Version: 3.x

Custom animations

INFO

This page was ported from an old version of the documentation.

As we're rewriting the documentation some of the pages might be a little outdated.

If our set of predefined animations is not enough for you then this tab is what you are looking for.

Custom Exiting Animation

What our exiting animation builders do under the hood is generating a worklet function that returns essential data for starting particular animation. The high level template looks like this:

```
function CustomExitingAnimation(values) {
  'worklet';
  const animations = {
    // your animations
  };
  const initialValues = {
    // initial values for animations
  };
  const callback = (finished: boolean) => {
    // optional callback that will fire when layout animation ends
  };
  return {
    initialValues,
    animations,
    callback,
  };
}
```

- `values` - contains information about where view was displayed and what were its dimensions
  - `values.currentOriginX` - X coordinate of top left corner in parent's coordinate system

  - `values.currentOriginY` - Y coordinate of top left corner in parent's coordinate system
  - `values.currentWidth` - view's width
  - `values.currentHeight` - view's height
  - `values.currentGlobalOriginX` - X coordinate of top left corner in global coordinate system
  - `values.currentGlobalOriginY` - Y coordinate of top left corner in global coordinate system

Example

```
function CardView() {
  const exiting = (values) => {
    'worklet';
    const animations = {
      originX: withTiming(width, { duration: 3000 }),
      opacity: withTiming(0.5, { duration: 2000 }),
    };
    const initialValues = {
      originX: values.currentOriginX,
      opacity: 1,
    };
    return {
      initialValues,
      animations,
    };
  };

  return (
    <Animated.View style={[styles.animatedView]} exiting={exiting}>
      <Text> Card Example </Text>
    </Animated.View>
  );
}
```

Custom Entering Animation

What our entering animation builders do under the hood is generating a worklet function that returns essential data for starting particular animation. The high level template looks like this:

```
function CustomEnteringAnimation(values) {
  'worklet';
  const animations = {
    // your animations
  };
  const initialValues = {
    // initial values for animations
  };
  const callback = (finished: boolean) => {
    // optional callback that will fire when layout animation ends
  };
  return {
    initialValues,
    animations,
    callback,
  };
}
```

- `values` - contains information about where view wants to be displayed and what are its dimensions
  - `values.targetOriginX` - X coordinate of top left corner in parent's coordinate system
  - `values.targetOriginY` - Y coordinate of top left corner in parent's coordinate system
  - `values.targetWidth` - view's width
  - `values.targetHeight` - view's height
  - `values.targetGlobalOriginX` - X coordinate of top left corder in global coordinate system
  - `values.targetGlobalOriginY` - Y coordinate of top left corder in global coordinate system

Example

```
function CardView() {
  const entering = (targetValues) => {
    'worklet';
    const animations = {
      originX: withTiming(targetValues.originX, { duration: 3000 }),
      opacity: withTiming(1, { duration: 2000 }),
      borderRadius: withDelay(4000, withTiming(30, { duration: 3000 })),
      transform: [
        { rotate: withTiming('0deg', { duration: 4000 }) },
        { scale: withTiming(1, { duration: 3500 }) },
      ],
    };
    const initialValues = {
      originX: -width,
      opacity: 0,
      borderRadius: 10,
      transform: [{ rotate: '90deg' }, { scale: 0.5 }],
    };
    return {
      initialValues,
      animations,
    };
  };

  return (
    <Animated.View style={[styles.animatedView]} entering={entering}>
      <Text> Card Example </Text>
    </Animated.View>
  );
}
```

Custom Layout Transition

What our layout transition builders do under the hood is generating a worklet function that returns essential data for starting particular transition. The high level template looks like this:

```
function CustomLayoutTransition(values) {
  'worklet';
  const animations = {
    // your animations
  };
  const initialValues = {
    // initial values for animations
  };
  const callback = (finished: boolean) => {
    // optional callback that will fire when layout animation ends
  };
  return {
    initialValues,
    animations,
    callback,
  };
}
```

- `values` - contains before and after information about the view's origin and dimensions
  - `values.targetOriginX` - X coordinate of top left corner in parent's coordinate system
  - `values.targetOriginY` - Y coordinate of top left corner in parent's coordinate system
  - `values.targetWidth` - view's width
  - `values.targetHeight` - view's height
  - `values.targetGlobalOriginX` - X coordinate of top left corder in global coordinate system
  - `values.targetGlobalOriginY` - Y coordinate of top left corder in global coordinate system
  - `values.currentOriginX` - X coordinate of top left corner in parent's coordinate system (before)
  - `values.currentOriginY` - Y coordinate of top left corner in parent's coordinate system (before)
  - `values.currentWidth` - view's width (before)
  - `values.currentHeight` - view's height (before)
  - `values.currentGlobalOriginX` - X coordinate of top left corner in global coordinate system (before)
  - `values.currentGlobalOriginY` - Y coordinate of top left corner in global coordinate system (before)

Example

0:00

```
function CustomLayoutTransition(values) {
  'worklet';
  return {
    animations: {
      originX: withTiming(values.targetOriginX, { duration: 1000 }),
      originY: withDelay(
        1000,
        withTiming(values.targetOriginY, { duration: 1000 })
      ),
      width: withSpring(values.targetWidth),
      height: withSpring(values.targetHeight),
    },
    initialValues: {
      originX: values.currentOriginX,
      originY: values.currentOriginY,
      width: values.currentWidth,
      height: values.currentHeight,
    },
  };
}

function Box({ label, state }: { label: string, state: boolean }) {
  const ind = label.charCodeAt(0) - 'A'.charCodeAt(0);
  const delay = 300 * ind;
  return (
    <Animated.View
      layout={CustomLayoutTransition}
      style={[
        styles.box,
        {
          flexDirection: state ? 'row' : 'row-reverse',
          height: state ? 30 : 60,
        },
      ]}>
      <Text> {label} </Text>
    </Animated.View>
  );
}

export function CustomLayoutTransitionExample() {
  const [state, setState] = useState(true);
  return (
    <View style={{ marginTop: 30 }}>
      <View style={{ height: 300 }}>
        <View style={{ flexDirection: state ? 'row' : 'column' }}>
          {state && <Box key="a" label="A" state={state} />}
          <Box key="b" label="B" state={state} />
          {!state && <Box key="a" label="A" state={state} />}
          <Box key="c" label="C" state={state} />
        </View>
      </View>

      <Button
        onPress={() => {
          setState(!state);
        }}
        title="toggle"
      />
    </View>
  );
}
```

Other Facts

Each Reanimated component has its shared value that keeps current animations assigned to that particular component. If you want to start a new animation for a specific prop and you don't provide an initial value for the prop then the initial value will be taken from the last animation that has been assigned to the component. The only exception is Entering animation because we have no way to get the previous animation values.

Edit this page