PanResponder

PanResponder reconciles several touches into a single gesture. It makes single-touch gestures resilient to extra touches, and can be used to recognize basic multi-touch gestures.

By default, PanResponder holds an InteractionManager handle to block long-running JS events from interrupting active gestures.

It provides a predictable wrapper of the responder handlers provided by the gesture responder system. For each handler, it provides a new gestureState object alongside the native event object:

```
onPanResponderMove: (event, gestureState) => {}
```

A native event is a synthetic touch event with form of PressEvent.

A gestureState object has the following:

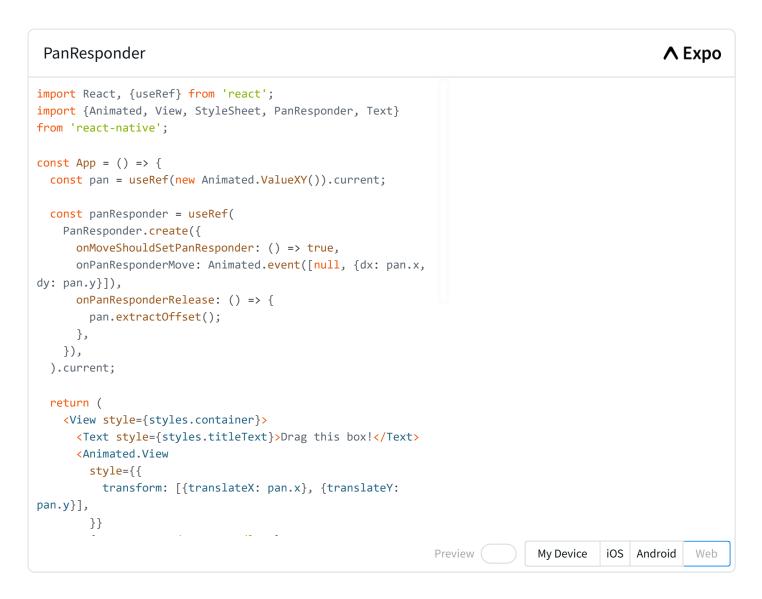
- stateID ID of the gestureState- persisted as long as there's at least one touch on screen
- moveX the latest screen coordinates of the recently-moved touch
- movey the latest screen coordinates of the recently-moved touch
- x0 the screen coordinates of the responder grant
- yo the screen coordinates of the responder grant
- dx accumulated distance of the gesture since the touch started
- dy accumulated distance of the gesture since the touch started
- vx current velocity of the gesture
- vy current velocity of the gesture
- numberActiveTouches Number of touches currently on screen

Usage Pattern

```
const ExampleComponent = () => {
  const panResponder = React.useRef(
    PanResponder.create({
      // Ask to be the responder:
      onStartShouldSetPanResponder: (evt, gestureState) => true,
      onStartShouldSetPanResponderCapture: (evt, gestureState) =>
      onMoveShouldSetPanResponder: (evt, gestureState) => true,
      onMoveShouldSetPanResponderCapture: (evt, gestureState) =>
       true,
      onPanResponderGrant: (evt, gestureState) => {
        // The gesture has started. Show visual feedback so the user knows
        // what is happening!
       // gestureState.d{x,y} will be set to zero now
      },
      onPanResponderMove: (evt, gestureState) => {
       // The most recent move distance is gestureState.move{X,Y}
       // The accumulated gesture distance since becoming responder is
       // gestureState.d{x,y}
      },
      onPanResponderTerminationRequest: (evt, gestureState) =>
      onPanResponderRelease: (evt, gestureState) => {
        // The user has released all touches while this view is the
        // responder. This typically means a gesture has succeeded
      },
      onPanResponderTerminate: (evt, gestureState) => {
        // Another component has become the responder, so this gesture
       // should be cancelled
      },
      onShouldBlockNativeResponder: (evt, gestureState) => {
        // Returns whether this component should block native components from becoming
the JS
       // responder. Returns true by default. Is currently only supported on android.
       return true;
     },
   }),
  ).current;
  return <View {...panResponder.panHandlers} />;
};
```

Example

PanResponder works with Animated API to help build complex gestures in the UI. The following example contains an animated View component which can be dragged freely across the screen



Try the PanResponder example in RNTester.

Reference

Methods

create()

```
static create(config: PanResponderCallbacks): PanResponderInstance;
```

Parameters:

NAME	TYPE	DESCRIPTION
config Required	object	Refer below

The config object provides enhanced versions of all of the responder callbacks that provide not only the PressEvent, but also the PanResponder gesture state, by replacing the word Responder with PanResponder in each of the typical onResponder* callbacks. For example, the config object would look like:

```
onMoveShouldSetPanResponder: (e, gestureState) => {...}
onMoveShouldSetPanResponderCapture: (e, gestureState) => {...}
onStartShouldSetPanResponder: (e, gestureState) => {...}
onStartShouldSetPanResponderCapture: (e, gestureState) => {...}
onPanResponderReject: (e, gestureState) => {...}
onPanResponderGrant: (e, gestureState) => {...}
onPanResponderStart: (e, gestureState) => {...}
onPanResponderEnd: (e, gestureState) => {...}
onPanResponderRelease: (e, gestureState) => {...}
onPanResponderMove: (e, gestureState) => {...}
onPanResponderTerminate: (e, gestureState) => {...}
onPanResponderTerminate: (e, gestureState) => {...}
onPanResponderTerminationRequest: (e, gestureState) => {...}
onPanResponderTerminationRequest: (e, gestureState) => {...}
```

In general, for events that have capture equivalents, we update the gestureState once in the capture phase and can use it in the bubble phase as well.

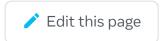
Be careful with onStartShould* callbacks. They only reflect updated gestureState for start/end events that bubble/capture to the Node. Once the node is the responder, you

can rely on every start/end event being processed by the gesture and gestureState being updated accordingly. (numberActiveTouches) may not be totally accurate unless you are the responder.









Last updated on Jun 21, 2023