Layout Props

> More detailed examples about those properties can be found on the Layout with Flexbox page.

Example

The following example shows how different properties can affect or shape a React Native layout. You can try for example to add or remove squares from the UI while changing the values of the property `flexWrap`.

TypeScript JavaScript

LayoutProps Example Expo

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

const App = () => {
  const flexDirections = [
    'row',
    'row-reverse',
    'column',
    'column-reverse',
  ] as const;
  const justifyContents = [
    'flex-start',
    'flex-end',
    'center',
    'space-between',
    'space-around',
    'space-evenly',
  ] as const;
  const alignItemsArr = [
```

Preview My Device iOS Android Web

Reference

Props

`alignContent`

`alignContent` controls how rows align in the cross direction, overriding the `alignContent` of the parent. See https://developer.mozilla.org/en-US/docs/Web/CSS/align-content for more details.

TYPE	REQUIRED
enum('flex-start', 'flex-end', 'center', 'stretch', 'space-between', 'space-around')	No

`alignItems`

`alignItems` aligns children in the cross direction. For example, if children are flowing vertically, `alignItems` controls how they align horizontally. It works like `align-items` in CSS (default: stretch). See https://developer.mozilla.org/en-US/docs/Web/CSS/align-items for more details.

TYPE	REQUIRED
enum('flex-start', 'flex-end', 'center', 'stretch', 'baseline')	No

`alignSelf`

`alignSelf` controls how a child aligns in the cross direction, overriding the `alignItems` of the parent. It works like `align-self` in CSS (default: auto). See https://developer.mozilla.org/en-US/docs/Web/CSS/align-self for more details.

TYPE	REQUIRED
enum('auto', 'flex-start', 'flex-end', 'center', 'stretch', 'baseline')	No

`aspectRatio`

Aspect ratio controls the size of the undefined dimension of a node. See https://developer.mozilla.org/en-US/docs/Web/CSS/aspect-ratio for more details.

- On a node with a set width/height, aspect ratio controls the size of the unset dimension
- On a node with a set flex basis, aspect ratio controls the size of the node in the cross axis if unset
- On a node with a measure function, aspect ratio works as though the measure function measures the flex basis
- On a node with flex grow/shrink, aspect ratio controls the size of the node in the cross axis if unset
- Aspect ratio takes min/max dimensions into account

TYPE	REQUIRED
number, string	No

`borderBottomWidth`

`borderBottomWidth` works like `border-bottom-width` in CSS. See https://developer.mozilla.org/en-US/docs/Web/CSS/border-bottom-width for more details.

TYPE	REQUIRED
number	No

`borderEndWidth`

When direction is `ltr`, `borderEndWidth` is equivalent to `borderRightWidth`. When direction is `rtl`, `borderEndWidth` is equivalent to `borderLeftWidth`.

TYPE	REQUIRED
number	No

`borderLeftWidth`

`borderLeftWidth` works like `border-left-width` in CSS. See https://developer.mozilla.org/en-US/docs/Web/CSS/border-left-width for more details.

TYPE	REQUIRED
number	No

`borderRightWidth`

`borderRightWidth` works like `border-right-width` in CSS. See https://developer.mozilla.org/en-US/docs/Web/CSS/border-right-width for more details.

TYPE	REQUIRED
number	No

`borderStartWidth`

When direction is `ltr`, `borderStartWidth` is equivalent to `borderLeftWidth`. When direction is `rtl`, `borderStartWidth` is equivalent to `borderRightWidth`.

TYPE	REQUIRED
number	No

`borderTopWidth`

`borderTopWidth` works like `border-top-width` in CSS. See https://developer.mozilla.org/en-US/docs/Web/CSS/border-top-width for more details.

TYPE	REQUIRED
number	No

`borderWidth`

`borderWidth` works like `border-width` in CSS. See https://developer.mozilla.org/en-US/docs/Web/CSS/border-width for more details.

TYPE	REQUIRED
number	No

`bottom`

`bottom` is the number of logical pixels to offset the bottom edge of this component.

It works similarly to `bottom` in CSS, but in React Native you must use points or percentages. Ems and other units are not supported.

See https://developer.mozilla.org/en-US/docs/Web/CSS/bottom for more details of how `bottom` affects layout.

TYPE	REQUIRED
number, string	No

`columnGap`

`columnGap` works like `column-gap` in CSS. Only pixel units are supported in React Native. See https://developer.mozilla.org/en-US/docs/Web/CSS/column-gap for more details.

TYPE	REQUIRED
number	No

