Text Field

Text Fields let users enter and edit text.

Text fields allow users to enter text into a UI. They typically appear in forms and dialogs.

View as Markdown Feedback Bundle size Source Material Design Figma Sketch

Basic TextField

The `TextField` wrapper component is a complete form control including a label, input, and help text. It comes with three variants: outlined (default), filled, and standard.

Outlined

Filled

Standard

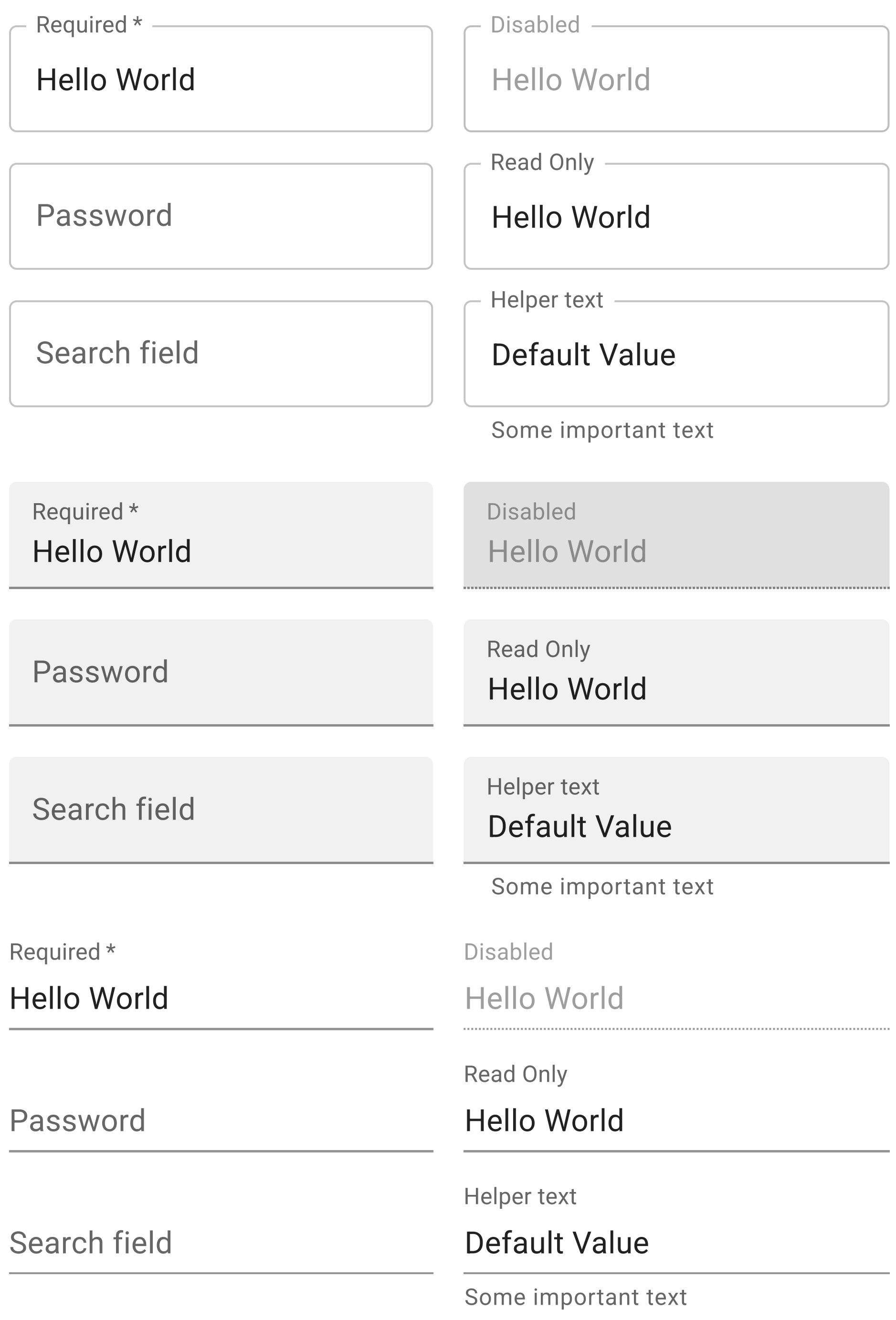
Edit in Chat JS TS Collapse code

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

export default function BasicTextFields() {
  return (
    <Box
      component="form"
      sx={{ '& > :not(style)': { m: 1, width: '25ch' } }}
      noValidate
      autoComplete="off"
    >
      <TextField id="outlined-basic" label="Outlined" variant="outlined" />
      <TextField id="filled-basic" label="Filled" variant="filled" />
      <TextField id="standard-basic" label="Standard" variant="standard" />
    </Box>
  );
}
```

The standard variant of the Text Field is no longer documented in the Material Design guidelines ↗ (this article explains why), but Material UI will continue to support it.

Form props

Standard form attributes are supported, for example `required`, `disabled`, `type`, etc. as well as a `helperText` which is used to give context about a field's input, such as how the input will be used.



Required * Hello World	Disabled Hello World
Password	Read Only Hello World
Search field	Helper text Default Value Some important text
Required * Hello World	Disabled Hello World
Password	Read Only Hello World
Search field	Helper text Default Value Some important text
Required * Hello World	Disabled Hello World
Password	Read Only Hello World
Search field	Helper text Default Value Some important text

Edit in Chat JS TS Hide code

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

export default function FormPropsTextFields() {
  return (
    <Box
      component="form"
      sx={{ '& .MuiTextField-root': { m: 1, width: '25ch' } }}
      noValidate
      autoComplete="off"
    >
      <div>
        <TextField
          required
          id="outlined-required"
          label="Required"
          defaultValue="Hello World"
        />
        <TextField
          disabled
          id="outlined-disabled"
          label="Disabled"
          defaultValue="Hello World"
        />
        <TextField
          id="outlined-password-input"
          label="Password"
          type="password"
          autoComplete="current-password"
        />
        <TextField
          id="outlined-read-only-input"
          label="Read Only"
          defaultValue="Hello World"
          slotProps={{
            input: {
              readOnly: true,
            },
```

Controlling the HTML input

Use `slotProps.htmlInput` to pass attributes to the underlying `<input>` element.

```
<TextField slotProps={{ htmlInput: { 'data-testid': '…' } }} />
```

Copy

The rendered HTML input will look like this:

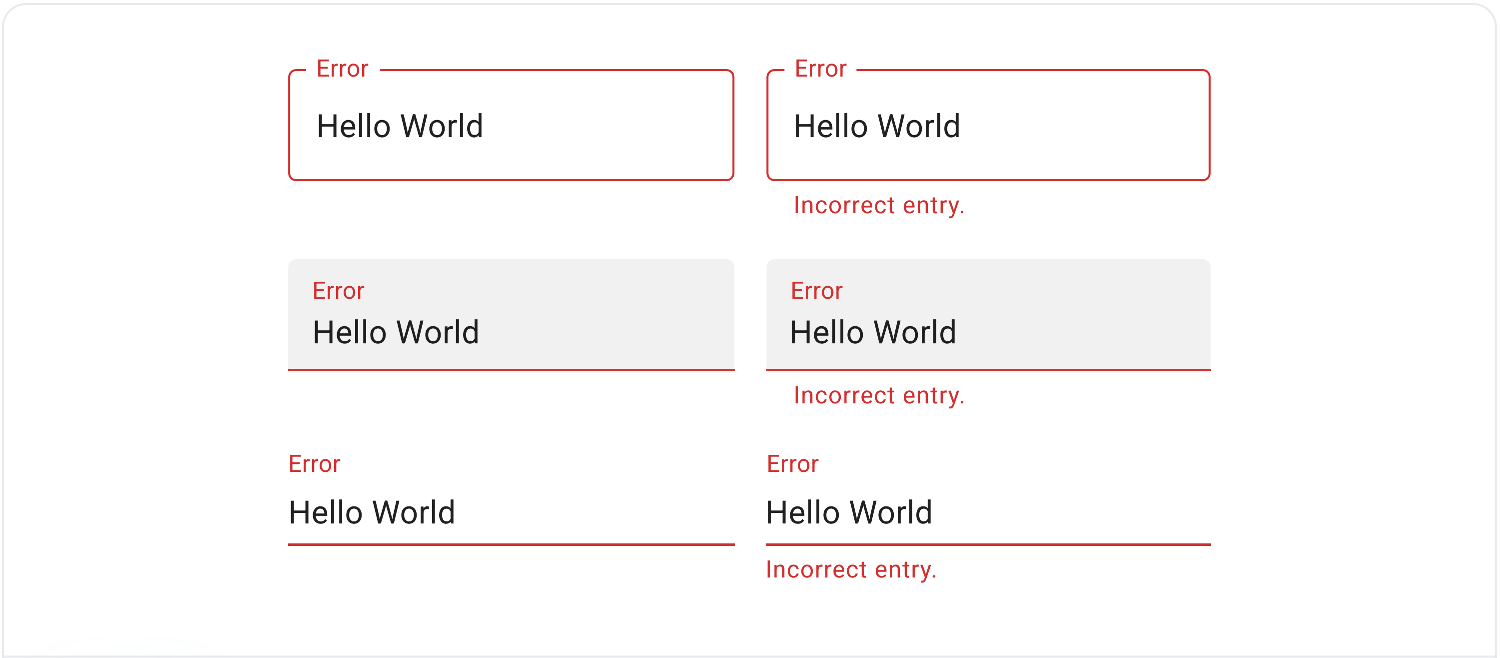
```
<input
  aria-invalid="false"
  class="MuiInputBase-input MuiOutlinedInput-input"
  type="text"
  data-testid="…"
/>
```

Copy

⚠ `slotProps.htmlInput` is not the same as `slotProps.input`. `slotProps.input` refers to the React `<Input />` component that's rendered based on the specified variant prop. `slotProps.htmlInput` refers to the HTML `<input>` element rendered within that Input component, regardless of the variant.

