





useMediaQuery

This React hook listens for matches to a CSS media query. It allows the rendering of components based on whether the query matches or not.



For Figma - A large UI kit with over 600 handcrafted Material UI, MUI X, Joy UI components 🎨.
ad by MUI

Some of the key features:

-  It has an idiomatic React API.
-  It's performant, it observes the document to detect when its media queries change, instead of polling the values periodically.
-  [1.1 kB gzipped](#).
-  It supports server-side rendering.

[View as Markdown](#)[Feedback](#)[Bundle size](#)[Source](#)

Basic media query



You should provide a media query to the first argument of the hook. The media query string can be any valid CSS media query, for example `'(prefers-color-scheme: dark)'`.

(min-width:600px) matches: true

[Edit in Chat](#)

JS

TS

[Collapse code](#)

```
import useMediaQuery from '@mui/material/useMediaQuery';

export default function SimpleMediaQuery() {
  const matches = useMediaQuery('(min-width:600px)');

  return <span>{(min-width:600px) matches: ${matches}}</span>;
}
```

⚠ Using the query `'print'` to modify a document for printing is not supported, as changes made in re-rendering may not be accurately reflected. You can use the `sx` prop's `displayPrint` field for this purpose instead. See [MUI System—Display in print](#) for more details.

Using breakpoint helpers



You can use Material UI's [breakpoint helpers](#) as follows:

```
import { useTheme } from '@mui/material/styles';
import useMediaQuery from '@mui/material/useMediaQuery';

function MyComponent() {
  const theme = useTheme();
  const matches = useMediaQuery(theme.breakpoints.up('sm'));

  return <span>{`theme.breakpoints.up('sm') matches: ${matches}`}</span>;
}
```

Copy

theme.breakpoints.up('sm') matches: true

✨ Edit in Chat

JS

TS

Hide code



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

function MyComponent() {
  const theme = useTheme();
  const matches = useMediaQuery(theme.breakpoints.up('sm'));

  return <span>{`theme.breakpoints.up('sm') matches: ${matches}`}</span>;
}

const theme = createTheme();

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

Alternatively, you can use a callback function, accepting the theme as a first argument:

```
import useMediaQuery from '@mui/material/useMediaQuery';

function MyComponent() {
  const matches = useMediaQuery((theme) => theme.breakpoints.up('sm'));

  return <span>{`theme.breakpoints.up('sm') matches: ${matches}`}</span>;
}
```

Copy

⚠ There is **no default** theme support, you have to inject it in a parent theme provider.

Using JavaScript syntax



You can use [json2mq](#) to generate media query string from a JavaScript object.

```
{ minWidth: 600 } matches: true
```

[Edit in Chat](#)

JS

TS

[Collapse code](#)


```
import json2mq from 'json2mq';
import useMediaQuery from '@mui/material/useMediaQuery';

export default function JavaScriptMedia() {
  const matches = useMediaQuery(
    json2mq({
      minWidth: 600,
    }),
  );

  return <span>{`${ { minWidth: 600 } matches: ${matches}`}</span>;
}
```

Testing



You need an implementation of [matchMedia](#) in your test environment.

For instance, [jsdom doesn't support it yet](#). You should polyfill it. Using [css-mediaquery](#) to emulate it is recommended.

```
import mediaQuery from 'css-mediaquery';

function createMatchMedia(width) {
  return (query) => ({
    matches: mediaQuery.match(query, {
      width,
    }),
    addEventListener: () => {},
    removeEventListener: () => {},
  });
}
```

Copy

```

}

describe('MyTests', () => {
  beforeAll(() => {
    window.matchMedia = createMatchMedia(window.innerWidth);
  });
});

```

Client-side only rendering +

To perform the server-side hydration, the hook needs to render twice. A first time with `defaultMatches`, the value of the server, and a second time with the resolved value. This double pass rendering cycle comes with a drawback: it's slower. You can set the `noSsr` option to `true` if you use the returned value **only** client-side.

```
const matches = useMediaQuery('(min-width:600px)', { noSsr: true });
```

Copy

or it can turn it on globally with the theme:

```

const theme = createTheme({
  components: {
    MuiUseMediaQuery: {
      defaultProps: {
        noSsr: true,
      },
    },
  },
});

```

Copy

i Note that `noSsr` has no effects when using the `createRoot()` API (the client-side only API introduced in React 18).

Server-side rendering +

⚠ Server-side rendering and client-side media queries are fundamentally at odds. Be aware of the tradeoff. The support can only be partial.

Try relying on client-side CSS media queries first. For instance, you could use:

- `<Box display>`
- `themes.breakpoints.up(x)`
- or `sx_prop`

If none of the above alternatives are an option, you can proceed reading this section of the documentation.

First, you need to guess the characteristics of the client request, from the server. You have the choice between using:

- **User agent.** Parse the user agent string of the client to extract information. Using [ua-parser-js](#) to parse the user agent is recommended.
- **Client hints.** Read the hints the client is sending to the server. Be aware that this feature is [not supported everywhere](#).

