Version: 2.6.0 – 2.12.0

Installation

Requirements

version	`react-native` version
2.0.0+	0.63.0+
1.4.0+	0.60.0+
1.1.0+	0.57.2+
<1.1.0	0.50.0+

It may be possible to use newer versions of react-native-gesture-handler on React Native with version <= 0.59 by reverse Jetifying. Read more on that here https://github.com/mikehardy/jetifier#to-reverse-jetify--convert-node_modules-dependencies-to-support-libraries

Note that if you wish to use `React.createRef()` support for interactions you need to use v16.3 of React

In order to fully utilize the touch events you also need to use `react-native-reanimated` 2.3.0-beta.4 or newer.

Expo

Managed Expo

To use the version of react-native-gesture-handler that is compatible with your managed Expo project, run `expo install react-native-gesture-handler`.

The Expo SDK incorporates the latest version of react-native-gesture-handler available at the time of each SDK release, so managed Expo apps might not always support all our latest features as soon as they are available.

Bare React Native

Since the library uses native support for handling gestures, it requires an extended installation to the norm. If you are starting a new project, you may want to initialize it with expo-cli and use a bare template, they come pre-installed with react-native-gesture-handler.

JS

First, install the library using `yarn`:

```
yarn add react-native-gesture-handler
```

or with `npm` if you prefer:

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

After installation, wrap your entry point with `<GestureHandlerRootView>` or `gestureHandlerRootHOC`.

For example:

```
export default function App() {
  return <GestureHandlerRootView style={{ flex: 1 }}>{/* content */}
</GestureHandlerRootView>;
}
```

> INFO
>
> If you use props such as `shouldCancelWhenOutside`, `simultaneousHandlers`, `waitFor` etc. with gesture handlers, the handlers need to be mounted under a single `GestureHandlerRootView`. So it's important to keep the `GestureHandlerRootView` as close to the actual root view as possible.

> Note that `GestureHandlerRootView` acts like a normal `View`. So if you want it to fill the screen, you will need to pass `{ flex: 1 }` like you'll need to do with a normal `View`. By default, it'll take the size of the content nested inside.

> **TIP**
>
> If you're using gesture handler in your component library, you may want to wrap your library's code in the GestureHandlerRootView component. This will avoid extra configuration for the user.

Linking

> **Important**: You only need to do this step if you're using React Native 0.59 or lower. Since v0.60, linking happens automatically.

```
react-native link react-native-gesture-handler
```

Fabric

Starting with version 2.3.0, Gesture Handler now supports Fabric!. To use Gesture Handler in your Fabric application you have to:

on iOS:

Install pods using `RCT_NEW_ARCH_ENABLED=1 pod install` – this is the same command you run to prepare a Fabric build but you also need to run it after a new native library gets added.

on Android:

There are no additional steps required so long as your app is configured to build with Fabric – this is typically configured by setting `newArchEnabled=true` in `gradle.properties` file in your project.

With wix/react-native-navigation

If you are using a native navigation library like wix/react-native-navigation you need to make sure that every screen is wrapped with `GestureHandlerRootView` (you can do this using

`gestureHandlerRootHOC` function). This can be done for example at the stage when you register your screens. Here's an example:

```
import { gestureHandlerRootHOC } from 'react-native-gesture-handler';
import { Navigation } from 'react-native-navigation';
import FirstTabScreen from './FirstTabScreen';
import SecondTabScreen from './SecondTabScreen';
import PushedScreen from './PushedScreen';
// register all screens of the app (including internal ones)
export function registerScreens() {
  Navigation.registerComponent('example.FirstTabScreen',
    () => gestureHandlerRootHOC(FirstTabScreen),
    () => FirstTabScreen
  );
  Navigation.registerComponent('example.SecondTabScreen',
    () => gestureHandlerRootHOC(SecondTabScreen),
    () => SecondTabScreen
  );
  Navigation.registerComponent('example.PushedScreen',
    () => gestureHandlerRootHOC(PushedScreen),
    () => PushedScreen
  );
}
```

You can check out this example project to see this kind of set up in action.

Remember that you need to wrap each screen that you use in your app with `GestureHandlerRootView` (you can do this using `gestureHandlerRootHOC` function) as with native navigation libraries each screen maps to a separate root view. It will not be enough to wrap the main screen only.

Android

Usage with modals on Android

On Android RNGH does not work by default because modals are not located under React Native Root view in native hierarchy. To fix that, components need to be wrapped with `gestureHandlerRootHOC` (it's no-op on iOS and web).

For example:

```
const ExampleWithHoc = gestureHandlerRootHOC(() => (
    <View>
      <DraggableBox />
    </View>
  );
);

export default function Example() {
  return (
    <Modal>
      <ExampleWithHoc />
    </Modal>
  );
}
```

Kotlin

Since version `2.0.0` RNGH has been rewritten with Kotlin. The default version of the Kotlin plugin used in this library is `1.5.20`.

If you need to use a different Kotlin version, set the `kotlinVersion` ext property in `android/build.gradle` file and RNGH will use that version:

```
buildscript {
    ext {
        ...
        kotlinVersion = "1.5.20"
    }
}
```

The minimal version of the Kotlin plugin supported by RNGH is `1.4.10`.

iOS

There is no additional configuration required on iOS except what follows in the next steps.

If you're in a CocoaPods project (the default setup since React Native 0.60), make sure to install pods before you run your app:

```
cd ios && pod install
```

For React Native 0.61 or greater, add the library as the first import in your index.js file:

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

Web

There is no additional configuration required for the web, however, since the Gesture Handler 2.10.0 the new web implementation is enabled by default. It is recommended to check if the gestures in your app are working as expected since their behavior should now resemble the native platforms. If you don't want to use the new implementation, you can still revert back to the legacy one by enabling it at the beginning of your `index.js` file:

```
import { enableLegacyWebImplementation } from 'react-native-gesture-handler';

enableLegacyWebImplementation(true);
```

Nonetheless, it's recommended to adapt to the new implementation, as the legacy one will be dropped at some point in the future.

If you want to start using the new implementation but don't want to upgrade Gesture Handler, you can enable it (starting with Gesture Handler 2.6.0) at the beginning of your `index.js` file:

```
import { enableExperimentalWebImplementation } from 'react-native-gesture-handler';

enableExperimentalWebImplementation(true);
```