Tooltip

Tooltips display informative text when users hover over, focus on, or tap an element.

Design and Development tips in your inbox. Every weekday.

ads via Carbon

When activated, Tooltips display a text label identifying an element, such as a description of its function.

View as Markdown
Feedback
Bundle size
Source
WAI-ARIA
Material Design
Figma
Sketch

Basic tooltip

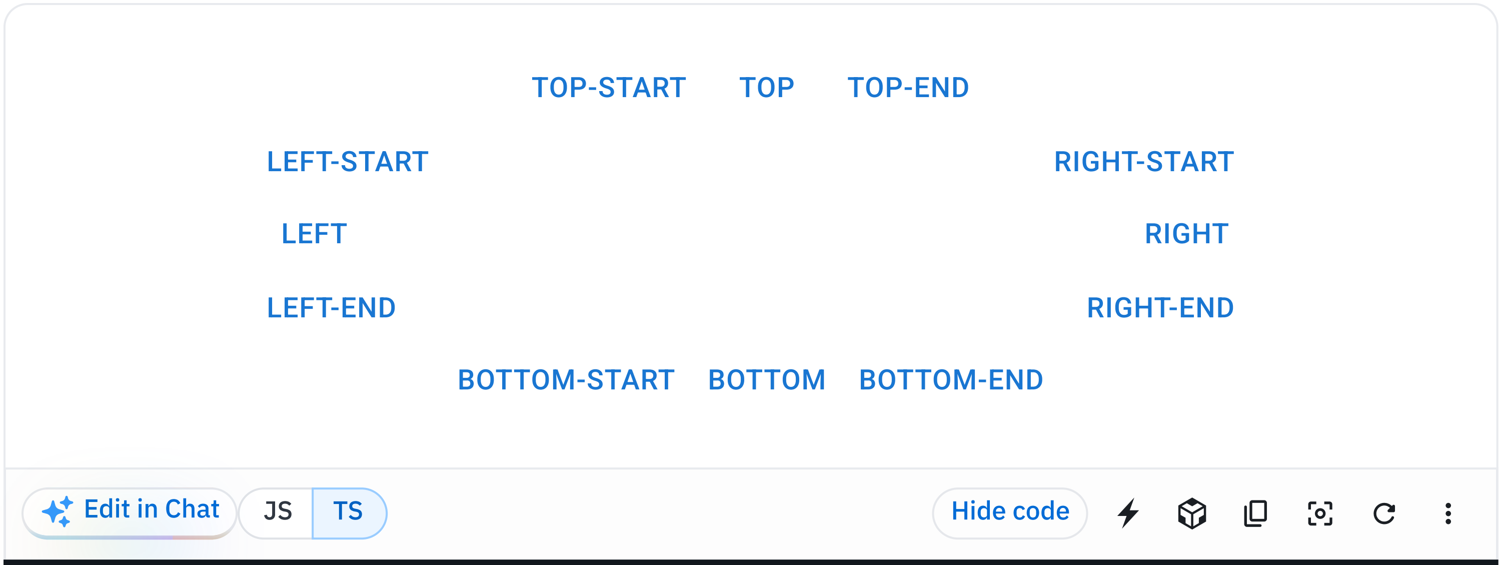
Edit in Chat
JS
TS
Collapse code

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

export default function BasicTooltip() {
  return (
    <Tooltip title="Delete">
      <IconButton>
        <DeleteIcon />
      </IconButton>
    </Tooltip>
  );
}
```

Positioned tooltips

The `Tooltip` has 12 **placement** choices. They don't have directional arrows; instead, they rely on motion emanating from the source to convey direction.



