Navigators Native Stack

Version: 6.x

Native Stack Navigator

Native Stack Navigator provides a way for your app to transition between screens where each new screen is placed on top of a stack.

This navigator uses the native APIs `UINavigationController` on iOS and `Fragment` on Android so that navigation built with `createNativeStackNavigator` will behave exactly the same and have the same performance characteristics as apps built natively on top of those APIs. It also offers basic Web support using `react-native-web`.

One thing to keep in mind is that while `@react-navigation/native-stack` offers native performance and exposes native features such as large title on iOS etc., it may not be as customizable as `@react-navigation/stack` depending on your needs. So if you need more customization than what's possible in this navigator, consider using `@react-navigation/stack` instead - which is a more customizable JavaScript based implementation.

Installation

To use this navigator, ensure that you have `@react-navigation/native` and its dependencies (follow this guide), then install `@react-navigation/native-stack`:

npm Yarn

```
npm install @react-navigation/native-stack
```

API Definition

> 💡 If you encounter any bugs while using `createNativeStackNavigator`, please open issues on `react-native-screens` rather than the `react-navigation` repository!

To use this navigator, import it from `@react-navigation/native-stack`:

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

const Stack = createNativeStackNavigator();

function MyStack() {
  return (
    <Stack.Navigator>
      <Stack.Screen name="Home" component={Home} />
      <Stack.Screen name="Notifications" component={Notifications} />
      <Stack.Screen name="Profile" component={Profile} />
      <Stack.Screen name="Settings" component={Settings} />
    </Stack.Navigator>
  );
}
```

Try this example on Snack

Props

The `Stack.Navigator` component accepts following props:

`id`

Optional unique ID for the navigator. This can be used with `navigation.getParent` to refer to this navigator in a child navigator.

`initialRouteName`

The name of the route to render on first load of the navigator.

`screenOptions`

Default options to use for the screens in the navigator.

Options

The following options can be used to configure the screens in the navigator:

`title`

String that can be used as a fallback for `headerTitle`.

`headerBackButtonMenuEnabled`

Boolean indicating whether to show the menu on longPress of iOS >= 14 back button. Defaults to `true`.

Requires `react-native-screens` version >=3.3.0.

Only supported on iOS.

`headerBackVisible`

Whether the back button is visible in the header. You can use it to show a back button alongside `headerLeft` if you have specified it.

This will have no effect on the first screen in the stack.

`headerBackTitle`

Title string used by the back button on iOS. Defaults to the previous scene's title, or "Back" if there's not enough space. Use `headerBackTitleVisible: false` to hide it.

Only supported on iOS.

`headerBackTitleVisible`

Whether the back button title should be visible or not.

Only supported on iOS.

`headerBackTitleStyle`

Style object for header back title. Supported properties:

- `fontFamily`
- `fontSize`

Only supported on iOS.

`headerBackImageSource`

Image to display in the header as the icon in the back button. Defaults to back icon image for the platform

- A chevron on iOS
- An arrow on Android

`headerLargeStyle`

Style of the header when a large title is shown. The large title is shown if `headerLargeTitle` is `true` and the edge of any scrollable content reaches the matching edge of the header.

Supported properties:

- backgroundColor

Only supported on iOS.

`headerLargeTitle`

Whether to enable header with large title which collapses to regular header on scroll.

For large title to collapse on scroll, the content of the screen should be wrapped in a scrollable view such as `ScrollView` or `FlatList`. If the scrollable area doesn't fill the screen, the large title won't collapse on scroll. You also need to specify `contentInsetAdjustmentBehavior="automatic"` in your `ScrollView`, `FlatList` etc.

Only supported on iOS.

`headerLargeTitleShadowVisible`

Whether drop shadow of header is visible when a large title is shown.

`headerLargeTitleStyle`

Style object for large title in header. Supported properties:

- `fontFamily`
- `fontSize`
- `fontWeight`

- `color`

Only supported on iOS.

`headerShown`

Whether to show the header. The header is shown by default. Setting this to `false` hides the header.

`headerStyle`

Style object for header. Supported properties:

- `backgroundColor`

`headerShadowVisible`

Whether to hide the elevation shadow (Android) or the bottom border (iOS) on the header.

`headerTransparent`

Boolean indicating whether the navigation bar is translucent.

Defaults to `false`. Setting this to `true` makes the header absolutely positioned - so that the header floats over the screen so that it overlaps the content underneath, and changes the background color to `transparent` unless specified in `headerStyle`.

This is useful if you want to render a semi-transparent header or a blurred background.

Note that if you don't want your content to appear under the header, you need to manually add a top margin to your content. React Navigation won't do it automatically.

To get the height of the header, you can use `HeaderHeightContext` with React's Context API or `useHeaderHeight`.

`headerBlurEffect`

Blur effect for the translucent header. The `headerTransparent` option needs to be set to `true` for this to work.

Supported values:

- `extraLight`
- `light`
- `dark`
- `regular`
- `prominent`
- `systemUltraThinMaterial`
- `systemThinMaterial`
- `systemMaterial`
- `systemThickMaterial`
- `systemChromeMaterial`
- `systemUltraThinMaterialLight`
- `systemThinMaterialLight`
- `systemMaterialLight`
- `systemThickMaterialLight`
- `systemChromeMaterialLight`
- `systemUltraThinMaterialDark`
- `systemThinMaterialDark`
- `systemMaterialDark`
- `systemThickMaterialDark`
- `systemChromeMaterialDark`

Only supported on iOS.

`headerBackground`

Function which returns a React Element to render as the background of the header. This is useful for using backgrounds such as an image or a gradient.

`headerTintColor`

Tint color for the header. Changes the color of back button and title.

`headerLeft`

Function which returns a React Element to display on the left side of the header. This replaces the back button. See `headerBackVisible` to show the back button along side left element.

`headerRight`

Function which returns a React Element to display on the right side of the header.

`headerTitle`

String or a function that returns a React Element to be used by the header. Defaults to `title` or name of the screen.

When a function is passed, it receives `tintColor` and `children` in the options object as an argument. The title string is passed in `children`.

Note that if you render a custom element by passing a function, animations for the title won't work.

`headerTitleAlign`

How to align the header title. Possible values:

- `left`
- `center`

Defaults to `left` on platforms other than iOS.

Not supported on iOS. It's always `center` on iOS and cannot be changed.

`headerTitleStyle`

Style object for header title. Supported properties:

- `fontFamily`
- `fontSize`
- `fontWeight`
- `color`

`headerSearchBarOptions`

Options to render a native search bar on iOS. Search bars are rarely static so normally it is controlled by passing an object to `headerSearchBarOptions` navigation option in the component's body. You also need to specify `contentInsetAdjustmentBehavior="automatic"` in your `ScrollView`, `FlatList` etc. If you don't have a `ScrollView`, specify `headerTransparent: false`.

Only supported on iOS and Android.

Example:

