SVELTE • REFERENCE

svelte

```js
import {
  afterUpdate,
  beforeUpdate,
  createEventDispatcher,
  createRawSnippet,
  flushSync,
  getAllContexts,
  getContext,
  hasContext,
  hydrate,
  mount,
  onDestroy,
  onMount,
  setContext,
  tick,
  unmount,
  untrack
} from 'svelte';
```

afterUpdate

> Deprecated
>
> Use `$effect` instead — see https://svelte.dev/docs/svelte/$effect

Schedules a callback to run immediately after the component has been updated.

The first time the callback runs will be after the initial `onMount` .

In runes mode use `$effect` instead.

beforeUpdate

> Deprecated
> Use `$effect.pre` instead — see https://svelte.dev/docs/svelte/$effect#$effect.pre

Schedules a callback to run immediately before the component is updated after any state change.

The first time the callback runs will be before the initial `onMount` .

In runes mode use `$effect.pre` instead.

```
function beforeUpdate(fn: () => void): void;
```

createEventDispatcher

> Deprecated
> Use callback props and/or the `$host()` rune instead — see https://svelte.dev/docs/svelte/v5-migration-guide#Event-changes-Component-events

Creates an event dispatcher that can be used to dispatch component events. Event dispatchers are functions that can take two arguments: `name` and `detail` .

Component events created with `createEventDispatcher` create a CustomEvent. These events do not bubble. The `detail` argument corresponds to the CustomEvent.detail property and can contain any type of data.

The event dispatcher can be typed to narrow the allowed event names and the type of the `detail` argument:

```
  change: string; // takes a detail argument of type string, which is required
  optional: number | null; // takes an optional detail argument of type number
}>();
```

```
function createEventDispatcher<
  EventMap extends Record<string, any> = any
>(): EventDispatcher<EventMap>;
```

createRawSnippet

Create a snippet programmatically

```
function createRawSnippet<Params extends unknown[]>(
  fn: (...params: Getters<Params>) => {
    render: () => string;
    setup?: (element: Element) => void | (() => void);
  }
): Snippet<Params>;
```

flushSync

Synchronously flushes any pending state changes and those that result from it.

```
function flushSync(fn?: (() => void) | undefined): void;
```

getAllContexts

Retrieves the whole context map that belongs to the closest parent component. Must be called during component initialisation. Useful, for example, if you programmatically create a component and want to pass the existing context to it.

```
>(): T;
```

getContext

Retrieves the context that belongs to the closest parent component with the specified `key`. Must be called during component initialisation.

```
function getContext<T>(key: any): T;
```

hasContext

Checks whether a given `key` has been set in the context of a parent component. Must be called during component initialisation.

```
function hasContext(key: any): boolean;
```

hydrate

Hydrates a component on the given target and returns the exports and potentially the props (if compiled with `accessors: true`) of the component

```
function hydrate<
  Props extends Record<string, any>,
  Exports extends Record<string, any>
>(
  component:
    | ComponentType<SvelteComponent<Props>>
    | Component<Props, Exports, any>,
  options: {} extends Props
    ? {
        target: Document | Element | ShadowRoot;
        props?: Props;
        intro?: boolean;
        recover?: boolean;
      }
    : {
        target: Document | Element | ShadowRoot;
        props: Props;
        events?: Record<string, (e: any) => any>;
        context?: Map<any, any>;
        intro?: boolean;
        recover?: boolean;
      }
): Exports;
```

mount

Mounts a component to the given target and returns the exports and potentially the props (if compiled with `accessors: true`) of the component. Transitions will play during the initial render unless the `intro` option is set to `false` .

```
function mount<
  Props extends Record<string, any>,
  Exports extends Record<string, any>
>(
  component:
    | ComponentType<SvelteComponent<Props>>
    | Component<Props, Exports, any>,
  options: MountOptions<Props>
): Exports;
```

onDestroy

Schedules a callback to run immediately before the component is unmounted.

Out of `onMount` , `beforeUpdate` , `afterUpdate` and `onDestroy` , this is the only one that runs inside a server-side component.

onMount

The `onMount` function schedules a callback to run as soon as the component has been mounted to the DOM. It must be called during the component's initialisation (but doesn't need to live *inside* the component; it can be called from an external module).

If a function is returned *synchronously* from `onMount`, it will be called when the component is unmounted.

