Autocomplete

The autocomplete is a normal text input enhanced by a panel of suggested options.

The widget is useful for setting the value of a single-line textbox in one of two types of scenarios:

1. The value for the textbox must be chosen from a predefined set of allowed values, e.g., a location field must contain a valid location name: combo box.
2. The textbox may contain any arbitrary value, but it is advantageous to suggest possible values to the user, e.g., a search field may suggest similar or previous searches to save the user time: free solo.

It's meant to be an improved version of the "react-select" and "downshift" packages.

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

Combo box

The value must be chosen from a predefined set of allowed values.

{{"demo": "ComboBox.js"}}

Options structure

By default, the component accepts the following options structures:

```
interface AutocompleteOption {
  label: string;
}
// or
type AutocompleteOption = string;
```

for instance:

```
const options = [
  { label: 'The Godfather', id: 1 },
  { label: 'Pulp Fiction', id: 2 },
];
// or
const options = ['The Godfather', 'Pulp Fiction'];
```

However, you can use different structures by providing a `getOptionLabel` prop.

Playground

Each of the following examples demonstrates one feature of the Autocomplete component.

{{"demo": "Playground.js"}}

Country select

Choose one of the 248 countries.

{{"demo": "CountrySelect.js"}}

Controlled states

The component has two states that can be controlled:

1. the "value" state with the `value` / `onChange` props combination. This state represents the value selected by the user, for instance when pressing `Enter`.
2. the "input value" state with the `inputValue` / `onInputChange` props combination. This state represents the value displayed in the textbox.

> ⚠️ *These two states are isolated, they should be controlled independently.*

{{"demo": "ControllableStates.js"}}

Free solo

Set `freeSolo` to true so the textbox can contain any arbitrary value.

Search input

The prop is designed to cover the primary use case of a **search input** with suggestions, e.g. Google search or react-autowhatever.

{{"demo": "FreeSolo.js"}}

Creatable

If you intend to use this mode for a combo box like experience (an enhanced version of a select element) we recommend setting:

- `selectOnFocus` to help the user clear the selected value.
- `clearOnBlur` to help the user enter a new value.
- `handleHomeEndKeys` to move focus inside the popup with the `Home` and `End` keys.
- A last option, for instance: `Add "YOUR SEARCH"`.

{{"demo": "FreeSoloCreateOption.js"}}

You could also display a dialog when the user wants to add a new value.

{{"demo": "FreeSoloCreateOptionDialog.js"}}

Grouped

You can group the options with the `groupBy` prop. If you do so, make sure that the options are also sorted with the same dimension that they are grouped by, otherwise, you will notice duplicate headers.

{{"demo": "Grouped.js"}}

Disabled options

{{"demo": "DisabledOptions.js"}}

`useAutocomplete`

For advanced customization use cases, a headless `useAutocomplete()` hook is exposed. It accepts almost the same options as the Autocomplete component minus all the props related to the rendering of JSX. The Autocomplete component is built on this hook.

```
import { useAutocomplete } from '@mui/base/AutocompleteUnstyled';
```

The `useAutocomplete` hook is also reexported from @mui/material for convenience and backward compatibility.

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

- 4.5 kB gzipped.

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

Customized hook

{{"demo": "CustomizedHook.js"}}

Head to the customization section for an example with the `Autocomplete` component instead of the hook.

Asynchronous requests

The component supports two different asynchronous use-cases:

- Load on open: it waits for the component to be interacted with to load the options.
- Search as you type: a new request is made for each keystroke.

Load on open

It displays a progress state as long as the network request is pending.

{{"demo": "Asynchronous.js"}}

Search as you type

If your logic is fetching new options on each keystroke and using the current value of the textbox to filter on the server, you may want to consider throttling requests.

Additionally, you will need to disable the built-in filtering of the `Autocomplete` component by overriding the `filterOptions` prop:

```
<Autocomplete filterOptions={(x) => x} />
```

Google Maps place

A customized UI for Google Maps Places Autocomplete. For this demo, we need to load the Google Maps JavaScript and Google Places API.

{{"demo": "GoogleMaps.js"}}

> ⚠️ *Before you can start using the Google Maps JavaScript API and Places API, you must sign up and create a billing account.*

Multiple values

Also known as tags, the user is allowed to enter more than one value.

{{"demo": "Tags.js"}}

Fixed options

In the event that you need to lock certain tags so that they can't be removed, you can set the chips disabled.

{{"demo": "FixedTags.js"}}

Checkboxes

{{"demo": "CheckboxesTags.js"}}

Limit tags

You can use the `limitTags` prop to limit the number of displayed options when not focused.

