InteractionManager

InteractionManager allows long-running work to be scheduled after any interactions/animations have completed. In particular, this allows JavaScript animations to run smoothly.

Applications can schedule tasks to run after interactions with the following:

```
InteractionManager.runAfterInteractions(() => {
  // ...long-running synchronous task...
});
```

Compare this to other scheduling alternatives:

- `requestAnimationFrame()` for code that animates a view over time.
- `setImmediate/setTimeout()` run code later, note this may delay animations.
- `runAfterInteractions()` run code later, without delaying active animations.

The touch handling system considers one or more active touches to be an 'interaction' and will delay `runAfterInteractions()` callbacks until all touches have ended or been cancelled.

InteractionManager also allows applications to register animations by creating an interaction 'handle' on animation start, and clearing it upon completion:

```
const handle = InteractionManager.createInteractionHandle();
// run animation... (`runAfterInteractions` tasks are queued)
// later, on animation completion:
InteractionManager.clearInteractionHandle(handle);
// queued tasks run if all handles were cleared
```

`runAfterInteractions` takes either a plain callback function, or a `PromiseTask` object with a `gen` method that returns a `Promise`. If a `PromiseTask` is supplied, then it is fully resolved (including asynchronous dependencies that also schedule more tasks via

`runAfterInteractions`) before starting on the next task that might have been queued up synchronously earlier.

By default, queued tasks are executed together in a loop in one `setImmediate` batch. If `setDeadline` is called with a positive number, then tasks will only be executed until the deadline (in terms of js event loop run time) approaches, at which point execution will yield via setTimeout, allowing events such as touches to start interactions and block queued tasks from executing, making apps more responsive.

Example

Basic

TypeScript JavaScript

Advanced

TypeScript JavaScript

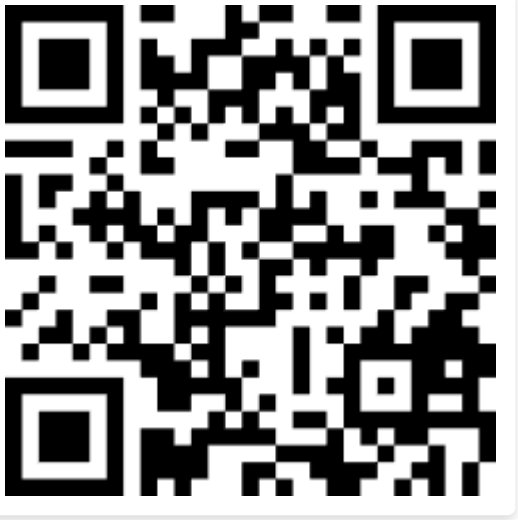
InteractionManager Function Component Advanced Example

Expo

```
import React, {useEffect} from 'react';
import {
  Alert,
  Animated,
  InteractionManager,
  Platform,
  StyleSheet,
  Text,
  View,
} from 'react-native';

const instructions = Platform.select({
  ios: 'Press Cmd+R to reload,\n' + 'Cmd+D or shake for dev
menu',
  android:
    'Double tap R on your keyboard to reload,\n' +
    'Shake or press menu button for dev menu',
});

// You can create a custom interaction/animation and add
// support for InteractionManager
const useCustomInteraction = (timeLocked = 2000) => {
  useEffect(() => {
    const handle =
InteractionManager.createInteractionHandle();
```



0

Log in or set a Device ID to open this Snack from Expo Go on your device or simulator.

Preview My Device iOS Android

> **Note**: `InteractionManager.runAfterInteractions()` is not working properly on web. It triggers immediately without waiting until the interaction is finished.

Reference

Methods

`runAfterInteractions()`

```
static runAfterInteractions(task?: (() => any) | SimpleTask | PromiseTask);
```

Schedule a function to run after all interactions have completed. Returns a cancellable "promise".

`createInteractionHandle()`

```
static createInteractionHandle(): Handle;
```

Notify manager that an interaction has started.

`clearInteractionHandle()`

```
static clearInteractionHandle(handle: Handle);
```

Notify manager that an interaction has completed.

`setDeadline()`

```
static setDeadline(deadline: number);
```

A positive number will use setTimeout to schedule any tasks after the eventLoopRunningTime hits the deadline value, otherwise all tasks will be executed in one setImmediate batch (default).

*Last updated on **Jun 21, 2023***