```
import Box from '@mui/material/Box';
import Grid from '@mui/material/Grid';
import Button from '@mui/material/Button';
import Tooltip from '@mui/material/Tooltip';

export default function PositionedTooltips() {
  return (
    <Box sx={{ width: 500 }}>
      <Grid container sx={{ justifyContent: 'center' }}>
        <Grid>
          <Tooltip title="Add" placement="top-start">
            <Button>top-start</Button>
          </Tooltip>
          <Tooltip title="Add" placement="top">
            <Button>top</Button>
          </Tooltip>
          <Tooltip title="Add" placement="top-end">
            <Button>top-end</Button>
          </Tooltip>
        </Grid>
      </Grid>
      <Grid container sx={{ justifyContent: 'center' }}>
        <Grid size={6}>
          <Tooltip title="Add" placement="left-start">
            <Button>left-start</Button>
          </Tooltip>
          <br />
          <Tooltip title="Add" placement="left">
            <Button>left</Button>
          </Tooltip>
          <br />
          <Tooltip title="Add" placement="left-end">
            <Button>left-end</Button>
          </Tooltip>
        </Grid>
        <Grid container direction="column" sx={{ alignItems: 'flex-end' }} size={6}>
          <Grid>
            <Tooltip title="Add" placement="right-start">
```

Customization

Here are some examples of customizing the component. You can learn more about this in the overrides documentation page.

LIGHT BOOTSTRAP HTML

Edit in Chat JS TS Hide code

```tsx
import * as React from 'react';
import { styled } from '@mui/material/styles';
import Button from '@mui/material/Button';
import Tooltip, { TooltipProps, tooltipClasses } from '@mui/material/Tooltip';
import Typography from '@mui/material/Typography';

const LightTooltip = styled(({ className, ...props }: TooltipProps) => (
  <Tooltip {...props} classes={{ popper: className }} />
))(({ theme }) => ({
  [`& .${tooltipClasses.tooltip}`]: {
    backgroundColor: theme.palette.common.white,
    color: 'rgba(0, 0, 0, 0.87)',
    boxShadow: theme.shadows[1],
    fontSize: 11,
  },
}));

const BootstrapTooltip = styled(({ className, ...props }: TooltipProps) => (
  <Tooltip {...props} arrow classes={{ popper: className }} />
))(({ theme }) => ({
  [`& .${tooltipClasses.arrow}`]: {
    color: theme.palette.common.black,
  },
  [`& .${tooltipClasses.tooltip}`]: {
    backgroundColor: theme.palette.common.black,
  },
}));

const HtmlTooltip = styled(({ className, ...props }: TooltipProps) => (
  <Tooltip {...props} classes={{ popper: className }} />
))(({ theme }) => ({
  [`& .${tooltipClasses.tooltip}`]: {
    backgroundColor: '#f5f5f9',
    color: 'rgba(0, 0, 0, 0.87)',
    maxWidth: 220,
    fontSize: theme.typography.pxToRem(12),
    border: '1px solid #dadde9',
  },
```

Arrow tooltips

You can use the `arrow` prop to give your tooltip an arrow indicating which element it refers to.

ARROW

Edit in Chat JS TS Collapse code

```
import Button from '@mui/material/Button';
import Tooltip from '@mui/material/Tooltip';

export default function ArrowTooltips() {
  return (
    <Tooltip title="Add" arrow>
      <Button>Arrow</Button>
    </Tooltip>
  );
}
```

Distance from anchor

To adjust the distance between the tooltip and its anchor, you can use the `slotProps` prop to modify the [offset](https://popper.js.org/docs/v2/modifiers/offset/) of the popper.

OFFSET

Edit in Chat JS TS Hide code

```
import Button from '@mui/material/Button';
import Tooltip from '@mui/material/Tooltip';

export default function TooltipOffset() {
  return (
    <Tooltip
      title="Add"
      slotProps={{
        popper: {
          modifiers: [
            {
              name: 'offset',
              options: {
                offset: [0, -14],
              },
            },
          ],
        },
      }}
    >
      <Button>Offset</Button>
    </Tooltip>
  );
}
```

Alternatively, you can use the `slotProps` prop to customize the margin of the popper.

MARGIN

Edit in Chat JS TS Hide code

```
import Button from '@mui/material/Button';
import Tooltip, { tooltipClasses } from '@mui/material/Tooltip';

export default function TooltipMargin() {
  return (
    <Tooltip
      title="Add"
      slotProps={{
        popper: {
          sx: {
            [`&.${tooltipClasses.popper}[data-popper-placement*="bottom"] .${tooltipClasses.tooltip}`]
              {
                marginTop: '0px',
              },
            [`&.${tooltipClasses.popper}[data-popper-placement*="top"] .${tooltipClasses.tooltip}`]:
              {
                marginBottom: '0px',
              },
            [`&.${tooltipClasses.popper}[data-popper-placement*="right"] .${tooltipClasses.tooltip}`]:
              {
                marginLeft: '0px',
              },
            [`&.${tooltipClasses.popper}[data-popper-placement*="left"] .${tooltipClasses.tooltip}`]:
              {
                marginRight: '0px',
              },
          },
        },
      }}
    >
      <Button>Margin</Button>
    </Tooltip>
  );
}
```

Custom child element

The tooltip needs to apply DOM event listeners to its child element. If the child is a custom React element, you need to make sure that it spreads its props to the underlying DOM element.

