Fast Refresh is an implementation of Hot Reloading with full support from React. It replaces unofficial solutions like `react-hot-loader`.

With Fast Refresh, changes to the code for your React components immediately update in the browser, without losing component state. If you have errors in your code, the browser view displays an error overlay with more details about what went wrong. This direct feedback on your changes results in a better overall development experience (DX).

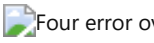
Fast Refresh is enabled by default in Gatsby's development server and should show your changes within a second.

How It Works

- If you edit a file that only exports React component(s), Fast Refresh will update the code only for that file, and re-render your component. You can edit anything in that file, including styles, rendering logic, event handlers, or effects.
- If you edit a file with exports that aren't React components, Fast Refresh will re-run both that file, and the other files importing it. So if both `Button.js` and `Modal.js` import `theme.js`, editing `theme.js` will update both components.
- Finally, if you edit a file that's imported by files outside of the React tree, Fast Refresh will fall back to doing a full reload. You might have a file which renders a React component but also exports a value that is imported by a non-React component. For example, maybe your component also exports a constant, and a non-React utility file imports it. In that case, consider migrating the constant to a separate file and importing it into both files. This will re-enable Fast Refresh to work. Other cases can usually be solved in a similar way.

Error Resilience

The Fast Refresh integration is good at catching errors instead of silently failing. Gatsby takes this opportunity to present you with a custom overlay that further explains the error and how to resolve it. Once it's resolved Fast Refresh will recover from that state. The overlay will indicate whether it's a compile error, GraphQL error, runtime error, or `getServerData` error.

 Four error overlays from left to right, top to bottom: Compile error, GraphQL errors, runtime errors, and getServerData error

Compile Errors

Compile errors originate from webpack and can't be dismissed. Those errors are blocking the compilation of the code itself, so you'll need to fix them in order to see the page. The Debugging HTML builds guide may also be helpful in some cases.

GraphQL Errors

GraphQL errors can occur due to a number of reasons, most often due to an error inside your page query or static query. In addition to an error message, you'll be given an error ID (e.g. `#85923`) that you can use to scan the documentation and internet for more information.

Runtime Errors

If the runtime error didn't occur during rendering, the component state will be retained. However, if the error happened during rendering, React will remount your application using the updated code.

If you have error boundaries in your application (which is a good idea for graceful failures in production), they will retry rendering on the next edit after a rendering error. This means having an error boundary can prevent you from always getting reset to the root app state. However, keep in mind that error boundaries shouldn't be too granular. They are used by React in production, and should always be designed intentionally.

`getServerData` Error

These errors originate from your `getServerData` function in your page and can't be dismissed. When using Server-Side Rendering (SSR) your code is run at runtime and will show up on your deployed site. Your own code can be faulty but also third-party requests, e.g. to an external API, thus it's a good idea to add error handling to your function.

Limitations

Fast Refresh tries to preserve local React state in the component you're editing, but only if it's safe to do so. There are a few reasons you might see local state being reset on every edit to a file:

- Local state is not preserved for class components. Only function components and hooks preserve state.
- The file you're editing might have other exports in addition to a React component. Gatsby includes an ESLint rule by default to warn against this pattern in pages: Page templates must only export one default export (the page) and `query` as a named export.
- Anonymous arrow functions like `export default () => <div />` cause Fast Refresh to not preserve local component state. Gatsby includes an ESLint rule by default to warn against this pattern.
- Sometimes a file will export the result of a higher-order component, like `HOC(WrappedComponent)`. If the returned component is a class or lowercase, state will be reset.
- Lowercase components like `function example() {}` cause Fast Refresh to not preserve local component state. Components and HOCs must be in `PascalCase`.

Tips

- Sometimes you want to force the state to be reset and a component to be remounted. For example, this can be handy if you're tweaking an animation that only happens on mount. To do this, you can add `// @refresh reset` anywhere in the file you're editing. This directive is local to the file, and instructs Fast Refresh to remount components defined in that file on every edit.

React Hooks

When possible, Fast Refresh attempts to preserve the state of your component between edits. In particular, `useState` and `useRef` preserve their previous values as long as you don't change their arguments or the order of the Hook calls.

Hooks with dependencies — such as `useEffect`, `useMemo`, and `useCallback` — will always update during Fast Refresh. Their list of dependencies will be ignored while Fast Refresh is happening.

For example, when you edit `useMemo(() => x * 2, [x])` to `useMemo(() => x * 10, [x])`, it will re-run even though `x` (the dependency) has not changed. If React didn't do that, your edit wouldn't reflect on the screen!

Sometimes, this can lead to unexpected results. For example, even a `useEffect` with an empty array of dependencies will re-run once during Fast Refresh.