`direction`

`direction` specifies the directional flow of the user interface. The default is `inherit`, except for root node which will have value based on the current locale. See https://yogalayout.com/docs/layout-direction for more details.

TYPE	REQUIRED	PLATFORM
enum('inherit', 'ltr', 'rtl')	No	iOS

`display`

`display` sets the display type of this component.

It works similarly to `display` in CSS but only supports 'flex' and 'none'. 'flex' is the default.

TYPE	REQUIRED
enum('none', 'flex')	No

`end`

When the direction is `ltr`, `end` is equivalent to `right`. When the direction is `rtl`, `end` is equivalent to `left`.

This style takes precedence over the `left` and `right` styles.

TYPE	REQUIRED
number, string	No

`flex`

In React Native `flex` does not work the same way that it does in CSS. `flex` is a number rather than a string, and it works according to the Yoga layout engine.

When `flex` is a positive number, it makes the component flexible, and it will be sized proportional to its flex value. So a component with `flex` set to 2 will take twice the space as a component with `flex` set to 1. `flex: <positive number>` equates to `flexGrow: <positive number>, flexShrink: 1, flexBasis: 0`.

When `flex` is 0, the component is sized according to `width` and `height`, and it is inflexible.

When `flex` is -1, the component is normally sized according to `width` and `height`. However, if there's not enough space, the component will shrink to its `minWidth` and `minHeight`.

`flexGrow`, `flexShrink`, and `flexBasis` work the same as in CSS.

TYPE	REQUIRED
number	No

`flexBasis`

`flexBasis` is an axis-independent way of providing the default size of an item along the main axis. Setting the `flexBasis` of a child is similar to setting the `width` of that child if its parent is a container with `flexDirection: row` or setting the `height` of a child if its parent is a container with `flexDirection: column`. The `flexBasis` of an item is the default size of that item, the size of the item before any `flexGrow` and `flexShrink` calculations are performed.

TYPE	REQUIRED
number, string	No

`flexDirection`

`flexDirection` controls which directions children of a container go. `row` goes left to right, `column` goes top to bottom, and you may be able to guess what the other two do. It works like `flex-direction` in CSS, except the default is `column`. See https://developer.mozilla.org/en-US/docs/Web/CSS/flex-direction for more details.

TYPE	REQUIRED
enum('row', 'row-reverse', 'column', 'column-reverse')	No

`flexGrow`

`flexGrow` describes how any space within a container should be distributed among its children along the main axis. After laying out its children, a container will distribute any remaining space according to the flex grow values specified by its children.

`flexGrow` accepts any floating point value >= 0, with 0 being the default value. A container will distribute any remaining space among its children weighted by the children's `flexGrow` values.

TYPE	REQUIRED
number	No

`flexShrink`

`flexShrink` describes how to shrink children along the main axis in the case in which the total size of the children overflows the size of the container on the main axis. `flexShrink` is very similar to `flexGrow` and can be thought of in the same way if any overflowing size is considered to be negative remaining space. These two properties also work well together by allowing children to grow and shrink as needed.

`flexShrink` accepts any floating point value >= 0, with 0 being the default value. A container will shrink its children weighted by the children's `flexShrink` values.

TYPE	REQUIRED
number	No

`flexWrap`

`flexWrap` controls whether children can wrap around after they hit the end of a flex container. It works like `flex-wrap` in CSS (default: nowrap). See https://developer.mozilla.org/en-US/docs/Web/CSS/flex-wrap for more details. Note it does not work anymore with `alignItems: stretch` (the default), so you may want to use `alignItems: flex-start` for example (breaking change details: https://github.com/facebook/react-native/releases/tag/v0.28.0).

TYPE	REQUIRED
enum('wrap', 'nowrap', 'wrap-reverse')	No

`gap`

`gap` works like `gap` in CSS. Only pixel units are supported in React Native. See https://developer.mozilla.org/en-US/docs/Web/CSS/gap for more details.

TYPE	REQUIRED
number	No

`height`

`height` sets the height of this component.

