React Native Hooks - Complete Guide

React Native Hooks - Complete Guide (Markdown Render)

~~~~
# React Native Hooks - Complete Guide

This guide covers all common hooks you will use in React Native projects:
- React core hooks (from React 18)
- React Native built-in hooks
- React Navigation hooks
- Reanimated and gesture hooks
- Safe Area hooks
- React Query hooks
- Expo hooks
- Custom hook patterns

All examples are Expo-friendly and written to be pasted into a Snack or RN app.
Flow diagrams use simple ASCII so they are readable everywhere.

Rules of Hooks:
1) Call hooks only at the top level of a component or another hook.
2) Call hooks only from React function components or custom hooks.
~~~~

Core React Hooks

~~~~
Below are the React core hooks you can use in React Native projects.

## useState

Add state to a function component.

```tsx
import React, { useState } from 'react';
import { Button, Text, View } from 'react-native';

export default function UseStateDemo() {
  const [count, setCount] = useState(0);
  return (
    <View style={{ padding: 16 }}>
      <Text>Count: {count}</Text>
      <Button title="Increment" onPress={() => setCount(c => c + 1)} />
    </View>
  );
}
```
**Flow diagram**

```
Render -> read state -> on event setState -> schedule re-render
+-------------------+
| Initial render    |
+---------+---------+
          |
          v
~~~~

React Native Hooks - Complete Guide

~~~~
+-------------------+
| setState called   |
+---------+---------+
          |
          v
+-------------------+
| Re-render with    |
| new state         |
+-------------------+
```
## useEffect

Run side effects after render; return a cleanup to run before next effect or unmount. Depend on an
array.

```tsx
import React, { useEffect, useState } from 'react';
import { Text, View } from 'react-native';

export default function UseEffectDemo() {
  const [data, setData] = useState('');
  useEffect(() => {
    const id = setTimeout(() => setData('Loaded'), 500);
    return () => clearTimeout(id);
  }, []);
  return <View style={{ padding: 16 }}><Text>{data}</Text></View>;
}
```
**Flow diagram**

```
Render -> Run effect -> (later) Cleanup -> Next effect
+---------+      +-----------+      +-----------+
| Render  | ---> | Effect    | ---> | Cleanup   |
+---------+      +-----------+      +-----------+
        Dependencies change or unmount triggers Cleanup
```
## useContext

Read a value from React context.

```tsx
import React, { createContext, useContext } from 'react';
import { Text, View } from 'react-native';

const ThemeContext = createContext('light');

function Child() {
  const theme = useContext(ThemeContext);
  return <Text>Theme: {theme}</Text>;
}
~~~~

~~~
export default function UseContextDemo() {
  return (
    <ThemeContext.Provider value="dark">
      <View style={{ padding: 16 }}><Child /></View>
    </ThemeContext.Provider>
  );
}
```
**Flow diagram**

```
Provider -> Consumer reads context -> Re-render when value changes
```
## useReducer

Predictable state transitions for complex logic.

```tsx
import React, { useReducer } from 'react';
import { Button, Text, View } from 'react-native';

function reducer(state, action) {
  switch (action.type) {
    case 'inc': return { count: state.count + 1 };
    case 'dec': return { count: state.count - 1 };
    default: return state;
  }
}

export default function UseReducerDemo() {
  const [state, dispatch] = useReducer(reducer, { count: 0 });
  return (
    <View style={{ padding: 16 }}>
      <Text>Count: {state.count}</Text>
      <Button title="+" onPress={() => dispatch({ type: 'inc' })} />
      <Button title="-" onPress={() => dispatch({ type: 'dec' })} />
    </View>
  );
}
```
**Flow diagram**

```
dispatch(action) -> reducer(prev, action) -> new state -> re-render
```
## useCallback

Memoize a function reference; helps avoid unnecessary re-renders.

```tsx
import React, { useCallback, useState } from 'react';
import { Button, Text, View } from 'react-native';
~~~

~~~
export default function UseCallbackDemo() {
  const [count, setCount] = useState(0);
  const inc = useCallback(() => setCount(c => c + 1), []);
  return (
    <View style={{ padding: 16 }}>
      <Text>Count: {count}</Text>
      <Button title="Increment" onPress={inc} />
    </View>
  );
}
```
**Flow diagram**

```
Dependencies stable -> same function identity; change -> new function created
```
## useMemo

Memoize an expensive computed value.

