Version: 2.6.0 – 2.12.0

Gesture

`Gesture` is the object that allows you to create and compose gestures.

TIP

Consider wrapping your gesture configurations with `useMemo`, as it will reduce the amount of work Gesture Handler has to do under the hood when updating gestures. For example:

```
const gesture = useMemo(
  () =>
    Gesture.Tap().onStart(() => {
      console.log('Number of taps:', tapNumber + 1);
      setTapNumber((value) => value + 1);
    }),
  [tapNumber, setTapNumber]
);
```

Gesture.Tap()

Creates a new instance of `TapGesture` with its default config and no callbacks.

Gesture.Pan()

Creates a new instance of `PanGesture` with its default config and no callbacks.

Gesture.LongPress()

Creates a new instance of `LongPressGesture` with its default config and no callbacks.

Gesture.Fling()

Creates a new instance of `FlingGesture` with its default config and no callbacks.

Gesture.Pinch()

Creates a new instance of `PinchGesture` with its default config and no callbacks.

Gesture.Rotation()

Creates a new instance of `RotationGesture` with its default config and no callbacks.

Gesture.ForceTouch()

Creates a new instance of `ForceTouchGesture` with its default config and no callbacks.

Gesture.Manual()

Creates a new instance of `ManualGesture` with its default config and no callbacks.

Gesture.Native()

Creates a new instance of `NativeGesture` with its default config and no callbacks.

Gesture.Race(gesture1, gesture2, gesture3, ...): ComposedGesture

Creates a gesture composed of those provided as arguments. Only one of those can become active and there are no restrictions to the activation of the gesture. The first one to activate will cancel all the others.

Gesture.Simultaneous(gesture1, gesture2, gesture3, ...): ComposedGesture

Creates a gesture composed of those provided as arguments. All of them can become active without cancelling the others.

Gesture.Exclusive(gesture1, gesture2, gesture3, ...): ComposedGesture

Creates a gesture composed of those provided as arguments. Only one of them can become active, but the first one has a higher priority than the second one, the second one has a higher priority than the third one, and so on. When all gestures are in the `BEGAN` state and the activation criteria for the second one is met, instead of activating it will wait until the first one fails (and only then it will activate) or until the first one activates (and then the second one will get cancelled). It is useful when you want to compose gestures with similar activation criteria (e.g. single and double tap at the same component, without Exclusive the single tap would activate every time user taps thus cancelling the double tap).