Version: 6.x

Troubleshooting

This section attempts to outline issues that users frequently encounter when first getting accustomed to using React Navigation. These issues may or may not be related to React Navigation itself.

Before troubleshooting an issue, make sure that you have upgraded to **the latest available versions** of the packages. You can install the latest versions by installing the packages again (e.g. `npm install package-name`).

I'm getting an error "Unable to resolve module" after updating to the latest version

This might happen for 3 reasons:

Stale cache of Metro bundler

If the module points to a local file (i.e. the name of the module starts with `./`), then it's probably due to stale cache. To fix this, try the following solutions.

If you're using Expo, run:

```
expo start -c
```

If you're not using Expo, run:

```
npx react-native start --reset-cache
```

If that doesn't work, you can also try the following:

```
rm -rf $TMPDIR/metro-bundler-cache-*
```

Missing peer dependency

If the module points to an npm package (i.e. the name of the module doesn't with `./`), then it's probably due to a missing dependency. To fix this, install the dependency in your project:

npm Yarn

```
npm install name-of-the-module
```

Sometimes it might even be due to a corrupt installation. If clearing cache didn't work, try deleting your `node_modules` folder and run `npm install` again.

Missing extensions in metro configuration

Sometimes the error may look like this:

```
Error: While trying to resolve module "@react-navigation/native" from file
"/path/to/src/App.js", the package "/path/to/node_modules/@react-
navigation/native/package.json" was successfully found. However, this package
itself specifies a "main" module field that could not be resolved
("/path/to/node_modules/@react-navigation/native/src/index.tsx"
```

This can happen if you have a custom configuration for metro and haven't specified `ts` and `tsx` as valid extensions. These extensions are present in the default configuration. To check if this is the issue, look for a `metro.config.js` file in your project and check if you have specified the `sourceExts` option. It should at least have the following configuration:

```
sourceExts: ['js', 'json', 'ts', 'tsx'];
```

If it's missing these extensions, add them and then clear metro cache as shown in the section above.

I'm getting "SyntaxError in @react-navigation/xxx/xxx.tsx" or "SyntaxError: /xxx/@react-navigation/xxx/xxx.tsx: Unexpected token"

This might happen if you have an old version of the `metro-react-native-babel-preset` package. Try upgrading it to the latest version.

npm Yarn

```
npm install --save-dev metro-react-native-babel-preset
```

If you have `@babel/core` installed, also upgrade it to latest version.

npm Yarn

```
npm install --save-dev @babel/core
```

If upgrading the packages don't help, you can also try deleting your `node_modules` as well as lock the file and reinstall your dependencies.

If you use `npm`:

```
rm -rf node_modules
rm package-lock.json
npm install
```

If you use `yarn`:

```
rm -rf node_modules
rm yarn.lock
yarn
```

After upgrading or reinstalling the packages, you should also clear Metro bundler's cache following the instructions earlier in the page.

I'm getting "Module '[...]' has no exported member 'xxx' when using TypeScript

This might happen if you have an old version of TypeScript in your project. You can try upgrading it:

npm Yarn

```
npm install --save-dev typescript
```

I'm getting an error "null is not an object (evaluating 'RNGestureHandlerModule.default.Direction')"

This and some similar errors might occur if you have a bare React Native project and the library `react-native-gesture-handler` library isn't linked.

Linking is automatic from React Native 0.60, so if you have linked the library manually, first unlink it:

```
react-native unlink react-native-gesture-handler
```

If you're testing on iOS and use Mac, make sure you have run `pod install` in the `ios/` folder:

```
cd ios
pod install
cd ..
```

Now rebuild the app and test on your device or simulator.

I'm getting an error "requireNativeComponent: "RNCSafeAreaProvider" was not found in the UIManager"

This and some similar errors might occur if you have a bare React Native project and the library `react-native-safe-area-context` library isn't linked.

Linking is automatic from React Native 0.60, so if you have linked the library manually, first unlink it:

```
react-native unlink react-native-safe-area-context
```

If you're testing on iOS and use Mac, make sure you have run `pod install` in the `ios/` folder:

```
cd ios
pod install
cd ..
```

Now rebuild the app and test on your device or simulator.

I'm getting an error "Tried to register two views with the same name RNCSafeAreaProvider"

This might occur if you have multiple versions of `react-native-safe-area-context` installed.

If you're using Expo managed workflow, it's likely that you have installed an incompatible version. To install the correct version, run:

```
npx expo install react-native-safe-area-context
```

If it didn't fix the error or you're not using Expo managed workflow, you'll need to check which package depends on a different version of `react-native-safe-area-context`.

If you use `yarn`, run:

```
yarn why react-native-safe-area-context
```

If you use `npm`, run:

```
npm ls react-native-safe-area-context
```

This will tell you if a package you use has a dependency on `react-native-safe-area-context`. If it's a third-party package, you should open an issue on the relevant repo's issue tracker explaining the problem. Generally for libraries, dependencies containing native code should be defined in `peerDependencies` instead of `dependencies` to avoid such issues.

If it's already in `peerDependencies` and not in `dependencies`, and you use `npm`, it might be because of incompatible version range defined for the package. The author of the library will need to relax the version range in such cases to allow a wider range of versions to be installed.