`onMount` does not run inside server-side components.

```
function onMount<T>(
  fn: () =>
    | NotFunction<T>
    | Promise<NotFunction<T>>
    | (() => any)
): void;
```

setContext

Associates an arbitrary `context` object with the current component and the specified `key` and returns that object. The context is then available to children of the component (including slotted content) with `getContext`.

Like lifecycle functions, this must be called during component initialisation.

```
function setContext<T>(key: any, context: T): T;
```

tick

Returns a promise that resolves once any pending state changes have been applied.

unmount

Unmounts a component that was previously mounted using `mount` or `hydrate` .

```
function unmount(component: Record<string, any>): void;
```

untrack

When used inside a `$derived` or `$effect` , any state read inside `fn` will not be treated as a dependency.

```
$effect(() => {
  // this will run when `data` changes, but not when `time` changes
  save(data, {
    timestamp: untrack(() => time)
  });
});
```

```
function untrack<T>(fn: () => T): T;
```

Component

Can be used to create strongly typed Svelte components.

Example:

You have component library on npm called `component-library` , from which you export a component called `MyComponent` . For Svelte+TypeScript users, you want to provide typings. Therefore you create a `index.d.ts` :

```
import type { Component } from 'svelte';
export declare const MyComponent: Component<{ foo: string }> {}
```

```svelte
<script lang="ts">
  import { MyComponent } from "component-library";
</script>
<MyComponent foo={'bar'} />
```

```ts
interface Component<
  Props extends Record<string, any> = {},
  Exports extends Record<string, any> = {},
  Bindings extends keyof Props | '' = string
> {…}
```

```ts
(
  this: void,
  internals: ComponentInternals,
  props: Props
): {
  /**
   * @deprecated This method only exists when using one of the legacy compatibility helpe
   * is a stop-gap solution. See https://svelte.dev/docs/svelte/v5-migration-guide#Compon
   * for more info.
   */
  $on?(type: string, callback: (e: any) => void): () => void;
  /**
   * @deprecated This method only exists when using one of the legacy compatibility helpe
   * is a stop-gap solution. See https://svelte.dev/docs/svelte/v5-migration-guide#Compon
   * for more info.
   */
  $set?(props: Partial<Props>): void;
} & Exports;
```

`internal` An internal object used by Svelte. Do not use or modify.

`props` The props passed to the component.

```ts
element?: typeof HTMLElement;
```

ComponentConstructorOptions

> Deprecated
>
> In Svelte 4, components are classes. In Svelte 5, they are functions. Use `mount` instead to instantiate components. See migration guide for more info.

```
interface ComponentConstructorOptions<
  Props extends Record<string, any> = Record<string, any>
> {…}
```

```
target: Element | Document | ShadowRoot;
```

```
anchor?: Element;
```

```
props?: Props;
```

```
context?: Map<any, any>;
```

```
hydrate?: boolean;
```

```
intro?: boolean;
```

```
recover?: boolean;
```

```
sync?: boolean;
```

```
$$inline?: boolean;
```

> Deprecated
>
> The new `Component` type does not have a dedicated Events type. Use `ComponentProps` instead.

```
type ComponentEvents<Comp extends SvelteComponent> =
  Comp extends SvelteComponent<any, infer Events>
    ? Events
    : never;
```

ComponentInternals

Internal implementation details that vary between environments

```
type ComponentInternals = Branded<{}, 'ComponentInternals'>;
```

ComponentProps

Convenience type to get the props the given component expects.

Example: Ensure a variable contains the props expected by `MyComponent` :

```
import type { ComponentProps } from 'svelte';
import MyComponent from './MyComponent.svelte';

// Errors if these aren't the correct props expected by MyComponent.
const props: ComponentProps<typeof MyComponent> = { foo: 'bar' };
```

> In Svelte 4, you would do `ComponentProps<MyComponent>` because `MyComponent` was a class.