```tsx
import React, { useMemo, useState } from 'react';
import { Button, Text, View } from 'react-native';

function heavy(n) { for (let i=0;i<100000;i++){} return n*2; }

export default function UseMemoDemo() {
  const [n, setN] = useState(1);
  const doubled = useMemo(() => heavy(n), [n]);
  return (
    <View style={{ padding: 16 }}>
      <Text>Doubled: {doubled}</Text>
      <Button title="Inc" onPress={() => setN(x => x + 1)} />
    </View>
  );
}
```
**Flow diagram**

```
Dependencies change -> recompute; else reuse cached value
```
## useRef

Mutable container whose .current persists without causing re-renders.

```tsx
import React, { useRef, useState } from 'react';
import { Button, Text, View } from 'react-native';

export default function UseRefDemo() {
~~~

~~~~
  const clicks = useRef(0);
  const [_, force] = useState(0);
  return (
    <View style={{ padding: 16 }}>
      <Text>Clicks stored in ref: {clicks.current}</Text>
      <Button title="Click" onPress={() => { clicks.current++; force(x=>x+1); }} />
    </View>
  );
}
```
**Flow diagram**

```
Ref updated -> no render; only reading it is side-effect free
```
## useImperativeHandle

Customize the ref API exposed by a child component.

```tsx
import React, { forwardRef, useImperativeHandle, useRef } from 'react';
import { Button, TextInput, View } from 'react-native';

const FocusInput = forwardRef((props, ref) => {
  const inputRef = useRef(null);
  useImperativeHandle(ref, () => ({ focus: () => inputRef.current?.focus() }));
  return <TextInput ref={inputRef} style={{ borderWidth: 1, padding: 8 }} />;
});

export default function UseImperativeHandleDemo() {
  const ref = useRef(null);
  return (
    <View style={{ padding: 16 }}>
      <Button title="Focus" onPress={() => ref.current?.focus()} />
      <FocusInput ref={ref} />
    </View>
  );
}
```
**Flow diagram**

```
Parent holds ref -> child exposes imperative methods via useImperativeHandle
```
## useLayoutEffect

Like useEffect but runs after layout, before paint; avoid long work.

```tsx
import React, { useLayoutEffect } from 'react';
import { Text, View } from 'react-native';
~~~~

export default function UseLayoutEffectDemo() {
  useLayoutEffect(() => {
    // Measure or sync layout here
  }, []);
  return <View style={{ padding: 16 }}><Text>Layout effect demo</Text></View>;
}
```
**Flow diagram**

```
Render -> layout -> useLayoutEffect -> paint
~~~
## useDebugValue

Label custom hooks in React DevTools.

```tsx
import React, { useDebugValue } from 'react';
function useToggle(initial=false) {
  const [val, setVal] = React.useState(initial);
  useDebugValue(val ? 'ON' : 'OFF');
  return [val, () => setVal(v => !v)];
}
~~~
Flow diagram

```
Used inside custom hooks only; shows a label in DevTools
```
## useDeferredValue

Defer a value to keep UI responsive.

```tsx
import React from 'react';
import { TextInput, Text, View } from 'react-native';

export default function UseDeferredValueDemo() {
  const [text, setText] = React.useState('');
  const deferred = React.useDeferredValue(text);
  return (
    <View style={{ padding: 16 }}>
      <TextInput value={text} onChangeText={setText} />
      <Text>Deferred: {deferred}</Text>
    </View>
  );
}
```
Flow diagram

```
User types -> state updates immediately -> deferred updates later
```

~~~~
```
## useTransition

Mark state updates as non-urgent.

```tsx
import React from 'react';
import { Button, Text, View } from 'react-native';

export default function UseTransitionDemo() {
  const [list, setList] = React.useState([]);
  const [isPending, startTransition] = React.useTransition();
  return (
    <View style={{ padding: 16 }}>
      <Button title="Big update" onPress={() => {
        startTransition(() => setList(Array.from({length:5000}, (_,i)=>i)));
      }} />
      {isPending ? <Text>Loading...</Text> : list.slice(0,20).map(i => <Text key={i}>{i}</Text>)}
    </View>
  );
}
```
**Flow diagram**

```
startTransition -> low-priority update scheduled -> UI stays responsive
```
## useId

Stable unique IDs for accessibility.

```tsx
import React from 'react';
import { Text, View } from 'react-native';
export default function UseIdDemo() {
  const id = React.useId();
  return <View style={{ padding: 16 }}><Text>Generated id: {id}</Text></View>;
}
```
**Flow diagram**

```
Unique per tree; stable across renders
```
## useSyncExternalStore