```
const MyComponent = React.forwardRef(function MyComponent(props, ref) {
  //  Spread the props to the underlying DOM element.
  return (
    <div {...props} ref={ref}>
      Bin
    </div>
  );
});

// ...

<Tooltip title="Delete">
  <MyComponent />
</Tooltip>;
```

If using a class component as a child, you'll also need to ensure that the ref is forwarded to the underlying DOM element. (A ref to the class component itself will not work.)

```
class MyComponent extends React.Component {
  render() {
    const { innerRef, ...props } = this.props;
    //  Spread the props to the underlying DOM element.
    return (
      <div {...props} ref={innerRef}>
        Bin
      </div>
    );
  }
}

// Wrap MyComponent to forward the ref as expected by Tooltip
const WrappedMyComponent = React.forwardRef(function WrappedMyComponent(props, ref) {
  return <MyComponent {...props} innerRef={ref} />;
});

// ...

<Tooltip title="Delete">
```

Triggers

You can define the types of events that cause a tooltip to show.

The touch action requires a long press due to the `enterTouchDelay` prop being set to `700` ms by default.

HOVER OR TOUCH FOCUS OR TOUCH HOVER CLICK

Edit in Chat JS TS Hide code

```
import * as React from 'react';
import Grid from '@mui/material/Grid';
import Button from '@mui/material/Button';
import Tooltip from '@mui/material/Tooltip';
import ClickAwayListener from '@mui/material/ClickAwayListener';

export default function TriggersTooltips() {
  const [open, setOpen] = React.useState(false);

  const handleTooltipClose = () => {
    setOpen(false);
  };

  const handleTooltipOpen = () => {
    setOpen(true);
  };

  return (
    <div>
      <Grid container sx={{ justifyContent: 'center' }}>
        <Grid>
          <Tooltip disableFocusListener title="Add">
            <Button>Hover or touch</Button>
          </Tooltip>
        </Grid>
        <Grid>
          <Tooltip disableHoverListener title="Add">
            <Button>Focus or touch</Button>
          </Tooltip>
        </Grid>
        <Grid>
          <Tooltip disableFocusListener disableTouchListener title="Add">
            <Button>Hover</Button>
          </Tooltip>
        </Grid>
        <Grid>
          <ClickAwayListener onClickAway={handleTooltipClose}>
            <div>
```

Controlled tooltips

You can use the `open`, `onOpen` and `onClose` props to control the behavior of the tooltip.

CONTROLLED

Edit in Chat JS TS Collapse code

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

export default function ControlledTooltips() {
  const [open, setOpen] = React.useState(false);

  const handleClose = () => {
    setOpen(false);
  };

  const handleOpen = () => {
    setOpen(true);
  };

  return (
    <Tooltip open={open} onClose={handleClose} onOpen={handleOpen} title="Add">
      <Button>Controlled</Button>
    </Tooltip>
  );
}
```

Variable width

The `Tooltip` wraps long text by default to make it readable.

