useId

`useId` is a React Hook for generating unique IDs that can be passed to accessibility attributes.

```
const id = useId()
```

Reference

useId()

Call `useId` at the top level of your component to generate a unique ID:

```
import { useId } from 'react';

function PasswordField() {
  const passwordHintId = useId();
  // ...
```

See more examples below.

Parameters

`useId` does not take any parameters.

Returns

`useId` returns a unique ID string associated with this particular `useId` call in this particular component.

Caveats

- `useId` is a Hook, so you can only call it **at the top level of your component** or your own Hooks. You can't call it inside loops or conditions. If you need that, extract a new component and move the state into it.
- `useId` **should not be used to generate keys** in a list. Keys should be generated from your data.

Usage

Pitfall

Do not call `useId` to generate keys in a list. Keys should be generated from your data.

Generating unique IDs for accessibility attributes

Call `useId` at the top level of your component to generate a unique ID:

```
import { useId } from 'react';

function PasswordField() {
  const passwordHintId = useId();
  // ...
```

You can then pass the generated ID to different attributes:

```
<>
  <input type="password" aria-describedby={passwordHintId} />
  <p id={passwordHintId}>
</>
```

Let's walk through an example to see when this is useful.

HTML accessibility attributes like `aria-describedby` let you specify that two tags are related to each other. For example, you can specify that an element (like an input) is described by another element (like a paragraph).

In regular HTML, you would write it like this:

```
<label>
  Password:
  <input
    type="password"
    aria-describedby="password-hint"
  />
</label>
<p id="password-hint">
  The password should contain at least 18 characters
</p>
```

However, hardcoding IDs like this is not a good practice in React. A component may be rendered more than once on the page—but IDs have to be unique! Instead of hardcoding an ID, generate a unique ID with `useId`:

```
import { useId } from 'react';

function PasswordField() {
  const passwordHintId = useId();
  return (
    <>
      <label>
        Password:
        <input
          type="password"
          aria-describedby={passwordHintId}
        />
      </label>
      <p id={passwordHintId}>
        The password should contain at least 18 characters
      </p>
    </>
  );
}
```

Now, even if `PasswordField` appears multiple times on the screen, the generated IDs won't clash.

App.js Download Reset

```
24        <h2>Choose password</h2>
25        <PasswordField />
26        <h2>Confirm password</h2>
27        <PasswordField />
```

Show more

Watch this video to see the difference in the user experience with assistive technologies.

Pitfall

With server rendering, `useId` **requires an identical component tree on the server and the client**. If the trees you render on the server and the client don't match exactly, the generated IDs won't match.

DEEP DIVE

Why is useId better than an incrementing counter?

Show Details

Generating IDs for several related elements

If you need to give IDs to multiple related elements, you can call `useId` to generate a shared prefix for them:

App.js

Download Reset

```
import { useId } from 'react';

export default function Form() {
  const id = useId();
  return (
    <form>
      <label htmlFor={id + '-firstName'}>First Name:</label>
      <input id={id + '-firstName'} type="text" />
      <hr />
      <label htmlFor={id + '-lastName'}>Last Name:</label>
      <input id={id + '-lastName'} type="text" />
    </form>
```

This lets you avoid calling `useId` for every single element that needs a unique ID.

Specifying a shared prefix for all generated IDs

If you render multiple independent React applications on a single page, pass `identifierPrefix` as an option to your `createRoot` or `hydrateRoot` calls. This ensures that the IDs generated by the two different apps never clash because every identifier generated with `useId` will start with the distinct prefix you've specified.

index.html App.js index.js Reset

```
1  import { createRoot } from 'react-dom/client';
2  import App from './App.js';
3  import './styles.css';
4
5  const root1 = createRoot(document.getElementById('root1'), {
6    identifierPrefix: 'my-first-app-'
7  });
8  root1.render(<App />);
9
10 const root2 = createRoot(document.getElementById('root2'), {
11   identifierPrefix: 'my-second-app-'
12 });
```

PREVIOUS
useEffect

NEXT
useImperativeHandle

How do you like these docs?

Take our survey!

Meta Open Source

Learn React

Quick Start

Installation

Describing the UI

Adding Interactivity

Managing State

Escape Hatches

API Reference

React APIs

React DOM APIs

Community

Code of Conduct

Meet the Team

Docs Contributors

Acknowledgements

More

Blog

React Native

Privacy

Terms