API (LEGACY)

The API reference of @mui/styles.

> ⚠️ `@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.*

`createGenerateClassName([options]) => class name generator`

A function which returns a class name generator function.

Arguments

1. `options` (*object* [optional]):
   - `options.disableGlobal` (*bool* [optional]): Defaults to `false`. Disable the generation of deterministic class names.
   - `options.productionPrefix` (*string* [optional]): Defaults to `'jss'`. The string used to prefix the class names in production.
   - `options.seed` (*string* [optional]): Defaults to `''`. The string used to uniquely identify the generator. It can be used to avoid class name collisions when using multiple generators in the same document.

Returns

`class name generator`: The generator should be provided to JSS.

Examples

```
import * as React from 'react';
import { StylesProvider, createGenerateClassName } from '@mui/styles';

const generateClassName = createGenerateClassName({
  productionPrefix: 'c',
});

export default function App() {
  return <StylesProvider generateClassName={generateClassName}>...</StylesProvider>;
}
```

`createStyles(styles) => styles`

This function doesn't really "do anything" at runtime, it's just the identity function. Its only purpose is to defeat **TypeScript**'s type widening when providing style rules to `makeStyles` / `withStyles` which are a function of the `Theme`.

Arguments

1. `styles` (*object*): A styles object.

Returns

`styles`: A styles object.

Examples

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

const useStyles = makeStyles((theme: Theme) =>
  createStyles({
    root: {
      backgroundColor: theme.palette.red,
    },
  }),
);

const theme = createTheme();

export default function MyComponent() {
  const classes = useStyles();
  return (
    <ThemeProvider theme={theme}>
      <div className={classes.root} />
    </ThemeProvider>
  );
}
```

`makeStyles(styles, [options]) => hook`

Link a style sheet with a function component using the **hook** pattern.

Arguments

1. `styles` (*Function* | *Object*): A function generating the styles or a styles object. It will be linked to the component. Use the function signature if you need to have access to the theme. It's provided as the first argument.
2. `options` (*object* [optional]):

- `options.defaultTheme` (*object* [optional]): The default theme to use if a theme isn't supplied through a Theme Provider.
- `options.name` (*string* [optional]): The name of the style sheet. Useful for debugging.
- `options.flip` (*bool* [optional]): When set to `false`, this sheet will opt-out the `rtl` transformation. When set to `true`, the styles are inversed. When set to `null`, it follows `theme.direction`.
- The other keys are forwarded to the options argument of jss.createStyleSheet([styles], [options]).

Returns

`hook` : A hook. This hook can be used in a function component. The documentation often calls this returned hook `useStyles`. It accepts one argument: the props that will be used for "interpolation" in the style sheet.

Examples

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

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

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

`ServerStyleSheets`

This is a class helper to handle server-side rendering. You can follow this guide for a practical approach.

```
import ReactDOMServer from 'react-dom/server';
import { ServerStyleSheets } from '@mui/styles';

const sheets = new ServerStyleSheets();
const html = ReactDOMServer.renderToString(sheets.collect(<App />));
const cssString = sheets.toString();

const response = `
<!DOCTYPE html>
<html>
  <head>
    <style id="jss-server-side">${cssString}</style>
  </head>
  <body>${html}</body>
</html>
`;
```

`new ServerStyleSheets([options])`

The instantiation accepts an options object as a first argument.

1. `options` (*object* [optional]): The options are spread as props to the `StylesProvider` component.

`sheets.collect(node) => React element`

The method wraps your React node in a provider element. It collects the style sheets during the rendering so they can be later sent to the client.

`sheets.toString() => CSS string`

The method returns the collected styles.

⚠ You must call `.collect()` before using this method.

`sheets.getStyleElement() => CSS React element`

The method is an alternative to `.toString()` when you are rendering the whole page with React.

⚠ You must call `.collect()` before using this method.

`styled(Component)(styles, [options]) => Component`

Link a style sheet with a function component using the **styled components** pattern.

Arguments

1. `Component`: The component that will be wrapped.
2. `styles` (*Function* | *Object*): A function generating the styles or a styles object. It will be linked to the component. Use the function signature if you need to have access to the theme. It's provided as property of the first argument.
3. `options` (*object* [optional]):

- `options.defaultTheme` (*object* [optional]): The default theme to use if a theme isn't supplied through a Theme Provider.
- `options.withTheme` (*bool* [optional]): Defaults to `false`. Provide the `theme` object to the component as a prop.
- `options.name` (*string* [optional]): The name of the style sheet. Useful for debugging. If the value isn't provided, it will try to fallback to the name of the component.
- `options.flip` (*bool* [optional]): When set to `false`, this sheet will opt-out the `rtl` transformation. When set to `true`, the styles are inversed. When set to `null`, it follows `theme.direction`.
- The other keys are forwarded to the options argument of [jss.createStyleSheet([styles], [options])](https://cssinjs.org/jss-api/#create-style-sheet).

Returns

`Component`: The new component created.

Examples

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

const MyComponent = styled('div')({
  backgroundColor: 'red',
});

const MyThemeComponent = styled('div')(({ theme }) => ({
  padding: theme.spacing(1),
}));

const theme = createTheme();

export default function StyledComponents() {
  return (
    <ThemeProvider theme={theme}>
      <MyThemeComponent>
        <MyComponent />
      </MyThemeComponent>
    <ThemeProvider>
  );
}
```

`StylesProvider`

This component allows you to change the behavior of the styling solution. It makes the options available down the React tree thanks to the context.

