SVELTEKIT • REFERENCE

Configuration

ON THIS PAGE

Your project's configuration lives in a `svelte.config.js` file at the root of your project. As well as SvelteKit, this config object is used by other tooling that integrates with Svelte such as editor extensions.

svelte.config.js

```js
import adapter from '@sveltejs/adapter-auto';

/** @type {import('@sveltejs/kit').Config} */
const config = {
  kit: {
    adapter: adapter()
  }
};

export default config;
```

Config

```ts
interface Config {…}
```

```ts
compilerOptions?: CompileOptions;
```

DEFAULT `{}`

Options passed to `svelte.compile`.

Docs

DEFAULT `[".svelte"]`

List of file extensions that should be treated as Svelte files.

```
kit?: KitConfig;
```

SvelteKit options

```
preprocess?: any;
```

Preprocessor options, if any. Preprocessing can alternatively also be done through Vite's preprocessor capabilities.

```
vitePlugin?: PluginOptions;
```

`vite-plugin-svelte` plugin options.

```
[key: string]: any;
```

Any additional options required by tooling that integrates with Svelte.

KitConfig

The `kit` property configures SvelteKit, and can have the following properties:

adapter

DEFAULT `undefined`

Docs

DEFAULT `{}`

An object containing zero or more aliases used to replace values in `import` statements. These aliases are automatically passed to Vite and TypeScript.

svelte.config.js

```js
/** @type {import('@sveltejs/kit').Config} */
const config = {
  kit: {
    alias: {
      // this will match a file
      'my-file': 'path/to/my-file.js',

      // this will match a directory and its contents
      // (`my-directory/x` resolves to `path/to/my-directory/x`)
      'my-directory': 'path/to/my-directory',

      // an alias ending /* will only match
      // the contents of a directory, not the directory itself
      'my-directory/*': 'path/to/my-directory/*'
    }
  }
};
```

> The built-in `$lib` alias is controlled by `config.kit.files.lib` as it is used for packaging.

> You will need to run `npm run dev` to have SvelteKit automatically generate the required alias configuration in `jsconfig.json` or `tsconfig.json` .

appDir

DEFAULT `"_app"`

The directory where SvelteKit keeps its stuff, including static assets (such as JS and CSS) and internally-used routes.

Content Security Policy configuration. CSP helps to protect your users against cross-site scripting (XSS) attacks, by limiting the places resources can be loaded from. For example, a configuration like this...

svelte.config.js

```js
/** @type {import('@sveltejs/kit').Config} */
const config = {
  kit: {
    csp: {
      directives: {
        'script-src': ['self']
      },
      // must be specified with either the `report-uri` or `report-to` directives, or bot
      reportOnly: {
        'script-src': ['self'],
        'report-uri': ['/']
      }
    }
  }
};

export default config;
```

...would prevent scripts loading from external sites. SvelteKit will augment the specified directives with nonces or hashes (depending on `mode`) for any inline styles and scripts it generates.

To add a nonce for scripts and links manually included in `src/app.html` , you may use the placeholder `%sveltekit.nonce%` (for example `<script nonce="%sveltekit.nonce%">`).

When pages are prerendered, the CSP header is added via a `<meta http-equiv>` tag (note that in this case, `frame-ancestors` , `report-uri` and `sandbox` directives will be ignored).

> When `mode` is `'auto'` , SvelteKit will use nonces for dynamically rendered pages and hashes for prerendered pages. Using nonces with prerendered pages is insecure and therefore

in your app, you must either leave the `style-src` directive unspecified or add `unsafe-inline` .

If this level of configuration is insufficient and you have more dynamic requirements, you can use the `handle` hook to roll your own CSP.

```
mode?: 'hash' | 'nonce' | 'auto';
```

Whether to use hashes or nonces to restrict `<script>` and `<style>` elements. `'auto'` will use hashes for prerendered pages, and nonces for dynamically rendered pages.

```
directives?: CspDirectives;
```

Directives that will be added to `Content-Security-Policy` headers.

```
reportOnly?: CspDirectives;
```

Directives that will be added to `Content-Security-Policy-Report-Only` headers.

csrf

Protection against cross-site request forgery (CSRF) attacks.

```
checkOrigin?: boolean;
```

DEFAULT `true`

Whether to check the incoming `origin` header for `POST` , `PUT` , `PATCH` , or `DELETE` form submissions and verify that it matches the server's origin.

