Version: 6.x

Navigating without the navigation prop

Sometimes you need to trigger a navigation action from places where you do not have access to the `navigation` prop, such as a Redux middleware. For such cases, you can dispatch navigation actions use a `ref` on the navigation container.

Do not use the `ref` if:

- You need to navigate from inside a component without needing to pass the `navigation` prop down, see `useNavigation` instead. The `ref` behaves differently, and many helper methods specific to screens aren't available.
- You need to handle deep links or universal links. Doing this with the `ref` has many edge cases. See configuring links for more information on handling deep linking.
- You need to integrate with third party libraries, such as push notifications, branch etc. See third party integrations for deep linking instead.

Do use the `ref` if:

- You use a state management library such as Redux, where you need to dispatch navigation actions from a middleware.

Note that it's usually better to trigger navigation from user actions such as button presses, rather than from a Redux middleware. Navigating on user action makes the app feel more responsive and provides better UX. So consider this before using the `ref` for navigation. The `ref` is an escape hatch for scenarios that can't be handled with the existing APIs and should only be used in rare situations.

Usage

You can get access to the root navigation object through a `ref` and pass it to the `RootNavigation` which we will later use to navigate.

```
// App.js

import { NavigationContainer } from '@react-navigation/native';
import { navigationRef } from './RootNavigation';

export default function App() {
  return (
    <NavigationContainer ref={navigationRef}>{/* ... */}</NavigationContainer>
  );
}
```

In the next step, we define `RootNavigation`, which is a simple module with functions that dispatch user-defined navigation actions.

```
// RootNavigation.js

import { createNavigationContainerRef } from '@react-navigation/native';

export const navigationRef = createNavigationContainerRef()

export function navigate(name, params) {
  if (navigationRef.isReady()) {
    navigationRef.navigate(name, params);
  }
}

// add other navigation functions that you need and export them
```

Then, in any of your javascript modules, import the `RootNavigation` and call functions which you exported from it. You may use this approach outside of your React components and, in fact, it works as well when used from within them.

```
// any js module
import * as RootNavigation from './path/to/RootNavigation.js';

// ...

RootNavigation.navigate('ChatScreen', { userName: 'Lucy' });
```

Try this example on Snack

Apart from `navigate`, you can add other navigation actions:

```
import { StackActions } from '@react-navigation/native';

// ...

export function push(...args) {
  if (navigationRef.isReady()) {
    navigationRef.dispatch(StackActions.push(...args));
  }
}
```

Note that a stack navigators needs to be rendered to handle this action. You may want to check the docs for nesting for more details.

When writing tests, you may mock the navigation functions, and make assertions on whether the correct functions are called with the correct parameters.

Handling initialization

When using this pattern, you need to keep few things in mind to avoid navigation from failing in your app.

- The `ref` is set only after the navigation container renders, this can be async when handling deep links
- A navigator needs to be rendered to be able to handle actions, the `ref` won't be ready without a navigator

If you try to navigate without rendering a navigator or before the navigator finishes mounting, it will print an error and do nothing. So you'll need to add an additional check to decide what to do until your app mounts.

For an example, consider the following scenario, you have a screen somewhere in the app, and that screen dispatches a redux action on `useEffect`/`componentDidMount`. You are listening for this action in your middleware and try to perform navigation when you get it. This will throw an error,

because by this time, the parent navigator hasn't finished mounting and isn't ready. Parent's `useEffect`/`componentDidMount` is always called **after** child's `useEffect`/`componentDidMount`.

To avoid this, you can use the `isReady()` method available on the ref as shown in the above examples.

```
// RootNavigation.js

import * as React from 'react';

export const navigationRef = createNavigationContainerRef()

export function navigate(name, params) {
  if (navigationRef.isReady()) {
    // Perform navigation if the react navigation is ready to handle actions
    navigationRef.navigate(name, params);
  } else {
    // You can decide what to do if react navigation is not ready
    // You can ignore this, or add these actions to a queue you can call later
  }
}
```

Try this example on Snack

If you're unsure if a navigator is rendered, you can call `navigationRef.current.getRootState()`, and it'll return a valid state object if any navigators are rendered, otherwise it will return `undefined`.

Edit this page