It works similarly to `height` in CSS, but in React Native you must use points or percentages. Ems and other units are not supported. See https://developer.mozilla.org/en-US/docs/Web/CSS/height for more details.

TYPE	REQUIRED
number, string	No

`justifyContent`

`justifyContent` aligns children in the main direction. For example, if children are flowing vertically, `justifyContent` controls how they align vertically. It works like `justify-content` in CSS (default: flex-start). See https://developer.mozilla.org/en-US/docs/Web/CSS/justify-content for more details.

TYPE	REQUIRED
enum('flex-start', 'flex-end', 'center', 'space-between', 'space-around', 'space-evenly')	No

`left`

`left` is the number of logical pixels to offset the left edge of this component.

It works similarly to `left` in CSS, but in React Native you must use points or percentages. Ems and other units are not supported.

See https://developer.mozilla.org/en-US/docs/Web/CSS/left for more details of how `left` affects layout.

TYPE	REQUIRED
number, string	No

`margin`

Setting `margin` has the same effect as setting each of `marginTop`, `marginLeft`, `marginBottom`, and `marginRight`. See https://developer.mozilla.org/en-US/docs/Web/CSS/margin for more details.

TYPE	REQUIRED
number, string	No

`marginBottom`

`marginBottom` works like `margin-bottom` in CSS. See https://developer.mozilla.org/en-US/docs/Web/CSS/margin-bottom for more details.

TYPE	REQUIRED
number, string	No

`marginEnd`

When direction is `ltr`, `marginEnd` is equivalent to `marginRight`. When direction is `rtl`, `marginEnd` is equivalent to `marginLeft`.

TYPE	REQUIRED
number, string	No

`marginHorizontal`

Setting `marginHorizontal` has the same effect as setting both `marginLeft` and `marginRight`.

TYPE	REQUIRED
number, string	No

`marginLeft`

`marginLeft` works like `margin-left` in CSS. See https://developer.mozilla.org/en-US/docs/Web/CSS/margin-left for more details.

TYPE	REQUIRED
number, string	No

`marginRight`

`marginRight` works like `margin-right` in CSS. See https://developer.mozilla.org/en-US/docs/Web/CSS/margin-right for more details.

TYPE	REQUIRED
number, string	No

`marginStart`

When direction is `ltr`, `marginStart` is equivalent to `marginLeft`. When direction is `rtl`, `marginStart` is equivalent to `marginRight`.

TYPE	REQUIRED
number, string	No

`marginTop`

`marginTop` works like `margin-top` in CSS. See https://developer.mozilla.org/en-US/docs/Web/CSS/margin-top for more details.

TYPE	REQUIRED
number, string	No

`marginVertical`

Setting `marginVertical` has the same effect as setting both `marginTop` and `marginBottom`.

TYPE	REQUIRED
number, string	No

`maxHeight`

`maxHeight` is the maximum height for this component, in logical pixels.

It works similarly to `max-height` in CSS, but in React Native you must use points or percentages. Ems and other units are not supported.

See https://developer.mozilla.org/en-US/docs/Web/CSS/max-height for more details.

TYPE	REQUIRED
number, string	No

`maxWidth`

`maxWidth` is the maximum width for this component, in logical pixels.

It works similarly to `max-width` in CSS, but in React Native you must use points or percentages. Ems and other units are not supported.

See https://developer.mozilla.org/en-US/docs/Web/CSS/max-width for more details.

TYPE	REQUIRED
number, string	No

`minHeight`

`minHeight` is the minimum height for this component, in logical pixels.

It works similarly to `min-height` in CSS, but in React Native you must use points or percentages. Ems and other units are not supported.

See https://developer.mozilla.org/en-US/docs/Web/CSS/min-height for more details.

TYPE	REQUIRED
number, string	No

`minWidth`

`minWidth` is the minimum width for this component, in logical pixels.

It works similarly to `min-width` in CSS, but in React Native you must use points or percentages. Ems and other units are not supported.

See https://developer.mozilla.org/en-US/docs/Web/CSS/min-width for more details.

TYPE	REQUIRED
number, string	No

`overflow`

`overflow` controls how children are measured and displayed. `overflow: hidden` causes views to be clipped while `overflow: scroll` causes views to be measured independently of their parents' main axis. It works like `overflow` in CSS (default: visible). See https://developer.mozilla.org/en/docs/Web/CSS/overflow for more details.

TYPE	REQUIRED
enum('visible', 'hidden', 'scroll')	No

`padding`

