Circular, Linear progress React components

Progress

Progress indicators commonly known as spinners, express an unspecified wait time or display the length of a process.

Progress indicators inform users about the status of ongoing processes, such as loading an app, submitting a form, or saving updates.

- **Determinate** indicators display how long an operation will take.
- **Indeterminate** indicators visualize an unspecified wait time.

The animations of the components rely on CSS as much as possible to work even before the JavaScript is loaded.

{{“component”: “modules/components/ComponentLinkHeader.js”}}

Circular

Circular indeterminate

{{“demo”: “CircularIndeterminate.js”}}

Circular color

{{“demo”: “CircularColor.js”}}

Circular determinate

{{“demo”: “CircularDeterminate.js”}}

Interactive integration

{{“demo”: “CircularIntegration.js”}}

Circular with label

{{“demo”: “CircularWithValueLabel.js”}}

Linear

Linear indeterminate

{{“demo”: “LinearIndeterminate.js”}}

Linear color

{{“demo”: “LinearColor.js”}}

Linear determinate

{{“demo”: “LinearDeterminate.js”}}

Linear buffer

{{“demo”: “LinearBuffer.js”}}

Linear with label

{{“demo”: “LinearWithValueLabel.js”}}

Non-standard ranges

The progress components accept a value in the range 0 - 100. This simplifies things for screen-reader users, where these are the default min / max values. Sometimes, however, you might be working with a data source where the values fall outside this range. Here’s how you can easily transform a value in any range to a scale of 0 - 100:

```
// MIN = Minimum expected value
// MAX = Maximum expected value
// Function to normalise the values (MIN / MAX could be integrated)
const normalise = (value) => ((value - MIN) * 100) / (MAX - MIN);

// Example component that utilizes the `normalise` function at the point of render.
function Progress(props) {
  return (
    <React.Fragment>
      <CircularProgress variant="determinate" value={normalise(props.value)} />
      <LinearProgress variant="determinate" value={normalise(props.value)} />
    </React.Fragment>
  );
}
```

Customization

Here are some examples of customizing the component. You can learn more about this in the overrides documentation page.

{{“demo”: “CustomizedProgressBars.js”, “defaultCodeOpen”: false}}

Delaying appearance

There are 3 important limits to know around response time. The ripple effect of the `ButtonBase` component ensures that the user feels that the system is reacting instantaneously. Normally, no special feedback is necessary during delays of more than 0.1 but less than 1.0 second. After 1.0 second, you can display a loader to keep user's flow of thought uninterrupted.

{{“demo”: “DelayingAppearance.js”}}

Limitations

High CPU load

Under heavy load, you might lose the stroke dash animation or see random `CircularProgress` ring widths. You should run processor intensive operations in a web worker or by batch in order not to block the main rendering thread.

heavy load

When it's not possible, you can leverage the `disableShrink` prop to mitigate the issue. See this issue.

{{“demo”: “CircularUnderLoad.js”}}

High frequency updates

The `LinearProgress` uses a transition on the CSS transform property to provide a smooth update between different values. The default transition duration is 200ms. In the event a parent component updates the `value` prop too quickly, you will at least experience a 200ms delay between the re-render and the progress bar fully updated.

If you need to perform 30 re-renders per second or more, we recommend disabling the transition:

```
.MuiLinearProgress-bar {
  transition: none;
}
```

IE 11

The circular progress component animation on IE 11 is degraded. The stroke dash animation is not working (equivalent to `disableShrink`) and the circular animation wobbles. You can solve the latter with:

```
.MuiCircularProgress-indeterminate {
  animation: circular-rotate 1.4s linear infinite;
```

```
}

@keyframes circular-rotate {
  0% {
    transform: rotate(0deg);
    /* Fix IE11 wobbly */
    transform-origin: 50% 50%;
  }
  100% {
    transform: rotate(360deg);
  }
}
```