Version: 2.6.0 – 2.12.0

How does it work?

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.

iOS

All gestures are implemented using UIGestureRecognizers, some of them have been slightly modified to allow for more customization and to conform to the state flow of RNGH. When you assign a gesture configuration to the `GestureDetector`, it creates all the required recognizers and assigns them to the child view of the detector. From this point most of the heavy lifting is handled by the UIKit (with our help to correctly implement interactions between gestures).

Android

Unfortunately, Android doesn't provide an easy way of handling gestures hence most of them were implemented from scratch, including a system for managing how the gestures should interact with each other. Here's a quick overview of how it works: When you wrap a component with `GestureHandlerRootView` it allows for the RNGH to intercept all touch events on that component and process them, deciding whether they should be handled by one of the gesture handlers or passed to the underlying view. Gesture handlers are created when you assign a gesture configuration to the `GestureDetector`, it initializes all of the necessary handlers natively. Every `GestureHandlerRootView` also has a specific handler to decide whether to pass the touch events or to consume them. It can never activate, only begin, end or be cancelled. When this handler is in the `UNDETERMINED` state it means that there is no touch in progress, however when the touch starts it transitions to the `BEGAN` state. As long as it stays in that state, no touch event is consumed, but as soon as it gets cancelled (meaning that some handler has activated) all incoming touch events get consumed, preventing underlying view from receiving them.

When a pointer touches the screen the view tree is traversed in order to extract all handlers attached to the views below the finger (including the one attached to the `GestureHandlerRootView`) and all extracted handlers transition to the `BEGAN` state, signalling that the gesture may heve began. The touch events continue to be delivered to all extracted handlers until one of them recognizes the gesture and tries to activate. At this point the orchestrator checks whether this gesture should wait for any other of the extracted gestures to fail. If it does, it's put to the waiting list, if it doesn't, it gets activated and all other gestures (that are not simultaneous with it) get cancelled. When a gesture handlers transitions to a finished state (the gesture recognized by it stops, it fails or gets cancelled) the orchestrator check the waiting handlers. Every one of them that waited for the gesture that just failed tries to activate again (and again the orchestrator checks if it should wait for any of the extracted gestures...).