It should preferably be used at **the root of your component tree**.

Props

Name	Type	Default	Description
children *	node		Your component tree.
disableGeneration	bool	false	You can disable the generation of the styles with this option. It can be useful when traversing the React tree outside of the HTML rendering step on the server. Let's say you are using react-apollo to extract all the queries made by the interface server-side. You can significantly speed up the traversal with this prop.
generateClassName	func		JSS's class name generator.
injectFirst	bool	false	By default, the styles are injected last in the `<head>` element of the page. As a result, they gain more specificity than any other style sheet. If you want to override MUI's styles, set this prop.
jss	object		JSS's instance.

Examples

```
import * as React from 'react';
import ReactDOM from 'react-dom';
import { StylesProvider } from '@mui/styles';

function App() {
  return <StylesProvider jss={jss}>...</StylesProvider>;
}
```

```
ReactDOM.render(<App />, document.querySelector('#app'));
```

`ThemeProvider`

This component takes a `theme` prop, and makes it available down the React tree thanks to the context. It should preferably be used at **the root of your component tree**.

Props

Name	Type	Default	Description
children *	node		Your component tree.
theme *	union: object | func		A theme object. You can provide a function to extend the outer theme.

Examples

```
import * as React from 'react';
import ReactDOM from 'react-dom';
import { ThemeProvider } from '@mui/material/styles';

const theme = {};

function App() {
  return <ThemeProvider theme={theme}>...</ThemeProvider>;
}

ReactDOM.render(<App />, document.querySelector('#app'));
```

`useTheme() => theme`

This hook returns the `theme` object so it can be used inside a function component.

Returns

`theme`: The theme object previously injected in the context.

Examples

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

export default function MyComponent() {
  const theme = useTheme();

  return <div>{`spacing ${theme.spacing}`}</div>;
}
```

`withStyles(styles, [options]) => higher-order component`

Link a style sheet with a component using the **higher-order component** pattern. It does not modify the component passed to it; instead, it returns a new component with a `classes` prop. This `classes` object contains the name of the class names injected in the DOM.

Some implementation details that might be interesting to being aware of:

- It adds a `classes` prop so you can override the injected class names from the outside.
- It forwards refs to the inner component.
- It does **not** copy over statics. For instance, it can be used to define a `getInitialProps()` static method (next.js).

Arguments

1. `styles` (*Function* | *Object*): A function generating the styles or a styles object. It will be linked to the component. Use the function signature if you need to have access to the theme. It's provided as the first argument.
2. `options` (*object* [optional]):

- `options.defaultTheme` (*object* [optional]): The default theme to use if a theme isn't supplied through a Theme Provider.
- `options.withTheme` (*bool* [optional]): Defaults to `false`. Provide the `theme` object to the component as a prop.
- `options.name` (*string* [optional]): The name of the style sheet. Useful for debugging. If the value isn't provided, it will try to fallback to the name of the component.
- `options.flip` (*bool* [optional]): When set to `false`, this sheet will opt-out the `rtl` transformation. When set to `true`, the styles are inversed. When set to `null`, it follows `theme.direction`.
- The other keys are forwarded to the options argument of jss.createStyleSheet([styles], [options]).

Returns

`higher-order component`: Should be used to wrap a component.

Examples

```
import * as React from 'react';
import { withStyles } from '@mui/styles';

const styles = {
  root: {
    backgroundColor: 'red',
  },
};

function MyComponent(props) {
  return <div className={props.classes.root} />;
}

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

Also, you can use as decorators like so:

```
import * as React from 'react';
import { withStyles } from '@mui/styles';

const styles = {
  root: {
    backgroundColor: 'red',
  },
};

@withStyles(styles)
class MyComponent extends React.Component {
  render() {
    return <div className={this.props.classes.root} />;
  }
}

export default MyComponent;
```

`withTheme(Component) => Component`

Provide the `theme` object as a prop of the input component so it can be used in the render method.

Arguments

1. `Component`: The component that will be wrapped.

Returns

`Component`: The new component created. Does forward refs to the inner component.

Examples

```
import * as React from 'react';
import { withTheme } from '@mui/styles';

function MyComponent(props) {
  return <div>{props.theme.direction}</div>;
}

export default withTheme(MyComponent);
```