we're excited to announce our Release Candidate (RC) of TypeScript 4.7! Between now and the stable release of TypeScript 4.7, we expect no further changes apart ...

Daniel Rosenwasser

May 11, 2022

2 comments

## 54 comments

Comments are closed. [Login to edit/delete your existing comments](#)

1 2 3 4 >



**Steve Davis** March 9, 2022 12:21 pm 0

I love this.



**Ashley Claymore** March 9, 2022 12:51 pm 0

Most excellent!



**Daniel Brain** March 9, 2022 12:59 pm 0

Unless this is a 1:1 match with TypeScript, we'll still need to transpile everything right? Otherwise people who don't want to transpile will need to know and use a subset of TS syntax.

That problem will be exacerbated by any new TS version which introduces new syntax, keywords, and so on.



**Daniel Rosenwasser** March 9, 2022 3:19 pm 0

There will be certain constructs that may need to be compiled away given TypeScript authored today; however, not everyone uses all of those features in every project, and this proposal provides an easy on-ramp for adding type-checking to those codebases.



**Jacob Stamm** March 10, 2022 8:03 am 0

I'm definitely one of those users. I could happily do everything I need to do with a version of TypeScript from 4 years ago and none of the more advanced features that have come out recently (except maybe the improvements to type guards... I gotta have those).

**Artur Diniz Adam** March 10, 2022 7:06 pm 0

It being a subset of TypeScript makes it a partial solution and adds another complexity layer for developers to think while working on those codebases 😞

My fear is getting locked out of nice future language features due to syntax conflicts with this proposal while it doesn't solve the "build tools setup" problem entirely =(

If the idea is to completely avoid setting up build tools, wouldn't it be better to implement a client+server side lib that enables full runtime typescript? Something like:

```
<script src="lib-enable-runtime-typescript.js">
<script src="my-ts-codebase-entrypoint.ts" type="typescript">
```

Shipping TS compiler to the client would be a crazy thing to do, so the lib can just delegate compiling `my-ts-code-entrypoint.ts` to a server side solution that's already set up, hosted on same-origin, or perhaps a <http://typescript.microsoft.com/compile> \*\*

f  
t  
in

**Gabriel Dibble** March 9, 2022 1:53 pm 0

🔥👍🔥

**Jiahao Chen** March 9, 2022 4:21 pm 0

Can't wait for this exciting proposal to be implemented and shipped!

**Patricio Ezequiel Hondagneu Roig** March 9, 2022 4:38 pm 0

I absolutely adore this proposal, I hope it becomes part of the standard.

Great job everybody!

**Haoyang Gao** March 9, 2022 6:40 pm 0

Sounds great!

**Łukasz Polowczyk** March 9, 2022 7:29 pm 0

PROBLEM:

\* API is introduced.

\* people use type-comments in other ways (they put garbage in type-comments)

\* we block the syntax for the future because it would break the web. it can never be introduced in earnest again.

What do you guys think about this problem?

I mean, throwing this into type-comments doesn't bypass all the problems at all, it just might create new problems for the future.

Type-comments are NOT types, but ignored by the browser. This will be code that you can write ANY garbage in and it will be valid, working code, even though TypeScript would return errors.

This is an odd situation, potentially blocking future actual type composition.

So:

Either we add types to JavaScript for real, or it's better not to add it at all.

Because otherwise we already have to accept "hey, if you use type-comments according to the specification, but not the way we imagine it in x years, your code will break in x years".

—

How to get around it?

Make these type-comments allowed, only in a special dev browser mode...

This is mainly for such a purpose anyway, so that the developer doesn't have to wait for the code to compile for him.

Then, let the developer turn on the type-comment mode in his browser. 😊

But as far as I understand, it's not only about compilation time? It's also about being able to upload this "typed" code to the web without compiling at all?

I don't see how to solve the second problem.

Then, the problem of breaking the web disappears, as does the problem of performance degradation – it would only work in the developer's browser.