Validation

The `error` prop toggles the error state. The `helperText` prop can then be used to provide feedback to the user about the error.

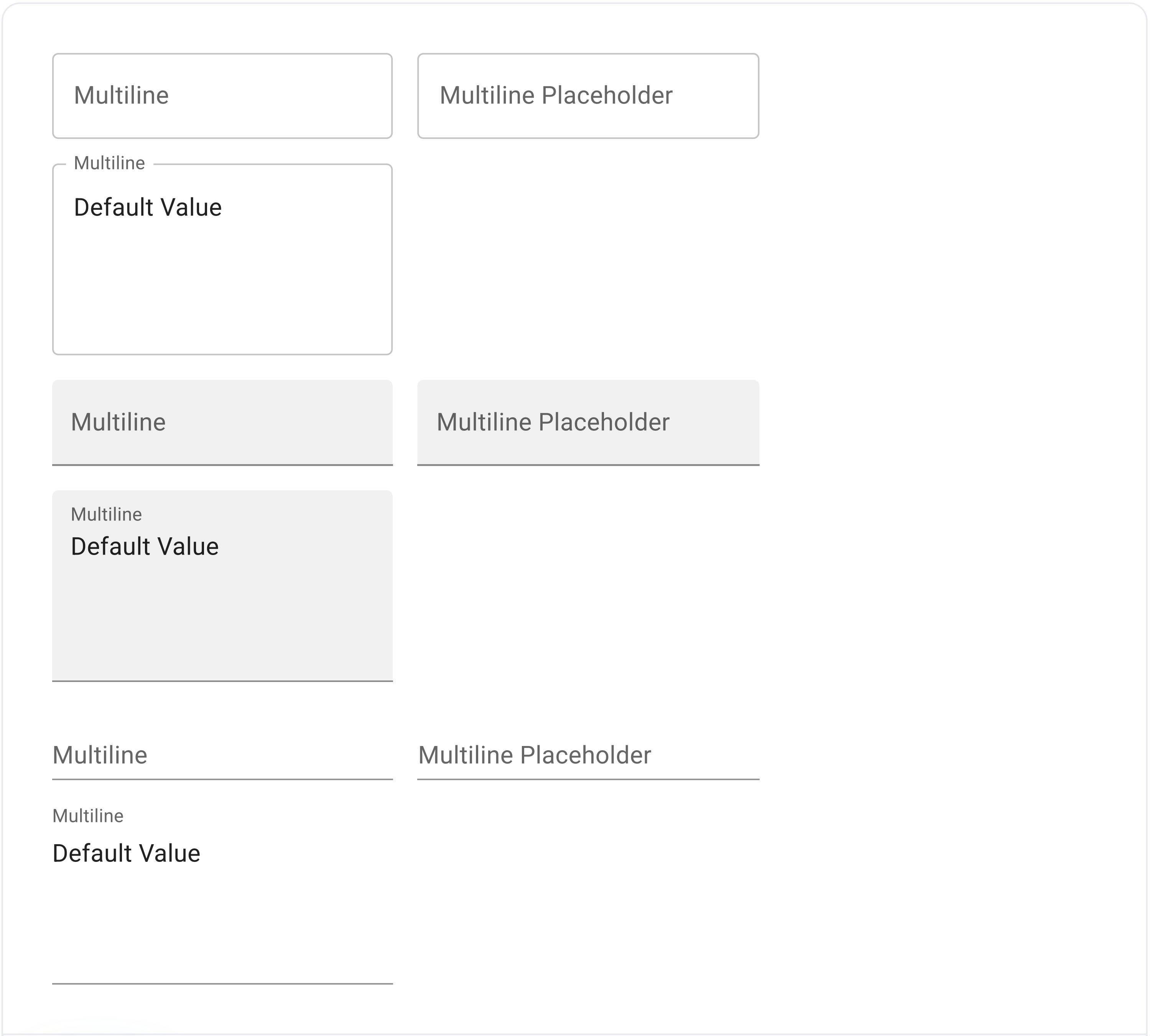


Edit in Chat JS TS Hide code

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

export default function ValidationTextFields() {
  return (
    <Box
      component="form"
      sx={{ '& .MuiTextField-root': { m: 1, width: '25ch' } }}
      noValidate
      autoComplete="off"
    >
      <div>
        <TextField
          error
          id="outlined-error"
          label="Error"
          defaultValue="Hello World"
        />
        <TextField
          error
          id="outlined-error-helper-text"
          label="Error"
          defaultValue="Hello World"
          helperText="Incorrect entry."
        />
      </div>
      <div>
        <TextField
          error
          id="filled-error"
          label="Error"
          defaultValue="Hello World"
          variant="filled"
        />
        <TextField
          error
          id="filled-error-helper-text"
          label="Error"
```

Multiline

The `multiline` prop transforms the Text Field into a [Textarea Autosize](#) element. Unless the `rows` prop is set, the height of the text field dynamically matches its content. You can use the `minRows` and `maxRows` props to bound it.

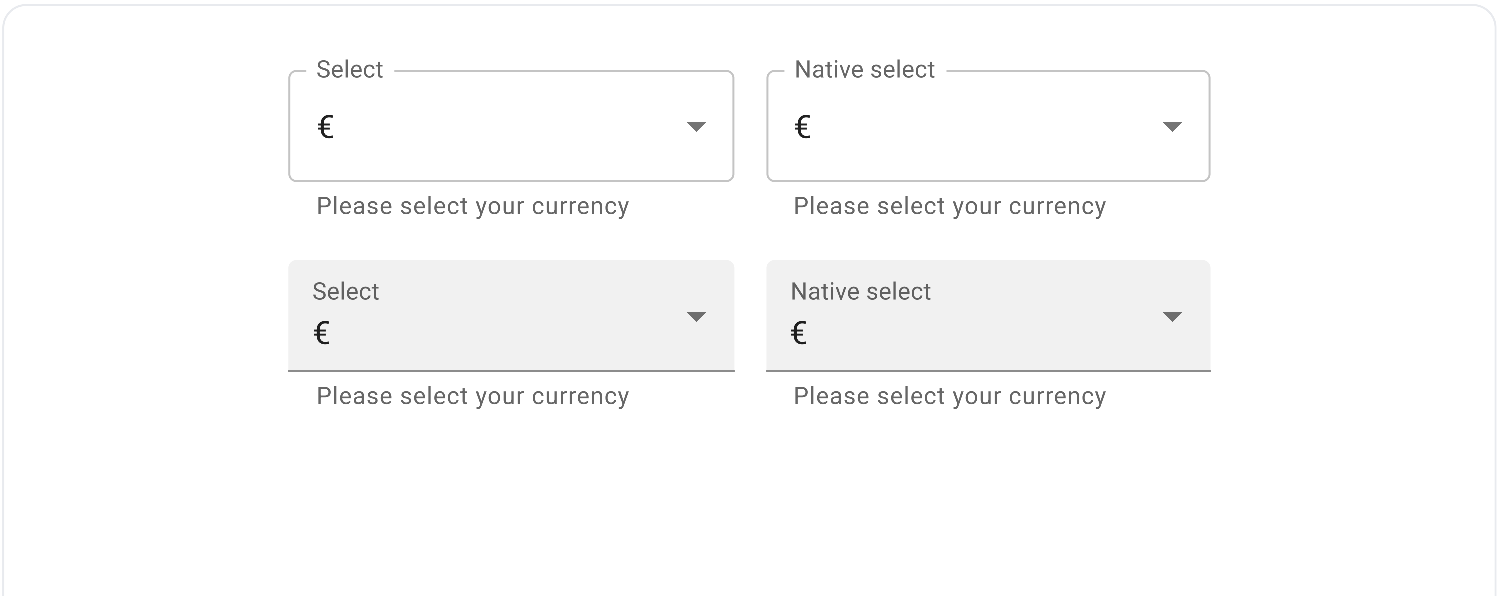


Edit in Chat JS TS Hide code

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

export default function MultilineTextFields() {
  return (
    <Box
      component="form"
      sx={{ '& .MuiTextField-root': { m: 1, width: '25ch' } }}
      noValidate
      autoComplete="off"
    >
      <div>
        <TextField
          id="outlined-multiline-flexible"
          label="Multiline"
          multiline
          maxRows={4}
        />
        <TextField
          id="outlined-textarea"
          label="Multiline Placeholder"
          placeholder="Placeholder"
          multiline
        />
        <TextField
          id="outlined-multiline-static"
          label="Multiline"
          multiline
          rows={4}
          defaultValue="Default Value"
        />
      </div>
      <div>
        <TextField
          id="filled-multiline-flexible"
          label="Multiline"
          multiline
```

Select

The `select` prop makes the text field use the [Select](#) component internally.


