TouchableWithoutFeedback

> If you're looking for a more extensive and future-proof way to handle touch-based input, check out the Pressable API.

Do not use unless you have a very good reason. All elements that respond to press should have a visual feedback when touched.

`TouchableWithoutFeedback` supports only one child. If you wish to have several child components, wrap them in a View. Importantly, `TouchableWithoutFeedback` works by cloning its child and applying responder props to it. It is therefore required that any intermediary components pass through those props to the underlying React Native component.

Usage Pattern

```
function MyComponent(props: MyComponentProps) {
  return (
    <View {...props} style={{flex: 1, backgroundColor: '#fff'}}>
      <Text>My Component</Text>
    </View>
  );
}

<TouchableWithoutFeedback onPress={() => alert('Pressed!')}>
  <MyComponent />
</TouchableWithoutFeedback>;
```

Example

TouchableWithoutFeedback Expo

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

const TouchableWithoutFeedbackExample = () => {
  const [count, setCount] = useState(0);

  const onPress = () => {
    setCount(count + 1);
  };

  return (
    <View style={styles.container}>
      <View style={styles.countContainer}>
        <Text style={styles.countText}>Count: {count}</Text>
      </View>
      <TouchableWithoutFeedback onPress={onPress}>
        <View style={styles.button}>
          <Text>Touch Here</Text>
        </View>
      </TouchableWithoutFeedback>
    </View>
  );
};