Finally, you need to provide an implementation of [matchMedia](#) to the `useMediaQuery` with the previously guessed characteristics. Using [css-mediaquery](#) to emulate matchMedia is recommended.

For instance on the server-side:

```
import * as ReactDOMServer from 'react-dom/server';
import parser from 'ua-parser-js';
import mediaQuery from 'css-mediaquery';
import { createTheme, ThemeProvider } from '@mui/material/styles';

function handleRender(req, res) {
  const deviceType = parser(req.headers['user-agent']).device.type || 'desktop';
  const ssrMatchMedia = (query) => ({
    matches: mediaQuery.match(query, {
      // The estimated CSS width of the browser.
      width: deviceType === 'mobile' ? '0px' : '1024px',
    }),
  });

  const theme = createTheme({
    components: {
      // Change the default options of useMediaQuery
      MuiUseMediaQuery: {
        defaultProps: {
          ssrMatchMedia
        }
      }
    }
  });

  ReactDOMServer.renderToString(
    <ThemeProvider theme={theme}>
      <MyComponent />
    </ThemeProvider>
  );
}
```

Copy

(min-width:600px) matches: true

 Edit in Chat

JS

TS

Hide code



```
import mediaQuery from 'css-mediaquery';
import { ThemeProvider, Theme } from '@mui/material/styles';
import useMediaQuery from '@mui/material/useMediaQuery';

function MyComponent() {
  const matches = useMediaQuery('(min-width:600px)');

  return <span>{(min-width:600px) matches: ${matches}}</span>;
}

export default function ServerSide() {
```

```
const ssrMatchMedia = (query: string) => ({
  matches: mediaQuery.match(query, {
    // The estimated CSS width of the browser.
    width: 800,
  }),
});

return (
  <ThemeProvider<Theme>
    theme={{
      components: {
        MuiUseMediaQuery: {
          // Change the default options of useMediaQuery
          defaultProps: { ssrMatchMedia },
        },
      },
    }}
  >
    <MyComponent />
  </ThemeProvider>
);
}
```

Make sure you provide the same custom match media implementation to the client-side to guarantee a hydration match.

Migrating from `withWidth()`



The `withWidth()` higher-order component injects the screen width of the page. You can reproduce the same behavior with a `useWidth` hook:

width: sm

Edit in Chat

JS

TS

Collapse code



```
import {
  Breakpoint,
  Theme,
  ThemeProvider,
  useTheme,
  createTheme,
} from '@mui/material/styles';
import useMediaQuery from '@mui/material/useMediaQuery';

type BreakpointOrNull = Breakpoint | null;

/**
 * Be careful using this hook. It only works because the number of
```

```

* breakpoints in theme is static. It will break once you change the number of
* breakpoints. See https://legacy.reactjs.org/docs/hooks-rules.html#only-call-hooks-at-the-top-level
*/
function useWidth() {
  const theme: Theme = useTheme();
  const keys: readonly Breakpoint[] = [...theme.breakpoints.keys].reverse();
  return (
    keys.reduce((output: BreakpointOrNull, key: Breakpoint) => {
      // TODO: uncomment once we enable eslint-plugin-react-compiler // eslint-disable-next-line react
      // eslint-disable-next-line react-hooks/rules-of-hooks
      const matches = useMediaQuery(theme.breakpoints.up(key));
      return !output && matches ? key : output;
    }, null) || 'xs'
  );
}

function MyComponent() {
  const width = useWidth();
  return <span>{`width: ${width}`}</span>;
}

const theme = createTheme();

export default function UseWidth() {
  return (

```

API

`useMediaQuery(query, [options]) => matches`

Arguments

1. `query` (*string* | *func*): A string representing the media query to handle or a callback function accepting the theme (in the context) that returns a string.
2. `options` (*object* [optional]):
 - `options.defaultMatches` (*bool* [optional]): As `window.matchMedia()` is unavailable on the server, it returns a default matches during the first mount. The default value is `false`.
 - `options.matchMedia` (*func* [optional]): You can provide your own implementation of *matchMedia*. This can be used for handling an iframe content window.
 - `options.noSsr` (*bool* [optional]): Defaults to `false`. To perform the server-side hydration, the hook needs to render twice. A first time with `defaultMatches`, the value of the server, and a second time with the resolved value. This double pass rendering cycle comes with a drawback: it's slower. You can set this option to `true` if you use the returned value **only** client-side.
 - `options.ssrMatchMedia` (*func* [optional]): You can provide your own implementation of *matchMedia*, it's used when rendering server-side.

Note: You can change the default options using the `defaultProps` feature of the theme with the `MuiUseMediaQuery` key.

Returns

`matches`: Matches is `true` if the document currently matches the media query and `false` when it does not.


Examples



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

export default function SimpleMediaQuery() {
  const matches = useMediaQuery('(min-width:600px)');

  return <span>`(min-width:600px) matches: ${matches}`</span>;
}
```

Copy

 Edit this page

Was this page helpful?  

[< Transitions](#)

[Data Grid >](#)