TextInput

A foundational component for inputting text into the app via a keyboard. Props provide configurability for several features, such as auto-correction, auto-capitalization, placeholder text, and different keyboard types, such as a numeric keypad.

The most basic use case is to plop down a `TextInput` and subscribe to the `onChangeText` events to read the user input. There are also other events, such as `onSubmitEditing` and `onFocus` that can be subscribed to. A minimal example:

TextInput

Expo

```
import React from 'react';
import {SafeAreaView, StyleSheet, TextInput} from 'react-
native';

const TextInputExample = () => {
  const [text, onChangeText] = React.useState('Useless
Text');
  const [number, onChangeNumber] = React.useState('');

  return (
    <SafeAreaView>
      <TextInput
        style={styles.input}
        onChangeText={onChangeText}
        value={text}
      />
      <TextInput
        style={styles.input}
        onChangeText={onChangeNumber}
        value={number}
        placeholder="useless placeholder"
        keyboardType="numeric"
      />
    </SafeAreaView>
  );
};
```

Preview My Device iOS Android Web

Two methods exposed via the native element are .focus() and .blur() that will focus or blur the TextInput programmatically.

Note that some props are only available with `multiline={true/false}`. Additionally, border styles that apply to only one side of the element (e.g., `borderBottomColor`, `borderLeftWidth`, etc.) will not be applied if `multiline=true`. To achieve the same effect, you can wrap your `TextInput` in a `View`:

TextInput

Expo

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

const MultilineTextInputExample = () => {
  const [value, onChangeText] = React.useState('Useless
Multiline Placeholder');

