Version: 2.6.0 – 2.12.0

Common handler properties

> DANGER
>
> Consider using the new gestures API instead. The old API is not actively supported and is not receiving the new features. Check out RNGH 2.0 section in Introduction for more information.

This page covers the common set of properties all gesture handler components expose.

Units

All handler component properties and event attributes that represent onscreen dimensions are expressed in screen density independent units we refer to as "points". These are the units commonly used in React Native ecosystem (e.g. in the layout system). They do not map directly to physical pixels but instead to iOS's points and to dp units on Android.

Properties

This section describes properties that can be used with all gesture handler components:

`enabled`

Accepts a boolean value. Indicates whether the given handler should be analyzing stream of touch events or not. When set to `false` we can be sure that the handler's state will **never** become `ACTIVE`. If the value gets updated while the handler already started recognizing a gesture, then the handler's state it will immediately change to `FAILED` or `CANCELLED` (depending on its current state). Default value is `true`.

`shouldCancelWhenOutside`

Accepts a boolean value. When `true` the handler will cancel or fail recognition (depending on its current state) whenever the finger leaves the area of the connected view. Default value of this property is different depending on the handler type. Most handlers' `shouldCancelWhenOutside`

property defaults to `false` except for the `LongPressGestureHandler` and `TapGestureHandler` which default to `true`.

`cancelsTouchesInView` (iOS only)

Accepts a boolean value. When `true`, the handler will cancel touches for native UI components (`UIButton`, `UISwitch`, etc) it's attached to when it becomes `ACTIVE`. Default value is `true`.

`simultaneousHandlers`

Accepts a react ref object or an array of refs to other handler components (refs should be created using `React.createRef()`). When set, the handler will be allowed to activate even if one or more of the handlers provided by their refs are in an `ACTIVE` state. It will also prevent the provided handlers from cancelling the current handler when they activate. Read more in the cross handler interaction section.

`waitFor`

Accepts a react ref object or an array of refs to other handler components (refs should be created using `React.createRef()`). When set the handler will not activate as long as the handlers provided by their refs are in the `BEGAN` state. Read more in the cross handler interaction section.

`hitSlop`

This parameter enables control over what part of the connected view area can be used to begin recognizing the gesture. When a negative number is provided the bounds of the view will reduce the area by the given number of points in each of the sides evenly.

Instead you can pass an object to specify how each boundary side should be reduced by providing different number of points for `left`, `right`, `top` or `bottom` sides. You can alternatively provide `horizontal` or `vertical` instead of specifying directly `left`, `right` or `top` and `bottom`. Finally, the object can also take `width` and `height` attributes. When `width` is set it is only allow to specify one of the sides `right` or `left`. Similarly when `height` is provided only `top` or `bottom` can be set. Specifying `width` or `height` is useful if we only want the gesture to activate on the edge of the view. In which case for example we can set `left: 0` and `width: 20` which would make it possible for the gesture to be recognize when started no more than 20 points from the left edge.

IMPORTANT: Note that this parameter is primarily designed to reduce the area where gesture can activate. Hence it is only supported for all the values (except `width` and `height`) to be non positive (0 or lower). Although on Android it is supported for the values to also be positive and therefore allow to expand beyond view bounds but not further than the parent view bounds. To achieve this effect on both platforms you can use React Native's View hitSlop property.

`userSelect` (web only)

This parameter allows to specify which `userSelect` property should be applied to underlying view. Possible values are `"none" | "auto" | "text"`. Defaults to `"none"`. **Available since version 2.8.0**

`onGestureEvent`

Takes a callback that is going to be triggered for each subsequent touch event while the handler is in an ACTIVE state. Event payload depends on the particular handler type. Common set of event data attributes is documented below and handler specific attributes are documented on the corresponding handler pages. E.g. event payload for `PinchGestureHandler` contains `scale` attribute that represents how the distance between fingers changed since when the gesture started.

Instead of a callback `Animated.event` object can be used. Also Animated events with `useNativeDriver` flag enabled **are fully supported**.

`onHandlerStateChange`

Takes a callback that is going to be triggered when state of the given handler changes.

The event payload contains the same payload as in case of `onGestureEvent` including handler specific event attributes some handlers may provide.

In addition `onHandlerStateChange` event payload contains `oldState` attribute which represents the state of the handler right before the change.

Instead of a callback `Animated.event` object can be used. Also Animated events with `useNativeDriver` flag enabled **are fully supported**.

Event data

This section describes the attributes of event object being provided to `onGestureEvent` and `onHandlerStateChange` callbacks:

`state`

Current state of the handler. Expressed as one of the constants exported under `State` object by the library. Refer to the section about handler state to learn more about how to use it.

`numberOfPointers`

Represents the number of pointers (fingers) currently placed on the screen.