Version: 2.6.0 – 2.12.0

PanGestureHandler

> **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.

A continuous gesture handler that can recognize a panning (dragging) gesture and track its movement.

The handler activates when a finger is placed on the screen and moved some initial distance.

Configurations such as a minimum initial distance, specific vertical or horizontal pan detection and number of fingers required for activation (allowing for multifinger swipes) may be specified.

Gesture callback can be used for continuous tracking of the pan gesture. It provides information about the gesture such as its XY translation from the starting point as well as its instantaneous velocity.

The handler is implemented using UIPanGestureRecognizer on iOS and PanGestureHandler on Android.

Custom activation criteria

The `PanGestureHandler` component exposes a number of properties that can be used to customize the criteria under which a handler will activate or fail when recognizing a gesture.

When more than one of such a property is set, `PanGestureHandler` expects all criteria to be met for successful recognition and at most one of the criteria to be overstepped to fail recognition. For example when both `minDeltaX` and `minDeltaY` are set to 20 we expect the finger to travel by 20 points in both the X and Y axis before the handler activates. Another example would be setting both `maxDeltaX` and `maxDeltaY` to 20 and `minDist` to 23. In such a case, if we move a finger along the X-axis by 20 points and along the Y-axis by 0 points, the handler will fail even though the finger is still within the bounds of translation along Y-axis.

Multi touch pan handling

If your app relies on multi touch pan handling this section provides some information how the default behavior differs between the platform and how (if necessary) it can be unified.

The difference in multi touch pan handling lies in the way how translation properties during the event are being calculated. On iOS the default behavior when more than one finger is placed on the screen is to treat this situation as if only one pointer was placed in the center of mass (average position of all the pointers). This applies also to many platform native components that handle touch even if not primarily interested in multi touch interactions like for example UIScrollView component.

The default behavior for native components like scroll view, pager views or drawers is different and hence gesture handler defaults to that when it comes to pan handling. The difference is that instead of treating the center of mass of all the fingers placed as a leading pointer it takes the latest placed finger as such. This behavior can be changed on Android using `avgTouches` flag.

Note that on both Android and iOS when the additional finger is placed on the screen that translation prop is not affected even though the position of the pointer being tracked might have changed. Therefore it is safe to rely on translation most of the time as it only reflects the movement that happens regardless of how many fingers are placed on the screen and if that number changes over time. If you wish to track the "center of mass" virtual pointer and account for its changes when the number of finger changes you can use relative or absolute position provided in the event (`x` and `y` or `absoluteX` and `absoluteY`).

Properties

See set of properties inherited from base handler class. Below is a list of properties specific to `PanGestureHandler` component:

`minDist`

Minimum distance the finger (or multiple finger) need to travel before the handler activates. Expressed in points.

`minPointers`

A number of fingers that is required to be placed before handler can activate. Should be a higher or equal to 0 integer.

`maxPointers`

When the given number of fingers is placed on the screen and handler hasn't yet activated it will fail recognizing the gesture. Should be a higher or equal to 0 integer.

`activeOffsetX`

Range along X axis (in points) where fingers travels without activation of handler. Moving outside of this range implies activation of handler. Range can be given as an array or a single number. If range is set as an array, first value must be lower or equal to 0, a the second one higher or equal to 0. If only one number `p` is given a range of `(-inf, p)` will be used if `p` is higher or equal to 0 and `(-p, inf)` otherwise.

`activeOffsetY`

Range along Y axis (in points) where fingers travels without activation of handler. Moving outside of this range implies activation of handler. Range can be given as an array or a single number. If range is set as an array, first value must be lower or equal to 0, a the second one higher or equal to 0. If only one number `p` is given a range of `(-inf, p)` will be used if `p` is higher or equal to 0 and `(-p, inf)` otherwise.

`failOffsetY`

When the finger moves outside this range (in points) along Y axis and handler hasn't yet activated it will fail recognizing the gesture. Range can be given as an array or a single number. If range is set as an array, first value must be lower or equal to 0, a the second one higher or equal to 0. If only one number `p` is given a range of `(-inf, p)` will be used if `p` is higher or equal to 0 and `(-p, inf)` otherwise.

`failOffsetX`

When the finger moves outside this range (in points) along X axis and handler hasn't yet activated it will fail recognizing the gesture. Range can be given as an array or a single number. If range is set as an array, first value must be lower or equal to 0, a the second one higher or equal to 0. If only

one number `p` is given a range of `(-inf, p)` will be used if `p` is higher or equal to 0 and `(-p, inf)` otherwise.

`maxDeltaX`

> This method is deprecated but supported for backward compatibility. Instead of using `maxDeltaX={N}` you can do `failOffsetX={[-N, N]}`.

When the finger travels the given distance expressed in points along X axis and handler hasn't yet activated it will fail recognizing the gesture.

`maxDeltaY`

> This method is deprecated but supported for backward compatibility. Instead of using `maxDeltaY={N}` you can do `failOffsetY={[-N, N]}`.

