AccessibilityInfo

Sometimes it's useful to know whether or not the device has a screen reader that is currently active. The `AccessibilityInfo` API is designed for this purpose. You can use it to query the current state of the screen reader as well as to register to be notified when the state of the screen reader changes.

Example

AccessibilityInfo Example

Expo

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

const App = () => {
  const [reduceMotionEnabled, setReduceMotionEnabled] =
useState(false);
  const [screenReaderEnabled, setScreenReaderEnabled] =
useState(false);

  useEffect(() => {
    const reduceMotionChangedSubscription =
AccessibilityInfo.addEventListener(
      'reduceMotionChanged',
      isReduceMotionEnabled => {
        setReduceMotionEnabled(isReduceMotionEnabled);
      },
    );
    const screenReaderChangedSubscription =
AccessibilityInfo.addEventListener(
      'screenReaderChanged',
      isScreenReaderEnabled => {
        setScreenReaderEnabled(isScreenReaderEnabled);
      },
    );
```

Download Expo Go and scan the QR code to get started.

Connected devices 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

Reference

Methods

`addEventListener()`

```
static addEventListener(
  eventName: AccessibilityChangeEventName | AccessibilityAnnouncementEventName,
  handler: (
    event: AccessibilityChangeEvent | AccessibilityAnnouncementFinishedEvent,
  ) => void,
): EmitterSubscription;
```

Add an event handler. Supported events:

EVENT NAME	DESCRIPTION
`accessibilityServiceChanged` Android	Fires when some services such as TalkBack, other Android assistive technologies, and third-party accessibility services are enabled. The argument to the event handler is a boolean. The boolean is `true` when a some accessibility services is enabled and `false` otherwise.
`announcementFinished` iOS	Fires when the screen reader has finished making an announcement. The argument to the event handler is a dictionary with these keys: • `announcement`: The string announced by the screen reader. • `success`: A boolean indicating whether the announcement was successfully made.
`boldTextChanged` iOS	Fires when the state of the bold text toggle changes. The argument to the event handler is a boolean. The boolean is `true` when bold text is enabled and `false` otherwise.
`grayscaleChanged` iOS	Fires when the state of the gray scale toggle changes. The argument to the event handler is a boolean. The boolean is `true` when a gray scale is enabled and `false` otherwise.
`invertColorsChanged` iOS	Fires when the state of the invert colors toggle changes. The argument to the event handler is a boolean. The boolean is `true` when invert colors is enabled and `false` otherwise.
`reduceMotionChanged`	Fires when the state of the reduce motion toggle changes. The argument to the event handler is a boolean. The boolean is `true`

EVENT NAME	DESCRIPTION
	when a reduce motion is enabled (or when "Transition Animation Scale" in "Developer options" is "Animation off") and `false` otherwise.
`reduceTransparencyChanged` iOS	Fires when the state of the reduce transparency toggle changes. The argument to the event handler is a boolean. The boolean is `true` when reduce transparency is enabled and `false` otherwise.
`screenReaderChanged`	Fires when the state of the screen reader changes. The argument to the event handler is a boolean. The boolean is `true` when a screen reader is enabled and `false` otherwise.

`announceForAccessibility()`

```
static announceForAccessibility(announcement: string);
```

Post a string to be announced by the screen reader.

`announceForAccessibilityWithOptions()`

```
static announceForAccessibilityWithOptions(
  announcement: string,
  options: options: {queue?: boolean},
);
```

Post a string to be announced by the screen reader with modification options. By default announcements will interrupt any existing speech, but on iOS they can be queued behind existing speech by setting `queue` to `true` in the options object.

Parameters:

NAME	TYPE	DESCRIPTION
announcement Required	string	The string to be announced
options Required	object	`queue` - queue the announcement behind existing speech iOS

`getRecommendedTimeoutMillis()` Android

```
static getRecommendedTimeoutMillis(originalTimeout: number): Promise<number>;
```

Gets the timeout in millisecond that the user needs. This value is set in "Time to take action (Accessibility timeout)" of "Accessibility" settings.

Parameters:

NAME	TYPE	DESCRIPTION
originalTimeout Required	number	The timeout to return if "Accessibility timeout" is not set. Specify in milliseconds.

`isAccessibilityServiceEnabled()` Android

```
static isAccessibilityServiceEnabled(): Promise<boolean>;
```

Check whether any accessibility service is enabled. This includes TalkBack but also any third-party accessibility app that may be installed. To only check whether TalkBack is enabled, use isScreenReaderEnabled. Returns a promise which resolves to a boolean. The result is `true` when some accessibility services is enabled and `false` otherwise.

> **Note**: Please use isScreenReaderEnabled if you only want to check the status of TalkBack.

`isBoldTextEnabled()` ◀ iOS

```
static isBoldTextEnabled(): Promise<boolean>:
```

Query whether a bold text is currently enabled. Returns a promise which resolves to a boolean. The result is `true` when bold text is enabled and `false` otherwise.

`isGrayscaleEnabled()` ◀ iOS

```
static isGrayscaleEnabled(): Promise<boolean>;
```

Query whether grayscale is currently enabled. Returns a promise which resolves to a boolean. The result is `true` when grayscale is enabled and `false` otherwise.

`isInvertColorsEnabled()` ◀ iOS

```
static isInvertColorsEnabled(): Promise<boolean>;
```

Query whether invert colors is currently enabled. Returns a promise which resolves to a boolean. The result is `true` when invert colors is enabled and `false` otherwise.

`isReduceMotionEnabled()`

```
static isReduceMotionEnabled(): Promise<boolean>;
```

Query whether reduce motion is currently enabled. Returns a promise which resolves to a boolean. The result is `true` when reduce motion is enabled and `false` otherwise.

`isReduceTransparencyEnabled()` iOS

```
static isReduceTransparencyEnabled(): Promise<boolean>;
```

Query whether reduce transparency is currently enabled. Returns a promise which resolves to a boolean. The result is `true` when a reduce transparency is enabled and `false` otherwise.

`isScreenReaderEnabled()`

```
static isScreenReaderEnabled(): Promise<boolean>;
```

Query whether a screen reader is currently enabled. Returns a promise which resolves to a boolean. The result is `true` when a screen reader is enabled and `false` otherwise.

`prefersCrossFadeTransitions()` iOS

```
static prefersCrossFadeTransitions(): Promise<boolean>;
```

Query whether reduce motion and prefer cross-fade transitions settings are currently enabled. Returns a promise which resolves to a boolean. The result is `true` when prefer cross-fade transitions is enabled and `false` otherwise.

`setAccessibilityFocus()`

```
static setAccessibilityFocus(reactTag: number);
```

Set accessibility focus to a React component.

On Android, this calls `UIManager.sendAccessibilityEvent` method with passed `reactTag` and `UIManager.AccessibilityEventTypes.typeViewFocused` arguments.

> **Note**: Make sure that any `View` you want to receive the accessibility focus has `accessible={true}`.

Is this page useful?

Edit this page

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