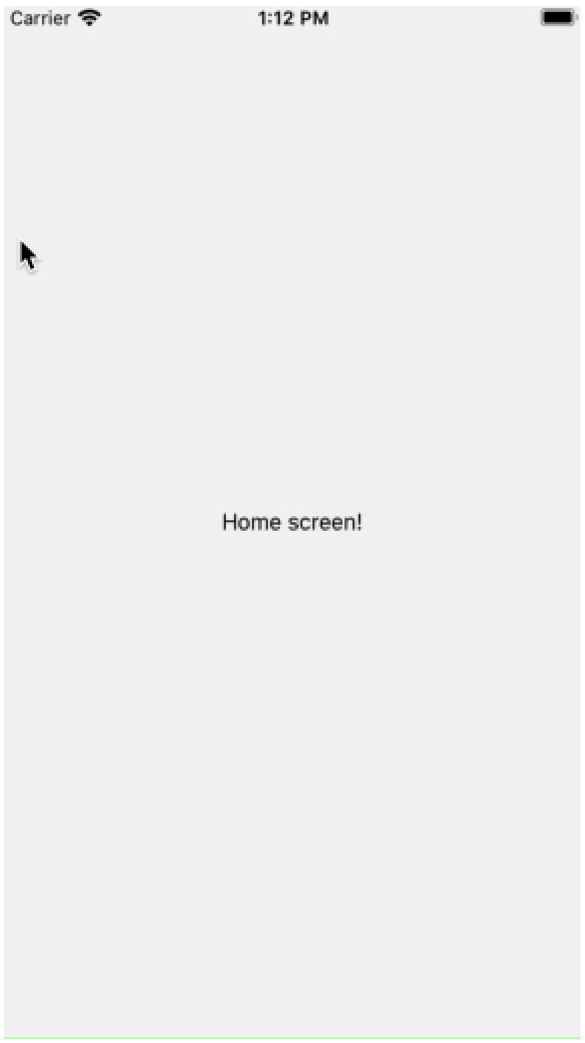
Libraries Drawer Layout

**Version: 6.x**

# React Native Drawer Layout

A cross-platform Drawer component for React Native implemented using `react-native-gesture-handler` and `react-native-reanimated`.



This package doesn't integrate with React Navigation. If you want to integrate the drawer layout with React Navigation's navigation system, e.g. want to show screens in the drawer and be able to navigate between them using `navigation.navigate` etc, use Drawer Navigator instead.

## Installation

To use this package, open a Terminal in the project root and run:

**npm** Yarn

```
npm install react-native-drawer-layout
```

Then, you need to install and configure the libraries that are required by the drawer:

1. First, install `react-native-gesture-handler` and `react-native-reanimated`.

   If you have a Expo managed project, in your project directory, run:

   ```
   npx expo install react-native-gesture-handler react-native-reanimated
   ```

   If you have a bare React Native project, in your project directory, run:

   **npm** Yarn

   ```
   npm install react-native-gesture-handler react-native-reanimated
   ```

   The Drawer supports both Reanimated 1 and Reanimated 2. If you want to use Reanimated 2, make sure to configure it following the installation guide.

2. To finalize installation of `react-native-gesture-handler`, add the following at the **top** (make sure it's at the top and there's nothing else before it) of your entry file, such as `index.js` or `App.js`:

   ```
   import 'react-native-gesture-handler';
   ```

   > Note: If you are building for Android or iOS, do not skip this step, or your app may crash in production even if it works fine in development. This is not applicable to other platforms.

3. If you're on a Mac and developing for iOS, you also need to install the pods (via Cocoapods) to complete the linking.

```
npx pod-install ios
```

We're done! Now you can build and run the app on your device/simulator.

## Quick start

```
import * as React from 'react';
import { Button, Text } from 'react-native';
import { Drawer } from 'react-native-drawer-layout';

export default function DrawerExample() {
  const [open, setOpen] = React.useState(false);

  return (
    <Drawer
      open={open}
      onOpen={() => setOpen(true)}
      onClose={() => setOpen(false)}
      renderDrawerContent={() => {
        return <Text>Drawer content</Text>;
      }}
    >
      <Button
        onPress={() => setOpen((prevOpen) => !prevOpen)}
        title={`${open ? 'Close' : 'Open'} drawer`}
      />
    </Drawer>
  );
}
```

## API reference

The package exports a `Drawer` component which is the one you'd use to render the drawer.

