How to customize

Learn how to customize Material UI components by taking advantage of different strategies for specific use cases.

Material UI provides several different ways to customize a component's styles. Your specific context will determine which one is ideal. From narrowest to broadest use case, here are the options:

1. One-off customization
2. Reusable component
3. Global theme overrides
4. Global CSS override

1. One-off customization

To change the styles of *one single instance* of a component, you can use one of the following options:

The `sx` prop

The `sx` prop is the best option for adding style overrides to a single instance of a component in most cases. It can be used with all Material UI components.

```
<Slider
  defaultValue={30}
  sx={{
    width: 300,
    color: 'success.main',
```

```
  }}
/>
```

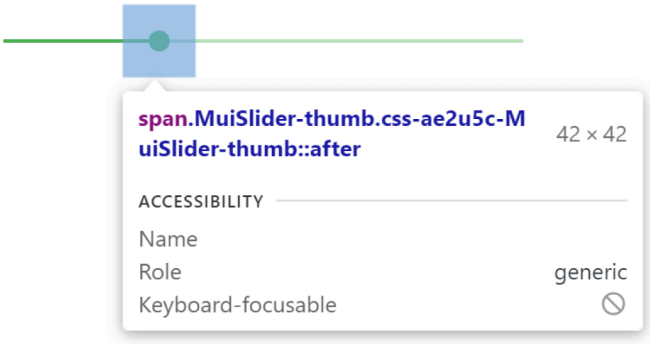
Overriding nested component styles

To customize a specific part of a component, you can use the class name provided by Material UI inside the `sx` prop. As an example, let's say you want to change the `Slider` component's thumb from a circle to a square.

First, use your browser's dev tools to identify the class for the component slot you want to override.

The styles injected into the DOM by Material UI rely on class names that all follow a standard pattern: `[hash]-Mui[Component name]-[name of the slot]`.

In this case, the styles are applied with `.css-ae2u5c-MuiSlider-thumb` but you only really need to target the `.MuiSlider-thumb`, where `Slider` is the component and `thumb` is the slot. Use this class name to write a CSS selector within the `sx` prop (`& .MuiSlider-thumb`), and add your overrides.



```
<Slider
  defaultValue={30}
  sx={{
    width: 300,
    color: 'success.main',
    '& .MuiSlider-thumb': {
      borderRadius: '1px',
    },
  }}
/>
```

⚠ These class names can't be used as CSS selectors because they are unstable.

Overriding styles with class names

If you want to override a component's styles using custom classes, you can use the `className` prop, available on each component. To override the styles of a specific part of the component, use the global classes provided by Material UI, as described in the previous section **"Overriding nested component styles"** under the `sx` prop section.

Visit the Style library interoperability guide to find examples of this approach using different styling libraries.

State classes

States like *hover*, *focus*, *disabled* and *selected*, are styled with a higher CSS specificity. To customize them, you'll need to **increase specificity**.

Here is an example with the *disabled* state and the `Button` component using a pseudo-class (`:disabled`):

```
.Button {
  color: black;
}

/* Increase the specificity */
.Button:disabled {
  color: white;
}
```

```
<Button disabled className="Button">
```

You can't always use a CSS pseudo-class, as the state doesn't exist in the web specification. Let's take the `MenuItem` component and its *selected* state as an example. In this situation, you can use Material UI's **state classes**, which act just like CSS pseudo-classes. Target the `.Mui-selected` global class name to customize the special state of the `MenuItem` component:

```
.MenuItem {
  color: black;
}

/* Increase the specificity */
.MenuItem.Mui-selected {
  color: blue;
}
```

```
<MenuItem selected className="MenuItem">
```

If you'd like to learn more about this topic, we recommend checking out the MDN Web Docs on CSS Specificity.

Why do I need to increase specificity to override one component state?

CSS pseudo-classes have a high level of specificity. For consistency with native elements, Material UI's state classes have the same level of specificity as CSS pseudo-classes, making it possible to target an individual component's state.

What custom state classes are available in Material UI?

