Version: 6.x

Screen

`Screen` components are used to configure various aspects of screens inside a navigator.

A `Screen` is returned from a `createXNavigator` function:

```
const Stack = createNativeStackNavigator(); // Stack contains Screen & Navigator
properties
```

After creating the navigator, it can be used as children of the `Navigator` component:

```
<Stack.Navigator>
  <Stack.Screen name="Home" component={HomeScreen} />
  <Stack.Screen name="Profile" component={ProfileScreen} />
</Stack.Navigator>
```

You need to provide at least a name and a component to render for each screen.

Props

`name`

The name to use for the screen. It accepts a string:

```
<Stack.Screen name="Profile" component={ProfileScreen} />
```

This name is used to navigate to the screen:

```
navigation.navigate('Profile');
```

It is also used for the `name` property in the `route`.

While it is supported, we recommend to avoid spaces or special characters in screen names and keep them simple.

`options`

Options to configure how the screen gets presented in the navigator. It accepts either an object or a function returning an object:

```
<Stack.Screen
  name="Profile"
  component={ProfileScreen}
  options={{
    title: 'Awesome app',
  }}
/>
```

When you pass a function, it'll receive the `route` and `navigation`:

```
<Stack.Screen
  name="Profile"
  component={ProfileScreen}
  options={({ route, navigation }) => ({
    title: route.params.userId,
  })}
/>
```

See Options for screens for more details and examples.

`initialParams`

Initial params to use for the screen. If a screen is used as `initialRouteName`, it'll contain the params from `initialParams`. If you navigate to a new screen, the params passed are shallow merged with the initial params.

```
<Stack.Screen
  name="Details"
  component={DetailsScreen}
  initialParams={{ itemId: 42 }}
/>
```

`getId`

Callback to return an unique ID to use for the screen. It receives an object with the route params:

```
<Stack.Screen
  name="Profile"
  component={ProfileScreen}
  getId={({ params }) => params.userId}
/>
```

By default, `navigate('ScreenName', params)` identifies the screen by its name. So if you're on `ScreenName` and navigate to `ScreenName` again, it won't add a new screen even if the params are different - it'll update the current screen with the new params instead:

```
// Let's say you're on `Home` screen
// Then you navigate to `Profile` screen with `userId: 1`
navigation.navigate('Profile', { userId: 1 });

// Now the stack will have: `Home` -> `Profile` with `userId: 1`

// Then you navigate to `Profile` screen again with `userId: 2`
navigation.navigate('Profile', { userId: 2 });

// The stack will now have: `Home` -> `Profile` with `userId: 2`
```

If you specify `getId` and it doesn't return `undefined`, the screen is identified by both the screen name and the returned ID. Which means that if you're on `ScreenName` and navigate to `ScreenName` again with different params - and return a different ID from the `getId` callback, it'll add a new screen to the stack:

```
// Let's say you're on `Home` screen
// Then you navigate to `Profile` screen with `userId: 1`
navigation.navigate('Profile', { userId: 1 });

// Now the stack will have: `Home` -> `Profile` with `userId: 1`
// Then you navigate to `Profile` screen again with `userId: 2`
navigation.navigate('Profile', { userId: 2 });

// The stack will now have: `Home` -> `Profile` with `userId: 1` -> `Profile`
with `userId: 2`
```

The `getId` callback can also be used to ensure that the screen with the same ID doesn't appear multiple times in the stack:

```
// Let's say you have a stack with the screens: `Home` -> `Profile` with
`userId: 1` -> `Settings`
// Then you navigate to `Profile` screen with `userId: 1` again
navigation.navigate('Profile', { userId: 1 });

// Now the stack will have: `Home` -> `Profile` with `userId: 1`
```

In the above examples, `params.userId` is used as an ID, subsequent navigation to the screen with the same `userId` will navigate to the existing screen instead of adding a new one to the stack. If the navigation was with a different `userId`, then it'll add a new screen.

If `getId` is specified in a tab or drawer navigator, the screen will remount if the ID changes.

`component`

The React Component to render for the screen:

```
<Stack.Screen name="Profile" component={ProfileScreen} />
```

`getComponent`

Callback to return the React Component to render for the screen:

```
<Stack.Screen
  name="Profile"
  getComponent={() => require('./ProfileScreen').default}
/>
```

You can use this approach instead of the `component` prop if you want the `ProfileScreen` module to be lazily evaluated when needed. This is especially useful when using ram bundles to improve initial load.

`children`

Render callback to return React Element to use for the screen:

```
<Stack.Screen name="Profile">
  {(props) => <ProfileScreen {...props} />}
</Stack.Screen>
```

You can use this approach instead of the `component` prop if you need to pass additional props. Though we recommend using React context for passing data instead.

> Note: By default, React Navigation applies optimizations to screen components to prevent unnecessary renders. Using a render callback removes those optimizations. So if you use a render callback, you'll need to ensure that you use `React.memo` or `React.PureComponent` for your screen components to avoid performance issues.

`navigationKey`

Optional key for this screen. This doesn't need to be unique. If the key changes, existing screens with this name will be removed (if used in a stack navigator) or reset (if used in a tab or drawer navigator).

This can be useful when we have some screens which we want to be removed or reset when the condition changes:

```
<Stack.Screen
  navigationKey={isSignedIn ? 'user' : 'guest'}
  name="Profile"
  component={ProfileScreen}
/>
```

`listeners`

Event listeners to subscribe to. See `listeners` prop on `Screen` for more details.

Edit this page