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.

{{"component": "modules/components/ComponentLinkHeader.js"}}

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.

{{"demo": "BasicTextFields.js"}}

Note: The standard variant of the `TextField` is no longer documented in the Material Design guidelines (here's why), but MUI will continue to support it.

Form props

Standard form attributes are supported e.g. `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.

{{"demo": "FormPropsTextFields.js"}}

Validation

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

{{"demo": "ValidationTextFields.js"}}

Multiline

The `multiline` prop transforms the text field into a TextareaAutosize element. Unless the `rows` prop is set, the height of the text field dynamically matches its content (using TextareaAutosize). You can use the `minRows` and `maxRows` props to bound it.

{{"demo": "MultilineTextFields.js"}}

Select

The `select` prop makes the text field use the Select component internally.

{{"demo": "SelectTextFields.js"}}

Icons

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

{{"demo": "InputWithIcon.js"}}

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.

{{"demo": "InputAdornments.js"}}

Sizes

Fancy smaller inputs? Use the `size` prop.

{{"demo": "TextFieldSizes.js"}}

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

{{"demo": "TextFieldHiddenLabel.js"}}

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.

{{"demo": "LayoutTextFields.js"}}

Full width

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

{{"demo": "FullWidthTextField.js"}}

Uncontrolled vs. Controlled

The component can be controlled or uncontrolled.

{{"demo": "StateTextFields.js"}}

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 `inputProps` (and `InputProps` , `InputLabelProps` properties) if you want to avoid some boilerplate.

{{"demo": "ComposedTextField.js"}}

Inputs

{{"demo": "Inputs.js"}}

Color

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

{{"demo": "ColorTextFields.js"}}

Customization

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

{{"demo": "CustomizedInputs.js"}}

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.

{{"demo": "CustomizedInputBase.js", "bg": true}}

🎨 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

{{"demo": "UseFormControl.js"}}

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 (number input, datetime input, Stripe input). You might notice an overlap.

shrink

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

```
<TextField InputLabelProps={{ 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"

Inputs of type="number" have potential usability issues:

- Allowing certain non-numeric characters ('e', '+', '-', '.') and silently discarding others
- The functionality of scrolling to increment/decrement the number can cause accidental and hard-to-notice changes

and more - see this article by the GOV.UK Design System team for a more detailed explanation.

For number validation, one viable alternative is to use the default input type="text" with the *pattern* attribute, for example:

```
<TextField inputProps={{ inputMode: 'numeric', pattern: '[0-9]*' }} />
```

In the future, we might provide a number input component.

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:

{{"demo": "HelperTextMisaligned.js"}}

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

{{"demo": "HelperTextAligned.js"}}

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 and react-number-format libraries. The same concept could be applied to e.g. react-stripe-element.

{{"demo": "FormattedInputs.js"}}

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
  InputProps={{
    inputComponent: MyInputComponent,
    inputProps: {
      component: SomeThirdPartyComponent,
    },
  }}
/>;
```

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>
```

Complementary projects

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

- react-hook-form: React hook for form validation.
- react-hook-form-mui: MUI and react-hook-form combined.
- formik-material-ui: Bindings for using MUI with formik.
- redux-form-material-ui: Bindings for using MUI with Redux Form.
- mui-rff: Bindings for using MUI with React Final Form.