Example: A generic function that accepts some component and infers the type of its props:

```
import type { Component, ComponentProps } from 'svelte';
import MyComponent from './MyComponent.svelte';
```

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

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

```
type ComponentProps<
  Comp extends SvelteComponent | Component<any, any>
> =
  Comp extends SvelteComponent<infer Props>
    ? Props
    : Comp extends Component<infer Props, any>
      ? Props
      : never;
```

ComponentType

> Deprecated
>
> This type is obsolete when working with the new `Component` type.

```
type ComponentType<
  Comp extends SvelteComponent = SvelteComponent
> = (new (
  options: ComponentConstructorOptions<
    Comp extends SvelteComponent<infer Props>
      ? Props
      : Record<string, any>
  >
) => Comp) & {
  /** The custom element version of the component. Only present if compiled with the `cus
  element?: typeof HTMLElement;
};
```

EventDispatcher

```
> {…}
```

```
<Type extends keyof EventMap>(
  ...args: null extends EventMap[Type]
    ? [type: Type, parameter?: EventMap[Type] | null | undefined, options?: DispatchOptio
    : undefined extends EventMap[Type]
      ? [type: Type, parameter?: EventMap[Type] | null | undefined, options?: DispatchOpt
      : [type: Type, parameter: EventMap[Type], options?: DispatchOptions]
): boolean;
```

MountOptions

Defines the options accepted by the `mount()` function.

```
type MountOptions<
  Props extends Record<string, any> = Record<string, any>
> = {
  /**
   * Target element where the component will be mounted.
   */
  target: Document | Element | ShadowRoot;
  /**
   * Optional node inside `target`. When specified, it is used to render the component im
   */
  anchor?: Node;
  /**
   * Allows the specification of events.
   * @deprecated Use callback props instead.
   */
  events?: Record<string, (e: any) => any>;
  /**
   * Can be accessed via `getContext()` at the component level.
   */
  context?: Map<any, any>;
  /**
   * Whether or not to play transitions on initial render.
   * @default true
   */
  intro?: boolean;
```

```
      /**
       * Component properties.
       */
      props?: Props;
    }
  : {
      /**
       * Component properties.
       */
      props: Props;
    });
```

Snippet

The type of a `#snippet` block. You can use it to (for example) express that your component expects a snippet of a certain type:

```
let { banner }: { banner: Snippet<[{ text: string }]> } = $props();
```

You can only call a snippet through the `{@render ...}` tag.

/docs/svelte/snippet

```
interface Snippet<Parameters extends unknown[] = []> {…}
```

```
(
  this: void,
  // this conditional allows tuples but not arrays. Arrays would indicate a
  // rest parameter type, which is not supported. If rest parameters are added
  // in the future, the condition can be removed.
  ...args: number extends Parameters['length'] ? never : Parameters
): {
  '{@render ...} must be called with a Snippet': "import type { Snippet } from 'svelte'";
} & typeof SnippetReturn;
```

This was the base class for Svelte components in Svelte 4. Svelte 5+ components are completely different under the hood. For typing, use `Component` instead. To instantiate components, use `mount` instead`. See migration guide for more info.

```
class SvelteComponent<
  Props extends Record<string, any> = Record<string, any>,
  Events extends Record<string, any> = any,
  Slots extends Record<string, any> = any
> {…}
```

```
static element?: typeof HTMLElement;
```

The custom element version of the component. Only present if compiled with the `customElement` compiler option

```
[prop: string]: any;
```

```
constructor(options: ComponentConstructorOptions<Properties<Props, Slots>>);
```

DEPRECATED This constructor only exists when using the `asClassComponent` compatibility helper, which is a stop-gap solution. Migrate towards using `mount` instead. See https://svelte.dev/docs/svelte/v5-migration-guide#Components-are-no-longer-classes for more info.

```
$destroy(): void;
```

DEPRECATED This method only exists when using one of the legacy compatibility helpers, which is a stop-gap solution. See https://svelte.dev/docs/svelte/v5-migration-guide#Components-are-no-longer-classes for more info.

```
$on<K extends Extract<keyof Events, string>>(
  type: K,
```

DEPRECATED This method only exists when using one of the legacy compatibility helpers, which is a stop-gap solution. See https://svelte.dev/docs/svelte/v5-migration-guide#Components-are-no-longer-classes for more info.

```
$set(props: Partial<Props>): void;
```

DEPRECATED This method only exists when using one of the legacy compatibility helpers, which is a stop-gap solution. See https://svelte.dev/docs/svelte/v5-migration-guide#Components-are-no-longer-classes for more info.

SvelteComponentTyped

> Deprecated
>
> Use `Component` instead. See migration guide for more information.

```
class SvelteComponentTyped<
  Props extends Record<string, any> = Record<string, any>,
  Events extends Record<string, any> = any,
  Slots extends Record<string, any> = any
> extends SvelteComponent<Props, Events, Slots> {}
```