# Fast Refresh

Fast Refresh is a React Native feature that allows you to get near-instant feedback for changes in your React components. Fast Refresh is enabled by default, and you can toggle "Enable Fast Refresh" in the React Native Dev Menu. With Fast Refresh enabled, most edits should be visible within a second or two.

## How It Works

- If you edit a module that **only exports React component(s)**, Fast Refresh will update the code only for that module, and re-render your component. You can edit anything in that file, including styles, rendering logic, event handlers, or effects.
- If you edit a module with exports that *aren't* React components, Fast Refresh will re-run both that module, and the other modules 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 modules 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 module 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

If you make a **syntax error** during a Fast Refresh session, you can fix it and save the file again. The redbox will disappear. Modules with syntax errors are prevented from running, so you won't need to reload the app.

If you make a **runtime error during the module initialization** (for example, typing `Style.create` instead of `StyleSheet.create`), the Fast Refresh session will continue once you fix the error. The redbox will disappear, and the module will be updated.

If you make a mistake that leads to a **runtime error inside your component**, the Fast Refresh session will *also* continue after you fix the error. In that case, React will remount your application using the updated code.

If you have error boundaries in your app (which is a good idea for graceful failures in production), they will retry rendering on the next edit after a redbox. In that sense, having an error boundary can prevent you from always getting kicked out to the root app screen. 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.

## Limitations

Fast Refresh tries to preserve local React state in the component you're editing, but only if it's safe to do so. Here's a few reasons why 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 module you're editing might have *other* exports in addition to a React component.
- Sometimes, a module would export the result of calling higher-order component like `createNavigationContainer(MyScreen)`. If the returned component is a class, state will be reset.

In the longer term, as more of your codebase moves to function components and Hooks, you can expect state to be preserved in more cases.

## Tips

- Fast Refresh preserves React local state in function components (and Hooks) by default.
- Sometimes you might 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.

## Fast Refresh and 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 would still re-run once during Fast Refresh. However, writing code resilient to an occasional re-running of `useEffect` is a good practice even without Fast Refresh. This makes it easier for you to later introduce new dependencies to it.

Is this page useful?

Edit this page

*Last updated on **Jun 21, 2023***