Components

The theme's components key allows you to customize a component without wrapping it in another component. You can change the styles, the default props, and more.

Default props

You can change the default of every prop of a MUI component. A `defaultProps` key is exposed in the theme's `components` key for this use case.

```
const theme = createTheme({
  components: {
    // Name of the component
    MuiButtonBase: {
      defaultProps: {
        // The props to change the default for.
        disableRipple: true, // No more ripple!
      },
    },
  },
});
```

{{"demo": "DefaultProps.js"}}

To override lab component styles with TypeScript, check this page.

Global style overrides

You can use the theme's `styleOverrides` key to potentially change every single style injected by MUI into the DOM.

```
const theme = createTheme({
  components: {
    // Name of the component
    MuiButton: {
      styleOverrides: {
        // Name of the slot
        root: {
          // Some CSS
          fontSize: '1rem',
        },
      },
    },
  },
});
```

{{"demo": "GlobalThemeOverride.js"}}

The list of each component's classes is documented under the **CSS** section of its API page.

To override a lab component's styles with TypeScript, check this section of the documentation.

Overrides based on props

You can pass a callback as a value in each slot of the component's `styleOverrides` to apply styles based on props.

The `ownerState` prop is a combination of public props that you pass to the component + internal state of the component.

```
const finalTheme = createTheme({
  components: {
    MuiSlider: {
      styleOverrides: {
        valueLabel: ({ ownerState, theme }) => ({
          ...(ownerState.orientation === 'vertical' && {
            backgroundColor: 'transparent',
            color: theme.palette.grey[500],
          }),
        }),
      },
    },
  },
});
```

{{"demo": "GlobalThemeOverrideCallback.js"}}

Using `sx` (experimental) syntax

If you are not familiar `sx`, first check out the concept and the difference with the `styled`.

`sx` is also compatible with theme style overrides if you prefer the shorthand notation.

{{"demo": "GlobalThemeOverrideSx.js"}}

Adding new component variants

You can use the `variants` key in the theme's `components` section to add new variants to MUI components. These new variants can specify what styles the component should have when specific props are applied.

The definitions are specified in an array, under the component's name. For each of them a CSS class is added to the HTML `<head>`. The order is important, so make sure that the styles that should win are specified last.

```
const theme = createTheme({
  components: {
    MuiButton: {
      variants: [
        {
          props: { variant: 'dashed' },
          style: {
            textTransform: 'none',
            border: `2px dashed ${blue[500]}`,
          },
        },
```

```
        {
          props: { variant: 'dashed', color: 'secondary' },
          style: {
            border: `4px dashed ${red[500]}`,
          },
        },
      ],
    },
  },
});
```

If you're using TypeScript, you'll need to specify your new variants/colors, using module augmentation.

```
declare module '@mui/material/Button' {
  interface ButtonPropsVariantOverrides {
    dashed: true;
  }
}
```

{{"demo": "GlobalThemeVariants.js"}}

Theme variables

Another way to override the look of all component instances is to adjust the theme configuration variables.

```
const theme = createTheme({
  typography: {
    button: {
      fontSize: '1rem',
    },
  },
});
```

{{"demo": "ThemeVariables.js"}}