You can rely on the following global class names generated by Material UI:

State	Global class name
active	`.Mui-active`
checked	`.Mui-checked`
completed	`.Mui-completed`
disabled	`.Mui-disabled`
error	`.Mui-error`
expanded	`.Mui-expanded`
focus visible	`.Mui-focusVisible`
focused	`.Mui-focused`
readOnly	`.Mui-readOnly`
required	`.Mui-required`
selected	`.Mui-selected`

Never apply styles directly to state class names. This will impact all components with unclear side-effects. Always target a state class together with a component.

```
/* ❌ NOT OK */
.Mui-error {
  color: red;
}

/* ✅ OK */
.MuiOutlinedInput-root.Mui-error {
  color: red;
}
```

2. Reusable component

To reuse the same overrides in different locations across your application, create a reusable component using the `styled()` utility:

```
import * as React from 'react';
import Slider, { SliderProps } from '@mui/material/Slider';
import { alpha, styled } from '@mui/material/styles';

const SuccessSlider = styled(Slider)<SliderProps>(({ theme }) => ({
  width: 300,
  color: theme.palette.success.main,
  '& .MuiSlider-thumb': {
    '&:hover, &.Mui-focusVisible': {
      boxShadow: `0px 0px 0px 8px ${alpha(theme.palette.success.main, 0.16)}`,
    },
    '&.Mui-active': {
      boxShadow: `0px 0px 0px 14px ${alpha(theme.palette.success.main, 0.16)}`,
    },
  },
}));

export default function StyledCustomization() {
  return <SuccessSlider defaultValue={30} />;
}
```

Dynamic overrides

The `styled()` utility lets you add dynamic styles based on a component's props. You can do this with **dynamic CSS** or **CSS variables**.

Dynamic CSS

⚠ If you are using TypeScript, you will need to update the prop's types of the new component.


```tsx
import * as React from 'react';
import { styled } from '@mui/material/styles';
import Slider, { SliderProps } from '@mui/material/Slider';

interface StyledSliderProps extends SliderProps {
  success?: boolean;
}

const StyledSlider = styled(Slider, {
  shouldForwardProp: (prop) => prop !== 'success',
})<StyledSliderProps>(({ success, theme }) => ({
  ...(success &&
    {
      // the overrides added when the new prop is used
    }),
}));
```

CSS variables


```tsx
<React.Fragment>
  <FormControlLabel
    control={
      <Switch
        checked={vars === successVars}
        onChange={handleChange}
        color="primary"
        value="dynamic-class-name"
      />
```

```
    }
    label="Success"
  />
  <CustomSlider style={vars} defaultValue={30} sx={{ mt: 1 }} />
</React.Fragment>
```

3. Global theme overrides

Material UI provides theme tools for managing style consistency between all components across your user interface. Visit the Component theming customization page for more details.

4. Global CSS override

To add global baseline styles for some of the HTML elements, use the `GlobalStyles` component. Here is an example of how you can override styles for the `h1` elements:

Grey h1 element

```
<React.Fragment>
  <GlobalStyles styles={{ h1: { color: 'grey' } }} />
  <h1>Grey h1 element</h1>
</React.Fragment>
```

If you are already using the CssBaseline component for setting baseline styles, you can also add these global styles as overrides for this component. Here is how you can achieve the same by using this approach.

Grey h1 element

```
<ThemeProvider theme={theme}>
  <CssBaseline />
  <h1>Grey h1 element</h1>
</ThemeProvider>
```

The `styleOverrides` key in the `MuiCssBaseline` component slot also supports callback from which you can access the theme. Here is how you can achieve the same by using this approach.


```
<ThemeProvider theme={theme}>
  <CssBaseline />
  <h1>h1 element</h1>
</ThemeProvider>
```

It is a good practice to hoist the `<GlobalStyles />` to a static constant, to avoid rerendering. This will ensure that the `<style>` tag generated would not recalculate on each render.

```
 import * as React from 'react';
 import GlobalStyles from '@mui/material/GlobalStyles';

+const inputGlobalStyles = <GlobalStyles styles={...} />;

 function Input(props) {
   return (
     <React.Fragment>
-      <GlobalStyles styles={...} />
+      {inputGlobalStyles}
       <input {...props} />
     </React.Fragment>
   )
 }
```

API

See the documentation below for a complete reference to all of the props and classes available to the components mentioned here.

- `<GlobalStyles />`

Edit this page

Was this page helpful?

‹ Zoom

Dark mode ›

MUI • Blog ↗ • Store ↗