```
React.useLayoutEffect(() => {
  navigation.setOptions({
    headerSearchBarOptions: {
      // search bar options
    },
  });
}, [navigation]);
```

Supported properties are described below.

`autoCapitalize`

Controls whether the text is automatically auto-capitalized as it is entered by the user. Possible values:

- `none`
- `words`
- `sentences`
- `characters`

Defaults to `sentences`.

`autoFocus`

Whether to automatically focus search bar when it's shown. Defaults to `false`.

Only supported on Android.

`barTintColor`

The search field background color. By default bar tint color is translucent.

Only supported on iOS.

`tintColor`

The color for the cursor caret and cancel button text.

Only supported on iOS.

`cancelButtonText`

The text to be used instead of default `Cancel` button text.

Only supported on iOS.

`disableBackButtonOverride`

Whether the back button should close search bar's text input or not. Defaults to `false`.

Only supported on Android.

`hideNavigationBar`

Boolean indicating whether to hide the navigation bar during searching. Defaults to `true`.

Only supported on iOS.

`hideWhenScrolling`

Boolean indicating whether to hide the search bar when scrolling. Defaults to `true`.

Only supported on iOS.

`inputType`

The type of the input. Defaults to `"text"`.

Supported values:

- `"text"`
- `"phone"`

- `"number"`
- `"email"`

Only supported on Android.

`obscureBackground`

Boolean indicating whether to obscure the underlying content with semi-transparent overlay. Defaults to `true`.

`placeholder`

Text displayed when search field is empty.

`textColor`

The color of the text in the search field.

`hintTextColor`

The color of the hint text in the search field.

Only supported on Android.

`headerIconColor`

The color of the search and close icons shown in the header

Only supported on Android.

`shouldShowHintSearchIcon`

Whether to show the search hint icon when search bar is focused. Defaults to `true`.

Only supported on Android.

`onBlur`

A callback that gets called when search bar has lost focus.

`onCancelButtonPress`

A callback that gets called when the cancel button is pressed.

`onChangeText`

A callback that gets called when the text changes. It receives the current text value of the search bar.

Example:

```
const [search, setSearch] = React.useState('');