DEFAULT WIDTH [300PX] CUSTOM WIDTH [500PX] NO WRAPPING

Edit in Chat JS TS Collapse code

```
import { styled } from '@mui/material/styles';
import Button from '@mui/material/Button';
import Tooltip, { TooltipProps, tooltipClasses } from '@mui/material/Tooltip';

const CustomWidthTooltip = styled(({ className, ...props }: TooltipProps) => (
  <Tooltip {...props} classes={{ popper: className }} />
))({
  [`& .${tooltipClasses.tooltip}`]: {
    maxWidth: 500,
  },
});

const NoMaxWidthTooltip = styled(({ className, ...props }: TooltipProps) => (
  <Tooltip {...props} classes={{ popper: className }} />
))({
  [`& .${tooltipClasses.tooltip}`]: {
    maxWidth: 'none',
  },
});

const longText = `
Aliquam eget finibus ante, non facilisis lectus. Sed vitae dignissim est, vel aliquam tellus.
Praesent non nunc mollis, fermentum neque at, semper arcu.
Nullam eget est sed sem iaculis gravida eget vitae justo.
`;

export default function VariableWidth() {
  return (
    <div>
      <Tooltip title={longText}>
        <Button sx={{ m: 1 }}>Default Width [300px]</Button>
      </Tooltip>
      <CustomWidthTooltip title={longText}>
        <Button sx={{ m: 1 }}>Custom Width [500px]</Button>
      </CustomWidthTooltip>
      <NoMaxWidthTooltip title={longText}>
        <Button sx={{ m: 1 }}>No wrapping</Button>
```

Interactive

Tooltips are interactive by default (to pass [WCAG 2.1 success criterion 1.4.13](https://www.w3.org/TR/WCAG21/#content-on-hover-or-focus)). It won't close when the user hovers over the tooltip before the `leaveDelay` is expired. You can disable this behavior (thus failing the success criterion which is required to reach level AA) by passing `disableInteractive`.

NOT INTERACTIVE

Edit in Chat JS TS Collapse code

```
import Button from '@mui/material/Button';
import Tooltip from '@mui/material/Tooltip';

export default function NonInteractiveTooltips() {
  return (
    <Tooltip title="Add" disableInteractive>
      <Button>Not interactive</Button>
    </Tooltip>
  );
}
```

Disabled elements

By default disabled elements like `<button>` do not trigger user interactions so a `Tooltip` will not activate on normal events like hover. To accommodate disabled elements, add a simple wrapper element, such as a `span`.

⚠ In order to work with Safari, you need at least one display block or flex item below the tooltip wrapper.

A DISABLED BUTTON

Edit in Chat JS TS Collapse code

```
import Button from '@mui/material/Button';
import Tooltip from '@mui/material/Tooltip';

export default function DisabledTooltips() {
  return (
    <Tooltip title="You don't have permission to do this">
      <span>
        <Button disabled>A Disabled Button</Button>
      </span>
    </Tooltip>
  );
}
```

⚠ If you're not wrapping a Material UI component that inherits from `ButtonBase`, for instance, a native `<button>` element, you should also add the CSS property *pointer-events: none;* to your element when disabled:

```
<Tooltip title="You don't have permission to do this">
  <span>
    <button disabled={disabled} style={disabled ? { pointerEvents: 'none' } : {}}>
      A disabled button
    </button>
```

Copy

```
  </span>
</Tooltip>
```

Transitions

Use a different transition.

GROW FADE ZOOM

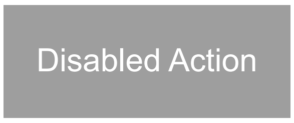
Edit in Chat JS TS Hide code

```
import Button from '@mui/material/Button';
import Tooltip from '@mui/material/Tooltip';
import Fade from '@mui/material/Fade';
import Zoom from '@mui/material/Zoom';

export default function TransitionsTooltips() {
  return (
    <div>
      <Tooltip title="Add">
        <Button>Grow</Button>
      </Tooltip>
      <Tooltip
        title="Add"
        slots={{
          transition: Fade,
        }}
        slotProps={{
          transition: { timeout: 600 },
        }}
      >
        <Button>Fade</Button>
      </Tooltip>
      <Tooltip
        title="Add"
        slots={{
          transition: Zoom,
        }}
      >
        <Button>Zoom</Button>
      </Tooltip>
    </div>
  );
}
```

Follow cursor

You can enable the tooltip to follow the cursor by setting `followCursor={true}`.



Edit in Chat JS TS Collapse code

```
import Box from '@mui/material/Box';
import Tooltip from '@mui/material/Tooltip';

export default function FollowCursorTooltips() {
  return (
    <Tooltip title="You don't have permission to do this" followCursor>
      <Box sx={{ bgcolor: 'text.disabled', color: 'background.paper', p: 2 }}>
        Disabled Action
      </Box>
    </Tooltip>
  );
}
```

Virtual element

In the event you need to implement a custom placement, you can use the `anchorEl` prop: The value of the `anchorEl` prop can be a reference to a fake DOM element. You need to create an object shaped like the `VirtualElement`.