Setting `padding` has the same effect as setting each of `paddingTop`, `paddingBottom`, `paddingLeft`, and `paddingRight`. See https://developer.mozilla.org/en-US/docs/Web/CSS/padding for more details.

TYPE	REQUIRED
number, string	No

`paddingBottom`

`paddingBottom` works like `padding-bottom` in CSS. See https://developer.mozilla.org/en-US/docs/Web/CSS/padding-bottom for more details.

TYPE	REQUIRED
number, string	No

`paddingEnd`

When direction is `ltr`, `paddingEnd` is equivalent to `paddingRight`. When direction is `rtl`, `paddingEnd` is equivalent to `paddingLeft`.

TYPE	REQUIRED
number, string	No

`paddingHorizontal`

Setting `paddingHorizontal` is like setting both of `paddingLeft` and `paddingRight`.

TYPE	REQUIRED
number, string	No

`paddingLeft`

`paddingLeft` works like `padding-left` in CSS. See https://developer.mozilla.org/en-US/docs/Web/CSS/padding-left for more details.

TYPE	REQUIRED
number, string	No

`paddingRight`

`paddingRight` works like `padding-right` in CSS. See https://developer.mozilla.org/en-US/docs/Web/CSS/padding-right for more details.

TYPE	REQUIRED
number, string	No

`paddingStart`

When direction is `ltr`, `paddingStart` is equivalent to `paddingLeft`. When direction is `rtl`, `paddingStart` is equivalent to `paddingRight`.

TYPE	REQUIRED
number, string	No

`paddingTop`

`paddingTop` works like `padding-top` in CSS. See https://developer.mozilla.org/en-US/docs/Web/CSS/padding-top for more details.

TYPE	REQUIRED
number, ,string	No

`paddingVertical`

Setting `paddingVertical` is like setting both of `paddingTop` and `paddingBottom`.

TYPE	REQUIRED
number, string	No

`position`

`position` in React Native is similar to regular CSS, but everything is set to `relative` by default, so `absolute` positioning is always relative to the parent.

If you want to position a child using specific numbers of logical pixels relative to its parent, set the child to have `absolute` position.

If you want to position a child relative to something that is not its parent, don't use styles for that. Use the component tree.

See https://github.com/facebook/yoga for more details on how `position` differs between React Native and CSS.

TYPE	REQUIRED
enum('absolute', 'relative')	No

`right`

`right` is the number of logical pixels to offset the right edge of this component.

It works similarly to `right` in CSS, but in React Native you must use points or percentages. Ems and other units are not supported.

See https://developer.mozilla.org/en-US/docs/Web/CSS/right for more details of how `right` affects layout.

TYPE	REQUIRED
number, string	No

`rowGap`

`rowGap` works like `row-gap` in CSS. Only pixel units are supported in React Native. See https://developer.mozilla.org/en-US/docs/Web/CSS/row-gap for more details.

TYPE	REQUIRED
number	No

`start`

When the direction is `ltr`, `start` is equivalent to `left`. When the direction is `rtl`, `start` is equivalent to `right`.

This style takes precedence over the `left`, `right`, and `end` styles.

TYPE	REQUIRED
number, string	No

`top`

`top` is the number of logical pixels to offset the top edge of this component.

It works similarly to `top` in CSS, but in React Native you must use points or percentages. Ems and other units are not supported.

See https://developer.mozilla.org/en-US/docs/Web/CSS/top for more details of how `top` affects layout.

TYPE	REQUIRED
number, string	No

`width`

`width` sets the width of this component.

It works similarly to `width` in CSS, but in React Native you must use points or percentages. Ems and other units are not supported. See https://developer.mozilla.org/en-US/docs/Web/CSS/width for more details.

TYPE	REQUIRED
number, string	No

`zIndex`

`zIndex` controls which components display on top of others. Normally, you don't use `zIndex`. Components render according to their order in the document tree, so later components draw over earlier ones. `zIndex` may be useful if you have animations or custom modal interfaces where you don't want this behavior.

It works like the CSS `z-index` property - components with a larger `zIndex` will render on top. Think of the z-direction like it's pointing from the phone into your eyeball. See https://developer.mozilla.org/en-US/docs/Web/CSS/z-index for more details.

On iOS, `zIndex` may require `View`s to be siblings of each other for it to work as expected.

TYPE	REQUIRED
number	No

Is this page useful?

Edit this page

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