Version: 2.6.0 – 2.12.0

Cross handler interactions

Gesture handlers can "communicate" with each other to support complex gestures and control how they *activate* in certain scenarios.

There are two means of achieving that described in the sections below. In each case, it is necessary to provide a reference of one handler as a property to the other. Gesture handler relies on ref objects created using `React.createRef()` and introduced in React 16.3.

Simultaneous recognition

By default, only one gesture handler is allowed to be in the `ACTIVE` state. So when a gesture handler recognizes a gesture it cancels all other handlers in the `BEGAN` state and prevents any new handlers from receiving a stream of touch events as long as it remains `ACTIVE`.

This behavior can be altered using the `simultaneousHandlers` property (available for all types of handlers). This property accepts a ref or an array of refs to other handlers. Handlers connected in this way will be allowed to remain in the `ACTIVE` state at the same time.

Use cases

Simultaneous recognition needs to be used when implementing a photo preview component that supports zooming (scaling) the photo, rotating and panning it while zoomed in. In this case we would use a `PinchGestureHandler`, `RotationGestureHandler` and `PanGestureHandler` that would have to simultaneously recognize gestures.

Example

See the "Scale, rotate & tilt" example from the GestureHandler Example App or view it directly on your phone by visiting our expo demo.

```
class PinchableBox extends React.Component {
  // ...take a look on full implementation in an Example app
  render() {
```

```
        const imagePinch = React.createRef();
        const imageRotation = React.createRef();
        return (
          <RotationGestureHandler
            ref={imageRotation}
            simultaneousHandlers={imagePinch}
            onGestureEvent={this._onRotateGestureEvent}
            onHandlerStateChange={this._onRotateHandlerStateChange}>
            <Animated.View>
              <PinchGestureHandler
                ref={imagePinch}
                simultaneousHandlers={imageRotation}
                onGestureEvent={this._onPinchGestureEvent}
                onHandlerStateChange={this._onPinchHandlerStateChange}>
                <Animated.View style={styles.container} collapsable={false}>
                  <Animated.Image
                    style={[
                      styles.pinchableImage,
                      {
                        /* events-related transformations */
                      },
                    ]}
                  />
                </Animated.View>
              </PinchGestureHandler>
            </Animated.View>
          </RotationGestureHandler>
        );
      }
    }
```

Awaiting other handlers

Use cases

A good example where awaiting is necessary is when we want to have single and double tap handlers registered for one view (a button). In such a case we need to make single tap handler await a double tap. Otherwise if we try to perform a double tap the single tap handler will fire just after we hit the button for the first time, consequently cancelling the double tap handler.

Example

See the "Multitap" example from GestureHandler Example App or view it directly on your phone by visiting our expo demo.

```
const doubleTap = React.createRef();
const PressBox = () => (
  <TapGestureHandler
    onHandlerStateChange={({ nativeEvent }) =>
      nativeEvent.state === State.ACTIVE && Alert.alert('Single tap!')
    }
    waitFor={doubleTap}>
    <TapGestureHandler
      ref={doubleTap}
      onHandlerStateChange={({ nativeEvent }) =>
        nativeEvent.state === State.ACTIVE && Alert.alert("You're so fast")
      }
      numberOfTaps={2}>
      <View style={styles.box} />
    </TapGestureHandler>
  </TapGestureHandler>
);
```