Animated.ValueXY

2D Value for driving 2D animations, such as pan gestures. Almost identical API to normal `Animated.Value`, but multiplexed. Contains two regular `Animated.Value`s under the hood.

Example

Animated.ValueXY

Expo

```
import React, {useRef} from 'react';
import {Animated, PanResponder, StyleSheet, View} from 'react-native';

const DraggableView = () => {
  const pan = useRef(new Animated.ValueXY()).current;

  const panResponder = PanResponder.create({
    onStartShouldSetPanResponder: () => true,
    onPanResponderMove: Animated.event([
      null,
      {
        dx: pan.x, // x,y are Animated.Value
        dy: pan.y,
      },
    ]),
    onPanResponderRelease: () => {
      Animated.spring(
        pan, // Auto-multiplexed
        {toValue: {x: 0, y: 0}, useNativeDriver: true}, // Back to zero
      ).start();
    },
  });

  return (
    <View style={styles.container}>
      <Animated.View
```

Preview My Device iOS Android Web

Reference

Methods

setValue()

```
setValue(value: {x: number; y: number});
```

Directly set the value. This will stop any animations running on the value and update all the bound properties.

Parameters:

NAME	TYPE	REQUIRED	DESCRIPTION
value	`{x: number; y: number}`	Yes	Value

setOffset()

```
setOffset(offset: {x: number; y: number});
```

Sets an offset that is applied on top of whatever value is set, whether via `setValue`, an animation, or `Animated.event`. Useful for compensating things like the start of a pan gesture.

Parameters:

NAME	TYPE	REQUIRED	DESCRIPTION
offset	`{x: number; y: number}`	Yes	Offset value

flattenOffset()

```
flattenOffset();
```

Merges the offset value into the base value and resets the offset to zero. The final output of the value is unchanged.

extractOffset()

```
extractOffset();
```

Sets the offset value to the base value, and resets the base value to zero. The final output of the value is unchanged.

addListener()

```
addListener(callback: (value: {x: number; y: number}) => void);
```

Adds an asynchronous listener to the value so you can observe updates from animations. This is useful because there is no way to synchronously read the value because it might be driven natively.

Returns a string that serves as an identifier for the listener.

Parameters:

NAME	TYPE	REQUIRED	DESCRIPTION
callback	function	Yes	The callback function which will receive an object with a `value` key set to the new value.

removeListener()

```
removeListener(id: string);
```

Unregister a listener. The `id` param shall match the identifier previously returned by `addListener()`.

Parameters:

NAME	TYPE	REQUIRED	DESCRIPTION
id	string	Yes	Id for the listener being removed.

`removeAllListeners()`

```
removeAllListeners();
```

Remove all registered listeners.

`stopAnimation()`

```
stopAnimation(callback?: (value: {x: number; y: number}) => void);
```

Stops any running animation or tracking. `callback` is invoked with the final value after stopping the animation, which is useful for updating state to match the animation position with layout.

Parameters:

NAME	TYPE	REQUIRED	DESCRIPTION
callback	function	No	A function that will receive the final value.

`resetAnimation()`

```
resetAnimation(callback?: (value: {x: number; y: number}) => void);
```

Stops any animation and resets the value to its original.

Parameters:

NAME	TYPE	REQUIRED	DESCRIPTION
callback	function	No	A function that will receive the original value.

getLayout()

```
getLayout(): {left: Animated.Value, top: Animated.Value};
```

Converts `{x, y}` into `{left, top}` for use in style, e.g.

```
style={this.state.anim.getLayout()}
```

getTranslateTransform()

```
getTranslateTransform(): [
  {translateX: Animated.Value},
  {translateY: Animated.Value},
];
```

Converts `{x, y}` into a useable translation transform, e.g.

```
style={{
  transform: this.state.anim.getTranslateTransform()
}}
```

Is this page useful? 

Edit this page

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