To allow people to make `POST` , `PUT` , `PATCH` , or `DELETE` requests with a `Content-Type` of `application/x-www-form-urlencoded` , `multipart/form-data` , or `text/plain` to your

DEFAULT `false`

Whether or not the app is embedded inside a larger app. If `true`, SvelteKit will add its event listeners related to navigation etc on the parent of `%sveltekit.body%` instead of `window`, and will pass `params` from the server rather than inferring them from `location.pathname`. Note that it is generally not supported to embed multiple SvelteKit apps on the same page and use client-side SvelteKit features within them (things such as pushing to the history state assume a single instance).

env

Environment variable configuration

```
dir?: string;
```

DEFAULT `"."`

The directory to search for `.env` files.

```
publicPrefix?: string;
```

DEFAULT `"PUBLIC_"`

A prefix that signals that an environment variable is safe to expose to client-side code. See `$env/static/public` and `$env/dynamic/public`. Note that Vite's `envPrefix` must be set separately if you are using Vite's environment variable handling - though use of that feature should generally be unnecessary.

```
privatePrefix?: string;
```

DEFAULT `""`

Docs

Environment variables matching neither the public nor the private prefix will be discarded completely. See `$env/static/private` and `$env/dynamic/private` .

files

Where to find various files within your project.

```
assets?: string;
```

DEFAULT `"static"`

a place to put static files that should have stable URLs and undergo no processing, such as `favicon.ico` or `manifest.json`

```
hooks?: {…}
```

```
client?: string;
```

DEFAULT `"src/hooks.client"`

The location of your client hooks.

```
server?: string;
```

DEFAULT `"src/hooks.server"`

The location of your server hooks.

```
universal?: string;
```

DEFAULT `"src/hooks"`

Docs

```
lib?: string;
```

DEFAULT `"src/lib"`

your app's internal library, accessible throughout the codebase as `$lib`

```
params?: string;
```

DEFAULT `"src/params"`

a directory containing parameter matchers

```
routes?: string;
```

DEFAULT `"src/routes"`

the files that define the structure of your app (see Routing)

```
serviceWorker?: string;
```

DEFAULT `"src/service-worker"`

the location of your service worker's entry point (see Service workers)

```
appTemplate?: string;
```

DEFAULT `"src/app.html"`

the location of the template for HTML responses

```
errorTemplate?: string;
```

inlineStyleThreshold

`DEFAULT 0`

Inline CSS inside a `<style>` block at the head of the HTML. This option is a number that specifies the maximum length of a CSS file in UTF-16 code units, as specified by the String.length property, to be inlined. All CSS files needed for the page and smaller than this value are merged and inlined in a `<style>` block.

> This results in fewer initial requests and can improve your First Contentful Paint score. However, it generates larger HTML output and reduces the effectiveness of browser caches. Use it advisedly.

moduleExtensions

`DEFAULT [".js", ".ts"]`

An array of file extensions that SvelteKit will treat as modules. Files with extensions that match neither `config.extensions` nor `config.kit.moduleExtensions` will be ignored by the router.

outDir

`DEFAULT ".svelte-kit"`

The directory that SvelteKit writes files to during `dev` and `build`. You should exclude this directory from version control.

output

DEFAULT `"modulepreload"`

AVAILABLE SINCE v1.8.4

SvelteKit will preload the JavaScript modules needed for the initial page to avoid import 'waterfalls', resulting in faster application startup. There are three strategies with different trade-offs:

- `modulepreload` - uses `<link rel="modulepreload">`. This delivers the best results in Chromium-based browsers, in Firefox 115+, and Safari 17+. It is ignored in older browsers.
- `preload-js` - uses `<link rel="preload">`. Prevents waterfalls in Chromium and Safari, but Chromium will parse each module twice (once as a script, once as a module). Causes modules to be requested twice in Firefox. This is a good setting if you want to maximise performance for users on iOS devices at the cost of a very slight degradation for Chromium users.
- `preload-mjs` - uses `<link rel="preload">` but with the `.mjs` extension which prevents double-parsing in Chromium. Some static webservers will fail to serve .mjs files with a `Content-Type: application/javascript` header, which will cause your application to break. If that doesn't apply to you, this is the option that will deliver the best performance for the largest number of users, until `modulepreload` is more widely supported.

paths

```
assets?: '' | `http://${string}` | `https://${string}`;
```

DEFAULT `""`

An absolute path that your app's files are served from. This is useful if your files are

DEFAULT `""`

A root-relative path that must start, but not end with `/` (e.g. `/base-path`), unless it is the empty string. This specifies where your app is served from and allows the app to live on a non-root path. Note that you need to prepend all your root-relative links with the base value or they will point to the root of your domain, not your `base` (this is how the browser works). You can use `base` from `$app/paths` for that: `<a href="{base}/your-page">Link</a>`. If you find yourself writing this often, it may make sense to extract this into a reusable component.

```
relative?: boolean;
```

DEFAULT `true`

AVAILABLE SINCE v1.9.0

Whether to use relative asset paths.

If `true`, `base` and `assets` imported from `$app/paths` will be replaced with relative asset paths during server-side rendering, resulting in more portable HTML. If `false`, `%sveltekit.assets%` and references to build artifacts will always be root-relative paths, unless `paths.assets` is an external URL

Single-page app fallback pages will always use absolute paths, regardless of this setting.

If your app uses a `<base>` element, you should set this to `false`, otherwise asset URLs will incorrectly be resolved against the `<base>` URL rather than the current page.

