Context

August 17, 2026

Context

Kind: Interface

Source: `src/jsx/context.ts`

Part of: Jsx

`Context<T>` defines a context container with the available `values` and a React `Provider` component. The provider receives a current `value` and wraps child content so the selected context value can be supplied within the JSX tree.

Properties

Property	Type
`values`	`T[]`
`Provider`	`FC<PropsWithChildren<{ value: T }>>`

Diagram

```
graph LR
  Context["Context&lt;T&gt;"]
  Values["values: T[]"]
  Provider["Provider"]
  Value["value: T"]
  Children["children"]

  Context --> Values
  Context --> Provider
  Provider --> Value
  Provider --> Children
```

Usage

```tsx
import type { FC, PropsWithChildren, ReactNode } from "react";
import type { Context } from "./context";

type Theme = "light" | "dark";

declare const ThemeProvider: FC<PropsWithChildren<{ value: Theme }>>;

const themeContext: Context<Theme> = {
  values: ["light", "dark"],
  Provider: ThemeProvider,
};

function WithTheme({
  value,
  children,
}: {
  value: Theme;
  children: ReactNode;
}) {
  const Provider = themeContext.Provider;

  return <Provider value={value}>{children}</Provider>;
}
```

AI Coding Instructions

- Keep `values` typed as the same generic type passed to `Context<T>`.
- Pass a `value` prop to `Provider` whenever rendering it.
- Preserve `children` when wrapping content with a context provider.
- Define the provider with `FC<PropsWithChildren<{ value: T }>>` so its value type matches the context values.

Relationships

- IMPORTS → `html`
- IMPORTS → `JSXFragmentNode`
- IMPORTS → `DOM_RENDERER`
- IMPORTS → `createContextProviderFunction`