When the finger travels the given distance expressed in points along Y axis and handler hasn't yet activated it will fail recognizing the gesture.

`minOffsetX`

> This method is deprecated but supported for backward compatibility. Instead of using `minOffsetX={N}` you can do `activeOffsetX={N}`.

Minimum distance along X (in points) axis the finger (or multiple finger) need to travel before the handler activates. If set to a lower or equal to 0 value we expect the finger to travel "left" by the given distance. When set to a higher or equal to 0 number the handler will activate on a movement to the "right". If you wish for the movement direction to be ignored use `minDeltaX` instead.

`minOffsetY`

> This method is deprecated but supported for backward compatibility. Instead of using `minOffsetY={N}` you can do `activeOffsetY={N}`.

Minimum distance along Y (in points) axis the finger (or multiple finger) need to travel before the handler activates. If set to a lower or equal to 0 value we expect the finger to travel "up" by the given distance. When set to a higher or equal to 0 number the handler will activate on a movement to the "bottom". If you wish for the movement direction to be ignored use `minDeltaY` instead.

`minDeltaX`

> This method is deprecated but supported for backward compatibility. Instead of using `minDeltaX={N}` you can do `activeOffsetX={[-N, N]}`.

Minimum distance along X (in points) axis the finger (or multiple finger) need to travel (left or right) before the handler activates. Unlike `minoffsetx` this parameter accepts only non-lower or equal to 0 numbers that represents the distance in point units. If you want for the handler to activate for the movement in one particular direction use `minOffsetX` instead.

`minDeltaY`

> This method is deprecated but supported for backward compatibility. Instead of using `minDeltaY={N}` you can do `activeOffsetY={[-N, N]}`.

Minimum distance along Y (in points) axis the finger (or multiple finger) need to travel (top or bottom) before the handler activates. Unlike `minOffsetY` this parameter accepts only non-lower or equal to 0 numbers that represents the distance in point units. If you want for the handler to activate for the movement in one particular direction use `minOffsetY` instead.

`avgTouches` (Android only)

`enableTrackpadTwoFingerGesture` (iOS only)

Enables two-finger gestures on supported devices, for example iPads with trackpads. If not enabled the gesture will require click + drag, with enableTrackpadTwoFingerGesture swiping with two fingers will also trigger the gesture.

Event data

See set of event attributes from base handler class. Below is a list of gesture event attributes specific to `PanGestureHandler`:

`translationX`

Translation of the pan gesture along X axis accumulated over the time of the gesture. The value is expressed in the point units.

`translationY`

Translation of the pan gesture along Y axis accumulated over the time of the gesture. The value is expressed in the point units.

`velocityX`

Velocity of the pan gesture along the X axis in the current moment. The value is expressed in point units per second.

`velocityY`

Velocity of the pan gesture along the Y axis in the current moment. The value is expressed in point units per second.

`x`

X coordinate of the current position of the pointer (finger or a leading pointer when there are multiple fingers placed) relative to the view attached to the handler. Expressed in point units.

`y`

Y coordinate of the current position of the pointer (finger or a leading pointer when there are multiple fingers placed) relative to the view attached to the handler. Expressed in point units.

`absoluteX`

X coordinate of the current position of the pointer (finger or a leading pointer when there are multiple fingers placed) relative to the window. The value is expressed in point units. It is recommended to use it instead of `x` in cases when the original view can be transformed as an effect of the gesture.

`absoluteY`

Y coordinate of the current position of the pointer (finger or a leading pointer when there are multiple fingers placed) relative to the window. The value is expressed in point units. It is recommended to use it instead of `y` in cases when the original view can be transformed as an effect of the gesture.

Example

See the draggable example from Gesture Handler Example App.

```
import React, { Component } from 'react';
import { Animated, Dimensions } from 'react-native';
import {
  GestureHandlerRootView,
  PanGestureHandler,
} from 'react-native-gesture-handler';

const { width } = Dimensions.get('screen');
const circleRadius = 30;

class Circle extends Component {
  _touchX = new Animated.Value(width / 2 - circleRadius);

  _onPanGestureEvent = Animated.event([{ nativeEvent: { x: this._touchX } }], {
    useNativeDriver: true,
  });

  render() {
    return (
      <GestureHandlerRootView>
        <PanGestureHandler onGestureEvent={this._onPanGestureEvent}>
          <Animated.View
            style={{
              height: 150,
              justifyContent: 'center',
            }}>
            <Animated.View
              style={[
                {
                  backgroundColor: '#42a5f5',
                  borderRadius: circleRadius,
                  height: circleRadius * 2,
                  width: circleRadius * 2,
                },
                {
                  transform: [
                    {
                      translateX: Animated.add(
                        this._touchX,
                        new Animated.Value(-circleRadius)
                      ),
                    },
                  ],
                },
              ]}
            />
```

```
            </Animated.View>
          </PanGestureHandler>
        </GestureHandlerRootView>
      );
    }
  }

  export default function App() {
    return <Circle />;
  }
```