Select
€
Native select
€
Please select your currency
Please select your currency
Select
€
Native select
€
Please select your currency
Please select your currency

Select
€
Please select your currency

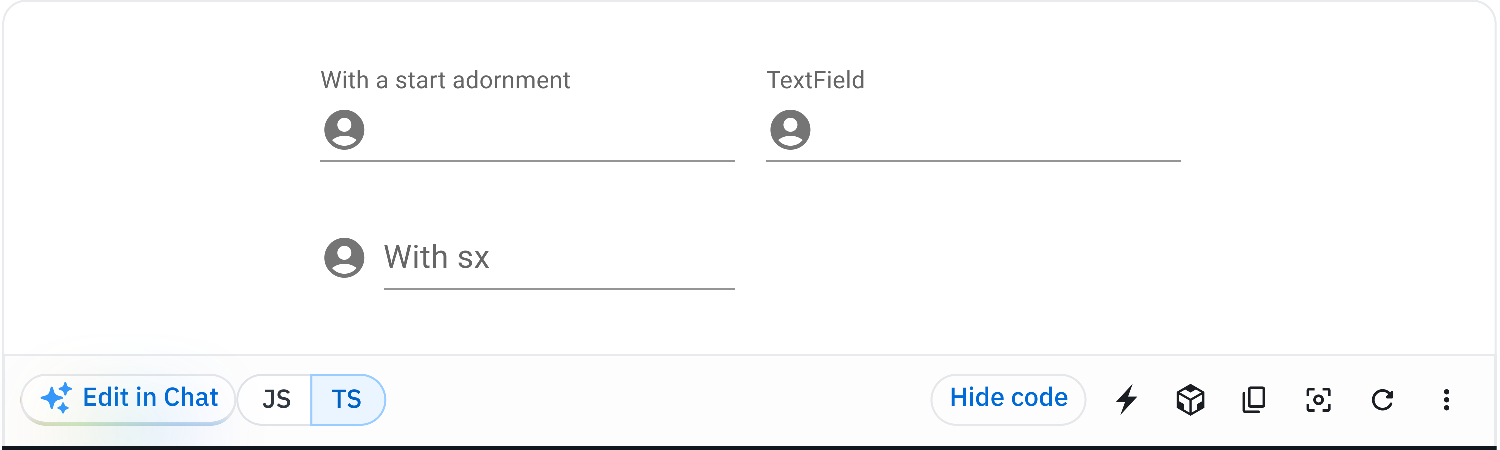
Native select
€
Please select your currency

Edit in Chat JS TS Hide code

```
import Box from '@mui/material/Box';
import TextField from '@mui/material/TextField';
import MenuItem from '@mui/material/MenuItem';

const currencies = [
  {
    value: 'USD',
    label: '$',
  },
  {
    value: 'EUR',
    label: '€',
  },
  {
    value: 'BTC',
    label: '฿',
  },
  {
    value: 'JPY',
    label: '¥',
  },
];

export default function SelectTextFields() {
  return (
    <Box
      component="form"
      sx={{ '& .MuiTextField-root': { m: 1, width: '25ch' } }}
      noValidate
      autoComplete="off"
    >
      <div>
        <TextField
          id="outlined-select-currency"
          select
          label="Select"
          defaultValue="EUR"
          helperText="Please select your currency"
```

Icons

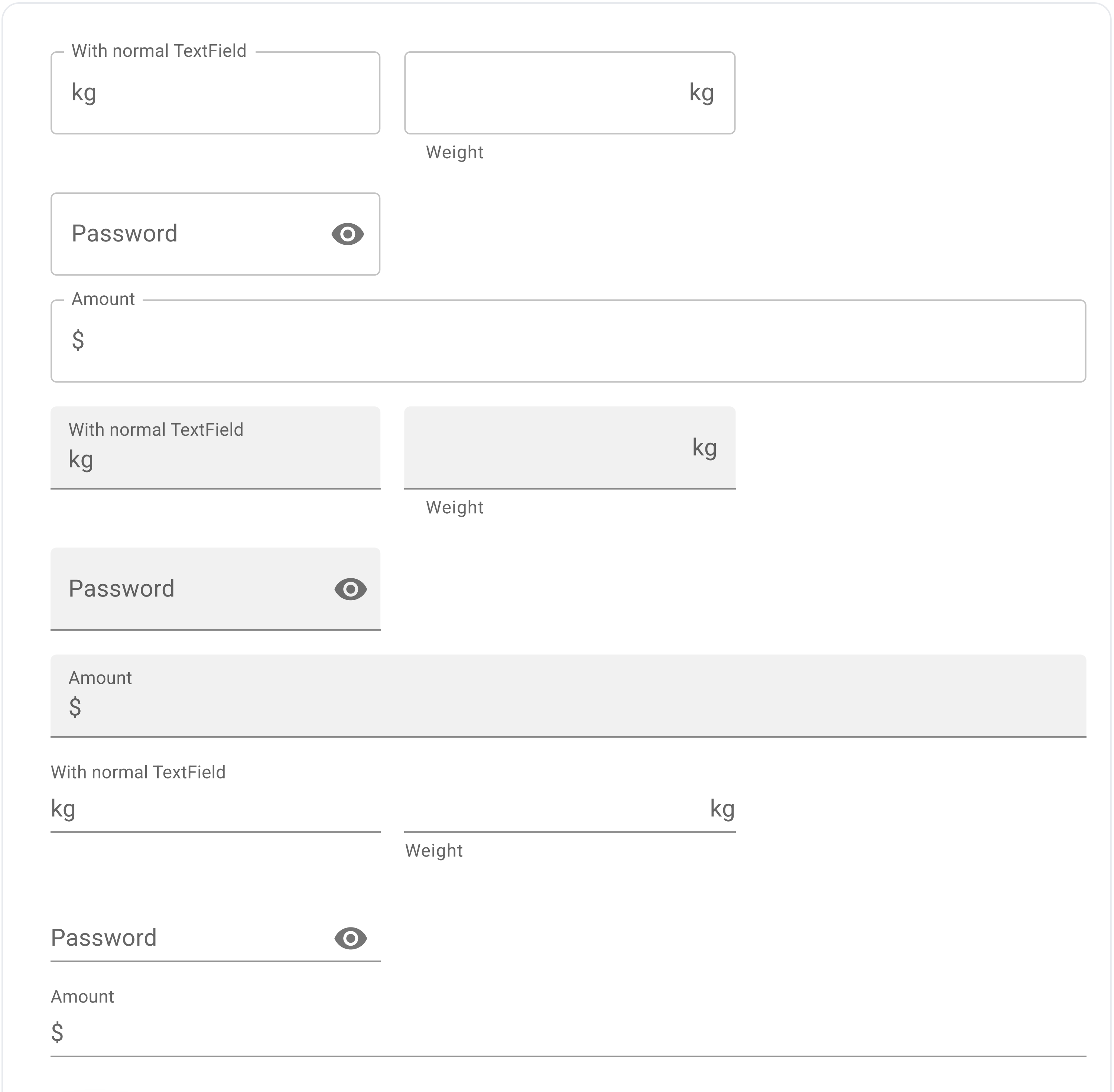
There are multiple ways to display an icon with a text field.



```tsx
import Box from '@mui/material/Box';
import Input from '@mui/material/Input';
import InputLabel from '@mui/material/InputLabel';
import InputAdornment from '@mui/material/InputAdornment';
import FormControl from '@mui/material/FormControl';
import TextField from '@mui/material/TextField';
import AccountCircle from '@mui/icons-material/AccountCircle';

export default function InputWithIcon() {
  return (
    <Box sx={{ '& > :not(style)': { m: 1 } }}>
      <FormControl variant="standard">
        <InputLabel htmlFor="input-with-icon-adornment">
          With a start adornment
        </InputLabel>
        <Input
          id="input-with-icon-adornment"
          startAdornment={
            <InputAdornment position="start">
              <AccountCircle />
            </InputAdornment>
          }
        />
      </FormControl>
      <TextField
        id="input-with-icon-textfield"
        label="TextField"
        slotProps={{
          input: {
            startAdornment: (
              <InputAdornment position="start">
                <AccountCircle />
              </InputAdornment>
            ),
          },
        }}
        variant="standard"
      />
```

Input Adornments

The main way is with an `InputAdornment`. This can be used to add a prefix, a suffix, or an action to an input. For instance, you can use an icon button to hide or reveal the password.

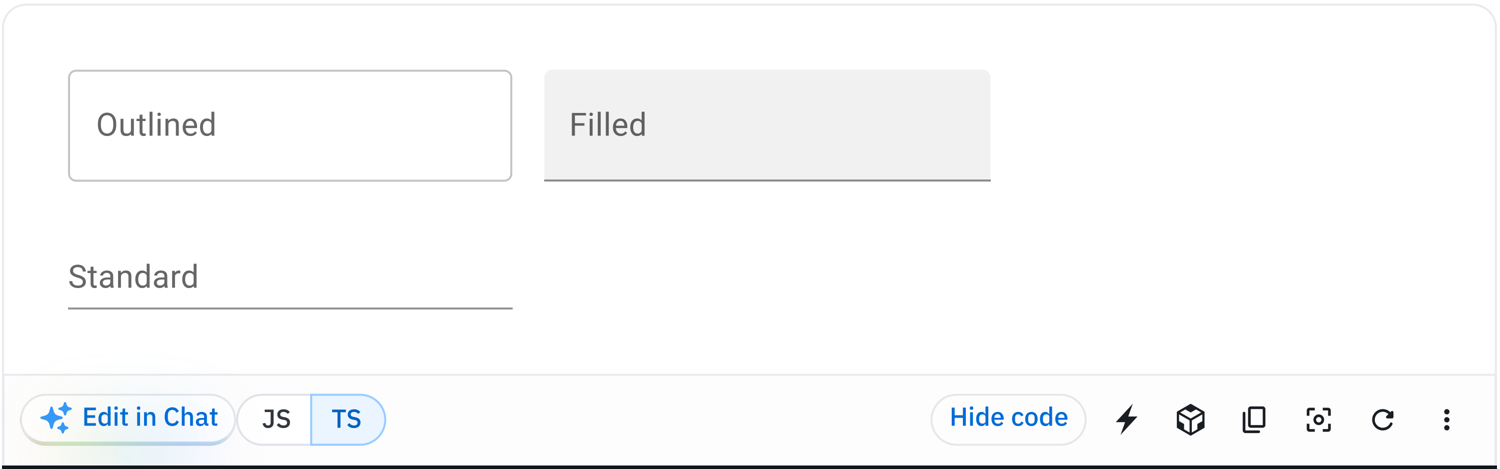


```
import * as React from 'react';
import Box from '@mui/material/Box';
import IconButton from '@mui/material/IconButton';
import Input from '@mui/material/Input';
import FilledInput from '@mui/material/FilledInput';
import OutlinedInput from '@mui/material/OutlinedInput';
import InputLabel from '@mui/material/InputLabel';
import InputAdornment from '@mui/material/InputAdornment';
import FormHelperText from '@mui/material/FormHelperText';
import FormControl from '@mui/material/FormControl';
import TextField from '@mui/material/TextField';
import Visibility from '@mui/icons-material/Visibility';
import VisibilityOff from '@mui/icons-material/VisibilityOff';
```

```
export default function InputAdornments() {
  const [showPassword, setShowPassword] = React.useState(false);

  const handleClickShowPassword = () => setShowPassword((show) => !show);

  const handleMouseDownPassword = (event: React.MouseEvent<HTMLButtonElement>) => {
    event.preventDefault();
  };

  const handleMouseUpPassword = (event: React.MouseEvent<HTMLButtonElement>) => {
    event.preventDefault();
  };

  return (
    <Box sx={{ display: 'flex', flexWrap: 'wrap' }}>
      <div>
        <TextField
          label="With normal TextField"
          id="outlined-start-adornment"
          sx={{ m: 1, width: '25ch' }}
          slotProps={{
            input: {
              startAdornment: <InputAdornment position="start">kg</InputAdornment>,
            }
```