Subscribe to an external store (for library authors).

```tsx
import React from 'react';
function subscribe(callback){ const id=setInterval(callback,1000); return () => clearInterval(id);
}
~~~~

~~~
function getSnapshot(){ return new Date().toISOString(); }

export default function UseSyncExternalStoreDemo() {
  const time = React.useSyncExternalStore(subscribe, getSnapshot);
  return <>{time}</>;
}
```
**Flow diagram**

```
Library-level hook to read from external sources safely
```
## useInsertionEffect

Runs before any DOM mutations; used by CSS-in-JS libs. Rare in React Native.

```tsx
// Generally not needed in RN apps
import React from 'react';
export default function UseInsertionEffectDemo(){
  React.useInsertionEffect(() => { /* style registry, etc. */ }, []);
  return null;
}
```
**Flow diagram**

```
Primarily for library authors; avoid in app code
```
~~~

React Native Built-in Hooks

~~~
Hooks provided by the react-native package.

## useWindowDimensions

Responsive layout: read width, height, scale, fontScale.

```tsx
import React from 'react';
import { Text, View, useWindowDimensions } from 'react-native';

export default function UseWindowDimensionsDemo() {
  const { width, height } = useWindowDimensions();
  return <View style={{ padding: 16 }}><Text>{width} x {height}</Text></View>;
}
```
## useColorScheme

Detect light or dark mode.

```tsx
~~~

React Native Hooks - Complete Guide

~~~~
import React from 'react';
import { Text, View, useColorScheme } from 'react-native';

export default function UseColorSchemeDemo() {
  const scheme = useColorScheme();
  return <View style={{ padding: 16 }}><Text>Scheme: {scheme}</Text></View>;
}
```
~~~~

React Navigation Hooks

~~~~
Hooks from @react-navigation/native.

## useNavigation

Access navigation methods without prop drilling.

```tsx
import React from '@react-navigation/native';

```
## useRoute

Read current route and params.

```tsx
import { useRoute } from '@react-navigation/native';

```
## useFocusEffect

Run an effect when the screen is focused.

```tsx
import { useFocusEffect } from '@react-navigation/native';
import React, { useCallback } from 'react';
export default function Screen() {
  useFocusEffect(
    useCallback(() => {
      // subscribe
      return () => {
        // unsubscribe
      };
    }, [])
  );
  return null;
}
```
## useIsFocused

Boolean flag: is this screen currently focused.
~~~~

React Native Hooks - Complete Guide

```tsx
import { useIsFocused } from '@react-navigation/native';

```
Focus flow diagram

```
User navigates -> Screen gains focus
-> useFocusEffect callback runs
-> When navigating away: cleanup runs
-> When coming back: callback runs again
```

Reanimated Hooks

Hooks from react-native-reanimated (v2+). Requires Reanimated properly configured.

## useSharedValue

Create a mutable value for animations.

```tsx
import React from 'react';
import Animated, { useSharedValue, useAnimatedStyle, withTiming } from 'react-native-reanimated';
import { Button, View } from 'react-native';

export default function SharedValueDemo(){
  const progress = useSharedValue(0);
  const style = useAnimatedStyle(() => ({ transform: [{ scale: 1 + progress.value }] }));
  return (
    <View style={{ padding: 16 }}>
      <Animated.View style={[{ width: 60, height: 60, backgroundColor: '#999' }, style]} />
      <Button title="Animate" onPress={() => { progress.value = withTiming(1, {}, () => { progress.value = 0; }); }} />
    </View>
  );
}
```
## useAnimatedStyle

Return styles based on shared values.

```tsx
// see example above
```
## useDerivedValue

Compute a value from other shared values.

```tsx
import { useSharedValue, useDerivedValue } from 'react-native-reanimated';
const a = useSharedValue(1);
const b = useDerivedValue(() => a.value * 2);
```
## useAnimatedProps

Animate props like strokeDashoffset or text.

```tsx
import Animated, { useSharedValue, useAnimatedProps, withTiming } from 'react-native-reanimated';
import { Text as RNText } from 'react-native';
const AnimatedText = Animated.createAnimatedComponent(RNText);
const val = useSharedValue(0);
const props = useAnimatedProps(() => ({ text: String(val.value) })); // library-specific components
support

```
## useAnimatedScrollHandler

Respond to scroll events on Animated.ScrollView.