In 1.0, `undefined` was a valid value, which was set by default. In that case, if `paths.assets` was not external, SvelteKit would replace `%sveltekit.assets%` with a relative path and use relative paths to reference build artifacts, but `base` and `assets` imported from `$app/paths` would be as specified in your config.

```
concurrency?: number;
```

DEFAULT `1`

How many pages can be prerendered simultaneously. JS is single-threaded, but in cases where prerendering performance is network-bound (for example loading content from a remote CMS) this can speed things up by processing other tasks while waiting on the network response.

```
crawl?: boolean;
```

DEFAULT `true`

Whether SvelteKit should find pages to prerender by following links from `entries` .

```
entries?: Array<'*' | `/${string}`>;
```

DEFAULT `["*"]`

An array of pages to prerender, or start crawling from (if `crawl: true`). The `*` string includes all routes containing no required `[parameters]` with optional parameters included as being empty (since SvelteKit doesn't know what value any parameters should have).

```
handleHttpError?: PrerenderHttpErrorHandlerValue;
```

DEFAULT `"fail"`

AVAILABLE SINCE v1.15.7

How to respond to HTTP errors encountered while prerendering the app.

`(details) => void` — a custom error handler that takes a `details` object with `status`, `path`, `referrer`, `referenceType` and `message` properties. If you `throw` from this function, the build will fail

svelte.config.js

```js
/** @type {import('@sveltejs/kit').Config} */
const config = {
  kit: {
    prerender: {
      handleHttpError: ({ path, referrer, message }) => {
        // ignore deliberate link to shiny 404 page
        if (path === '/not-found' && referrer === '/blog/how-we-built-our-404-page') {
          return;
        }

        // otherwise fail the build
        throw new Error(message);
      }
    }
  }
};
```

```ts
handleMissingId?: PrerenderMissingIdHandlerValue;
```

DEFAULT `"fail"`

AVAILABLE SINCE v1.15.7

How to respond when hash links from one prerendered page to another don't correspond to an `id` on the destination page.

- `'fail'` — fail the build
- `'ignore'` - silently ignore the failure and continue
- `'warn'` — continue, but print a warning
- `(details) => void` — a custom error handler that takes a `details` object with

```
handleEntryGeneratorMismatch?: PrerenderEntryGeneratorMismatchHandlerValue;
```

DEFAULT `"fail"`

AVAILABLE SINCE v1.16.0

How to respond when an entry generated by the `entries` export doesn't match the route it was generated from.

- `'fail'` — fail the build
- `'ignore'` - silently ignore the failure and continue
- `'warn'` — continue, but print a warning
- `(details) => void` — a custom error handler that takes a `details` object with `generatedFromId`, `entry`, `matchedId` and `message` properties. If you `throw` from this function, the build will fail

```
origin?: string;
```

DEFAULT `"http://sveltekit-prerender"`

The value of `url.origin` during prerendering; useful if it is included in rendered content.

serviceWorker

```
register?: boolean;
```

DEFAULT `true`

Whether to automatically register the service worker, if it exists.

Docs

DEFAULT `(filename) => !/\.DS_Store/.test(filename)`

Determine which files in your `static` directory will be available in `$service-worker.files`.

typescript

```
config?: (config: Record<string, any>) => Record<string, any> | void;
```

DEFAULT `(config) => config`

AVAILABLE SINCE v1.3.0

A function that allows you to edit the generated `tsconfig.json`. You can mutate the config (recommended) or return a new one. This is useful for extending a shared `tsconfig.json` in a monorepo root, for example.

version

Client-side navigation can be buggy if you deploy a new version of your app while people are using it. If the code for the new page is already loaded, it may have stale content; if it isn't, the app's route manifest may point to a JavaScript file that no longer exists. SvelteKit helps you solve this problem through version management. If SvelteKit encounters an error while loading the page and detects that a new version has been deployed (using the `name` specified here, which defaults to a timestamp of the build) it will fall back to traditional full-page navigation. Not all navigations will result in an error though, for example if the JavaScript for the next page is already loaded. If you still want to force a full-page navigation in these cases, use techniques such as setting the `pollInterval` and then using `beforeNavigate`:

Docs

```
  import { beforeNavigate } from '$app/navigation';
  import { updated } from '$app/stores';

  beforeNavigate(({ willUnload, to }) => {
    if ($updated && !willUnload && to?.url) {
      location.href = to.url.href;
    }
  });
</script>
```

If you set `pollInterval` to a non-zero value, SvelteKit will poll for new versions in the background and set the value of the `updated` store to `true` when it detects one.

```
name?: string;
```

The current app version string. If specified, this must be deterministic (e.g. a commit ref rather than `Math.random()` or `Date.now().toString()`), otherwise defaults to a timestamp of the build.

For example, to use the current commit hash, you could do use `git rev-parse HEAD`:

svelte.config.js

```
import * as child_process from 'node:child_process';

export default {
  kit: {
    version: {
      name: child_process.execSync('git rev-parse HEAD').toString().trim()
    }
  }
};
```

```
pollInterval?: number;
```

DEFAULT `0`

Docs

PREVIOUS
$service-worker

NEXT
Command Line Interface