Icons

Guidance and suggestions for using icons with MUI.

MUI provides icons support in three ways:

1. Standardized Material Design icons exported as React components (SVG icons).
2. With the SvgIcon component, a React wrapper for custom SVG icons.
3. With the Icon component, a React wrapper for custom font icons.

Material icons

Google has created over 2,000 official Material icons, each in five different "themes" (see below). For each SVG icon, we export the respective React component from the `@mui/icons-material` package. You can search the full list of these icons.

Installation

Install the package in your project directory with:

```
// with npm
npm install @mui/icons-material

// with yarn
yarn add @mui/icons-material
```

These components use the MUI `SvgIcon` component to render the SVG path for each icon, and so have a peer-dependency on `@mui/material`.

If you aren't already using Material UI in your project, you can add it following the installation guide.

Usage

Import icons using one of these two options:

- Option 1:

```
import AccessAlarmIcon from '@mui/icons-material/AccessAlarm';
import ThreeDRotation from '@mui/icons-material/ThreeDRotation';
```

- Option 2:

```
import { AccessAlarm, ThreeDRotation } from '@mui/icons-material';
```

The safest for bundle size is Option 1, but some developers prefer Option 2. Make sure you follow the minimizing bundle size guide before using the second approach.

Each Material icon also has a "theme": Filled (default), Outlined, Rounded, Two-tone, and Sharp. To import the icon component with a theme other than the default, append the theme name to the icon name. For example `@mui/icons-material/Delete` icon with:

- Filled theme (default) is exported as `@mui/icons-material/Delete`,
- Outlined theme is exported as `@mui/icons-material/DeleteOutlined`,
- Rounded theme is exported as `@mui/icons-material/DeleteRounded`,
- Twotone theme is exported as `@mui/icons-material/DeleteTwoTone`,
- Sharp theme is exported as `@mui/icons-material/DeleteSharp`.

> *Note: The Material Design guidelines name the icons using "snake_case" naming (for example `delete_forever`, `add_a_photo`), while `@mui/icons-material` exports the respective icons using "PascalCase" naming (for example `DeleteForever`, `AddAPhoto`). There are three exceptions to this naming rule: `3d_rotation` exported as `ThreeDRotation`, `4k` exported as `FourK`, and `360` exported as `ThreeSixty`.*

{{"demo": "SvgMaterialIcons.js"}}

Testing

For testing purposes, each icon exposed from `@mui/icons-material` has a `data-testid` attribute with the name of the icon. For instance:

```
import DeleteIcon from '@mui/icons-material/Delete';
```

has the following attribute once mounted:

```
<svg data-testid="DeleteIcon"></svg>
```

SvgIcon

If you need a custom SVG icon (not available in the Material Icons) you can use the `SvgIcon` wrapper. This component extends the native `<svg>` element:

- It comes with built-in accessibility.
- SVG elements should be scaled for a 24x24px viewport so that the resulting icon can be used as is, or included as a child for other MUI components that use icons. This can be customized with the `viewBox` attribute. To inherit the `viewBox` value from the original image, the `inheritViewBox` prop can be used.
- By default, the component inherits the current color. Optionally, you can apply one of the theme colors using the `color` prop.

```
function HomeIcon(props) {
  return (
    <SvgIcon {...props}>
      <path d="M10 20v-6h4v6h5v-8h3L12 3 2 12h3v8z" />
    </SvgIcon>
  );
}
```

Color

{{"demo": "SvgIconsColor.js"}}

Size

{{"demo": "SvgIconsSize.js"}}

Component prop

You can use the `SvgIcon` wrapper even if your icons are saved in the `.svg` format. svgr has loaders to import SVG files and use them as React components. For example, with webpack:

```
// webpack.config.js
{
  test: /\.svg$/,
  use: ['@svgr/webpack'],
}

// ---
import StarIcon from './star.svg';

<SvgIcon component={StarIcon} inheritViewBox />
```

It's also possible to use it with "url-loader" or "file-loader". This is the approach used by Create React App.

```
// webpack.config.js
{
  test: /\.svg$/,
  use: ['@svgr/webpack', 'url-loader'],
}

// ---
import { ReactComponent as StarIcon } from './star.svg';

<SvgIcon component={StarIcon} inheritViewBox />
```

createSvgIcon

The `createSvgIcon` utility component is used to create the Material icons. It can be used to wrap an SVG path with an SvgIcon component.

```
const HomeIcon = createSvgIcon(
  <path d="M10 20v-6h4v6h5v-8h3L12 3 2 12h3v8z" />,
  'Home',
);
```

{{"demo": "CreateSvgIcon.js"}}

Font Awesome

If you find that there are layout issues when using FontAwesomeIcon from `@fortawesome/react-fontawesome`, you can try passing the Font Awesome SVG data directly to SvgIcon.

