



Version: 6.x

Navigation events

You can listen to various events emitted by React Navigation to get notified of certain events, and in some cases, override the default action. There are few core events such as `focus`, `blur` etc. (documented below) that work for every navigator, as well as navigator specific events that work only for certain navigators.

Apart from the core events, each navigator can emit their own custom events. For example, stack navigator emits `transitionStart` and `transitionEnd` events, tab navigator emits `tabPress` event etc. You can find details about the events emitted on the individual navigator's documentation.

Core events

Following are the events available in every navigator:

`focus`

This event is emitted when the screen comes into focus.

For most cases, the `useFocusEffect` hook might be appropriate than adding the listener manually. See [this guide](#) for more details to decide which API you should use.

`blur`

This event is emitted when the screen goes out of focus.

`state`

This event is emitted when the navigator's state changes. This event receives the navigator's state in the event data (`event.data.state`).

`beforeRemove`

This event is emitted when the user is leaving the screen, there's a chance to prevent the user from leaving.

Listening to events

There are multiple ways to listen to events from the navigators. Each callback registered as an event listener receives an event object as its argument. The event object contains few properties:

- `data` - Additional data regarding the event passed by the navigator. This can be `undefined` if no data was passed.
- `target` - The route key for the screen that should receive the event. For some events, this maybe `undefined` if the event wasn't related to a specific screen.
- `preventDefault` - For some events, there may be a `preventDefault` method on the event object. Calling this method will prevent the default action performed by the event (such as switching tabs on `tabPress`). Support for preventing actions are only available for certain events like `tabPress` and won't work for all events.

You can listen to events with the following APIs:

`navigation.addListener`

Inside a screen, you can add listeners on the `navigation` prop with the `addListener` method. The `addListener` method takes 2 arguments: type of the event, and a callback to be called on the event. It returns a function that can be called to unsubscribe from the event.

Example:

```
const unsubscribe = navigation.addListener('tabPress', (e) => {  
  // Prevent default action  
  e.preventDefault();  
});
```

Normally, you'd add an event listener in `React.useEffect` for function components. For example:

```
function Profile({ navigation }) {  
  React.useEffect(() => {
```

```
const unsubscribe = navigation.addListener('focus', () => {  
  // do something  
});  
  
return unsubscribe;  
}, [navigation]);  
  
return <ProfileContent />;  
}
```

Try this example on Snack [↗](#)

The `unsubscribe` function can be returned as the cleanup function in the effect.

For class components, you can add the event in the `componentDidMount` lifecycle method and unsubscribe in `componentWillUnmount`:

```
class Profile extends React.Component {  
  componentDidMount() {  
    this._unsubscribe = navigation.addListener('focus', () => {  
      // do something  
    });  
  }  
  
  componentWillUnmount() {  
    this._unsubscribe();  
  }  
  
  render() {  
    // Content of the component  
  }  
}
```

One thing to keep in mind is that you can only listen to events from the immediate navigator with `addListener`. For example, if you try to add a listener in a screen that's inside a stack that's nested in a tab, it won't get the `tabPress` event. If you need to listen to an event from a parent navigator, you may use `navigation.getParent` to get a reference to parent navigator's navigation prop and add a listener.

```
const unsubscribe = navigation
  .getParent('MyTabs')
  .addListener('tabPress', (e) => {
    // Do something
  });
```

Here `'MyTabs'` refers to the value you pass in the `id` prop of the parent `Tab.Navigator` whose event you want to listen to.

listeners prop on Screen

Sometimes you might want to add a listener from the component where you defined the navigator rather than inside the screen. You can use the `listeners` prop on the `Screen` component to add listeners. The `listeners` prop takes an object with the event names as keys and the listener callbacks as values.

Example:

```
<Tab.Screen
  name="Chat"
  component={Chat}
  listeners={{
    tabPress: (e) => {
      // Prevent default action
      e.preventDefault();
    },
  }}
/>
```

You can also pass a callback which returns the object with listeners. It'll receive `navigation` and `route` as the arguments.

Example:

```
<Tab.Screen
  name="Chat"
  component={Chat}
  listeners={({ navigation, route }) => ({
```

```
tabPress: (e) => {  
  // Prevent default action  
  e.preventDefault();  
  
  // Do something with the `navigation` object  
  navigation.navigate('AnotherPlace');  
},  
}}}  
</>
```

screenListeners prop on the navigator

You can pass a prop named `screenListeners` to the navigator component, where you can specify listeners for events from all screens for this navigator. This can be useful if you want to listen to specific events regardless of the screen, or want to listen to common events such as `state` which is emitted to all screens.

Example:


```
<Stack.Navigator  
  screenListeners={{  
    state: (e) => {  
      // Do something with the state  
      console.log('state changed', e.data);  
    },  
  }}  
>  
  <Stack.Screen name="Home" component={HomeScreen} />  
  <Stack.Screen name="Profile" component={ProfileScreen} />  
</Stack.Navigator>
```

Similar to `listeners`, you can also pass a function to `screenListeners`. The function will receive the `navigation` prop and the `route` prop for each screen. This can be useful if you need access to the `navigation` object.

```
<Tab.Navigator  
  screenListeners={({ navigation }) => ({  
    state: (e) => {  
      // Do something with the state
```

```
    console.log('state changed', e.data);

    // Do something with the `navigation` object
    if (!navigation.canGoBack()) {
      console.log("we're on the initial screen");
    }
  },
  }}}
>
<Tab.Screen name="Home" component={HomeScreen} />
<Tab.Screen name="Profile" component={ProfileScreen} />
</Tab.Navigator>
```

 [Edit this page](#)