# Consuming a context in a strongly-typed class component

In this lesson, we'll learn how to use a context in strongly-typed class components

WE'LL COVER THE FOLLOWING

## Trying to use the context hook in a class component #

We are going to continue using the context we created in the last lesson that allows consumers to share and set a theme. Let's update the `Header` component to be a class component:

```
class Header extends React.Component {
  render() {
    const { theme, setTheme } = useTheme()!;

    return (
      <div style={{ backgroundColor: theme }}>
        <select
          value={theme}
          onChange={e => setTheme(e.currentTarget.value)}
        >
          <option value="white">White</option>
          <option value="lightblue">Blue</option>
          <option value="lightgreen">Green</option>
        </select>
        <span>Hello!</span>
      </div>
    );
  }
}
```

Can you spot a problem with this implementation?

Show Answer

## Using the `context` property #

React class components have a `context` property that we can use to consume a context. First, we need to tell the class what context it should use with a `static` `contextType` property and then we can access the `context` property:

```
class Header extends React.Component {
  static contextType = ThemeContext;

  render() {
    const { theme, setTheme } = this.context!;

    return (
      ...
    );
  }
}
```

Notice that we put an exclamation mark(`!`) after the `context` property to tell the TypeScript compiler that it isn't `undefined`.

What types have `theme` and `setTheme` been inferred as?

Show Answer

## Explicitly setting the type for the `context` property #

At the moment, the consumed context isn't strongly-typed. We can explicitly define the class's `context` property with a type annotation to make it strongly-typed:

```
class Header extends React.Component {
  static contextType = ThemeContext;
  context: React.ContextType<typeof ThemeContext>;

  render() {
    const { theme, setTheme } = this.context!;

    return (
      ...
    );
  }
}
```

Notice that we don’t use `React.ContextType<ThemeContextType>` as the type annotation for the `context` property because we get a type error if we do so.

A full working implementation is available by clicking the link below. Give it a try and change the theme value to see the background change color.

Open full implementation

## Using the `Consumer` component #

There is an alternative approach to consuming a context in a class component if we just need access to it in the JSX. This method is to use the contexts `Consumer` component:

```
class Header extends React.Component {
  render() {
    return (
      <ThemeContext.Consumer>
        {value => (
          <div style={{ backgroundColor: value!.theme }}>
            <select
              value={value!.theme}
              onChange={e => value!.setTheme(e.currentTarget.value)}
            >
              <option value="white">White</option>
              <option value="lightblue">Blue</option>
              <option value="lightgreen">Green</option>
            </select>
            <span>Hello!</span>
            <span>Hello!</span>
          </div>
        )}
      </ThemeContext.Consumer>
    );
  }
}
```

The child of the `Consumer` component is a function that has the value of the context passed into it and returns the JSX we want to render. Notice that we have put an exclamation mark (`!`) after we reference `value` to tell the TypeScript compiler that this isn't `undefined`.

The benefit of this approach is that the `contextType` static property doesn't need to be implemented. We don't need to declare the `context` property with its type annotation as well.

What do you think the inferred type of the `value` parameter is in the `Consumer` components child function?

Show Answer

## Wrap up #

We can use React's context in class components, but we can't use the `useContext` hook.

Using the `Consumer` component is a neat way of getting access to the context in the `render` method, which has its type inferred correctly.

The `context` property can be used in other lifecycle methods to get access to the context. We need to explicitly define a type annotation for the `context` property and specify the specific context in a `contextType` static property.

Next, let's double-check what we have learned from the last few lessons with a quiz.