**Riccardo Cecchini** April 8, 2022 10:32 am 0



I don't agree, for today's parser reading a little more characters is not a problem, also is not a problem ignoring type check during non-development execution.

TypeScript is now a community standard, and although I personally don't like it, I prefer that typing should be part of ECMA.



**Deider Alex** March 9, 2022 9:03 pm 0



What is the advantage of this proposal, other than visual, over jsDoc?



**Tony Brix** March 12, 2022 10:14 am 0



I'm curious about who this benefits as well. This isn't anything browser users will see. The browser just views this the same as jsdoc comments so there is no benefit there. Developers still need build systems for other things (minification, etc.). I don't see how anyone benefits from moving type comments from the typescript binary to browsers.



**Kieran Pilkington** March 9, 2022 9:08 pm 0



In my view, Javascript is getting very old, and has numerous problems and quirks that have been kept for the sake of compatibility.

While the proposal here is a nice step, the problem is it only adds to an already bloated and outdated system, lipstick on a pig if you will.

I would much rather see the web browser industry work move forward with a modern language, designed for speed, with inbuilt type checking.

For example, DartLang is very similar to Javascript, yet modern, type checked, well designed, free of quirks, and in my tests, much faster.

Browsers could support Javascript and Dart side by side. Dart also has the means to convert code to javascript compatible syntax for older browsers.



**Jacob Stamm** March 10, 2022 8:08 am 0



I doubt another interpreted language will come to browsers in the foreseeable future. I think WASM and incremental evolution of JS (with breaking, non-polyfillable changes eventually being inevitable) is the best of both worlds.



**Michael Taylor** March 11, 2022 8:34 am 0




This was tried decades ago with VBScript and it failed. Javascript wasn't really designed to be used like it is today. It was purposefully loose so it could be used however you wanted but that came at the cost of hard to diagnose issues, odd behavior, etc. Some of this has been resolved in newer updates but it breaks older code.

Ultimately any new language would require browser support which means somebody would have to write the language and then become popular enough for browsers to decide to allocate resources to integrate it. This could easily take a decade or more. Add on top of that the fact that not all devices can be upgraded to a new browser and therefore they would never work with a new language so your site would either need to break compatibility or support both.

This is ultimately why languages like Typescript compile down to JS so they run anywhere. If any language would stand a chance at being standalone it would be TS given it is popular enough but since it compiles down to JS there is not a big benefit to browser manufacturers.



### Top Bloggers

 [Daniel Rosenwasser](#)  
Senior Program Manager

### Archive

- [November 2022](#)
- [October 2022](#)
- [September 2022](#)
- [August 2022](#)
- [June 2022](#)
- [May 2022](#)
- [April 2022](#)
- [March 2022](#)
- [February 2022](#)
- [January 2022](#)
- [November 2021](#)

### Relevant Links

- [The TypeScript Website](#)
- [TypeScript on GitHub](#)
- [TypeScript on Twitter](#)

## Stay informed



### What's new

- Surface Pro 9
- Surface Laptop 5
- Surface Studio 2+
- Surface Laptop Go 2
- Surface Laptop Studio
- Surface Duo 2
- Microsoft 365
- Windows 11 apps



### Microsoft Store

- Account profile
- Download Center
- Microsoft Store support
- Returns
- Order tracking
- Personal shopping appointments
- Microsoft Store Promise
- Flexible Payments

### Education

- Microsoft in education
- Devices for education
- Microsoft Teams for Education
- Microsoft 365 Education
- Education consultation appointment
- Educator training and development
- Deals for students and parents
- Azure for students

### Business

- Microsoft Cloud
- Microsoft Security
- Dynamics 365
- Microsoft 365
- Microsoft Power Platform
- Microsoft Teams
- Microsoft Industry
- Small Business

### Developer & IT

- Azure
- Developer Center
- Documentation
- Microsoft Learn
- Microsoft Tech Community
- Azure Marketplace
- AppSource
- Visual Studio

### Company

- Careers
- About Microsoft
- Company news
- Privacy at Microsoft
- Investors
- Diversity and inclusion
- Accessibility
- Sustainability

