SVELTE • MISC

TypeScript

ON THIS PAGE

You can use TypeScript within Svelte components. IDE extensions like the Svelte VS Code extension will help you catch errors right in your editor, and `svelte-check` does the same on the command line, which you can integrate into your CI.

<script lang="ts">

To use TypeScript inside your Svelte components, add `lang="ts"` to your `script` tags:

```
<script lang="ts">
  let name: string = 'world';

  function greet(name: string) {
    alert(`Hello, ${name}!`);
  }
</script>

<button onclick={(e: Event) => greet(e.target.innerText)}>
  {name as string}
</button>
```

Doing so allows you to use TypeScript's *type-only* features. That is, all features that just disappear when transpiling to JavaScript, such as type annotations or interface declarations. Features that require the TypeScript compiler to output actual code are not supported. This includes:

- using enums

initializers

using features that are not yet part of the ECMAScript standard (i.e. not level 4 in the TC39 process) and therefore not implemented yet within Acorn, the parser we use for parsing JavaScript

If you want to use one of these features, you need to setup up a `script` preprocessor.

Preprocessor setup

To use non-type-only TypeScript features within Svelte components, you need to add a preprocessor that will turn TypeScript into JavaScript.

Using SvelteKit or Vite

The easiest way to get started is scaffolding a new SvelteKit project by typing `npx sv create`, following the prompts and choosing the TypeScript option.

svelte.config.js

```
import { vitePreprocess } from '@sveltejs/kit/vite';

const config = {
  preprocess: vitePreprocess()
};

export default config;
```

If you don't need or want all the features SvelteKit has to offer, you can scaffold a Svelte-flavoured Vite project instead by typing `npm create vite@latest` and selecting the `svelte-ts` option.

svelte.config.js

```
import { vitePreprocess } from '@sveltejs/vite-plugin-svelte';
```

```
export default config;
```

In both cases, a `svelte.config.js` with `vitePreprocess` will be added. Vite/SvelteKit will read from this config file.

Other build tools

If you're using tools like Rollup or Webpack instead, install their respective Svelte plugins. For Rollup that's rollup-plugin-svelte and for Webpack that's svelte-loader. For both, you need to install `typescript` and `svelte-preprocess` and add the preprocessor to the plugin config (see the respective READMEs for more info). If you're starting a new project, you can also use the rollup or webpack template to scaffold the setup from a script.

> If you're starting a new project, we recommend using SvelteKit or Vite instead

Typing $props

Type `$props` just like a regular object with certain properties.

```
<script lang="ts">
  import type { Snippet } from 'svelte';

  interface Props {
    requiredProperty: number;
    optionalProperty?: boolean;
    snippetWithStringArgument: Snippet<[string]>;
    eventHandler: (arg: string) => void;
    [key: string]: unknown;
  }

  let {
    requiredProperty,
    optionalProperty,
</script>

<button onclick={() => eventHandler('clicked button')}>
  {@render snippetWithStringArgument('hello')}
</button>
```

Generic $props

Components can declare a generic relationship between their properties. One example is a generic list component that receives a list of items and a callback property that receives an item from the list. To declare that the `items` property and the `select` callback operate on the same types, add the `generics` attribute to the `script` tag:

```
<script lang="ts" generics="Item extends { text: string }">
  interface Props {
    items: Item[];
    select(item: Item): void;
  }

  let { items, select }: Props = $props();
</script>

{#each items as item}
  <button onclick={() => select(item)}>
    {item.text}
  </button>
{/each}
```

The content of `generics` is what you would put between the `<...>` tags of a generic function. In other words, you can use multiple generics, `extends` and fallback types.

Typing wrapper components

In case you're writing a component that wraps a native element, you may want to expose all component:

```
<script lang="ts">
  import type { HTMLButtonAttributes } from 'svelte/elements';

  let { children, ...rest }: HTMLButtonAttributes = $props();
</script>

<button {...rest}>
  {@render children()}
</button>
```

Not all elements have a dedicated type definition. For those without one, use `SvelteHTMLElements` :

```
<script lang="ts">
  import type { SvelteHTMLElements } from 'svelte/elements';

  let { children, ...rest }: SvelteHTMLElements['div'] = $props();
</script>

<div {...rest}>
  {@render children()}
</div>
```

Typing $state

You can type `$state` like any other variable.

```
let count: number = $state(0);
```

If you don't give `$state` an initial value, part of its types will be `undefined` .

```
// Error: Type 'number | undefined' is not assignable to type 'number'
```

is especially useful in the context of classes:

```
class Counter {
  count = $state() as number;
  constructor(initial: number) {
    this.count = initial;
  }
}
```

The Component type

Svelte components are of type `Component`. You can use it and its related types to express a variety of constraints.

Using it together with dynamic components to restrict what kinds of component can be passed to it:

```
<script lang="ts">
  import type { Component } from 'svelte';

  interface Props {
    // only components that have at most the "prop"
    // property required can be passed
    DynamicComponent: Component<{ prop: string }>;
  }

  let { DynamicComponent }: Props = $props();
</script>

<DynamicComponent prop="foo" />
```

Legacy mode   show all

To extract the properties from a component, use `ComponentProps`.

```
  component: TComponent,
  props: ComponentProps<TComponent>
) {}

// Errors if the second argument is not the correct props expected
// by the component in the first argument.
withProps(MyComponent, { foo: 'bar' });
```

To declare that a variable expects the constructor or instance type of a component:

```
<script lang="ts">
  import MyComponent from './MyComponent.svelte';

  let componentConstructor: typeof MyComponent = MyComponent;
  let componentInstance: MyComponent;
</script>

<MyComponent bind:this={componentInstance} />
```

Enhancing built-in DOM types

Svelte provides a best effort of all the HTML DOM types that exist. Sometimes you may want to use experimental attributes or custom events coming from an action. In these cases, TypeScript will throw a type error, saying that it does not know these types. If it's a non-experimental standard attribute/event, this may very well be a missing typing from our HTML typings. In that case, you are welcome to open an issue and/or a PR fixing it.

In case this is a custom or experimental attribute/event, you can enhance the typings like this:

additional-svelte-typings.d.ts

```
declare namespace svelteHTML {
  // enhance elements
  interface IntrinsicElements {
    onbeforeinstallprompt?: (event: any) => any;
    // If you want to use myCustomAttribute={..} (note: all lowercase)
    mycustomattribute?: any; // You can replace any with something more specific if you l
  }
}
```

Then make sure that `d.ts` file is referenced in your `tsconfig.json`. If it reads something like `"include": ["src/**/*"]` and your `d.ts` file is inside `src`, it should work. You may need to reload for the changes to take effect.

You can also declare the typings by augmenting the `svelte/elements` module like this:

additional-svelte-typings.d.ts

```
import { HTMLButtonAttributes } from 'svelte/elements';

declare module 'svelte/elements' {
  export interface SvelteHTMLElements {
    'custom-button': HTMLButtonAttributes;
  }

  // allows for more granular control over what element to add the typings to
  export interface HTMLButtonAttributes {
    veryexperimentalattribute?: string;
  }
}

export {}; // ensure this is not an ambient module, else types will be overridden instead
```

Edit this page on GitHub