Version: 6.x

Screen options with nested navigators

In this document we'll explain how screen options work when there are multiple navigators. It's important to understand this so that you put your `options` in the correct place and can properly configure your navigators. If you put them in the wrong place, at best nothing will happen and at worst something confusing and unexpected will happen.

You can only modify navigation options for a navigator from one of its screen components. This applies equally to navigators that are nested as screens.

Let's take for example a tab navigator that contains a native stack in each tab. What happens if we set the `options` on a screen inside of the stack?

```
const Tab = createTabNavigator();
const HomeStack = createNativeStackNavigator();
const SettingsStack = createNativeStackNavigator();

function HomeStackScreen() {
  return (
    <HomeStack.Navigator>
      <HomeStack.Screen
        name="A"
        component={A}
        options={{ tabBarLabel: 'Home!' }}
      />
    </HomeStack.Navigator>
  );
}

function SettingsStackScreen() {
  return (
    <SettingsStack.Navigator>
      <SettingsStack.Screen
        name="B"
        component={B}
        options={{ tabBarLabel: 'Settings!' }}
      />
```

```
    </SettingsStack.Navigator>
  );
}

export default function App() {
  return (
    <NavigationContainer>
      <Tab.Navigator>
        <Tab.Screen name="Home" component={HomeStackScreen} />
        <Tab.Screen name="Settings" component={SettingsStackScreen} />
      </Tab.Navigator>
    </NavigationContainer>
  );
}
```

Try this example on Snack

As we mentioned earlier, you can only modify navigation options for a navigator from one of its screen components. `A` and `B` above are screen components in `HomeStack` and `SettingsStack` respectively, not in the tab navigator. So the result will be that the `tabBarLabel` property is not applied to the tab navigator. We can fix this though!

```
export default function App() {
  return (
    <NavigationContainer>
      <Tab.Navigator>
        <Tab.Screen
          name="Home"
          component={HomeStackScreen}
          options={{ tabBarLabel: 'Home!' }}
        />
        <Tab.Screen
          name="Settings"
          component={SettingsStackScreen}
          options={{ tabBarLabel: 'Settings!' }}
        />
      </Tab.Navigator>
    </NavigationContainer>
  );
}
```

Try this example on Snack

When we set the `options` directly on `Screen` components containing the `HomeStack` and `SettingsStack` component, it allows us to control the options for its parent navigator when its used as a screen component. In this case, the options on our stack components configure the label in the tab navigator that renders the stacks.

Setting parent screen options based on child navigator's state

Imagine the following configuration:

```
const Tab = createBottomTabNavigator();

function HomeTabs() {
  return (
    <Tab.Navigator>
      <Tab.Screen name="Feed" component={FeedScreen} />
      <Tab.Screen name="Profile" component={ProfileScreen} />
      <Tab.Screen name="Account" component={AccountScreen} />
    </Tab.Navigator>
  );
}

const Stack = createNativeStackNavigator();

export default function App() {
  return (
    <NavigationContainer>
      <Stack.Navigator>
        <Stack.Screen name="Home" component={HomeTabs} />
        <Stack.Screen name="Settings" component={SettingsScreen} />
      </Stack.Navigator>
    </NavigationContainer>
  );
}
```

Try this example on Snack

If we were to set the `headerTitle` with `options` for the `FeedScreen`, this would not work. This is because `App` stack will only look at its immediate children for configuration: `HomeTabs` and `SettingsScreen`.

But we can determine the `headerTitle` option based on the navigation state of our tab navigator using the `getFocusedRouteNameFromRoute` helper. Let's create a function to get the title first:

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

function getHeaderTitle(route) {
  // If the focused route is not found, we need to assume it's the initial
screen
  // This can happen during if there hasn't been any navigation inside the
screen
  // In our case, it's "Feed" as that's the first screen inside the navigator
  const routeName = getFocusedRouteNameFromRoute(route) ?? 'Feed';

  switch (routeName) {
    case 'Feed':
      return 'News feed';
    case 'Profile':
      return 'My profile';
    case 'Account':
      return 'My account';
  }
}
```

Then we can use this function with the `options` prop on `Screen`:

```
<Stack.Screen
  name="Home"
  component={HomeTabs}
  options={({ route }) => ({
    headerTitle: getHeaderTitle(route),
  })}
/>
```

Try this example on Snack

So what's happening here? With the `getFocusedRouteNameFromRoute` helper, we can get the currently active route name from this child navigator (in this case it's the tab navigator since that's what we're rendering) and setting an appropriate title for the header.

This approach can be used anytime you want to set options for a parent navigator based on a child navigator's state. Common use cases are:

1. Show tab title in stack header: a stack contains a tab navigator and you want to set the title on the stack header (above example)
2. Show screens without tab bar: a tab navigator contains a stack and you want to hide the tab bar on specific screens (not recommended, see Hiding tab bar in specific screens instead)
3. Lock drawer on certain screens: a drawer has a stack inside of it and you want to lock the drawer on certain screens

In many cases, similar behavior can be achieved by reorganizing our navigators. We usually recommend this option if it fits your use case.

For example, for the above use case, instead of adding a tab navigator inside a stack navigator, we can add a stack navigator inside each of the tabs.

```
const FeedStack = createNativeStackNavigator();

function FeedStackScreen() {
  return (
    <FeedStack.Navigator>
      <FeedStack.Screen name="Feed" component={FeedScreen} />
      {/* other screens */}
    </FeedStack.Navigator>
  );
}

const ProfileStack = createNativeStackNavigator();

function ProfileStackScreen() {
  return (
    <ProfileStack.Navigator>
      <ProfileStack.Screen name="Profile" component={ProfileScreen} />
      {/* other screens */}
    </ProfileStack.Navigator>
  );
}
```

```
const Tab = createBottomTabNavigator();

function HomeTabs() {
  return (
    <Tab.Navigator>
      <Tab.Screen name="Feed" component={FeedStackScreen} />
      <Tab.Screen name="Profile" component={ProfileStackScreen} />
    </Tab.Navigator>
  );
}

const RootStack = createNativeStackNavigator();

export default function App() {
  return (
    <NavigationContainer>
      <RootStack.Navigator>
        <RootStack.Screen name="Home" component={HomeTabs} />
        <RootStack.Screen name="Settings" component={SettingsScreen} />
      </RootStack.Navigator>
    </NavigationContainer>
  );
}
```

Try this example on Snack

Additionally, this lets you push new screens to the feed and profile stacks without hiding the tab bar by adding more routes to those stacks.

If you want to push screens on top of the tab bar (i.e. that don't show the tab bar), then you can add them to the `App` stack instead of adding them into the screens inside the tab navigator.

Edit this page