  // If you type something in the text box that is a color,
the background will change to that
  // color.
  return (
    <View
      style={{
        backgroundColor: value,
        borderBottomColor: '#000000',
        borderBottomWidth: 1,
      }}>
      <TextInput
        editable
        multiline
        numberOfLines={4}
        maxLength={40}
        onChangeText={text => onChangeText(text)}
        value={value}
        style={{padding: 10}}
      />
```

Preview My Device iOS Android Web

`TextInput` has by default a border at the bottom of its view. This border has its padding set by the background image provided by the system, and it cannot be changed. Solutions to avoid this are to either not set height explicitly, in which case the system will take care of displaying the border in the correct position, or to not display the border by setting `underlineColorAndroid` to transparent.

Note that on Android performing text selection in an input can change the app's activity `windowSoftInputMode` param to `adjustResize`. This may cause issues with components that have position: 'absolute' while the keyboard is active. To avoid this behavior either specify `windowSoftInputMode` in AndroidManifest.xml (

https://developer.android.com/guide/topics/manifest/activity-element.html) or control this param programmatically with native code.

Reference

Props

View Props

Inherits View Props.

`allowFontScaling`

Specifies whether fonts should scale to respect Text Size accessibility settings. The default is `true`.

TYPE
bool

`autoCapitalize`

Tells `TextInput` to automatically capitalize certain characters. This property is not supported by some keyboard types such as `name-phone-pad`.

- `characters`: all characters.
- `words`: first letter of each word.
- `sentences`: first letter of each sentence (*default*).
- `none`: don't auto capitalize anything.

TYPE
enum('none', 'sentences', 'words', 'characters')

`autoComplete`

Specifies autocomplete hints for the system, so it can provide autofill. On Android, the system will always attempt to offer autofill by using heuristics to identify the type of content. To disable autocomplete, set `autoComplete` to `off`.

The following values work across platforms:

- `additional-name`
- `address-line1`
- `address-line2`
- `cc-number`
- `country`
- `current-password`
- `email`
- `family-name`
- `given-name`
- `honorific-prefix`
- `honorific-suffix`
- `name`
- `new-password`
- `off`
- `one-time-code`
- `postal-code`
- `street-address`
- `tel`
- `username`

iOS

The following values work on iOS only:

- `nickname`
- `organization`
- `organization-title`
- `url`

Android

The following values work on Android only:

- `birthdate-day`
- `birthdate-full`
- `birthdate-month`
- `birthdate-year`
- `cc-csc`
- `cc-exp`
- `cc-exp-day`
- `cc-exp-month`
- `cc-exp-year`
- `gender`
- `name-family`
- `name-given`
- `name-middle`
- `name-middle-initial`
- `name-prefix`
- `name-suffix`
- `password`
- `password-new`
- `postal-address`
- `postal-address-country`

- `postal-address-extended`
- `postal-address-extended-postal-code`
- `postal-address-locality`
- `postal-address-region`
- `sms-otp`
- `tel-country-code`
- `tel-national`
- `tel-device`
- `username-new`

TYPE
enum('additional-name', 'address-line1', 'address-line2', 'birthdate-day', 'birthdate-full', 'birthdate-month', 'birthdate-year', 'cc-csc', 'cc-exp', 'cc-exp-day', 'cc-exp-month', 'cc-exp-year', 'cc-number', 'country', 'current-password', 'email', 'family-name', 'gender', 'given-name', 'honorific-prefix', 'honorific-suffix', 'name', 'name-family', 'name-given', 'name-middle', 'name-middle-initial', 'name-prefix', 'name-suffix', 'new-password', 'nickname', 'one-time-code', 'organization', 'organization-title', 'password', 'password-new', 'postal-address', 'postal-address-country', 'postal-address-extended', 'postal-address-extended-postal-code', 'postal-address-locality', 'postal-address-region', 'postal-code', 'street-address', 'sms-otp', 'tel', 'tel-country-code', 'tel-national', 'tel-device', 'url', 'username', 'username-new', 'off')

`autoCorrect`