Customizing adornments

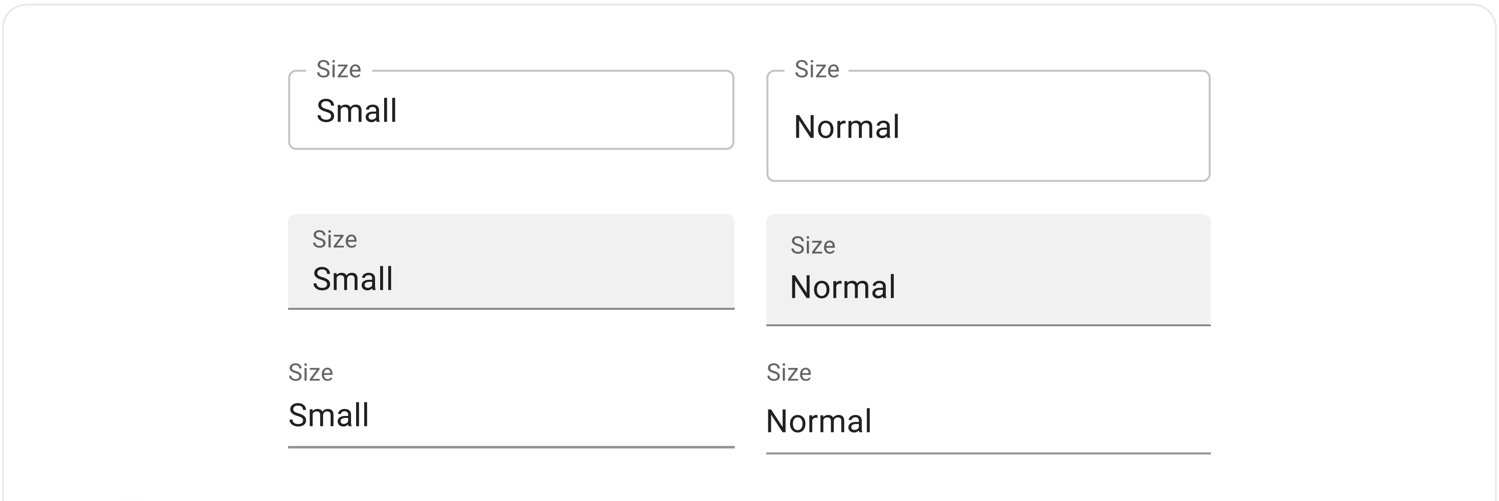
You can apply custom styles to adornments, and trigger changes to one based on attributes from another. For example, the demo below uses the label's `[data-shrink=true]` attribute to make the suffix visible (via opacity) when the label is in its shrunken state.



```
import Box from '@mui/material/Box';
import { filledInputClasses } from '@mui/material/FilledInput';
import { inputBaseClasses } from '@mui/material/InputBase';
import TextField from '@mui/material/TextField';
import InputAdornment from '@mui/material/InputAdornment';

export default function InputSuffixShrink() {
  return (
    <Box
      component="form"
      sx={{ '& > :not(style)': { m: 1, width: '25ch' } }}
      noValidate
      autoComplete="off"
    >
      <TextField
        id="outlined-suffix-shrink"
        label="Outlined"
        variant="outlined"
        slotProps={{
          input: {
            endAdornment: (
              <InputAdornment
                position="end"
                sx={{
                  opacity: 0,
                  pointerEvents: 'none',
                  [`[data-shrink=true] ~ .${inputBaseClasses.root} > &`]: {
                    opacity: 1,
                  },
                }}
              >
                lbs
              </InputAdornment>
            ),
          },
        }}
      />
      <TextField
```

Sizes

Fancy smaller inputs? Use the `size` prop.



Size
Small

Size
Normal

Size
Small

Size
Normal

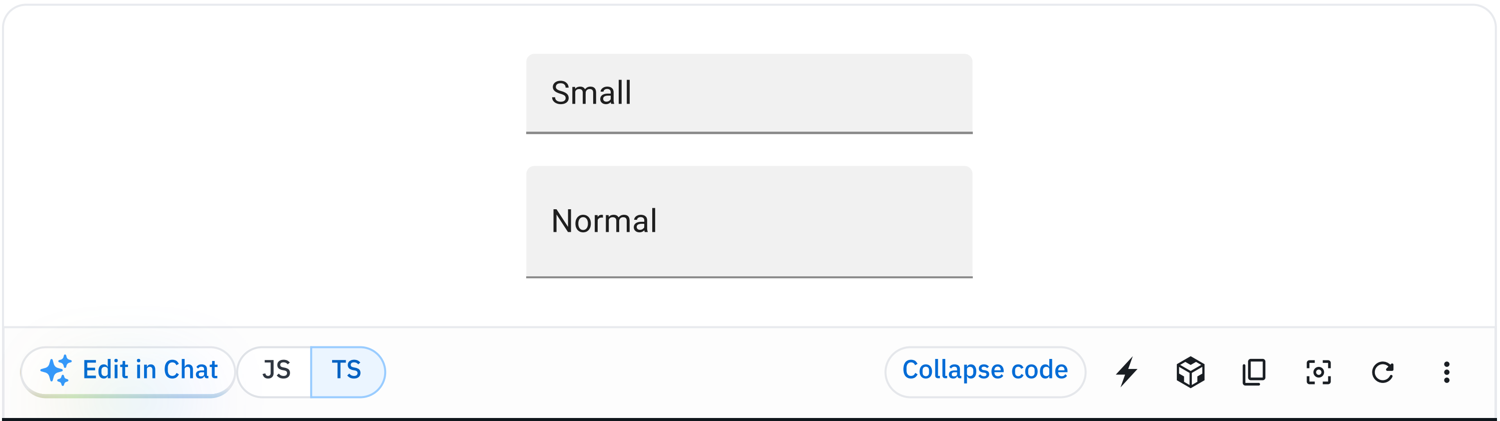
Size
Small

Size
Normal

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

```
export default function TextFieldSizes() {
  return (
    <Box
      component="form"
      sx={{ '& .MuiTextField-root': { m: 1, width: '25ch' } }}
      noValidate
      autoComplete="off"
    >
      <div>
        <TextField
          label="Size"
          id="outlined-size-small"
          defaultValue="Small"
          size="small"
        />
        <TextField label="Size" id="outlined-size-normal" defaultValue="Normal" />
      </div>
      <div>
        <TextField
          label="Size"
          id="filled-size-small"
          defaultValue="Small"
          variant="filled"
          size="small"
        />
        <TextField
          label="Size"
          id="filled-size-normal"
          defaultValue="Normal"
          variant="filled"
        />
      </div>
      <div>
        <TextField
```

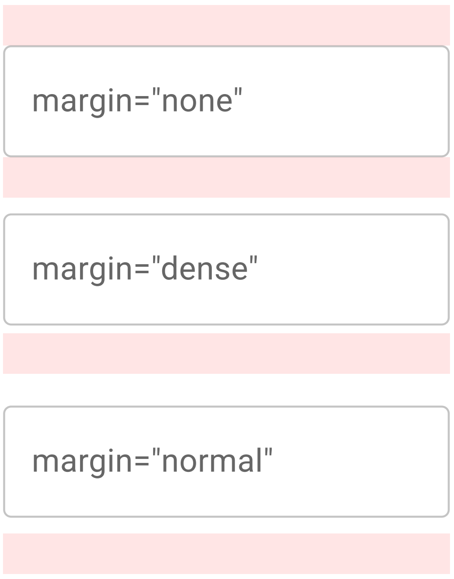
The `filled` variant input height can be further reduced by rendering the label outside of it.



```
import Stack from '@mui/material/Stack';
import TextField from '@mui/material/TextField';

export default function TextFieldHiddenLabel() {
  return (
    <Stack
      component="form"
      sx={{ width: '25ch' }}
      spacing={2}
      noValidate
      autoComplete="off"
    >
      <TextField
        hiddenLabel
        id="filled-hidden-label-small"
        defaultValue="Small"
        variant="filled"
        size="small"
      />
      <TextField
        hiddenLabel
        id="filled-hidden-label-normal"
        defaultValue="Normal"
        variant="filled"
      />
    </Stack>
  );
}
```

Margin

The `margin` prop can be used to alter the vertical spacing of the text field. Using `none` (default) doesn't apply margins to the `FormControl` whereas `dense` and `normal` do.

