@mui/styles (LEGACY)

The legacy styling solution of MUI.

> ⚠️ `@mui/styles` *is the* ***legacy*** *styling solution for MUI. It depends on JSS as a styling solution, which is not used in the* `@mui/material` *anymore, deprecated in v5. If you don't want to have both emotion & JSS in your bundle, please refer to the* `@mui/system` *documentation which is the recommended alternative.*

> ⚠️ `@mui/styles` *is not compatible with React.StrictMode or React 18.*

MUI aims to provide a strong foundation for building dynamic UIs. For the sake of simplicity, **we expose the styling solution used in MUI components** as the `@mui/styles` package. You can use it, but you don't have to, since MUI is also interoperable with all the other major styling solutions.

Why use MUI's styling solution?

In previous versions, MUI has used Less, and then a custom inline-style solution to write the component styles, but these approaches proved to be limited. A *CSS-in-JS* solution overcomes many of those limitations, and **unlocks many great features** (theme nesting, dynamic styles, self-support, etc.).

MUI's styling solution is inspired by many other styling libraries such as styled-components and emotion.

- 💅 You can expect the same advantages as styled-components.
- 🚀 It's blazing fast.
- 🧩 It's extensible via a plugin API.
- ⚡ It uses JSS at its core – a high performance JavaScript to CSS compiler which works at runtime and server-side.
- 📦 Less than 15 KB gzipped; and no bundle size increase if used alongside MUI.

Installation

To install and save in your `package.json` dependencies, run:

```
// with npm
npm install @mui/styles

// with yarn
yarn add @mui/styles
```

Getting started

There are 3 possible APIs you can use to generate and apply styles, however they all share the same underlying logic.

Hook API

```
import * as React from 'react';
import { makeStyles } from '@mui/styles';
import Button from '@mui/material/Button';

const useStyles = makeStyles({
```

```
  root: {
    background: 'linear-gradient(45deg, #FE6B8B 30%, #FF8E53 90%)',
    border: 0,
    borderRadius: 3,
    boxShadow: '0 3px 5px 2px rgba(255, 105, 135, .3)',
    color: 'white',
    height: 48,
    padding: '0 30px',
  },
});

export default function Hook() {
  const classes = useStyles();
  return <Button className={classes.root}>Hook</Button>;
}
```

{{"demo": "Hook.js"}}

Styled components API

Note: this only applies to the calling syntax – style definitions still use a JSS object. You can also change this behavior, with some limitations.

```
import * as React from 'react';
import { styled } from '@mui/styles';
import Button from '@mui/material/Button';

const MyButton = styled(Button)({
  background: 'linear-gradient(45deg, #FE6B8B 30%, #FF8E53 90%)',
  border: 0,
  borderRadius: 3,
  boxShadow: '0 3px 5px 2px rgba(255, 105, 135, .3)',
  color: 'white',
  height: 48,
  padding: '0 30px',
});

export default function StyledComponents() {
  return <MyButton>Styled Components</MyButton>;
}
```

{{"demo": "StyledComponents.js"}}

Higher-order component API

```
import * as React from 'react';
import PropTypes from 'prop-types';
import { withStyles } from '@mui/styles';
import Button from '@mui/material/Button';

const styles = {
```

```
  root: {
    background: 'linear-gradient(45deg, #FE6B8B 30%, #FF8E53 90%)',
    border: 0,
    borderRadius: 3,
    boxShadow: '0 3px 5px 2px rgba(255, 105, 135, .3)',
    color: 'white',
    height: 48,
    padding: '0 30px',
  },
};

function HigherOrderComponent(props) {
  const { classes } = props;
  return <Button className={classes.root}>Higher-order component</Button>;
}

HigherOrderComponent.propTypes = {
  classes: PropTypes.object.isRequired,
};

export default withStyles(styles)(HigherOrderComponent);
```

{{"demo": "HigherOrderComponent.js"}}

Nesting selectors

You can nest selectors to target elements inside the current class or component. The following example uses the Hook API, but it works the same way with the other APIs.

```
const useStyles = makeStyles({
  root: {
    color: 'red',
    '& p': {
      color: 'green',
      '& span': {
        color: 'blue',
      },
    },
  },
});
```

{{"demo": "NestedStylesHook.js", "defaultCodeOpen": false}}

Adapting based on props

You can pass a function to `makeStyles` ("interpolation") in order to adapt the generated value based on the component's props. The function can be provided at the style rule level, or at the CSS property level:

```
const useStyles = makeStyles({
  // style rule
```

```
  foo: (props) => ({
    backgroundColor: props.backgroundColor,
  }),
  bar: {
    // CSS property
    color: (props) => props.color,
  },
});

function MyComponent() {
  // Simulated props for the purpose of the example
  const props = {
    backgroundColor: 'black',
    color: 'white',
  };
  // Pass the props as the first argument of useStyles()
  const classes = useStyles(props);

  return <div className={`${classes.foo} ${classes.bar}`} />;
}
```

This button component has a `color` prop that changes its color:

Adapting the hook API

{{"demo": "AdaptingHook.js"}}

Adapting the styled components API

{{"demo": "AdaptingStyledComponents.js"}}

Adapting the higher-order component API

{{"demo": "AdaptingHOC.js"}}

Stress test

In the following stress test, you can update the *theme color* and the *background-color property* live:

```
const useStyles = makeStyles((theme) => ({
  root: (props) => ({
    backgroundColor: props.backgroundColor,
    color: theme.color,
  }),
}));
```

{{"demo": "StressTest.js"}}

Using the theme context

Starting from v5, MUI no longer uses JSS as its default styling solution. If you still want to use the utilities exported by `@mui/styles` and they depend on the `theme`, you will need to provide the `theme` as part of the context. For

this, you can use the `ThemeProvider` component available in `@mui/styles`, or, if you are already using `@mui/material`, you should use the one exported from `@mui/material/styles` so that the same `theme` is available for components from '@mui/material'.

```
import { makeStyles } from '@mui/styles';
import { createTheme, ThemeProvider } from '@mui/material/styles';

const theme = createTheme();

const useStyles = makeStyles((theme) => ({
  root: {
    color: theme.palette.primary.main,
  }
}));

const App = (props) => {
  const classes = useStyles();
  return <ThemeProvider theme={theme}><div {...props} className={classes.root}>
</ThemeProvider>;
}
```