If `false`, disables auto-correct. The default value is `true`.

TYPE
bool

`autoFocus`

If `true`, focuses the input on `componentDidMount` or `useEffect`. The default value is `false`.

TYPE
bool

`blurOnSubmit`

If `true`, the text field will blur when submitted. The default value is true for single-line fields and false for multiline fields. Note that for multiline fields, setting `blurOnSubmit` to `true` means that pressing return will blur the field and trigger the `onSubmitEditing` event instead of inserting a newline into the field.

TYPE
bool

`caretHidden`

If `true`, caret is hidden. The default value is `false`.

TYPE
bool

`clearButtonMode` iOS

When the clear button should appear on the right side of the text view. This property is supported only for single-line TextInput component. The default value is `never`.

TYPE
enum('never', 'while-editing', 'unless-editing', 'always')

`clearTextOnFocus` iOS

If `true`, clears the text field automatically when editing begins.

TYPE
bool

`contextMenuHidden`

If `true`, context menu is hidden. The default value is `false`.

TYPE
bool

`dataDetectorTypes` iOS

Determines the types of data converted to clickable URLs in the text input. Only valid if `multiline={true}` and `editable={false}`. By default no data types are detected.

You can provide one type or an array of many types.

Possible values for `dataDetectorTypes` are:

- `'phoneNumber'`
- `'link'`
- `'address'`
- `'calendarEvent'`
- `'none'`
- `'all'`

TYPE
enum('phoneNumber', 'link', 'address', 'calendarEvent', 'none', 'all'), ,array of enum('phoneNumber', 'link', 'address', 'calendarEvent', 'none', 'all')

`defaultValue`

Provides an initial value that will change when the user starts typing. Useful for use-cases where you do not want to deal with listening to events and updating the value prop to keep the controlled state in sync.

TYPE
string

`cursorColor` Android

When provided it will set the color of the cursor (or "caret") in the component. Unlike the behavior of `selectionColor` the cursor color will be set independently from the color of the text selection box.

TYPE
color

`disableFullscreenUI` Android

When `false`, if there is a small amount of space available around a text input (e.g. landscape orientation on a phone), the OS may choose to have the user edit the text inside of a full screen text input mode. When `true`, this feature is disabled and users will always edit the text directly inside of the text input. Defaults to `false`.

TYPE
bool

`editable`

If `false`, text is not editable. The default value is `true`.

TYPE
bool

`enablesReturnKeyAutomatically` iOS

If `true`, the keyboard disables the return key when there is no text and automatically enables it when there is text. The default value is `false`.

TYPE
bool

`enterKeyHint`

Determines what text should be shown to the return key. Has precedence over the `returnKeyType` prop.

The following values work across platforms:

- `enter`
- `done`
- `next`
- `search`
- `send`

Android Only

The following values work on Android only:

- `previous`

TYPE
enum('enter', 'done', 'next', 'previous', 'search', 'send')

`importantForAutofill` Android

Tells the operating system whether the individual fields in your app should be included in a view structure for autofill purposes on Android API Level 26+. Possible values are `auto`, `no`, `noExcludeDescendants`, `yes`, and `yesExcludeDescendants`. The default value is `auto`.

- `auto`: Let the Android System use its heuristics to determine if the view is important for autofill.
- `no`: This view isn't important for autofill.
- `noExcludeDescendants`: This view and its children aren't important for autofill.
- `yes`: This view is important for autofill.
- `yesExcludeDescendants`: This view is important for autofill, but its children aren't important for autofill.

TYPE
enum('auto', 'no', 'noExcludeDescendants', 'yes', 'yesExcludeDescendants')

`inlineImageLeft` Android

If defined, the provided image resource will be rendered on the left. The image resource must be inside `/android/app/src/main/res/drawable` and referenced like

