SVELTE • LEGACY APIS

# Imperative component API

ON THIS PAGE

In Svelte 3 and 4, the API for interacting with a component is different than in Svelte 5. Note that this page does *not* apply to legacy mode components in a Svelte 5 application.

## Creating a component

```
const component = new Component(options);
```

A client-side component — that is, a component compiled with `generate: 'dom'` (or the `generate` option left unspecified) is a JavaScript class.

```
import App from './App.svelte';

const app = new App({
  target: document.body,
  props: {
    // assuming App.svelte contains something like
    // `export let answer`:
    answer: 42
  }
});
```

The following initialisation options can be provided:

| option | default | description |
| --- | --- | --- |
| `target` | none | An `HTMLElement` or `ShadowRoot` to render to. This option is required |
| `anchor` | `null` | A child of `target` to render the component immediately before |

| | | |
|---|---|---|
| `props` | `{}` | An object of properties to supply to the component |
| `context` | `new Map()` | A `Map` of root-level context key-value pairs to supply to the component |
| `hydrate` | `false` | See below |
| `intro` | `false` | If `true`, will play transitions on initial render, rather than waiting for subsequent state changes |

Existing children of `target` are left where they are.

The `hydrate` option instructs Svelte to upgrade existing DOM (usually from server-side rendering) rather than creating new elements. It will only work if the component was compiled with the `hydratable: true` option. Hydration of `<head>` elements only works properly if the server-side rendering code was also compiled with `hydratable: true`, which adds a marker to each element in the `<head>` so that the component knows which elements it's responsible for removing during hydration.

Whereas children of `target` are normally left alone, `hydrate: true` will cause any children to be removed. For that reason, the `anchor` option cannot be used alongside `hydrate: true`.

The existing DOM doesn't need to match the component — Svelte will 'repair' the DOM as it goes.

index.js

```
import App from './App.svelte';

const app = new App({
  target: document.querySelector('#server-rendered-html'),
  hydrate: true
});
```

In Svelte 5+, use `mount` instead

```
component.$set(props);
```

Programmatically sets props on an instance. `component.$set({ x: 1 })` is equivalent to `x = 1` inside the component's `<script>` block.

Calling this method schedules an update for the next microtask — the DOM is *not* updated synchronously.

```
component.$set({ answer: 42 });
```

> In Svelte 5+, use `$state` instead to create a component props and update that

```
let props = $state({ answer: 42 });
const component = mount(Component, { props });
// ...
props.answer = 24;
```

## $on

```
component.$on(ev, callback);
```

Causes the `callback` function to be called whenever the component dispatches an `event`.

A function is returned that will remove the event listener when called.

```
const off = component.$on('selected', (event) => {
  console.log(event.detail.selection);
});

off();
```

> In Svelte 5+, pass callback props instead

```
component.$destroy();
```

Removes a component from the DOM and triggers any `onDestroy` handlers.

> In Svelte 5+, use `unmount` instead

## Component props

```
component.prop;
```

```
component.prop = value;
```

If a component is compiled with `accessors: true`, each instance will have getters and setters corresponding to each of the component's props. Setting a value will cause a *synchronous* update, rather than the default async update caused by `component.$set(...)`.

By default, `accessors` is `false`, unless you're compiling as a custom element.

```
console.log(component.count);
component.count += 1;
```

> In Svelte 5+, this concept is obsolete. If you want to make properties accessible from the outside, `export` them

## Server-side component API

```
const result = Component.render(...)
```

is somewhat different.

A server-side component exposes a `render` method that can be called with optional props. It returns an object with `head`, `html`, and `css` properties, where `head` contains the contents of any `<svelte:head>` elements encountered.

You can import a Svelte component directly into Node using `svelte/register`.

```
require('svelte/register');

const App = require('./App.svelte').default;

const { head, html, css } = App.render({
  answer: 42
});
```

The `.render()` method accepts the following parameters:

| parameter | default | description |
| --- | --- | --- |
| `props` | `{}` | An object of properties to supply to the component |
| `options` | `{}` | An object of options |

The `options` object takes in the following options:

| option | default | description |
| --- | --- | --- |
| `context` | `new Map()` | A `Map` of root-level context key-value pairs to supply to the component |

```
const { head, html, css } = App.render(
  // props
  { answer: 42 },
  // options
  {
    context: new Map([['context-key', 'context-value']])
  }
);
```

Edit this page on GitHub

PREVIOUS

NEXT

<svelte:self>