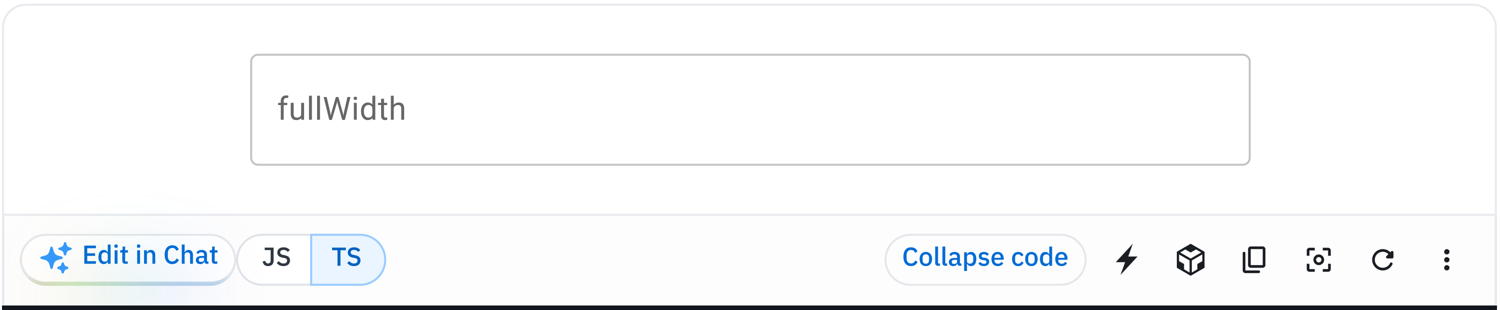


```
import Box from '@mui/material/Box';
import TextField from '@mui/material/TextField';
```

```
function RedBar() {
  return (
    <Box
      sx={(theme) => ({
        height: 20,
        backgroundColor: 'rgba(255, 0, 0, 0.1)',
        ...theme.applyStyles('dark', {
          backgroundColor: 'rgb(255 132 132 / 25%)',
        }),
      })}
    />
  );
}

export default function LayoutTextFields() {
  return (
    <Box
      sx={{
        display: 'flex',
        flexDirection: 'column',
        '& .MuiTextField-root': { width: '25ch' },
      }}
    >
      <RedBar />
      <TextField label={'margin="none"'} id="margin-none" />
      <RedBar />
      <TextField label={'margin="dense"'} id="margin-dense" margin="dense" />
      <RedBar />
      <TextField label={'margin="normal"'} id="margin-normal" margin="normal" />
      <RedBar />
    </Box>
  );
}
```

Full width

`fullWidth` can be used to make the input take up the full width of its container.



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

export default function FullWidthTextField() {
  return (
    <Box sx={{ width: 500, maxWidth: '100%' }}>
```

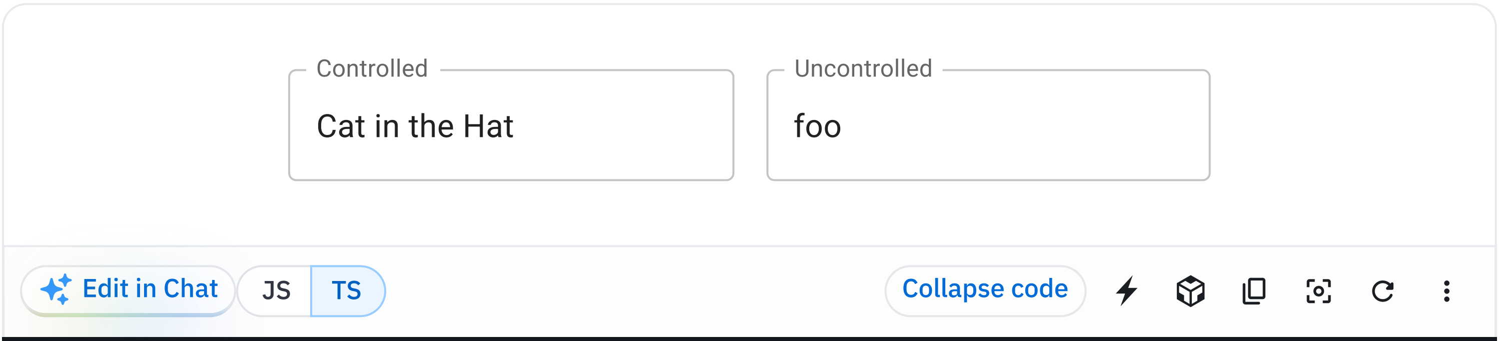
```
      <TextField fullWidth label="fullWidth" id="fullWidth" />
    </Box>
  );
}
```

Uncontrolled vs. Controlled

The component can be controlled or uncontrolled.

- A component is **controlled** when it's managed by its parent using props.
- A component is **uncontrolled** when it's managed by its own local state.

Learn more about controlled and uncontrolled components in the React documentation.

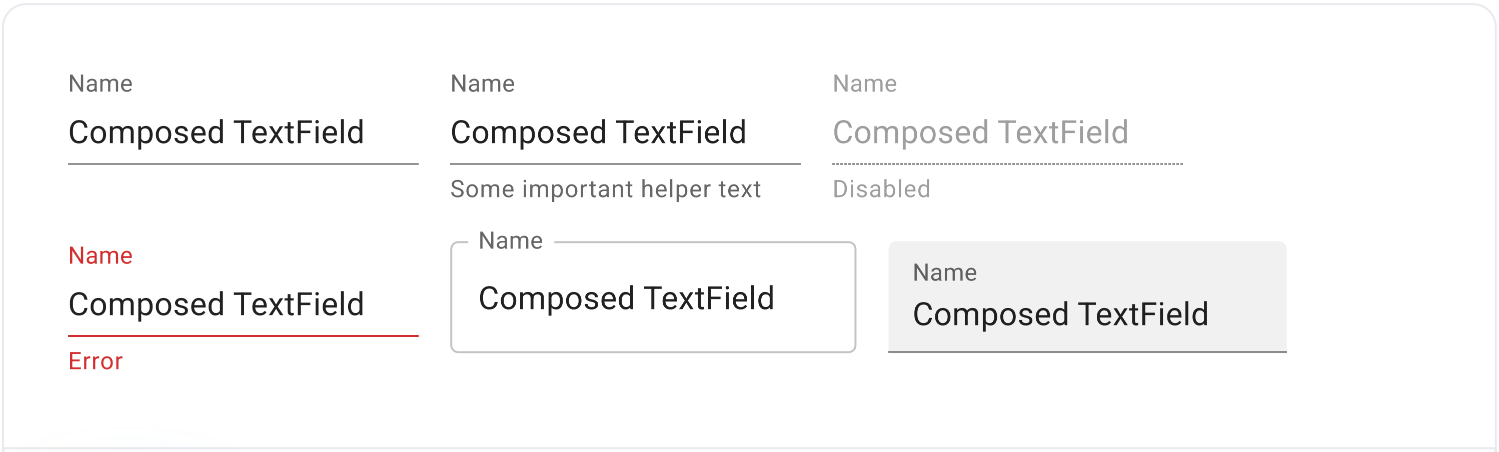


```
import * as React from 'react';
import Box from '@mui/material/Box';
import TextField from '@mui/material/TextField';

export default function StateTextFields() {
  const [name, setName] = React.useState('Cat in the Hat');

  return (
    <Box
      component="form"
      sx={{ '& > :not(style)': { m: 1, width: '25ch' } }}
      noValidate
      autoComplete="off"
    >
      <TextField
        id="outlined-controlled"
        label="Controlled"
        value={name}
        onChange={(event: React.ChangeEvent<HTMLInputElement>) => {
          setName(event.target.value);
        }}
      />
      <TextField
        id="outlined-uncontrolled"
        label="Uncontrolled"
        defaultValue="foo"
      />
    </Box>
  );
}
```

Components

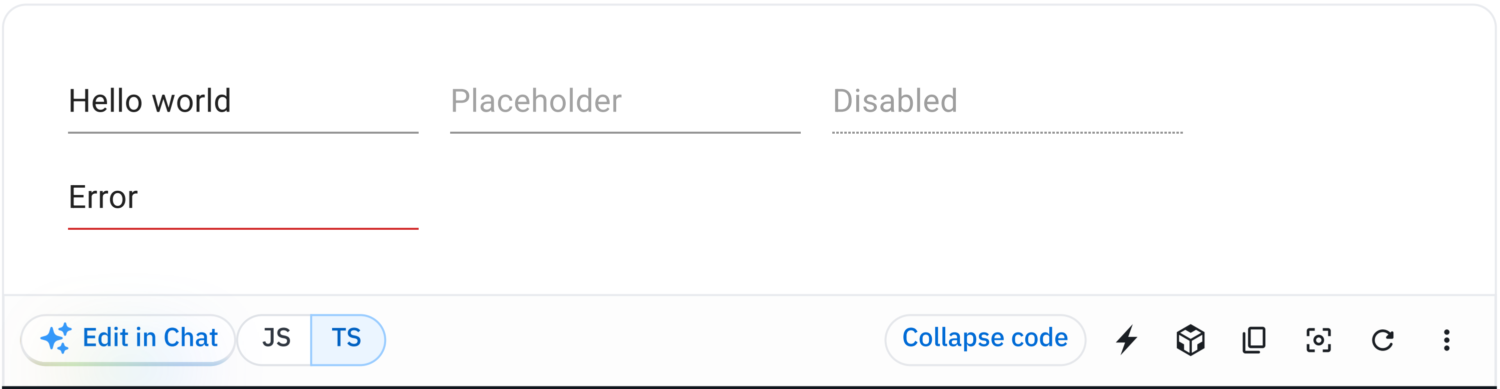
`TextField` is composed of smaller components (`FormControl`, `Input`, `FilledInput`, `InputLabel`, `OutlinedInput`, and `FormHelperText`) that you can leverage directly to significantly customize your form inputs.

You might also have noticed that some native HTML input properties are missing from the `TextField` component. This is on purpose. The component takes care of the most used properties. Then, it's up to the user to use the underlying component shown in the following demo. Still, you can use `slotProps.htmlInput` (and `slotProps.input`, `slotProps.inputLabel` properties) if you want to avoid some boilerplate.