### `Drawer`

Component responsible for rendering a drawer with animations and gestures.

### Drawer Props

#### `open`

Whether the drawer is open or not.

#### `onOpen`

Callback which is called when the drawer is opened.

#### `onClose`

Callback which is called when the drawer is closed.

#### `renderDrawerContent`

Callback which returns a react element to render as the content of the drawer.

#### `layout`

Object containing the layout of the container. Defaults to the dimensions of the application's window.

#### `drawerPosition`

Position of the drawer on the screen. Defaults to `right` in RTL mode, otherwise `left`.

#### `drawerType`

Type of the drawer. It determines how the drawer looks and animates.

- `front`: Traditional drawer which covers the screen with a overlay behind it.
- `back`: The drawer is revealed behind the screen on swipe.
- `slide`: Both the screen and the drawer slide on swipe to reveal the drawer.
- `permanent`: A permanent drawer is shown as a sidebar.

Defaults to `slide` on iOS and `front` on other platforms.

#### `drawerStyle`

Style object for the drawer. You can pass a custom background color for drawer or a custom width for the drawer.

#### `overlayStyle`

Style object for the overlay.

#### `hideStatusBarOnOpen`

Whether to hide the status bar when the drawer is open. Defaults to `false`.

#### `keyboardDismissMode`

Whether to dismiss the keyboard when the drawer is open. Supported values are:

- `none`: The keyboard will not be dismissed when the drawer is open.
- `on-drag`: The keyboard will be dismissed when the drawer is opened by a swipe gesture.

Defaults to `on-drag`.

#### `statusBarAnimation`

Animation to use when the status bar is hidden. Supported values are:

- `slide`: The status bar will slide out of view.
- `fade`: The status bar will fade out of view.
- `none`: The status bar will not animate.

Use it in combination with `hideStatusBarOnOpen`.

#### `swipeEnabled`

Whether to enable swipe gestures to open the drawer. Defaults to `true`.

Swipe gestures are only supported on iOS and Android.

#### `swipeEdgeWidth`

How far from the edge of the screen the swipe gesture should activate. Defaults to `32`.

This is only supported on iOS and Android.

#### `swipeMinDistance`

Minimum swipe distance that should activate opening the drawer. Defaults to `60`.

This is only supported on iOS and Android.

#### `swipeMinVelocity`

Minimum swipe velocity that should activate opening the drawer. Defaults to `500`.

This is only supported on iOS and Android.

#### `gestureHandlerProps`

Props to pass to the underlying pan gesture handler.

This is only supported on iOS and Android.

#### `children`

Content that the drawer should wrap.

## `useDrawerProgress`

The `useDrawerProgress` hook returns a Reanimated SharedValue (with modern implementation) or Reanimated Node (with legacy implementation) which represents the progress of the drawer. It can be used to animate the content of the screen.

Example with modern implementation:

```
import { Animated } from 'react-native-reanimated';
import { useDrawerProgress } from 'react-native-drawer-layout';

// ...

function MyComponent() {
  const progress = useDrawerProgress();

  const animatedStyle = useAnimatedStyle(() => {
    return {
      transform: [
        {
          translateX: interpolate(progress, [0, 1], [-100, 0]),
        },
      ],
    };
  });

  return <Animated.View style={animatedStyle}>{/* ... */}</Animated.View>;
}
```

Example with legacy implementation:

```
import { Animated } from 'react-native-reanimated';
import { useDrawerProgress } from 'react-native-drawer-layout';

// ...

function MyComponent() {
  const progress = useDrawerProgress();

  // If you are on react-native-reanimated 1.x, use `Animated.interpolate`
instead of `Animated.interpolateNode`
  const translateX = Animated.interpolateNode(progress, {
    inputRange: [0, 1],
    outputRange: [-100, 0],
  });

  return (
    <Animated.View style={{ transform: [{ translateX }] }}>
      {/* ... */}
    </Animated.View>
  );
}
```

If you are using class components, you can use the `DrawerProgressContext` to get the progress value.

```
import { DrawerProgressContext } from 'react-native-drawer-layout';

// ...
```

```
class MyComponent extends React.Component {
  static contextType = DrawerProgressContext;

  render() {
    const progress = this.context;

    // ...
  }
}
```

Edit this page