const styles = StyleSheet.create({
```

Preview My Device iOS Android Web

Reference

Props

`accessibilityIgnoresInvertColors`

TYPE
Boolean

`accessible`

When `true`, indicates that the view is an accessibility element. By default, all the touchable elements are accessible.

TYPE
bool

`accessibilityLabel`

Overrides the text that's read by the screen reader when the user interacts with the element. By default, the label is constructed by traversing all the children and accumulating all the `Text` nodes separated by space.

TYPE
string

`accessibilityLanguage` iOS

A value indicating which language should be used by the screen reader when the user interacts with the element. It should follow the BCP 47 specification.

See the iOS `accessibilityLanguage` doc for more information.

TYPE
string

`accessibilityHint`

An accessibility hint helps users understand what will happen when they perform an action on the accessibility element when that result is not clear from the accessibility label.

TYPE
string

`accessibilityRole`

`accessibilityRole` communicates the purpose of a component to the user of an assistive technology.

`accessibilityRole` can be one of the following:

- `'none'` - Used when the element has no role.
- `'button'` - Used when the element should be treated as a button.
- `'link'` - Used when the element should be treated as a link.
- `'search'` - Used when the text field element should also be treated as a search field.
- `'image'` - Used when the element should be treated as an image. Can be combined with button or link, for example.
- `'keyboardkey'` - Used when the element acts as a keyboard key.
- `'text'` - Used when the element should be treated as static text that cannot change.
- `'adjustable'` - Used when an element can be "adjusted" (e.g. a slider).
- `'imagebutton'` - Used when the element should be treated as a button and is also an image.
- `'header'` - Used when an element acts as a header for a content section (e.g. the title of a navigation bar).
- `'summary'` - Used when an element can be used to provide a quick summary of current conditions in the app when the app first launches.
- `'alert'` - Used when an element contains important text to be presented to the user.
- `'checkbox'` - Used when an element represents a checkbox which can be checked, unchecked, or have mixed checked state.
- `'combobox'` - Used when an element represents a combo box, which allows the user to select among several choices.
- `'menu'` - Used when the component is a menu of choices.

- `'menubar'` - Used when a component is a container of multiple menus.
- `'menuitem'` - Used to represent an item within a menu.
- `'progressbar'` - Used to represent a component which indicates progress of a task.
- `'radio'` - Used to represent a radio button.
- `'radiogroup'` - Used to represent a group of radio buttons.
- `'scrollbar'` - Used to represent a scroll bar.
- `'spinbutton'` - Used to represent a button which opens a list of choices.
- `'switch'` - Used to represent a switch which can be turned on and off.
- `'tab'` - Used to represent a tab.
- `'tablist'` - Used to represent a list of tabs.
- `'timer'` - Used to represent a timer.
- `'toolbar'` - Used to represent a tool bar (a container of action buttons or components).

TYPE
string

`accessibilityState`

Describes the current state of a component to the user of an assistive technology.

See the Accessibility guide for more information.

TYPE
object: `{disabled: bool, selected: bool, checked: bool or 'mixed', busy: bool, expanded: bool}`

`accessibilityActions`

Accessibility actions allow an assistive technology to programmatically invoke the actions of a component. The `accessibilityActions` property should contain a list of action objects. Each action object should contain the field name and label.

See the Accessibility guide for more information.

TYPE
array

`aria-busy`

Indicates an element is being modified and that assistive technologies may want to wait until the changes are complete before informing the user about the update.

TYPE	DEFAULT
boolean	false

`aria-checked`

Indicates the state of a checkable element. This field can either take a boolean or the "mixed" string to represent mixed checkboxes.

TYPE	DEFAULT
boolean, 'mixed'	false

`aria-disabled`

Indicates that the element is perceivable but disabled, so it is not editable or otherwise operable.

TYPE	DEFAULT
boolean	false

`aria-expanded`

Indicates whether an expandable element is currently expanded or collapsed.

TYPE	DEFAULT
boolean	false

`aria-hidden`

Indicates whether the accessibility elements contained within this accessibility element are hidden.

For example, in a window that contains sibling views `A` and `B`, setting `aria-hidden` to `true` on view `B` causes VoiceOver to ignore the elements in the view `B`.

TYPE	DEFAULT
boolean	false

`aria-label`

Defines a string value that labels an interactive element.

TYPE
string

`aria-live` Android

Indicates that an element will be updated, and describes the types of updates the user agents, assistive technologies, and user can expect from the live region.

- **off** Accessibility services should not announce changes to this view.

- **polite** Accessibility services should announce changes to this view.
- **assertive** Accessibility services should interrupt ongoing speech to immediately announce changes to this view.

TYPE	DEFAULT
enum(`'assertive'`, `'off'`, `'polite'`)	`'off'`

`aria-modal` iOS

Boolean value indicating whether VoiceOver should ignore the elements within views that are siblings of the receiver. Has precedence over the `accessibilityViewIsModal` prop.

TYPE	DEFAULT
boolean	false

`aria-selected`

Indicates whether a selectable element is currently selected or not.

TYPE
boolean

`onAccessibilityAction`

Invoked when the user performs the accessibility actions. The only argument to this function is an event containing the name of the action to perform.

See the Accessibility guide for more information.

TYPE
function

`accessibilityValue`

Represents the current value of a component. It can be a textual description of a component's value, or for range-based components, such as sliders and progress bars, it contains range information (minimum, current, and maximum).

See the Accessibility guide for more information.

TYPE
object: `{min: number, max: number, now: number, text: string}`

`aria-valuemax`

Represents the maximum value for range-based components, such as sliders and progress bars. Has precedence over the `max` value in the `accessibilityValue` prop.

TYPE
number

`aria-valuemin`

Represents the minimum value for range-based components, such as sliders and progress bars. Has precedence over the `min` value in the `accessibilityValue` prop.

TYPE
number

`aria-valuenow`

Represents the current value for range-based components, such as sliders and progress bars. Has precedence over the `now` value in the `accessibilityValue` prop.

TYPE
number

`aria-valuetext`

Represents the textual description of the component. Has precedence over the `text` value in the `accessibilityValue` prop.

TYPE
string

`delayLongPress`

Duration (in milliseconds) from `onPressIn` before `onLongPress` is called.

TYPE
number

`delayPressIn`

Duration (in milliseconds), from the start of the touch, before `onPressIn` is called.

TYPE
number

`delayPressOut`

Duration (in milliseconds), from the release of the touch, before `onPressOut` is called.

TYPE
number

`disabled`

If true, disable all interactions for this component.

TYPE
bool

`hitSlop`

This defines how far your touch can start away from the button. This is added to `pressRetentionOffset` when moving off of the button.

> The touch area never extends past the parent view bounds and the Z-index of sibling views always takes precedence if a touch hits two overlapping views.

TYPE
Rect or number

`id`

Used to locate this view from native code. Has precedence over `nativeID` prop.

TYPE
string

`onBlur`

Invoked when the item loses focus.

TYPE
function

`onFocus`

Invoked when the item receives focus.

TYPE
function

`onLayout`

Invoked on mount and on layout changes.

TYPE
`({nativeEvent: LayoutEvent}) => void`

`onLongPress`

Called if the time after `onPressIn` lasts longer than 370 milliseconds. This time period can be customized with `delayLongPress` .

TYPE
function

`onPress`

Called when the touch is released, but not if cancelled (e.g. by a scroll that steals the responder lock). The first function argument is an event in form of PressEvent.

TYPE
function

`onPressIn`

Called as soon as the touchable element is pressed and invoked even before onPress. This can be useful when making network requests. The first function argument is an event in form of PressEvent.

TYPE
function

`onPressOut`

Called as soon as the touch is released even before onPress. The first function argument is an event in form of PressEvent.

TYPE
function

`pressRetentionOffset`

When the scroll view is disabled, this defines how far your touch may move off of the button, before deactivating the button. Once deactivated, try moving it back and you'll see that the button is once again reactivated! Move it back and forth several times while the scroll view is disabled. Ensure you pass in a constant to reduce memory allocations.

TYPE
Rect or number

`nativeID`

TYPE
string

`testID`

Used to locate this view in end-to-end tests.

TYPE
string

`touchSoundDisabled` Android

If true, doesn't play a system sound on touch.

TYPE
Boolean

Is this page useful?

Edit this page

Last updated on ***Aug 17, 2023***