```
import Box from '@mui/material/Box';
import FilledInput from '@mui/material/FilledInput';
import FormControl from '@mui/material/FormControl';
import FormHelperText from '@mui/material/FormHelperText';
import Input from '@mui/material/Input';
import InputLabel from '@mui/material/InputLabel';
import OutlinedInput from '@mui/material/OutlinedInput';

export default function ComposedTextField() {
  return (
    <Box
      component="form"
      sx={{ '& > :not(style)': { m: 1 } }}
      noValidate
      autoComplete="off"
    >
      <FormControl variant="standard">
        <InputLabel htmlFor="component-simple">Name</InputLabel>
        <Input id="component-simple" defaultValue="Composed TextField" />
      </FormControl>
      <FormControl variant="standard">
        <InputLabel htmlFor="component-helper">Name</InputLabel>
        <Input
          id="component-helper"
          defaultValue="Composed TextField"
          aria-describedby="component-helper-text"
        />
        <FormHelperText id="component-helper-text">
          Some important helper text
        </FormHelperText>
      </FormControl>
      <FormControl disabled variant="standard">
        <InputLabel htmlFor="component-disabled">Name</InputLabel>
        <Input id="component-disabled" defaultValue="Composed TextField" />
        <FormHelperText>Disabled</FormHelperText>
      </FormControl>
      <FormControl error variant="standard">
```

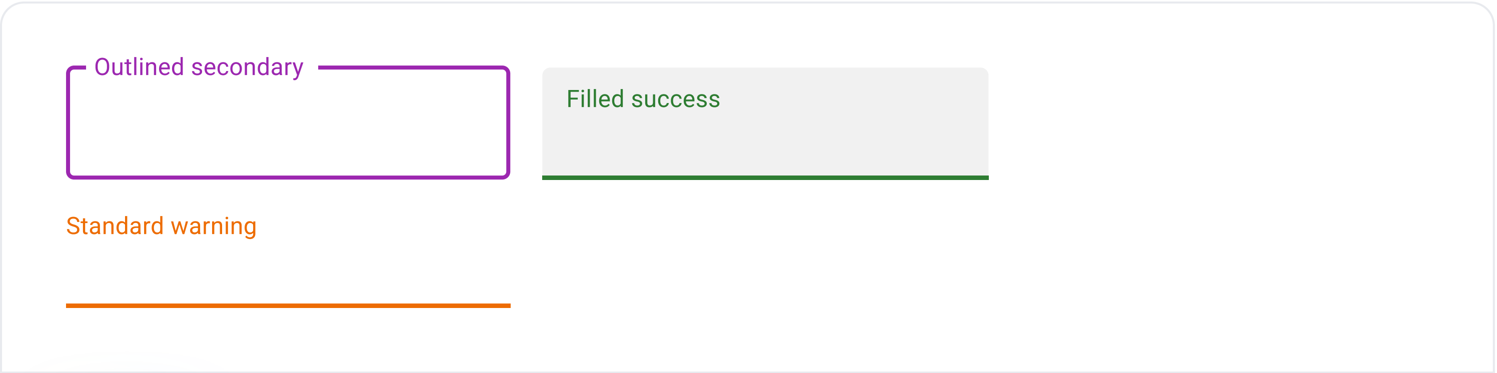
Inputs



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

const ariaLabel = { 'aria-label': 'description' };

export default function Inputs() {
  return (
    <Box
      component="form"
      sx={{ '& > :not(style)': { m: 1 } }}
      noValidate
      autoComplete="off"
    >
      <Input defaultValue="Hello world" inputProps={ariaLabel} />
      <Input placeholder="Placeholder" inputProps={ariaLabel} />
      <Input disabled defaultValue="Disabled" inputProps={ariaLabel} />
      <Input defaultValue="Error" error inputProps={ariaLabel} />
    </Box>
  );
}
```

Color

The `color` prop changes the highlight color of the text field when focused.

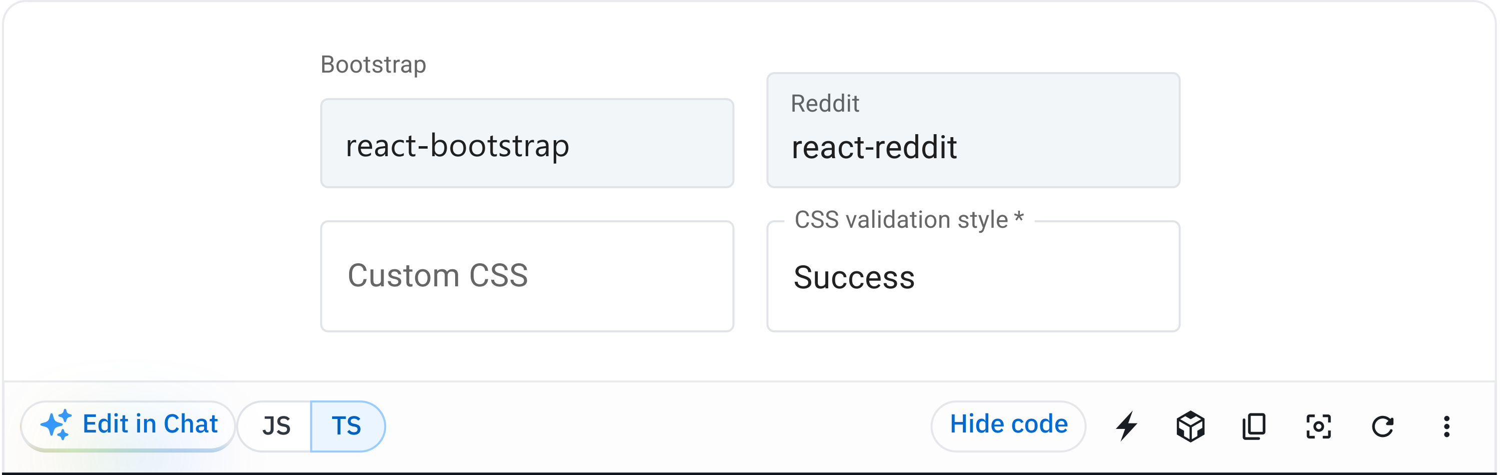


```tsx
import Box from '@mui/material/Box';
import TextField from '@mui/material/TextField';

export default function ColorTextFields() {
  return (
    <Box
      component="form"
      sx={{ '& > :not(style)': { m: 1, width: '25ch' } }}
      noValidate
      autoComplete="off"
    >
      <TextField label="Outlined secondary" color="secondary" focused />
      <TextField label="Filled success" variant="filled" color="success" focused />
      <TextField
        label="Standard warning"
        variant="standard"
        color="warning"
        focused
      />
    </Box>
  );
}
```

Customization

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

Using the styled API



Bootstrap

react-bootstrap

Reddit

react-reddit

Custom CSS

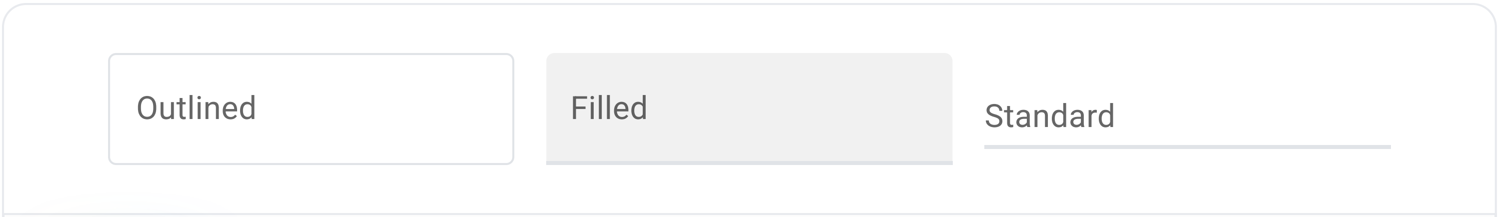
CSS validation style *

Success

```
import { alpha, styled } from '@mui/material/styles';
import InputBase from '@mui/material/InputBase';
import Box from '@mui/material/Box';
import InputLabel from '@mui/material/InputLabel';
import TextField, { TextFieldProps } from '@mui/material/TextField';
import FormControl from '@mui/material/FormControl';
import { OutlinedInputProps } from '@mui/material/OutlinedInput';

const CssTextField = styled(TextField)({
  '& label.Mui-focused': {
    color: '#A0AAB4',
  },
  '& .MuiInput-underline:after': {
    borderBottomColor: '#B2BAC2',
  },
  '& .MuiOutlinedInput-root': {
    '& fieldset': {
      borderColor: '#E0E3E7',
    },
    '&:hover fieldset': {
      borderColor: '#B2BAC2',
    },
    '&.Mui-focused fieldset': {
      borderColor: '#6F7E8C',
    },
  },
});

const BootstrapInput = styled(InputBase)(({ theme }) => ({
  'label + &': {
    marginTop: theme.spacing(3),
  },
  '& .MuiInputBase-input': {
    borderRadius: 4,
    position: 'relative',
    backgroundColor: '#F3F6F9',
    border: '1px solid',
    borderColor: '#E0E3E7',
```

Using the theme style overrides API

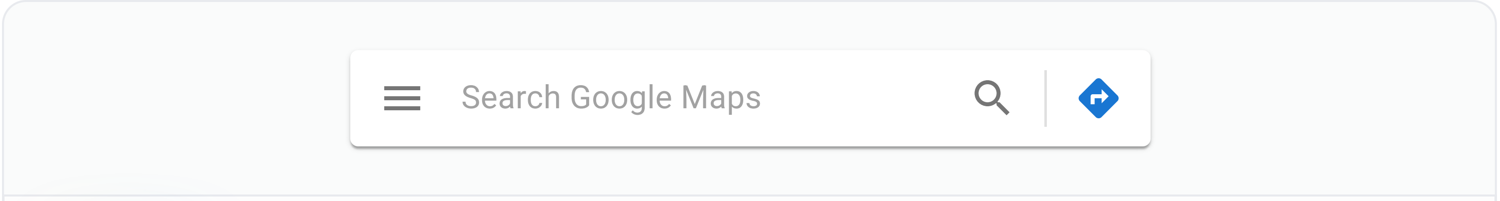
Use the `styleOverrides` key to change any style injected by Material UI into the DOM. See the theme style overrides documentation for further details.



