

Developing SPFx Web Parts using React.js



Agenda

- Getting Started with React.js
- Working with JSX and TSX files
- Component Properties vs. State
- Developing SPFx Web Parts using React.js
- Passing Web Part Properties to a Component



Introducing React

- React is a library for building user interfaces
 - Not as all-encompassing as a framework like Angular
 - Focused on building HTML-based user experiences
 - Based on reusable component-based architecture
 - Components *react* to state changes by updating UI
 - React uses shadow DOM for efficient event handling
- React was originally designed for Facebook
 - Also a good fit for building SPFx web parts



Hello World with React.js

- Obtain the React library with npm or from a CDN
 - npm install react --save
 - npm install react-dom --save

```
<> SimpleReactApp.html x
1  <!DOCTYPE html>
2  <html>
3  <head>
4    <meta charset="utf-8" />
5    <title>Simple React App</title>
6  </head>
7  <body>
8
9    <div id="app">
10     <!-- this is where your app renders -->
11   </div>
12
13   <!-- React Libraries -->
14   <script src="https://cdnjs.cloudflare.com/ajax/libs/react/15.5.4/react.min.js"></script>
15   <script src="https://cdnjs.cloudflare.com/ajax/libs/react/15.5.4/react-dom.min.js"></script>
16
17   <script>
18     var reactComponentent = React.DOM.h1(null, "Hello, React!");
19     var target = document.getElementById("app");
20     ReactDOM.render(reactComponentent, target);
21   </script>
22
23 </body>
24 </html>
```

Hello, React!



React versus ReactDOM

- **React** and **ReactDOM** are separate libraries
 - **React** (**react.js**) is the primary library used to build out user experiences
 - **ReactDOM** (**react-dom.js**) is used to render **React** user experience in the browser
- **React** library exposes global **React** object
 - **React** object is the main entry point into React API
 - **React.DOM** wraps standard HTML elements
- **ReactDOM** library exposes global **ReactDOM** object
 - **ReactDOM** object used to render React components on web page

```
var reactComponent = React.DOM.h1(null, "Hello, React!");  
  
var target = document.getElementById("app");  
  
ReactDOM.render(reactComponent, target);
```



React Component Created Using ES5

- React component can be created using EcmaScript 5
 - React component definition created using **React.createClass**
 - React component must be defined with **render** method
 - React component can be instantiated with **React.createElement**

```
var myComponent = React.createClass({
  render: () => {
    return React.DOM.h1(null, "Hello React!")
  }
});

ReactDOM.render(
  React.createElement(myComponent),
  document.getElementById("app")
);
```



Initializing Element Properties

- Elements created using properties object
 - Object properties used to initialize element properties
 - Use **className** instead of **class** to assign CSS class
 - Use **htmlFor** instead of **for** to define HTML label

```
render: () => {  
  
  var elementProperties = {  
    id: "myElementId",  
    className: "myCssClass"  
  };  
  
  return React.DOM.h1( elementProperties , "Hello React!");  
}
```



Initializing Element Styles

- Elements styles initialized using style object
 - style must be defined using an object not a string
 - CSS properties referenced using camel casing

```
render: () => {  
  
  var elementProperties = {  
    id: "myElementId",  
    style: {  
      backgroundColor: "yellow",  
      borderStyle: "Solid",  
      borderColor: "green",  
      padding: 8,  
      color: "Blue",  
      fontSize: 48  
    }  
  };  
  
  return React.DOM.h1( elementProperties , "Hello React!");  
}
```



React Provides Synthetic Events

- Replaces standard DOM-based event handling
 - React creates virtual DOM for elements in component
 - React interacts with real DOM when required
 - Provides faster event registration and processing
 - No need to write browser-specific code

```
private incrementCounter() {
  var previousCount: number = this.state.count;
  this.setState({ count: previousCount + 1 });
}

public render(): React.ReactElement<IBeanCounterProps> {
  return (
    <div className={styles.beanCounter}>
      <h3>Mr Bean Counter</h3>
      <div className={styles.toolbar}>
        <button onClick={(event) => { this.incrementCounter(); }} >Add another Bean</button>
      </div>
      <div className={styles.beanCounterDisplay} >
        Bean Count: {this.state.count}
      </div>
    </div>
  );
}
```



Understanding JSX (and TSX)