Edit in Chat JS TS Hide code

```
import * as React from 'react';
import Box from '@mui/material/Box';
import Tooltip from '@mui/material/Tooltip';
import { Instance } from '@popperjs/core';

export default function AnchorElTooltips() {
  const positionRef = React.useRef<{ x: number; y: number }>({
    x: 0,
    y: 0,
  });
  const popperRef = React.useRef<Instance>(null);
  const areaRef = React.useRef<HTMLDivElement>(null);

  const handleMouseMove = (event: React.MouseEvent) => {
    positionRef.current = { x: event.clientX, y: event.clientY };

    if (popperRef.current != null) {
      popperRef.current.update();
    }
  };

  return (
    <Tooltip
      title="Add"
      placement="top"
      arrow
      slotProps={{
        popper: {
          popperRef,
          anchorEl: {
            getBoundingClientRect: () => {
              return new DOMRect(
                positionRef.current.x,
                areaRef.current!.getBoundingClientRect().y,
                0,
                0,
              );
            },
```

Showing and hiding

The tooltip is normally shown immediately when the user's mouse hovers over the element, and hides immediately when the user's mouse leaves. A delay in showing or hiding the tooltip can be added through the `enterDelay` and `leaveDelay` props.

On mobile, the tooltip is displayed when the user longpresses the element and hides after a delay of 1500ms. You can disable this feature with the `disableTouchListener` prop.

[500MS, 200MS]

Edit in Chat JS TS Collapse code

```
import Button from '@mui/material/Button';
import Tooltip from '@mui/material/Tooltip';

export default function DelayTooltips() {
  return (
    <Tooltip title="Add" enterDelay={500} leaveDelay={200}>
      <Button>[500ms, 200ms]</Button>
    </Tooltip>
  );
}
```

Accessibility

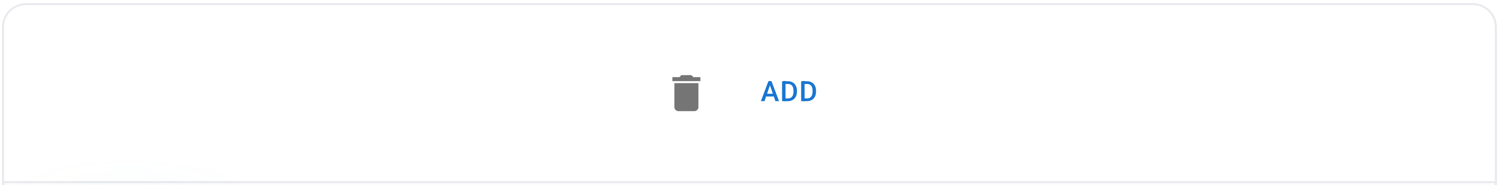
(WAI-ARIA: https://www.w3.org/WAI/ARIA/apg/patterns/tooltip/ ↗)

By default, the tooltip only labels its child element. This is notably different from `title` which can either label **or** describe its child depending on whether the child already has a label. For example, in:

```html
<button title="some more information">A button</button>
```

Copy

the `title` acts as an accessible description. If you want the tooltip to act as an accessible description you can pass `describeChild`. Note that you shouldn't use `describeChild` if the tooltip provides the only visual label. Otherwise, the child would have no accessible name and the tooltip would violate success criterion 2.5.3 in WCAG 2.1 ↗.



Edit in Chat | JS | TS | Collapse code

```tsx
import DeleteIcon from '@mui/icons-material/Delete';
import Button from '@mui/material/Button';
import IconButton from '@mui/material/IconButton';
import Tooltip from '@mui/material/Tooltip';

export default function AccessibilityTooltips() {
  return (
    <div>
      <Tooltip title="Delete">
        <IconButton>
          <DeleteIcon />
        </IconButton>
      </Tooltip>
      <Tooltip describeChild title="Does not add if it already exists.">
        <Button>Add</Button>
      </Tooltip>
    </div>
  );
}
```

API

See the documentation below for a complete reference to all of the props and classes available to the components mentioned here.

- `<Tooltip />`