Edit in Chat JS TS Collapse code

```tsx
import TextField from '@mui/material/TextField';
import { outlinedInputClasses } from '@mui/material/OutlinedInput';
import Box from '@mui/material/Box';
import { createTheme, ThemeProvider, Theme, useTheme } from '@mui/material/styles';

const customTheme = (outerTheme: Theme) =>
  createTheme({
    palette: {
      mode: outerTheme.palette.mode,
    },
    components: {
      MuiTextField: {
        styleOverrides: {
          root: {
            '--TextField-brandBorderColor': '#E0E3E7',
            '--TextField-brandBorderHoverColor': '#B2BAC2',
            '--TextField-brandBorderFocusedColor': '#6F7E8C',
            '& label.Mui-focused': {
              color: 'var(--TextField-brandBorderFocusedColor)',
            },
          },
        },
      },
      MuiOutlinedInput: {
        styleOverrides: {
          notchedOutline: {
            borderColor: 'var(--TextField-brandBorderColor)',
          },
          root: {
            [`&:hover .${outlinedInputClasses.notchedOutline}`]: {
              borderColor: 'var(--TextField-brandBorderHoverColor)',
            },
            [`&.Mui-focused .${outlinedInputClasses.notchedOutline}`]: {
              borderColor: 'var(--TextField-brandBorderFocusedColor)',
            },
          },
        },
      },
```

Customization does not stop at CSS. You can use composition to build custom components and give your app a unique feel. Below is an example using the `InputBase` component, inspired by Google Maps.



```tsx
import Paper from '@mui/material/Paper';
import InputBase from '@mui/material/InputBase';
import Divider from '@mui/material/Divider';
import IconButton from '@mui/material/IconButton';
import MenuIcon from '@mui/icons-material/Menu';
import SearchIcon from '@mui/icons-material/Search';
import DirectionsIcon from '@mui/icons-material/Directions';

export default function CustomizedInputBase() {
  return (
    <Paper
      component="form"
      sx={{ p: '2px 4px', display: 'flex', alignItems: 'center', width: 400 }}
    >
      <IconButton sx={{ p: '10px' }} aria-label="menu">
        <MenuIcon />
      </IconButton>
      <InputBase
        sx={{ ml: 1, flex: 1 }}
        placeholder="Search Google Maps"
        inputProps={{ 'aria-label': 'search google maps' }}
      />
      <IconButton type="button" sx={{ p: '10px' }} aria-label="search">
        <SearchIcon />
      </IconButton>
      <Divider sx={{ height: 28, m: 0.5 }} orientation="vertical" />
      <IconButton color="primary" sx={{ p: '10px' }} aria-label="directions">
        <DirectionsIcon />
      </IconButton>
    </Paper>
  );
}
```

🎨 If you are looking for inspiration, you can check MUI Treasury's customization examples.

`useFormControl`

For advanced customization use cases, a `useFormControl()` hook is exposed. This hook returns the context value of the parent `FormControl` component.

API

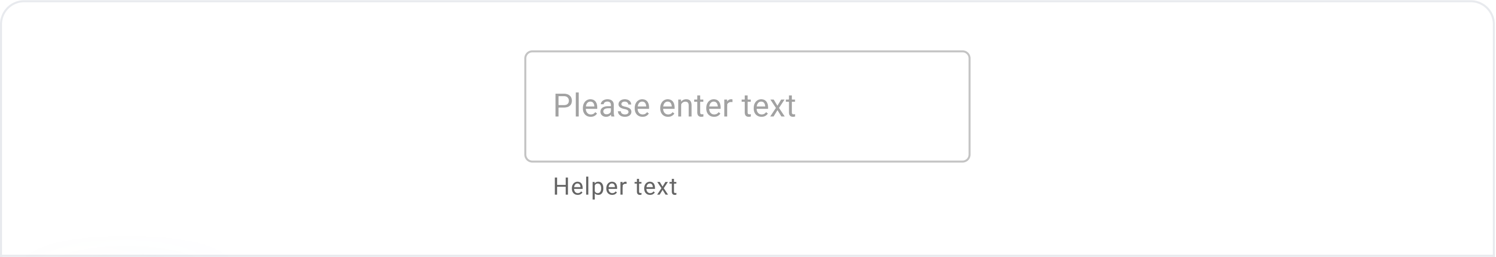
```
import { useFormControl } from '@mui/material/FormControl';
```

Returns

`value` (*object*):

- `value.adornedStart` (*bool*): Indicate whether the child `Input` or `Select` component has a start adornment.
- `value.setAdornedStart` (*func*): Setter function for `adornedStart` state value.
- `value.color` (*string*): The theme color is being used, inherited from `FormControl` `color` prop .
- `value.disabled` (*bool*): Indicate whether the component is being displayed in a disabled state, inherited from `FormControl` `disabled` prop.
- `value.error` (*bool*): Indicate whether the component is being displayed in an error state, inherited from `FormControl` `error` prop
- `value.filled` (*bool*): Indicate whether input is filled
- `value.focused` (*bool*): Indicate whether the component and its children are being displayed in a focused state
- `value.fullWidth` (*bool*): Indicate whether the component is taking up the full width of its container, inherited from `FormControl` `fullWidth` prop
- `value.hiddenLabel` (*bool*): Indicate whether the label is being hidden, inherited from `FormControl` `hiddenLabel` prop
- `value.required` (*bool*): Indicate whether the label is indicating that the input is required input, inherited from the `FormControl` `required` prop
- `value.size` (*string*): The size of the component, inherited from the `FormControl` `size` prop
- `value.variant` (*string*): The variant is being used by the `FormControl` component and its children, inherited from `FormControl` `variant` prop
- `value.onBlur` (*func*): Should be called when the input is blurred
- `value.onFocus` (*func*): Should be called when the input is focused
- `value.onEmpty` (*func*): Should be called when the input is emptied
- `value.onFilled` (*func*): Should be called when the input is filled

Example



Edit in Chat JS TS Collapse code

```
import * as React from 'react';
import FormControl, { useFormControl } from '@mui/material/FormControl';
import OutlinedInput from '@mui/material/OutlinedInput';
import FormHelperText from '@mui/material/FormHelperText';

function MyFormHelperText() {
  const { focused } = useFormControl() || {};

  const helperText = React.useMemo(() => {
    if (focused) {
      return 'This field is being focused';
    }

    return 'Helper text';
  }, [focused]);

  return <FormHelperText>{helperText}</FormHelperText>;
}

export default function UseFormControl() {
  return (
    <form noValidate autoComplete="off">
      <FormControl sx={{ width: '25ch' }}>
        <OutlinedInput placeholder="Please enter text" />
        <MyFormHelperText />
      </FormControl>
    </form>
  );
}
```

Performance

Global styles for the auto-fill keyframes are injected and removed on each mount and unmount, respectively. If you are loading a large number of Text Field components at once, it might be a good idea to change this default behavior by enabling `disableInjectingGlobalStyles` in `MuiInputBase`. Make sure to inject `GlobalStyles` for the auto-fill keyframes at the top of your application.

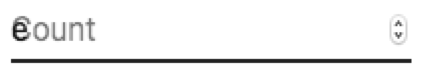
```
import { GlobalStyles, createTheme, ThemeProvider } from '@mui/material';

const theme = createTheme({
  components: {
    MuiInputBase: {
      defaultProps: {
        disableInjectingGlobalStyles: true,
      },
    },
  },
});

export default function App() {
  return (
    <ThemeProvider theme={theme}>
```

```
      <GlobalStyles
        styles={{
          '@keyframes mui-auto-fill': { from: { display: 'block' } },
          '@keyframes mui-auto-fill-cancel': { from: { display: 'block' } },
```

Limitations

Shrink

The input label "shrink" state isn't always correct. The input label is supposed to shrink as soon as the input is displaying something. In some circumstances, we can't determine the "shrink" state (datetime input, Stripe input). You might notice an overlap.



