SVELTE • TEMPLATE SYNTAX

Basic markup

ON THIS PAGE

Markup inside a Svelte component can be thought of as HTML++.

Tags

A lowercase tag, like `<div>`, denotes a regular HTML element. A capitalised tag or a tag that uses dot notation, such as `<Widget>` or `<my.stuff>`, indicates a *component*.

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

<div>
  <Widget />
</div>
```

Element attributes

By default, attributes work exactly like their HTML counterparts.

```
<div class="foo">
  <button disabled>can't touch this</button>
</div>
```

As in HTML, values may be unquoted.

Attribute values can contain JavaScript expressions.

```
<a href="page/{p}">page {p}</a>
```

Or they can *be* JavaScript expressions.

```
<button disabled={!clickable}>...</button>
```

Boolean attributes are included on the element if their value is truthy and excluded if it's falsy.

All other attributes are included unless their value is nullish (`null` or `undefined`).

```
<input required={false} placeholder="This input field is not required" />
<div title={null}>This div has no title attribute</div>
```

> Quoting a singular expression does not affect how the value is parsed, but in Svelte 6 it will cause the value to be coerced to a string:
>
> ```
> <button disabled="{number !== 42}">...</button>
> ```

When the attribute name and value match (`name={name}`), they can be replaced with `{name}` .

```
<button {disabled}>...</button>
<!-- equivalent to
<button disabled={disabled}>...</button>
-->
```

attributes, which are a feature of the DOM.

As with elements, `name={name}` can be replaced with the `{name}` shorthand.

```
<Widget foo={bar} answer={42} text="hello" />
```

Spread attributes allow many attributes or properties to be passed to an element or component at once.

An element or component can have multiple spread attributes, interspersed with regular ones.

```
<Widget {...things} />
```

Events

Listening to DOM events is possible by adding attributes to the element that start with `on`. For example, to listen to the `click` event, add the `onclick` attribute to a button:

```
<button onclick={() => console.log('clicked')}>click me</button>
```

Event attributes are case sensitive. `onclick` listens to the `click` event, `onClick` listens to the `Click` event, which is different. This ensures you can listen to custom events that have uppercase characters in them.

Because events are just attributes, the same rules as for attributes apply:

- you can use the shorthand form: `<button {onclick}>click me</button>`
- you can spread them: `<button {...thisSpreadContainsEventAttributes}>click me</button>`

Timing-wise, event attributes always fire after events from bindings (e.g. `oninput` always

When using `ontouchstart` and `ontouchmove` event attributes, the handlers are passive for better performance. This greatly improves responsiveness by allowing the browser to scroll the document immediately, rather than waiting to see if the event handler calls `event.preventDefault()`.

In the very rare cases that you need to prevent these event defaults, you should use `on` instead (for example inside an action).

Event delegation

To reduce memory footprint and increase performance, Svelte uses a technique called event delegation. This means that for certain events — see the list below — a single event listener at the application root takes responsibility for running any handlers on the event's path.

There are a few gotchas to be aware of:

- when you manually dispatch an event with a delegated listener, make sure to set the `{ bubbles: true }` option or it won't reach the application root
- when using `addEventListener` directly, avoid calling `stopPropagation` or the event won't reach the application root and handlers won't be invoked. Similarly, handlers added manually inside the application root will run *before* handlers added declaratively deeper in the DOM (with e.g. `onclick={...}`), in both capturing and bubbling phases. For these reasons it's better to use the `on` function imported from `svelte/events` rather than `addEventListener`, as it will ensure that order is preserved and `stopPropagation` is handled correctly.

The following event handlers are delegated:

```
beforeinput

click

change

dblclick
```

```
focusout

input

keydown

keyup

mousedown

mousemove

mouseout

mouseover

mouseup

pointerdown

pointermove

pointerout

pointerover

pointerup

touchend

touchmove

touchstart
```

Text expressions

A JavaScript expression can be included as text by surrounding it with curly braces.

```
{expression}
```

Curly braces can be included in a Svelte template by using their HTML entity strings

parentheses.

```
<h1>Hello {name}!</h1>
<p>{a} + {b} = {a + b}.</p>

<div>{(/^[A-Za-z ]+$/).test(value) ? x : y}</div>
```

The expression will be stringified and escaped to prevent code injections. If you want to render HTML, use the `{@html}` tag instead.

```
{@html potentiallyUnsafeHtmlString}
```

> Make sure that you either escape the passed string or only populate it with values that are under your control in order to prevent XSS attacks

Comments

You can use HTML comments inside components.

```
<!-- this is a comment! --><h1>Hello world</h1>
```

Comments beginning with `svelte-ignore` disable warnings for the next block of markup. Usually, these are accessibility warnings; make sure that you're disabling them for a good reason.

```
<!-- svelte-ignore a11y-autofocus -->
<input bind:value={name} autofocus />
```

You can add a special comment starting with `@component` that will show up when hovering over the component name in other files.

```
@component
- You can use markdown here.
- You can also use code blocks here.
- Usage:
  ```html
  <Main name="Arethra">
  ```
-->
<script>
  let { name } = $props();
</script>

<main>
  <h1>
    Hello, {name}
  </h1>
</main>
```

Edit this page on GitHub

PREVIOUS
$host

NEXT
{#if ...}