```
<TextInput
 inlineImageLeft='search_icon'
/>
```

TYPE
string

`inlineImagePadding` Android

Padding between the inline image, if any, and the text input itself.

TYPE
number

`inputAccessoryViewID` iOS

An optional identifier which links a custom InputAccessoryView to this text input. The InputAccessoryView is rendered above the keyboard when this text input is focused.

TYPE
string

`inputMode`

Works like the `inputmode` attribute in HTML, it determines which keyboard to open, e.g. `numeric` and has precedence over `keyboardType`.

Support the following values:

- `none`
- `text`
- `decimal`
- `numeric`
- `tel`
- `search`
- `email`
- `url`

TYPE
enum('decimal', 'email', 'none', 'numeric', 'search', 'tel', 'text', 'url')

`keyboardAppearance` iOS

Determines the color of the keyboard.

TYPE
enum('default', 'light', 'dark')

`keyboardType`

Determines which keyboard to open, e.g. `numeric` .

See screenshots of all the types here.

The following values work across platforms:

- `default`
- `number-pad`
- `decimal-pad`
- `numeric`
- `email-address`
- `phone-pad`
- `url`

iOS Only

The following values work on iOS only:

- `ascii-capable`
- `numbers-and-punctuation`
- `name-phone-pad`
- `twitter`
- `web-search`

Android Only

The following values work on Android only:

- `visible-password`

TYPE
enum('default', 'email-address', 'numeric', 'phone-pad', 'ascii-capable', 'numbers-and-punctuation', 'url', 'number-pad', 'name-phone-pad', 'decimal-pad', 'twitter', 'web-search', 'visible-password')

`maxFontSizeMultiplier`

Specifies largest possible scale a font can reach when `allowFontScaling` is enabled. Possible values:

- `null/undefined` (default): inherit from the parent node or the global default (0)
- `0`: no max, ignore parent/global default
- `>= 1`: sets the `maxFontSizeMultiplier` of this node to this value

TYPE
number

`maxLength`

Limits the maximum number of characters that can be entered. Use this instead of implementing the logic in JS to avoid flicker.

TYPE
number

`multiline`

If `true`, the text input can be multiple lines. The default value is `false`.

> **NOTE**
>
> It is important to note that this aligns the text to the top on iOS, and centers it on Android. Use with `textAlignVertical` set to `top` for the same behavior in both platforms.

TYPE
bool

`numberOfLines` Android

Sets the number of lines for a `TextInput`. Use it with multiline set to `true` to be able to fill the lines.

TYPE
number

`onBlur`

Callback that is called when the text input is blurred.

> Note: If you are attempting to access the `text` value from `nativeEvent` keep in mind that the resulting value you get can be `undefined` which can cause unintended errors. If you are trying to find the last value of TextInput, you can use the `onEndEditing` event, which is fired upon completion of editing.

TYPE
function

onChange

Callback that is called when the text input's text changes.

TYPE
`({nativeEvent: {eventCount, target, text}})` => void

onChangeText

Callback that is called when the text input's text changes. Changed text is passed as a single string argument to the callback handler.

TYPE
function

onContentSizeChange

Callback that is called when the text input's content size changes.

Only called for multiline text inputs.

TYPE
`({nativeEvent: {contentSize: {width, height} }})` => void

onEndEditing

Callback that is called when text input ends.

TYPE
function

onPressIn

Callback that is called when a touch is engaged.

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

onPressOut

Callback that is called when a touch is released.

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

onFocus

Callback that is called when the text input is focused.

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

onKeyPress

Callback that is called when a key is pressed. This will be called with object where `keyValue` is `'Enter'` or `'Backspace'` for respective keys and the typed-in character otherwise including `' '` for space. Fires before `onChange` callbacks. Note: on Android only the inputs from soft keyboard are handled, not the hardware keyboard inputs.

TYPE
`({nativeEvent: {key: keyValue} }) =>` void

onLayout

Invoked on mount and on layout changes.

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

onScroll

Invoked on content scroll. May also contain other properties from `ScrollEvent` but on Android `contentSize` is not provided for performance reasons.

TYPE
`({nativeEvent: {contentOffset: {x, y} }}) =>` void

onSelectionChange

Callback that is called when the text input selection is changed.

TYPE
`({nativeEvent: {selection: {start, end} }}) =>` void

onSubmitEditing

Callback that is called when the text input's submit button is pressed.

TYPE
`({nativeEvent: {text, eventCount, target}}) =>` void

Note that on iOS this method isn't called when using `keyboardType="phone-pad"`.

`placeholder`

The string that will be rendered before text input has been entered.

TYPE
string

`placeholderTextColor`

The text color of the placeholder string.

TYPE
color

`readOnly`

If `true`, text is not editable. The default value is `false`.

TYPE
bool

`returnKeyLabel` Android

Sets the return key to the label. Use it instead of `returnKeyType`.

TYPE
string

`returnKeyType`

Determines how the return key should look. On Android you can also use `returnKeyLabel`.

Cross platform

The following values work across platforms:

- `done`
- `go`
- `next`
- `search`
- `send`

Android Only

The following values work on Android only:

- `none`
- `previous`

iOS Only

The following values work on iOS only:

- `default`
- `emergency-call`
- `google`
- `join`
- `route`
- `yahoo`

TYPE
enum('done', 'go', 'next', 'search', 'send', 'none', 'previous', 'default', 'emergency-call', 'google', 'join', 'route', 'yahoo')

`rejectResponderTermination` iOS