To workaround the issue, you can force the "shrink" state of the label.

```
<TextField slotProps={{ inputLabel: { shrink: true } }} />
```

or

```
<InputLabel shrink>Count</InputLabel>
```

Floating label

The floating label is absolutely positioned. It won't impact the layout of the page. Make sure that the input is larger than the label to display correctly.

type="number"

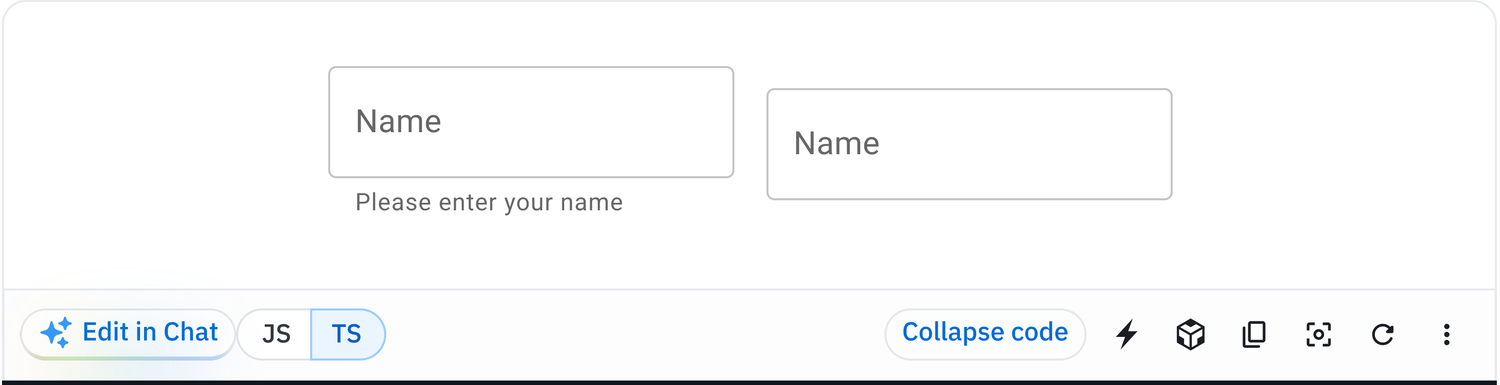
⚠ We do not recommend using `type="number"` with a Text Field due to potential usability issues:

- it allows certain non-numeric characters ('e', '+', '-', '.') and silently discards others
- the functionality of scrolling to increment/decrement the number can cause accidental and hard-to-notice changes
- and more—see Why the GOV.UK Design System team changed the input type for numbers for a more detailed explanation of the limitations of `<input type="number">`

If you need a text field with number validation, you can use Number Field instead.

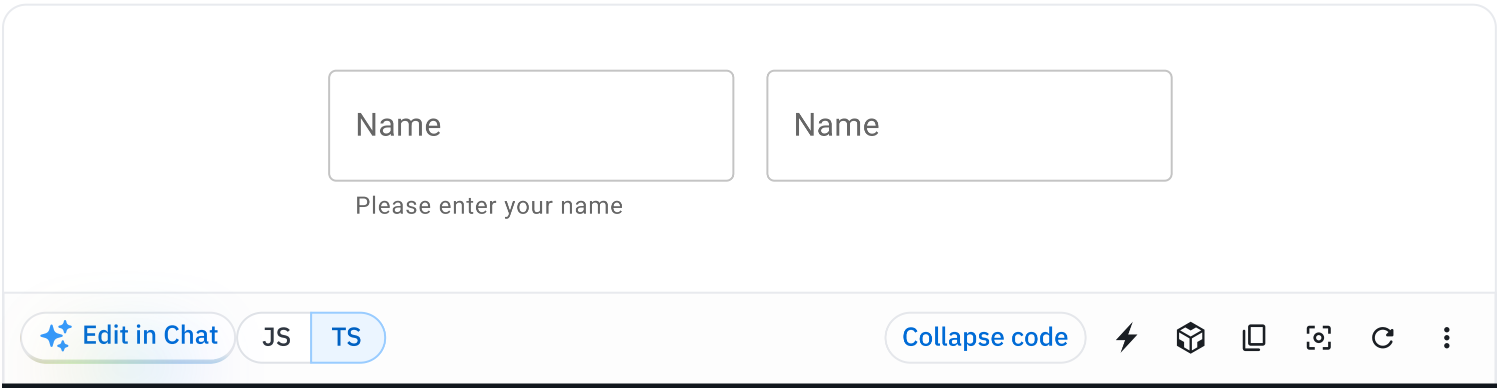
Helper text

The helper text prop affects the height of the text field. If two text fields are placed side by side, one with a helper text and one without, they will have different heights. For example:



```tsx
import Box from '@mui/material/Box';
import TextField from '@mui/material/TextField';

export default function HelperTextMisaligned() {
  return (
    <Box sx={{ display: 'flex', alignItems: 'center', '& > :not(style)': { m: 1 } }}>
      <TextField
        helperText="Please enter your name"
        id="demo-helper-text-misaligned"
        label="Name"
      />
      <TextField id="demo-helper-text-misaligned-no-helper" label="Name" />
    </Box>
  );
}
```

This can be fixed by passing a space character to the `helperText` prop:

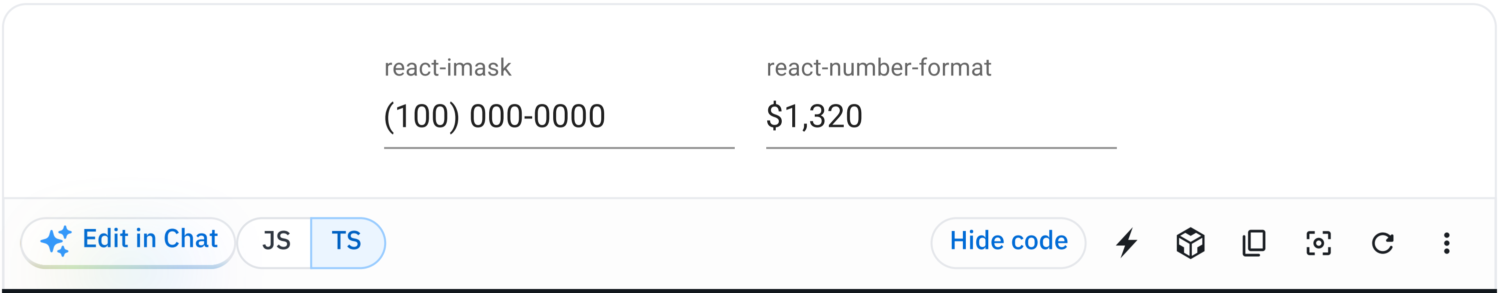


```tsx
import Box from '@mui/material/Box';
import TextField from '@mui/material/TextField';

export default function HelperTextAligned() {
  return (
    <Box sx={{ display: 'flex', alignItems: 'center', '& > :not(style)': { m: 1 } }}>
      <TextField
        helperText="Please enter your name"
        id="demo-helper-text-aligned"
        label="Name"
      />
      <TextField
        helperText=" "
        id="demo-helper-text-aligned-no-helper"
        label="Name"
      />
    </Box>
  );
}
```

Integration with 3rd party input libraries

You can use third-party libraries to format an input. You have to provide a custom implementation of the `<input>` element with the `inputComponent` property.

The following demo uses the [react-imask](https://github.com/uNmAnNeR/imaskjs) and [react-number-format](https://github.com/s-yadav/react-number-format) libraries. The same concept could be applied to, for example [react-stripe-element](https://github.com/stripe/react-stripe-elements).



react-imask
(100) 000-0000

react-number-format
$1,320

```
import * as React from 'react';
import { IMaskInput } from 'react-imask';
import { NumericFormat } from 'react-number-format';
import Stack from '@mui/material/Stack';
import Input from '@mui/material/Input';
import InputLabel from '@mui/material/InputLabel';
import TextField from '@mui/material/TextField';
import FormControl from '@mui/material/FormControl';

interface CustomProps {
  onChange: (event: { target: { name: string; value: string } }) => void;
  name: string;
}

const TextMaskCustom = React.forwardRef<HTMLInputElement, CustomProps>(
  function TextMaskCustom(props, ref) {
    const { onChange, ...other } = props;
    return (
      <IMaskInput
        {...other}
        mask="(#00) 000-0000"
        definitions={{
          '#': /[1-9]/,
        }}
        inputRef={ref}
        onAccept={(value: any) => onChange({ target: { name: props.name, value } })}
        overwrite
      />
    );
  },
);

export default function FormattedInputs() {
  const [values, setValues] = React.useState({
    textmask: '(100) 000-0000',
    numberformat: '1320',
  });
```

The provided input component should expose a ref with a value that implements the following interface:

```
interface InputElement {
  focus(): void;
  value?: string;
}
```

```
const MyInputComponent = React.forwardRef((props, ref) => {
  const { component: Component, ...other } = props;

  // implement `InputElement` interface
  React.useImperativeHandle(ref, () => ({
    focus: () => {
      // logic to focus the rendered component from 3rd party belongs here
    },
    // hiding the value e.g. react-stripe-elements
  }));

  // `Component` will be your `SomeThirdPartyComponent` from below
  return <Component {...other} />;
});

// usage
<TextField
  slotProps={{
    input: {
      inputComponent: MyInputComponent
```

Accessibility

In order for the text field to be accessible, **the input should be linked to the label and the helper text**. The underlying DOM nodes should have this structure:

```
<div class="form-control">
  <label for="my-input">Email address</label>
  <input id="my-input" aria-describedby="my-helper-text" />
  <span id="my-helper-text">We'll never share your email.</span>
</div>
```

- If you are using the `TextField` component, you just have to provide a unique `id` unless you're using the `TextField` only client-side. Until the UI is hydrated `TextField` without an explicit `id` will not have associated labels.
- If you are composing the component:

```
<FormControl>
  <InputLabel htmlFor="my-input">Email address</InputLabel>
  <Input id="my-input" aria-describedby="my-helper-text" />
  <FormHelperText id="my-helper-text">We'll never share your email.</FormHelperText>
</FormControl>
```

Supplementary projects

For more advanced use cases, you might be able to take advantage of:

- react-hook-form-mui: Material UI and react-hook-form combined.
- formik-material-ui: Bindings for using Material UI with formik.
- mui-rff: Bindings for using Material UI with React Final Form.
- @ui-schema/ds-material Bindings for using Material UI with UI Schema. JSON Schema compatible.

API

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

- `<FilledInput />`
- `<FormControl />`
- `<FormHelperText />`
- `<Input />`
- `<InputAdornment />`
- `<InputBase />`
- `<InputLabel />`
- `<OutlinedInput />`
- `<TextField />`