Below is a comparison of the `FontAwesomeIcon` component and a wrapped `SvgIcon` component.

{{"demo": "FontAwesomeSvgIconDemo.js"}}

FontAwesomeIcon's `fullWidth` prop can also be used to approximate the correct dimensions, but it isn't perfect.

Other Libraries

MDI

materialdesignicons.com provides over 2,000 icons. For the wanted icon, copy the SVG `path` they provide, and use it as the child of the `SvgIcon` component, or with `createSvgIcon()`.

Note: mdi-material-ui has already wrapped each of these SVG icons with the `SvgIcon` component, so you don't have to do it yourself.

Icon (Font icons)

The `Icon` component will display an icon from any icon font that supports ligatures. As a prerequisite, you must include one, such as the Material icon font in your project. To use an icon simply wrap the icon name (font ligature) with the `Icon` component, for example:

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

<Icon>star</Icon>;
```

By default, an Icon will inherit the current text color. Optionally, you can set the icon color using one of the theme color properties: `primary`, `secondary`, `action`, `error` & `disabled`.

Font Material icons

`Icon` will by default set the correct base class name for the Material Icons font (filled variant). All you need to do is load the font, for instance, via Google Web Fonts:

```
<link
  rel="stylesheet"
  href="https://fonts.googleapis.com/icon?family=Material+Icons"
/>
```

{{"demo": "Icons.js"}}

Custom font

For other fonts, you can customize the baseline class name using the `baseClassName` prop. For instance, you can display two-tone icons with Material Design:

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

<link
  rel="stylesheet"
  href="https://fonts.googleapis.com/css?family=Material+Icons+Two+Tone"
  // Import the two tones MD variant                          ^^^^^^^^
/>;
```

{{"demo": "TwoToneIcons.js"}}

Global base class name

Modifying the `baseClassName` prop for each component usage is repetitive. You can change the default prop globally with the theme

```
const theme = createTheme({
  components: {
    MuiIcon: {
      defaultProps: {
        // Replace the `material-icons` default value.
        baseClassName: 'material-icons-two-tone',
      },
    },
  },
});
```

Then, you can use the two-tone font directly:

```
<Icon>add_circle</Icon>
```

Font Awesome

Font Awesome can be used with the `Icon` component as follows:

{{"demo": "FontAwesomeIcon.js"}}

Note that the Font Awesome icons weren't designed like the Material Design icons (compare the two previous demos). The fa icons are cropped to use all the space available. You can adjust for this with a global override:

```
const theme = createTheme({
  components: {
    MuiIcon: {
      styleOverrides: {
        root: {
          // Match 24px = 3 * 2 + 1.125 * 16
          boxSizing: 'content-box',
          padding: 3,
          fontSize: '1.125rem',
        },
      },
    },
  },
});
```

{{"demo": "FontAwesomeIconSize.js"}}

Font vs SVG. Which approach to use?

Both approaches work fine, however there are some subtle differences, especially in terms of performance and rendering quality. Whenever possible SVG is preferred as it allows code splitting, supports more icons, and renders faster and better.

For more details, take a look at why GitHub migrated from font icons to SVG icons.

Accessibility

Icons can convey all sorts of meaningful information, so it's important to ensure they are accessible where appropriate. There are two use cases you'll want to consider:

- **Decorative icons** that are only being used for visual or branding reinforcement. If they were removed from the page, users would still understand and be able to use your page.
- **Semantic icons** are ones that you're using to convey meaning, rather than just pure decoration. This includes icons without text next to them that are used as interactive controls — buttons, form elements, toggles, etc.

Decorative icons

If your icons are purely decorative, you're already done! The `aria-hidden=true` attribute is added so that your icons are properly accessible (invisible).

Semantic icons

Semantic SVG icons

You should include the `titleAccess` prop with a meaningful value. The `role="img"` attribute and the `<title>` element are added so that your icons are correctly accessible.

In the case of focusable interactive elements, for example when used with an icon button, you can use the `aria-label` prop:

```
import IconButton from '@mui/material/IconButton';
import SvgIcon from '@mui/material/SvgIcon';

// ...

<IconButton aria-label="delete">
  <SvgIcon>
    <path d="M20 12l-1.41-1.41L13 16.17V4h-2v12.17l-5.58-5.59L4 12l8 8 8-8z" />
  </SvgIcon>
</IconButton>;
```

Semantic font icons

You need to provide a text alternative that is only visible to assistive technologies.

```
import Box from '@mui/material/Box';
import Icon from '@mui/material/Icon';
import { visuallyHidden } from '@mui/utils';

// ...
```

```
<Icon>add_circle</Icon>
<Box component="span" sx={visuallyHidden}>Create a user</Box>
```

Reference

- https://www.tpgi.com/using-aria-enhance-svg-accessibility/