```tsx
import { useAnimatedScrollHandler } from 'react-native-reanimated';
const onScroll = useAnimatedScrollHandler({
  onScroll: (e) => {
    // e.contentOffset.x / y
  },
});
```
Reanimated flow

```
Update sharedValue -> UI thread computes worklet -> frame renders
+------------+   +-------------+   +------------+
| JS thread  |-> | Worklet UI  |-> | Render     |
+------------+   +-------------+   +------------+
```

Safe Area Hooks

Hooks from react-native-safe-area-context.

## useSafeAreaInsets

Read safe area padding values.

```tsx
import { useSafeAreaInsets } from 'react-native-safe-area-context';
const insets = useSafeAreaInsets();
```
## useSafeAreaFrame

Read layout frame excluding safe areas.

React Native Hooks - Complete Guide

```tsx
import { useSafeAreaFrame } from 'react-native-safe-area-context';
const frame = useSafeAreaFrame();
```

React Query Hooks

Hooks from @tanstack/react-query.

## useQuery

Fetch and cache data.

```tsx
import { useQuery } from '@tanstack/react-query';
function useUser(userId){
  return useQuery({ queryKey: ['user', userId], queryFn: () =>
fetch('https://example.com/u/'+userId).then(r=>r.json()) });
}
```
## useMutation

Perform POST/PUT/DELETE with cache updates.

```tsx
import { useMutation, useQueryClient } from '@tanstack/react-query';
const qc = useQueryClient();
const mutation = useMutation({
  mutationFn: (payload) => fetch('/api', { method: 'POST', body: JSON.stringify(payload) }),
  onSuccess: () => qc.invalidateQueries({ queryKey: ['user'] }),
});
```
## useInfiniteQuery

Paginated or infinite lists.

```tsx
import { useInfiniteQuery } from '@tanstack/react-query';
const q = useInfiniteQuery({
  queryKey: ['feed'],
  queryFn: ({ pageParam = 0 }) => fetch('/api?page='+pageParam).then(r=>r.json()),
  getNextPageParam: (lastPage) => lastPage.nextPage,
});
```
## useIsFetching

Global is fetching count.

```tsx
import { useIsFetching } from '@tanstack/react-query';
const isFetching = useIsFetching();
```

~~~~
```
~~~~

Expo Hooks

~~~~
Common Expo SDK hooks.

## useFonts

Load fonts before rendering UI.

```tsx
import { useFonts } from 'expo-font';
const [fontsLoaded] = useFonts({ Inter: require('./assets/Inter.ttf') });
```
## useUpdates

Listen and apply OTA updates (expo-updates).

```tsx
import { useUpdates } from 'expo-updates';
const { isUpdateAvailable, isUpdatePending, checkForUpdateAsync, downloadUpdateAsync, reloadAsync
}
= useUpdates();
```
## useCameraPermissions

Camera permission hook (expo-camera).

```tsx
import { Camera, useCameraPermissions } from 'expo-camera';
const [permission, requestPermission] = useCameraPermissions();
```
~~~~

Custom Hooks

~~~~
Examples of reusable custom hooks.

## useDebounce
Delays a changing value until a period of inactivity.
```tsx
import React from 'react';
export function useDebounce(value, delay) {
  const [debounced, setDebounced] = React.useState(value);
  React.useEffect(() => {
    const id = setTimeout(() => setDebounced(value), delay);
    return () => clearTimeout(id);
  }, [value, delay]);
  return debounced;
}
```
## useInterval
~~~~

React Native Hooks - Complete Guide

Run a callback every N ms.

```tsx
import React from 'react';
export function useInterval(cb, ms) {
  const saved = React.useRef(cb);
  React.useEffect(() => { saved.current = cb; }, [cb]);
  React.useEffect(() => {
    if (ms == null) return;
    const id = setInterval(() => saved.current(), ms);
    return () => clearInterval(id);
  }, [ms]);
}
```

Summary Table

Quick reference of hooks and their source.

React core:
useState, useEffect, useContext, useReducer, useRef, useImperativeHandle, useLayoutEffect, useCallback, useMemo, useDebugValue, useDeferredValue, useTransition, useId, useSyncExternalStore, useInsertionEffect

React Native:
useWindowDimensions, useColorScheme

React Navigation:
useNavigation, useRoute, useFocusEffect, useIsFocused

Reanimated:
useSharedValue, useAnimatedStyle, useDerivedValue, useAnimatedProps, useAnimatedScrollHandler

Safe Area:
useSafeAreaInsets, useSafeAreaFrame

React Query:
useQuery, useMutation, useInfiniteQuery, useIsFetching

Expo:
useFonts, useUpdates, useCameraPermissions