If `true`, allows TextInput to pass touch events to the parent component. This allows components such as SwipeableListView to be swipeable from the TextInput on iOS, as is the case on Android by default. If `false`, TextInput always asks to handle the input (except when disabled). The default value is `true`.

TYPE
bool

`rows` Android

Sets the number of lines for a `TextInput`. Use it with multiline set to `true` to be able to fill the lines.

TYPE
number

`scrollEnabled` iOS

If `false`, scrolling of the text view will be disabled. The default value is `true`. Only works with `multiline={true}`.

TYPE
bool

`secureTextEntry`

If `true`, the text input obscures the text entered so that sensitive text like passwords stay secure. The default value is `false`. Does not work with `multiline={true}`.

TYPE
bool

`selection`

The start and end of the text input's selection. Set start and end to the same value to position the cursor.

TYPE
object: `{start: number,end: number}`

`selectionColor`

The highlight and cursor color of the text input.

TYPE
color

`selectTextOnFocus`

If `true`, all text will automatically be selected on focus.

TYPE
bool

`showSoftInputOnFocus`

When `false`, it will prevent the soft keyboard from showing when the field is focused. The default value is `true`.

TYPE
bool

`spellCheck` iOS

If `false`, disables spell-check style (i.e. red underlines). The default value is inherited from `autoCorrect`.

TYPE
bool

`textAlign`

Align the input text to the left, center, or right sides of the input field.

Possible values for `textAlign` are:

- `left`
- `center`
- `right`

TYPE
enum('left', 'center', 'right')

`textContentType` iOS

Give the keyboard and the system information about the expected semantic meaning for the content that users enter.

> **NOTE**

> `autoComplete`, provides the same functionality and is available for all platforms. You can use `Platform.select` for differing platform behaviors.
>
> Avoid using both `textContentType` and `autoComplete`. For backwards compatibility, `textContentType` takes precedence when both properties are set.

For iOS 11+ you can set `textContentType` to `username` or `password` to enable autofill of login details from the device keychain.

For iOS 12+ `newPassword` can be used to indicate a new password input the user may want to save in the keychain, and `oneTimeCode` can be used to indicate that a field can be autofilled by a code arriving in an SMS.

To disable autofill, set `textContentType` to `none`.

Possible values for `textContentType` are:

- `none`
- `URL`
- `addressCity`
- `addressCityAndState`
- `addressState`
- `countryName`
- `creditCardNumber`
- `emailAddress`
- `familyName`
- `fullStreetAddress`
- `givenName`
- `jobTitle`
- `location`
- `middleName`
- `name`
- `namePrefix`

- `nameSuffix`
- `nickname`
- `organizationName`
- `postalCode`
- `streetAddressLine1`
- `streetAddressLine2`
- `sublocality`
- `telephoneNumber`
- `username`
- `password`
- `newPassword`
- `oneTimeCode`

TYPE
enum('none', 'URL', 'addressCity', 'addressCityAndState', 'addressState', 'countryName', 'creditCardNumber', 'emailAddress', 'familyName', 'fullStreetAddress', 'givenName', 'jobTitle', 'location', 'middleName', 'name', 'namePrefix', 'nameSuffix', 'nickname', 'organizationName', 'postalCode', 'streetAddressLine1', 'streetAddressLine2', 'sublocality', 'telephoneNumber', 'username', 'password')

`passwordRules` iOS

When using `textContentType` as `newPassword` on iOS we can let the OS know the minimum requirements of the password so that it can generate one that will satisfy them. In order to create a valid string for `PasswordRules` take a look to the Apple Docs.

> If passwords generation dialog doesn't appear please make sure that:
>
> - AutoFill is enabled: **Settings** → **Passwords & Accounts** → toggle "On" the **AutoFill Passwords**,
> - iCloud Keychain is used: **Settings** → **Apple ID** → **iCloud** → **Keychain** → toggle "On" the **iCloud Keychain**.

TYPE
string

`style`

Note that not all Text styles are supported, an incomplete list of what is not supported includes:

- `borderLeftWidth`
- `borderTopWidth`
- `borderRightWidth`
- `borderBottomWidth`
- `borderTopLeftRadius`
- `borderTopRightRadius`
- `borderBottomRightRadius`
- `borderBottomLeftRadius`

see Issue#7070 for more detail.

Styles

TYPE
Text

`textBreakStrategy` Android

Set text break strategy on Android API Level 23+, possible values are `simple`, `highQuality`, `balanced` The default value is `simple`.

TYPE
enum('simple', 'highQuality', 'balanced')

`underlineColorAndroid` Android

The color of the `TextInput` underline.

TYPE
color

`value`

The value to show for the text input. `TextInput` is a controlled component, which means the native value will be forced to match this value prop if provided. For most uses, this works great, but in some cases this may cause flickering - one common cause is preventing edits by keeping value the same. In addition to setting the same value, either set `editable={false}`, or set/update `maxLength` to prevent unwanted edits without flicker.

TYPE
string

`lineBreakStrategyIOS` iOS

Set line break strategy on iOS 14+. Possible values are `none`, `standard`, `hangul-word` and `push-out`.

TYPE	DEFAULT
enum('none', 'standard', 'hangul-word', 'push-out')	'none'

Methods

`.focus()`

```
focus();
```

Makes the native input request focus.

`.blur()`

```
blur();
```

Makes the native input lose focus.

`clear()`

```
clear();
```

Removes all text from the `TextInput`.

`isFocused()`

```
isFocused(): boolean;
```

Returns `true` if the input is currently focused; `false` otherwise.

Known issues

- react-native#19096: Doesn't support Android's `onKeyPreIme`.
- react-native#19366: Calling .focus() after closing Android's keyboard via back button doesn't bring keyboard up again.
- react-native#26799: Doesn't support Android's `secureTextEntry` when `keyboardType="email-address"` or `keyboardType="phone-pad"`.

Is this page useful? 

Edit this page

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