If you use `yarn`, you can also temporarily override the version being installed using `resolutions`. Add the following in your `package.json`:

```
"resolutions": {
  "react-native-safe-area-context": "<version you want to use>"
}
```

And then run:

```
yarn
```

If you're on iOS and not using Expo managed workflow, also run:

```
cd ios
pod install
cd ..
```

Now rebuild the app and test on your device or simulator.

Nothing is visible on the screen after adding a `View`

If you wrap the container in a `View`, make sure the `View` stretches to fill the container using `flex: 1`:

```
import * as React from 'react';
import { View } from 'react-native';
import { NavigationContainer } from '@react-navigation/native';

export default function App() {
  return (
    <View style={{ flex: 1 }}>
      <NavigationContainer>{/* ... */}</NavigationContainer>
    </View>
  );
}
```

I get the warning "Non-serializable values were found in the navigation state"

This can happen if you are passing non-serializable values such as class instances, functions etc. in params. React Navigation warns you in this case because this can break other functionality such state persistence, deep linking etc.

Example of some use cases for passing functions in params are the following:

- To pass a callback to use in a header button. This can be achieved using `navigation.setOptions` instead. See the guide for header buttons for examples.
- To pass a callback to the next screen which it can call to pass some data back. You can usually achieve it using `navigate` instead. See the guide for params for examples.
- To pass complex data to another screen. Instead of passing the data `params`, you can store that complex data somewhere else (like a global store), and pass an id instead. Then the screen can get the data from the global store using the id. See what should be in params.
- Pass data, callbacks etc. from a parent to child screens. You can either use React Context, or pass a children callback to pass these down instead of using params. See passing additional props.

If you don't use state persistence or deep link to the screen which accepts functions in params, then the warning doesn't affect you and you can safely ignore it. To ignore the warning, you can use `LogBox.ignoreLogs`.

Example:

```
import { LogBox } from 'react-native';

LogBox.ignoreLogs([
  'Non-serializable values were found in the navigation state',
]);
```

I'm getting "Invalid hook call. Hooks can only be called inside of the body of a function component"

This can happen when you pass a React component to an option that accepts a function returning a react element. For example, the `headerTitle` option in native stack navigator expects a function returning a react element:

```
<Stack.Screen
  name="Home"
  component={Home}
  option={{ headerTitle: (props) => <MyTitle {...props} /> }}
/>
```

If you directly pass a function here, you'll get this error when using hooks:

```
<Stack.Screen
  name="Home"
  component={Home}
  option={{
    // This is not correct
    headerTitle: MyTitle,
  }}
/>
```

The same applies to other options like `headerLeft`, `headerRight`, `tabBarIcon` etc. as well as props such as `tabBar`, `drawerContent` etc.

Screens are unmounting/remounting during navigation

Sometimes you might have noticed that your screens unmount/remount, or your local component state or the navigation state resets when you navigate. This might happen if you are creating React components during render.

The simplest example is something like following:

```
function App() {
  return (
    <Stack.Navigator>
      <Stack.Screen
        name="Home"
        component={() => {
          return <SomeComponent />;
        }}
      />
    </Stack.Navigator>
  );
}
```

The `component` prop expects a React Component, but in the example, it's getting a function returning an React Element. While superficially a component and a function returning a React Element look the exact same, they don't behave the same way when used.

Here, every time the component re-renders, a new function will be created and passed to the `component` prop. React will see a new component and unmount the previous component before rendering the new one. This will cause any local state in the old component to be lost. React Navigation will detect and warn for this specific case but there can be other ways you might be creating components during render which it can't detect.

Another easy to identify example of this is when you create a component inside another component:

```
function App() {
  const Home = () => {
    return <SomeComponent />;
  };

  return (
    <Stack.Navigator>
```

```
      <Stack.Screen name="Home" component={Home} />
    </Stack.Navigator>
  );
}
```

Or when you use a higher order component (such as `connect` from Redux, or `withX` functions that accept a component) inside another component:

```
function App() {
  return (
    <Stack.Navigator>
      <Stack.Screen name="Home" component={withSomeData(Home)} />
    </Stack.Navigator>
  );
}
```

If you're unsure, it's always best to make sure that the components you are using as screens are defined outside of a React component. They could be defined in another file and imported, or defined at the top level scope in the same file:

```
const Home = () => {
  return <SomeComponent />;
};

function App() {
  return (
    <Stack.Navigator>
      <Stack.Screen name="Home" component={Home} />
    </Stack.Navigator>
  );
}
```

This is not React Navigation specific, but related to React in general. You should always avoid creating components during render, whether you are using React Navigation or not.

App is not working properly when connected to Chrome Debugger

When the app is connected to Chrome Debugger (or other tools that use Chrome Debugger such as React Native Debugger) you might encounter various issues related to timing.

This can result in issues such as button presses taking a long time to register or not working at all, gestures and animations being slow and buggy etc. There can be other functional issues such as promises not resolving, timeouts and intervals not working correctly etc. as well.

The issues are not related to React Navigation, but due to the nature of how the Chrome Debugger works. When connected to Chrome Debugger, your whole app runs on Chrome and communicates with the native app via sockets over the network, which can introduce latency and timing related issues.

So, unless you are trying to debug something, it's better to test the app without being connected to the Chrome Debugger. If you are using iOS, you can alternatively use Safari to debug your app which debugs the app on the device directly and does not have these issues, though it has other downsides.

Edit this page