

Chapter 3: Using ReactJS with TypeScript

Section 3.1: ReactJS component written in TypeScript

Actually you can use ReactJS's components in Typescript as in facebook's example. Just replace 'jsx' file's extension to 'tsx':

```
//helloMessage.tsx:
var HelloMessage = React.createClass({
  render: function() {
    return <div>Hello {this.props.name}</div>;
  }
});
ReactDOM.render(<HelloMessage name="John" />, mountNode);
```

But in order to make full use of Typescript's main feature (static type checking) should be done couple things:

1) convert React.createClass example to ES6 Class:

```
//helloMessage.tsx:
class HelloMessage extends React.Component {
  render() {
    return <div>Hello {this.props.name}</div>;
  }
}
ReactDOM.render(<HelloMessage name="John" />, mountNode);
```

2) next add Props and State interfaces:

```
interface IHelloMessageProps {
  name:string;
}

interface IHelloMessageState {
  //empty in our case
}

class HelloMessage extends React.Component<IHelloMessageProps, IHelloMessageState> {
  constructor(){
    super();
  }
  render() {
    return <div>Hello {this.props.name}</div>;
  }
}
ReactDOM.render(<HelloMessage name="Sebastian" />, mountNode);
```

Now Typescript will display an error if the programmer forgets to pass props. Or if they added props that are not defined in the interface.

Section 3.2: Installation and Setup

To use typescript with react in a node project, you must first have a project directory initialized with npm. To initialize the directory with npm `init`

Installing via npm or yarn

You can install React using [npm](#) by doing the following:

```
npm install --save react react-dom
```

Facebook released its own package manager named [Yarn](#), which can also be used to install React. After installing Yarn you just need to run this command:

```
yarn add react react-dom
```

You can then use React in your project in exactly the same way as if you had installed React via npm.

Installing react type definitions in Typescript 2.0+

To compile your code using typescript, add/install type definition files using npm or yarn.

```
npm install --save-dev @types/react @types/react-dom
```

or, using yarn

```
yarn add --dev @types/react @types/react-dom
```

Installing react type definitions in older versions of Typescript

You have to use a separate package called [tsd](#)

```
tsd install react react-dom --save
```

Adding or Changing the Typescript configuration

To use [JSX](#), a language mixing javascript with html/xml, you have to change the typescript compiler configuration. In the project's typescript configuration file (usually named `tsconfig.json`), you will need to add the JSX option as:

```
"compilerOptions": { "jsx": "react" },
```

That compiler option basically tells the typescript compiler to translate the JSX tags in code to javascript function calls.

To avoid typescript compiler converting JSX to plain javascript function calls, use

```
"compilerOptions": {  
  "jsx": "preserve"  
},
```

Section 3.3: Stateless React Components in TypeScript

React components that are pure functions of their props and do not require any internal state can be written as JavaScript functions instead of using the standard class syntax, as:

```
import React from 'react'  
  
const HelloWorld = (props) => (  
  <h1>Hello, {props.name}!</h1>  
)
```

The same can be achieved in Typescript using the `React.SFC` class:

```
import * as React from 'react';

class GreeterProps {
  name: string
}

const Greeter : React.SFC<GreeterProps> = props =>
  <h1>Hello, {props.name}!</h1>;
```

Note that, the name `React.SFC` is an alias for `React.StatelessComponent` So, either can be used.

Section 3.4: Stateless and property-less Components

The simplest react component without a state and no properties can be written as:

```
import * as React from 'react';

const Greeter = () => <span>Hello, World!</span>
```

That component, however, can't access `this.props` since typescript can't tell if it is a react component. To access its props, use:

```
import * as React from 'react';

const Greeter: React.SFC<{}> = props => () => <span>Hello, World!</span>
```

Even if the component doesn't have explicitly defined properties, it can now access `props.children` since all components inherently have children.

Another similar good use of stateless and property-less components is in simple page templating. The following is an exemplary simple Page component, assuming there are hypothetical Container, NavTop and NavBottom components already in the project:

```
import * as React from 'react';

const Page: React.SFC<{}> = props => () =>
  <Container>
    <NavTop />
    {props.children}
    <NavBottom />
  </Container>

const LoginPage: React.SFC<{}> = props => () =>
  <Page>
    Login Pass: <input type="password" />
  </Page>
```

In this example, the Page component can later be used by any other actual page as a base template.