

# styled()

Utilidade para criar componentes estilizados.

## Introduction

All the MUI components are styled with this `styled()` utility. This utility is built on top of the `styled()` module of [@mui/styled-engine](#) and provides additional features.

## Importação

You can use the utility coming from the `@mui/system` package, or if you are using `@mui/material`, you can import it from `@mui/material/styles`. The difference is in the default `theme` that is used (if no theme is available in the React context).

## What problems does it solve?

The utility can be used as a replacement for emotion's or styled-components' `styled()` utility. It aims to solve the same problem, but also provides the following benefits:

1. It uses MUI's default `theme` if no theme is available in React context.
2. It supports the theme's `styleOverrides` and `variants` to be applied, based on the `name` applied in the options (can be skipped).
3. It adds support for the `the sx .prop` (can be skipped).
4. It adds by default the `shouldForwardProp` option (that can be overridden), taking into account all props used internally in the MUI components: `ownerState`, `theme`, `sx`, and `as`.

## API

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

### Arguments

1. `Component`: The component that will be wrapped.
2. `options` (*object* [optional]):
  - `options.shouldForwardProp` (*(prop: string) => bool* [optional]): Indicates whether the `prop` should be forwarded to the `Component`.
  - `options.label` (*string* [optional]): The suffix of the style sheet. Useful for debugging.
  - `options.name` (*string* [optional]): The key used under `theme.components` for specifying `styleOverrides` and `variants`. Also used for generating the `label`.
  - `options.slot` (*string* [optional]): If `Root`, it automatically applies the theme's `styleOverrides` & `variants`.
  - `options.overridesResolver` (*((props: object, styles: Record<string, styles>) => styles* [optional]): Function that returns styles based on the props and the `theme.components[name].styleOverrides` object.
  - `options.skipVariantsResolver` (*bool*): Disables the automatic resolver for the `theme.components[name].variants`.
  - `options.skipSx` (*bool* [optional]): Disables the `sx` prop on the component.

- The other keys are forwarded to the `options` argument of emotion's `styled([Component], [options])`.

### Returns

`Component` : O novo componente criado.

## Basic usage

```
{{"demo": "BasicUsage.js", "defaultCodeOpen": true}}
```

## Using the theme

```
{{"demo": "ThemeUsage.js", "defaultCodeOpen": true}}
```

## Custom components

This example demonstrates how you can use the `styled` API to create custom components, with the same capabilities as the core components:

```
{{"demo": "UsingOptions.js", "defaultCodeOpen": true}}
```

If you inspect this element with the browser DevTools in development mode, you will notice that the class of the component now ends with the `MyThemeComponent-root`, which comes from the `name` and `slot` options that were provided.



browser DevTools showing the rendered component

In addition to this, the `color`, `sx`, and `variant` props are not propagated to the generated `div` element.

## Removing features

If you would like to remove some of the MUI specific features, you can do it like this:

```
const StyledComponent = styled('div', {}, {  
  name: 'MuiStyled',  
  slot: 'Root',  
  - overridesResolver: (props, styles) => styles.root, // disables  
  theme.components[name].styleOverrides  
  + skipVariantsResolver: true, // disables theme.components[name].variants  
  + skipSx: true, // disables the sx prop  
});
```

## Create custom `styled()` utility

If you want to have a different default theme for the `styled()` utility, you can create your own version of it, using the `createStyled()` utility.

```
import { createStyled, createTheme } from '@mui/system';
```

```
const defaultTheme = createTheme({
  // your custom theme values
});

const styled = createStyled({ defaultTheme });

export default styled;
```

## Difference with the `sx` prop

The `styled` function is an extension of the `styled` utility provided by the underlying style library used – either emotion or styled-components. It is guaranteed that it will produce the same output as the `styled` function coming from the style library for the same input.

The `sx` prop, on the other hand, is a new way of styling your components, focused on fast customization.

`styled` is a function, while `sx` is a prop of the MUI components.

