





useMediaQuery

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

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 kB gzipped](#).
-  It supports server-side rendering.

```
{{"component": "modules/components/ComponentLinkHeader.js", "design": false}}
```

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, e.g. `'(prefers-color-scheme: dark)'`.

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

⚠ You can't use `'print'` per browsers limitation, e.g. [Firefox](#).

Using MUI's breakpoint helpers

You can use MUI'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>;
}
```

```
{{"demo": "ThemeHelper.js", "defaultCodeOpen": false}}
```

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>;
}
```

⚠ 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.

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

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,
    }),
    addListener: () => {},
    removeListener: () => {},
  });
}

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 `false`, 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 are doing **client-side only** rendering.

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

or it can turn it on globally with the theme:

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

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 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,
        },
      },
    },
  });

  const html = ReactDOMServer.renderToString(
    <ThemeProvider theme={theme}>
      <App />
  );
}
```

```

    </ThemeProvider>,
  );

  // ...
}

```

```

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

```

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:

```

{"demo": "UseWidth.js"}

```

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, we return 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 `false`, 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 are doing **client-side only** rendering.
 - `options.ssrMatchMedia` (*func* [optional]): You can provide your own implementation of *matchMedia* in a [server-side rendering context](#).

Note: You can change the default options using the [default_props](#) 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>;
}
```