{{"demo": "LimitTags.js"}}

Sizes

Fancy smaller inputs? Use the `size` prop.

{{"demo": "Sizes.js"}}

Customization

Custom input

The `renderInput` prop allows you to customize the rendered input. The first argument of this render prop contains props that you need to forward. Pay specific attention to the `ref` and `inputProps` keys.

{{"demo": "CustomInputAutocomplete.js"}}

GitHub's picker

This demo reproduces GitHub's label picker:

{{"demo": "GitHubLabel.js"}}

Head to the Customized hook section for a customization example with the `useAutocomplete` hook instead of the component.

Highlights

The following demo relies on autosuggest-highlight, a small (1 kB) utility for highlighting text in autosuggest and autocomplete components.

{{"demo": "Highlights.js"}}

Custom filter

The component exposes a factory to create a filter method that can be provided to the `filterOptions` prop. You can use it to change the default option filter behavior.

```
import { createFilterOptions } from '@mui/material/Autocomplete';
```

`createFilterOptions(config) => filterOptions`

Arguments

1. `config` (*object* [optional]):

- `config.ignoreAccents` (*bool* [optional]): Defaults to `true`. Remove diacritics.
- `config.ignoreCase` (*bool* [optional]): Defaults to `true`. Lowercase everything.
- `config.limit` (*number* [optional]): Default to null. Limit the number of suggested options to be shown. For example, if `config.limit` is `100`, only the first `100` matching options are shown. It can be useful if a lot of options match and virtualization wasn't set up.
- `config.matchFrom` (*'any'* | *'start'* [optional]): Defaults to `'any'`.
- `config.stringify` (*func* [optional]): Controls how an option is converted into a string so that it can be matched against the input text fragment.
- `config.trim` (*bool* [optional]): Defaults to `false`. Remove trailing spaces.

Returns

`filterOptions`: the returned filter method can be provided directly to the `filterOptions` prop of the `Autocomplete` component, or the parameter of the same name for the hook.

In the following demo, the options need to start with the query prefix:

```
const filterOptions = createFilterOptions({
  matchFrom: 'start',
  stringify: (option) => option.title,
});

<Autocomplete filterOptions={filterOptions} />;
```

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

Advanced

For richer filtering mechanisms, like fuzzy matching, it's recommended to look at match-sorter. For instance:

```
import { matchSorter } from 'match-sorter';

const filterOptions = (options, { inputValue }) => matchSorter(options, inputValue);

<Autocomplete filterOptions={filterOptions} />;
```

Virtualization

Search within 10,000 randomly generated options. The list is virtualized thanks to react-window.

{{"demo": "Virtualize.js"}}

Events

If you would like to prevent the default key handler behavior, you can set the event's `defaultMuiPrevented` property to `true`:

```
<Autocomplete
  onKeyDown={(event) => {
    if (event.key === 'Enter') {
      // Prevent's default 'Enter' behavior.
      event.defaultMuiPrevented = true;
      // your handler code
    }
  }}
/>
```

Limitations

autocomplete/autofill

Browsers have heuristics to help the user fill in form inputs. However, this can harm the UX of the component.

By default, the component disables the input **autocomplete** feature (remembering what the user has typed for a given field in a previous session) with the `autoComplete="off"` attribute. Google Chrome does not currently support this attribute setting (Issue 587466). A possible workaround is to remove the `id` to have the component generate a random one.

In addition to remembering past entered values, the browser might also propose **autofill** suggestions (saved login, address, or payment details). In the event you want the avoid autofill, you can try the following:

- Name the input without leaking any information the browser can use. e.g. `id="field1"` instead of `id="country"`. If you leave the id empty, the component uses a random id.
- Set `autoComplete="new-password"` (some browsers will suggest a strong password for inputs with this attribute setting):

  ```
  <TextField
    {...params}
    inputProps={{
      ...params.inputProps,
      autoComplete: 'new-password',
    }}
  />
  ```

Read the guide on MDN for more details.

iOS VoiceOver

VoiceOver on iOS Safari doesn't support the `aria-owns` attribute very well. You can work around the issue with the `disablePortal` prop.

ListboxComponent

If you provide a custom `ListboxComponent` prop, you need to make sure that the intended scroll container has the `role` attribute set to `listbox`. This ensures the correct behavior of the scroll, for example when using the keyboard to navigate.

Accessibility

(WAI-ARIA: https://www.w3.org/TR/wai-aria-practices/#combobox)

We encourage the usage of a label for the textbox. The component implements the WAI-ARIA authoring practices.