Therefore, you will notice the following differences:

### `sx` provides more shortcuts than `styled`

With `styled`:

```
const MyStyledButton = styled('button') ({
  mx: 1, // ❌ don't use this! This shortcut is only provided by the `sx` prop
});
```

With `sx`:

```
const MyStyledButton = (props) => (
  <button sx={{
    mx: 1, // ✅ this shortcut is specific to the `sx` prop,
  }}>
    {props.children}
  </button>
)
```

### The style definition varies slightly

With `styled`:

```
const MyStyledButton = styled('button') ({
  padding: 1, // means "1px", NOT "theme.spacing(1)"
});
```

With `sx`:

```
const MyStyledButton = (props) => (
  <button sx={{
```

```
padding: 1 // means "theme.spacing(1)", NOT "1px"
}}>
    {props.children}
</button>
})
```

## Patterns for how to use props differ

With `styled`:

```
const MyStyledButton = styled('button')((props) => ({
  backgroundColor: props.myBackgroundColor,
}));
```

With `sx`:

```
const MyStyledButton = (props) => (
  <button sx={{backgroundColor: props.myCustomColor}}>
    {props.children}
  </button>
)
```

## Parameter when using function are different for each field

With `styled` (not recommended):

```
// You may find this syntax in the wild, but for code readability
// we recommend using only one top-level function
const MyStyledButtonPropsPerField = styled('button')({
  backgroundColor: (props) => props.myBackgroundColor,
});
```

With `sx`:

```
import { lighten } from "polished"
const MyStyledButton = (props) => (
  <button sx={{backgroundColor: theme => lighten(0.2,theme.palette.primary.main)}}>
    {props.children}
  </button>
)
// Note: for direct theme access without modification, you can also use a shortcut
// by providing the key as a string
const MyStyledButton = (props) => (
  <button sx={{backgroundColor: "primary.main"}}>
    {props.children}
  </button>
)
```

## How can I use the `sx` syntax with the `styled()` utility?

If you are one of those who prefers the `sx` syntax and wants to use it in both the `sx` prop and the `styled()` utility, you can use the `experimental_sx` utility from the `@mui/system` :

```
{{"demo": "UsingWithSx.js", "defaultCodeOpen": true}}
```

The overhead added by using the `experimental_sx` utility is the same as if you were to use the `sx` prop on the component.

*Note: You can use `experimental_sx` outside of the `styled()` utility, too; e.g., for defining `variants` in your custom theme.*

## How to use components selector API

If you've ever used the `styled()` API of either [emotion](#) or [styled-components](#) , you should have been able to use components as selectors.

```
import styled from '@emotion/styled';

const Child = styled.div`
  color: red;
`;

const Parent = styled.div`
  ${Child} {
    color: green;
  }
`;

render(
  <div>
    <Parent>
      <Child>Green because I am inside a Parent</Child>
    </Parent>
    <Child>Red because I am not inside a Parent</Child>
  </div>,
);
```

With MUI's `styled()` utility, you can use components as selectors, too. When using `@mui/styled-engine-sc` ( `styled-components` ), nothing needs to be done. When using `@mui/styled-engine` ( `emotion` ), the default engine, there are a few steps you should perform:

First, you should install [@emotion/babel-plugin](#) .

```
npm install @emotion/babel-plugin
```

Then, configure the plugin to know about the MUI version of the `styled()` utility:

**babel.config.js**

```
module.exports = {
  ...
  plugins: [
    [
      "@emotion",
      {
        importMap: {
          "@mui/system": {
            styled: {
              canonicalImport: ["@emotion/styled", "default"],
              styledBaseImport: ["@mui/system", "styled"]
            }
          },
          "@mui/material/styles": {
            styled: {
              canonicalImport: ["@emotion/styled", "default"],
              styledBaseImport: ["@mui/material/styles", "styled"]
            }
          }
        }
      }
    ]
  ]
};
```

Now you should be able to use components as your selectors!