React.useLayoutEffect(() => {
  navigation.setOptions({
    headerSearchBarOptions: {
      onChangeText: (event) => setSearch(event.nativeEvent.text),
    },
  });
}, [navigation]);
```

`header`

Custom header to use instead of the default header.

This accepts a function that returns a React Element to display as a header. The function receives an object containing the following properties as the argument:

- `navigation` - The navigation object for the current screen.
- `route` - The route object for the current screen.
- `options` - The options for the current screen
- `back` - Options for the back button, contains an object with a `title` property to use for back button label.

Example:

```
import { getHeaderTitle } from '@react-navigation/elements';

// ..

header: ({ navigation, route, options, back }) => {
  const title = getHeaderTitle(options, route.name);
```

```
  return (
    <MyHeader
      title={title}
      leftButton={
        back ? <MyBackButton onPress={navigation.goBack} /> : undefined
      }
      style={options.headerStyle}
    />
  );
};
```

To set a custom header for all the screens in the navigator, you can specify this option in the `screenOptions` prop of the navigator.

Note that if you specify a custom header, the native functionality such as large title, search bar etc. won't work.

`statusBarAnimation`

Sets the status bar animation (similar to the `StatusBar` component). Defaults to `fade` on iOS and `none` on Android.

Supported values:

- `"fade"`
- `"none"`
- `"slide"`

On Android, setting either `fade` or `slide` will set the transition of status bar color. On iOS, this option applies to appereance animation of the status bar.

Requires setting `View controller-based status bar appearance -> YES` (or removing the config) in your `Info.plist` file.

Only supported on Android and iOS.

`statusBarHidden`

Whether the status bar should be hidden on this screen.

Requires setting `View controller-based status bar appearance -> YES` (or removing the config) in your `Info.plist` file.

Only supported on Android and iOS.

`statusBarStyle`

Sets the status bar color (similar to the `StatusBar` component). Defaults to `auto`.

Supported values:

- `"auto"`
- `"inverted"` (iOS only)
- `"dark"`
- `"light"`

Requires setting `View controller-based status bar appearance -> YES` (or removing the config) in your `Info.plist` file.

Only supported on Android and iOS.

`statusBarColor`

Sets the status bar color (similar to the `StatusBar` component). Defaults to initial status bar color.

Only supported on Android.

`statusBarTranslucent`

Sets the translucency of the status bar (similar to the `StatusBar` component). Defaults to `false`.

Only supported on Android.

`contentStyle`

Style object for the scene content.

`customAnimationOnGesture`

Whether the gesture to dismiss should use animation provided to `animation` prop. Defaults to `false`.

Doesn't affect the behavior of screens presented modally.

Only supported on iOS.

`fullScreenGestureEnabled`

Whether the gesture to dismiss should work on the whole screen. Using gesture to dismiss with this option results in the same transition animation as `simple_push`. This behavior can be changed by setting `customAnimationOnGesture` prop. Achieving the default iOS animation isn't possible due to platform limitations. Defaults to `false`.

Doesn't affect the behavior of screens presented modally.

Only supported on iOS.

`gestureEnabled`

Whether you can use gestures to dismiss this screen. Defaults to `true`. Only supported on iOS.

`animationTypeForReplace`

The type of animation to use when this screen replaces another screen. Defaults to `pop`.

Supported values:

- `push`: the new screen will perform push animation.
- `pop`: the new screen will perform pop animation.

`animation`

How the screen should animate when pushed or popped.

Supported values:

- `default`: use the platform default animation
- `fade`: fade screen in or out
- `fade_from_bottom`: fade the new screen from bottom

- `flip`: flip the screen, requires `presentation: "modal"` (iOS only)
- `simple_push`: default animation, but without shadow and native header transition (iOS only, uses default animation on Android)
- `slide_from_bottom`: slide in the new screen from bottom
- `slide_from_right`: slide in the new screen from right (Android only, uses default animation on iOS)
- `slide_from_left`: slide in the new screen from left (Android only, uses default animation on iOS)
- `none`: don't animate the screen

Only supported on Android and iOS.

`presentation`

How should the screen be presented.

Supported values:

- `card`: the new screen will be pushed onto a stack, which means the default animation will be slide from the side on iOS, the animation on Android will vary depending on the OS version and theme.
- `modal`: the new screen will be presented modally. this also allows for a nested stack to be rendered inside the screen.
- `transparentModal`: the new screen will be presented modally, but in addition, the previous screen will stay so that the content below can still be seen if the screen has translucent background.
- `containedModal`: will use "UIModalPresentationCurrentContext" modal style on iOS and will fallback to "modal" on Android.
- `containedTransparentModal`: will use "UIModalPresentationOverCurrentContext" modal style on iOS and will fallback to "transparentModal" on Android.
- `fullScreenModal`: will use "UIModalPresentationFullScreen" modal style on iOS and will fallback to "modal" on Android. A screen using this presentation style can't be dismissed by gesture.
- `formSheet`: will use "UIModalPresentationFormSheet" modal style on iOS and will fallback to "modal" on Android.

Only supported on Android and iOS.

`orientation`

The display orientation to use for the screen.

Supported values:

- `default` - resolves to "all" without "portrait_down" on iOS. On Android, this lets the system decide the best orientation.
- `all`: all orientations are permitted.
- `portrait`: portrait orientations are permitted.
- `portrait_up`: right-side portrait orientation is permitted.
- `portrait_down`: upside-down portrait orientation is permitted.
- `landscape`: landscape orientations are permitted.
- `landscape_left`: landscape-left orientation is permitted.
- `landscape_right`: landscape-right orientation is permitted.

Only supported on Android and iOS.

`autoHideHomeIndicator`

Boolean indicating whether the home indicator should prefer to stay hidden. Defaults to `false`.

Only supported on iOS.

`gestureDirection`

Sets the direction in which you should swipe to dismiss the screen.

Supported values:

- `vertical` – dismiss screen vertically
- `horizontal` – dismiss screen horizontally (default)

When using `vertical` option, options `fullScreenGestureEnabled: true`, `customAnimationOnGesture: true` and `animation: 'slide_from_bottom'` are set by default.

Only supported on iOS.

`animationDuration`

Changes the duration (in milliseconds) of `slide_from_bottom`, `fade_from_bottom`, `fade` and `simple_push` transitions on iOS. Defaults to `350`.

The duration of `default` and `flip` transitions isn't customizable.

Only supported on iOS.

`navigationBarColor`

Sets the navigation bar color. Defaults to initial status bar color.

Only supported on Android.

`navigationBarHidden`

Boolean indicating whether the navigation bar should be hidden. Defaults to `false`.

Only supported on Android.

`freezeOnBlur`

Boolean indicating whether to prevent inactive screens from re-rendering. Defaults to `false`. Defaults to `true` when `enableFreeze()` from `react-native-screens` package is run at the top of the application.

Requires `react-native-screens` version >=3.16.0.

Only supported on iOS and Android.

Events

The navigator can emit events on certain actions. Supported events are:

`transitionStart`

This event is fired when the transition animation starts for the current screen.

Event data:

- `e.data.closing` - Boolean indicating whether the screen is being opened or closed.

Example:

