SVELTE • LEGACY APIS

# on:

In runes mode, event handlers are just like any other attribute or prop.

In legacy mode, we use the `on:` directive:

App.svelte

JS TS

```
<script lang="ts">
  let count = 0;

  function handleClick(event: MouseEvent) {
    count += 1;
  }
</script>

<button on:click={handleClick}>
  count: {count}
</button>
```

Handlers can be declared inline with no performance penalty:

```
<button on:click={() => (count += 1)}>
  count: {count}
</button>
```

Add *modifiers* to element event handlers with the `|` character.

```
<form on:submit|preventDefault={handleSubmit}>
  <!-- the `submit` event's default is prevented,
       so the page won't reload -->
```

- `preventDefault` — calls `event.preventDefault()` before running the handler
- `stopPropagation` — calls `event.stopPropagation()`, preventing the event reaching the next element
- `stopImmediatePropagation` - calls `event.stopImmediatePropagation()`, preventing other listeners of the same event from being fired.
- `passive` — improves scrolling performance on touch/wheel events (Svelte will add it automatically where it's safe to do so)
- `nonpassive` — explicitly set `passive: false`
- `capture` — fires the handler during the *capture* phase instead of the *bubbling* phase
- `once` — remove the handler after the first time it runs
- `self` — only trigger handler if `event.target` is the element itself
- `trusted` — only trigger handler if `event.isTrusted` is `true`. I.e. if the event is triggered by a user action.

Modifiers can be chained together, e.g. `on:click|once|capture={...}`.

If the `on:` directive is used without a value, the component will *forward* the event, meaning that a consumer of the component can listen for it.

```svelte
<button on:click>
  The component itself will emit the click event
</button>
```

It's possible to have multiple event listeners for the same event:

App.svelte

JS TS

```svelte
<script lang="ts">
  let count = 0;

  function increment() {
    console.log(event);
  }
</script>

<button on:click={increment} on:click={log}>
  clicks: {count}
</button>
```

## Component events

Components can dispatch events by creating a *dispatcher* when they are initialised:

Stepper.svelte -->

```
<script>
  import { createEventDispatcher } from 'svelte';
  const dispatch = createEventDispatcher();
</script>

<button on:click={() => dispatch('decrement')}>decrement</button>
<button on:click={() => dispatch('increment')}>increment</button>
```

`dispatch` creates a `CustomEvent`. If a second argument is provided, it becomes the `detail` property of the event object.

A consumer of this component can listen for the dispatched events:

```
<script>
  import Stepper from './Stepper.svelte';

  let n = 0;
</script>

<Stepper
  on:decrement={() => n -= 1}
  on:increment={() => n += 1}
/>
```

Docs

immediate children.

Other than `once`, modifiers are not valid on component event handlers.

> If you're planning an eventual migration to Svelte 5, use callback props instead. This will make upgrading easier as `createEventDispatcher` is deprecated:
>
> Stepper.svelte
>
> ```
> <script>
>   export let decrement;
>   export let increment;
> </script>
>
> <button on:click={decrement}>decrement</button>
> <button on:click={increment}>increment</button>
> ```

Edit this page on GitHub

PREVIOUS
$$props and $$restProps

NEXT
<slot>