- JSX provides better syntax for HTML composition
 - JSX allows extends JavaScript with XML-like syntax
 - JSX syntax must be transpiled into JavaScript code

```
var myHtml = <div id="myAppContainer" style={{ backgroundColor:"yellow", padding:8 }}>
  <h2>Hello JSX</h2>
  <p>I'm composing HTML elements using JSX syntax.</p>
</div>;

ReactDOM.render( myHtml , document.getElementById("app") );
```

- JSX/TSX is separate from React library
 - JSX/TSX commonly used in React development
 - Babel compiler used to transpile JSX to JavaScript
 - TypeScript compiler used to transpile TSX to JavaScript



Defining React Components using TypeScript

- Component is class extending `React.Component`
 - Component usually defined in its own **tsx** file
 - Component class must define **render** method

```
my-component.tsx •
import * as React from 'react';

export class MyComponent extends React.Component<any, any> {
  render() {
    return <h2>Hello from my component</h2>;
  }
}
```

- Component can be instantiated with JSX/TSX syntax

```
app.tsx •
import * as ReactDOM from 'react-dom';

import { MyComponent } from './components/my-component'

window.onload = () => {
  // Create and render component
  ReactDOM.render( <MyComponent/>, document.getElementById("app") );
}
```



Component Properties and State

- Component can contain properties and state
 - Properties are initialized by external components
 - Properties are read-only to hosting component
 - State is set internally by hosting component
 - Changing state triggers UI refresh by calling render
 - UI experience created by ***reacting*** to changes in state



Component Lifecycle

- `componentWillUpdate`
 - executed before component is rendered
- `componentDidUpdate`
 - executed after component is rendered
- `componentWillMount`
 - executed before node is added to the DOM
- `componentDidMount`
 - executed after node is added to the DOM
- `componentWillUnmount`
 - executed before node is removed from the DOM
- `shouldComponentUpdate(newProps, newState)`
 - executed before component is updated



React Component Properties

- Defining component with a property

```
component1.tsx •  
  
import * as React from 'react';  
  
export interface MyCustomProps {  
  Name: string;  
}  
  
export class Component1 extends React.Component<MyCustomProps, {}> {  
  render() {  
    return <div>Hello, my name is {this.props.Name}</div>;  
  }  
}
```

- Instantiating component with a property

```
ReactDOM.render(  
  <Component1 Name="Fred" />,  
  document.getElementById("app")  
)
```



Stateful Component

TS IBeanCounterProps.ts •

```
export interface IBeanCounterProps {  
  StartingValue: number;  
}
```

TS IBeanCounterState.ts •

```
export interface IBeanCounterState {  
  count: number;  
}
```

BeanCounter.tsx •

```
import * as React from 'react';  
import styles from './BeanCounter.module.scss';  
import { IBeanCounterProps } from './IBeanCounterProps';  
import { IBeanCounterState } from './IBeanCounterState';  
  
export default class BeanCounter extends React.Component<IBeanCounterProps, IBeanCounterState> {  
  constructor(props: any) {  
    super(props);  
    this.state = { count: this.props.StartingValue };  
  }  
  
  private incrementCounter() {  
    var previousCount: number = this.state.count;  
    this.setState({ count: previousCount + 1 });  
  }  
}
```



Stateful Component Rendering

BeanCounter.tsx

```
import * as React from 'react';
import styles from './BeanCounter.module.scss';
import { IBeanCounterProps } from './IBeanCounterProps';
import { IBeanCounterState } from './IBeanCounterState';

export default class BeanCounter extends React.Component<IBeanCounterProps, IBeanCounterState> {

  constructor(props: any) {
    super(props);
    this.state = { count: this.props.StartingValue };
  }

  private incrementCounter() {
    var previousCount: number = this.state.count;
    this.setState({ count: previousCount + 1 });
  }

  public render(): React.ReactElement<IBeanCounterProps> {
    return (
      <div className={styles.beanCounter}>
        <h3>Mr Bean Counter</h3>
        <div className={styles.toolbar}>
          <button onClick={(event) => { this.incrementCounter(); }} >Add another Bean</button>
        </div>
        <div className={styles.beanCounterDisplay} >
          Bean Count: {this.state.count}
        </div>
      </div>
    );
  }
}
```





DEMO

Creating Web Parts with React.js

Summary

- Getting Started with React.js
- Working with JSX and TSX files
- Component Properties vs. State
- Developing SPFx Web Parts using React.js
- Passing Web Part Properties to a Component