```
React.useEffect(() => {
  const unsubscribe = navigation.addListener('transitionStart', (e) => {
    // Do something
  });

  return unsubscribe;
}, [navigation]);
```

`transitionEnd`

This event is fired when the transition animation ends for the current screen.

Event data:

- `e.data.closing` - Boolean indicating whether the screen was opened or closed.

Example:

```
React.useEffect(() => {
  const unsubscribe = navigation.addListener('transitionEnd', (e) => {
    // Do something
  });

  return unsubscribe;
}, [navigation]);
```

Helpers

The native stack navigator adds the following methods to the navigation prop:

`replace`

Replaces the current screen with a new screen in the stack. The method accepts following arguments:

- `name` - *string* - Name of the route to push onto the stack.
- `params` - *object* - Screen params to pass to the destination route.

```
navigation.replace('Profile', { owner: 'Michaś' });
```

`push`

Pushes a new screen to top of the stack and navigate to it. The method accepts following arguments:

- `name` - *string* - Name of the route to push onto the stack.
- `params` - *object* - Screen params to pass to the destination route.

```
navigation.push('Profile', { owner: 'Michaś' });
```

`pop`

Pops the current screen from the stack and navigates back to the previous screen. It takes one optional argument (`count`), which allows you to specify how many screens to pop back by.

```
navigation.pop();
```

`popToTop`

Pops all of the screens in the stack except the first one and navigates to it.

```
navigation.popToTop();
```

Example

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

const Stack = createNativeStackNavigator();

function MyStack() {
  return (
    <Stack.Navigator
      initialRouteName="Home"
      screenOptions={{
        headerTintColor: 'white',
        headerStyle: { backgroundColor: 'tomato' },
      }}
    >
      <Stack.Screen
        name="Home"
        component={Home}
        options={{
          title: 'Awesome app',
        }}
      />
      <Stack.Screen
        name="Profile"
        component={Profile}
        options={{
          title: 'My profile',
        }}
      />
      <Stack.Screen
        name="Settings"
        component={Settings}
        options={{
          gestureEnabled: false,
        }}
      />
    </Stack.Navigator>
  );
}
```

Edit this page