Appendix G. TSX

Under the hood, TypeScript exposes a few hooks for typing TSX in a pluggable way. These are special types on the `global.JSX` namespace that TypeScript looks at as the source of truth for TSX types throughout your program.

NOTE

If you're just using React, you don't need to know about these low-level hooks, but if you're writing a TypeScript library that uses TSX without React, this appendix provides a helpful reference for the hooks you can use.

TSX supports two kinds of elements: built-in ones (*intrinsic elements*) and user-defined ones (*value-based elements*). Intrinsic elements always have lowercased names, and refer to built-in elements like `<li>`, `<h1>`, and `<div>`. Value-based elements have PascalCased names, and refer to those elements that you create with React (or whatever frontend framework you're using TSX with); they can be defined either as functions or as classes. See Figure G-1.

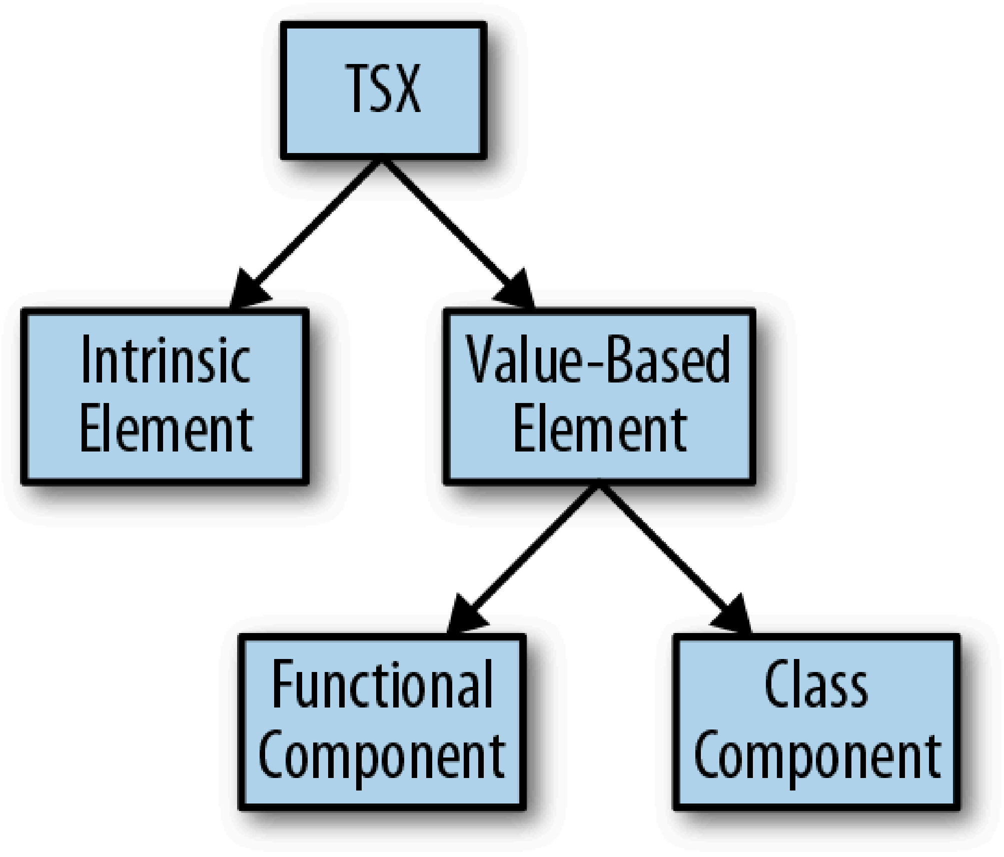

Figure G-1. The kinds of TSX elements

Using React’s type declarations as an example, we’ll walk through the hooks TypeScript uses to safely type TSX. Here’s how React hooks into TSX to type JSX safely:

```
declare global {
  namespace JSX {
    interface Element extends React.ReactElement<any> {
                               ❶

    interface ElementClass extends React.Component<any
                               ❷

      render(): React.ReactNode
    }
    interface ElementAttributesProperty {
                               ❸

      props: {}
    }
    interface ElementChildrenAttribute {
                               ❹

      children: {}
    }

    type LibraryManagedAttributes<C, P> = // ...
                               ❺
    interface IntrinsicAttributes extends React.Attribu
                          ❻

    interface IntrinsicClassAttributes<T> extends React
                          ❼


    interface IntrinsicElements {
                          ❽

      a: React.DetailedHTMLProps<
        React.AnchorHTMLAttributes<HTMLAnchorElement>,
        HTMLAnchorElement
      >
      abbr: React.DetailedHTMLProps<
        React.HTMLAttributes<HTMLElement>,
        HTMLElement
      >
      address: React.DetailedHTMLProps<
        React.HTMLAttributes<HTMLElement>,
        HTMLElement
      >
      // ...
    }
  }
}
```

`JSX.Element` ❶ is the type of a value-based TSX element.

`JSX.ElementClass` ❷ is the type of an instance of a value-based class component. Whenever you declare a class component that you plan to instantiate with TSX's `<MyComponent />` syntax, its class must satisfy this interface.

`JSX.ElementAttributesProperty` is the name of the property TypeScript looks at to figure out what attributes a component supports. For React, that means the `props` property. TypeScript looks for this value on a class instance.

`JSX.ElementChildrenAttribute` is the name of the property that TypeScript looks at to figure out what types of children a component supports. For React, that means the `children` property.

`JSX.IntrinsicAttributes` ❻ is the set of attributes that all intrinsic elements support. For React, that means the `key` attribute.

`JSX.IntrinsicClassAttributes` is the set of attributes that all

`JSX.IntrinsicClassAttributes` is the set of attributes that all class components (both intrinsic and value-based) support. For React, that means `ref`.

`JSX.LibraryManagedAttributes` specifies other places where JSX elements can declare and initialize property types. For React, that means `propTypes` as another place to declare property types, and `defaultProps` as the place to declare default values for properties.

`JSX.IntrinsicElements` enumerates every type of HTML element that you can use in TSX, mapping each element's tag name to its attribute and children types. Because JSX isn't HTML, React's type declarations have to tell TypeScript exactly what elements someone might use in a TSX expression, and because you can use any standard HTML element from TSX, the declarations have to manually enumerate each element along with its attribute types (for an `<a>` tag, for example, valid attributes include `href: string` and `rel: string`, but not `value`) and what types of children it might have.

By declaring any of these types in the global JSX namespace, you can hook into TypeScript's typechecking behavior for TSX and customize it however you want. Unless you're writing a library that uses TSX (and doesn't use React), you'll probably never touch these hooks.