Child Components Update Their Parents' state

How does a stateless, child component *update* the state of the parent component? Here's how that works:

1

The *parent* component class defines a method that calls `this.setState()`.

For an example, look in **Step1.js** at the `.handleClick()` method.

2

The parent component binds the newly-defined method to the current instance of the component in its constructor. This ensures that when we pass the method to the child component, it will still update the parent component.

For an example, look in **Step2.js** at the end of the `constructor()` method.

3

Once the *parent* has defined a method that updates its state and bound to it, the parent then passes that method down to a *child*.

Look in **Step2.js**, at the `prop` on line 28.

4

The *child* receives the passed-down function, and uses it as an event handler.

Look in **Step3.js**. When a user clicks on the `<button></button>`, a click event will fire. This will make the passed-down function get called, which will *update* the parent's state.

Instructions

Click through the three files in order, and try to follow their chronology. Whenever you're ready, click Next and we'll build an example!

Step1.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import { ChildClass } from './ChildClass';

class ParentClass extends React.Component {
  constructor(props) {
    super(props);

    this.state = { totalClicks: 0 };
  }

  handleClick() {
    const total = this.state.totalClicks;

    // calling handleClick will
    // result in a state change:
    this.setState(
      { totalClicks: total + 1 }
    );
  }
}
```

Step2.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import { ChildClass } from './ChildClass';

class ParentClass extends React.Component {
  constructor(props) {
    super(props);

    this.state = { totalClicks: 0 };

    this.handleClick = this.handleClick.bind(this);
  }

  handleClick() {
    const total = this.state.totalClicks;
```

```
    // calling handleClick will
    // result in a state change:
    this.setState(
      { totalClicks: total + 1 }
    );
  }

  // The stateful component class passes down
  // handleClick to a stateless component class:
  render() {
    return (
      <ChildClass onClick={this.handleClick} />
    );
  }
}
```

Step3.js

```
import React from 'react';
import ReactDOM from 'react-dom';

export class ChildClass extends React.Component {
  render() {
    return (
      // The stateless component class uses
      // the passed-down handleClick function,
      // accessed here as this.props.onClick,
      // as an event handler:
      <button onClick={this.props.onClick}>
        Click Me!
      </button>
    );
  }
}
```

Define an Event Handler

To make a child component update its parent's `state`, the first step is something that you've seen before: you must define a state-changing method on the parent.

Instructions

1.

Select **Child.js**.

Look at `Child`'s render function. It's similar to the last lesson, but you can see a `<select>` dropdown menu that wasn't there before.

Click Run. Try selecting different names from the dropdown menu in the browser.

It doesn't work! When you select a name, that name is supposed to replace "Frarthur" on the screen.

Look at line 8. Notice that the name inside of the `<h1></h1>` is equal to `this.props.name`. In order to make the dropdown menu *change* the `<h1></h1>`, you will need the dropdown menu to change the value of `this.props.name`!

Checkpoint 2 Passed

Hint

Click Run and try selecting different names from the dropdown menu in the browser.

2.

How can you change `Child`'s `this.props.name`?

Open **Parent.js** and look at line 13.

`Parent` *renders* a `<Child />`, passing in a `name` prop. This `name` prop is the same value that the `<Child />` displays in its `<h1></h1>`.

You need `Child`'s dropdown menu to change `Parent`'s `this.state.name`! That will cause `<Child />` to get passed a new `name` prop, which will change the name displayed on the screen.

In **Parent.js**, define a new function that can change `this.state.name`:

```
changeName(newName) {
  this.setState({
    name: newName
```

```
  });
}

render()                                                    {
  //                                                      ...
}
Checkpoint 3 Passed
```

Hint

Try copy-pasting this code in between the constructor and `render()` in **Parent.js**:

```
changeName(newName)                                         {
  this.setState({
    name:                                                newName
  });
}
```

Child.js

```
import React from 'react';

export class Child extends React.Component {
  render() {
    return (
      <div>
        <h1>
          Hey my name is {this.props.name}!
        </h1>
        <select id="great-names">
          <option value="Frarthur">
            Frarthur
          </option>

          <option value="Gromulus">
            Gromulus
          </option>

          <option value="Thinkpiece">
            Thinkpiece
          </option>
        </select>
```

```
      </div>
    );
  }
}
```

Parent.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import { Child } from './Child';

class Parent extends React.Component {
  constructor(props) {
    super(props);

    this.state = { name: 'Frarthur' };
  }

  changeName(newName) {
    this.setState({
      name: newName
    });
  }

  render() {
    return <Child name={this.state.name} />
  }
}

ReactDOM.render(
  <Parent />,
  document.getElementById('app')
);
```

Pass the Event Handler

In the last exercise, you defined a function in `Parent` that can change `Parent`'s state.

`Parent` must pass this function down to `Child`, so that `Child` can use it in an event listener on the dropdown menu.

Instructions

1.

We now need to make sure that the `.changeName()` method will always refer to the instance of `Parent`, even when we pass it down to `Child` to use.

In the constructor method of `Parent`, bind `this.changeName` to the current value of `this` and store it in `this.changeName`.

Checkpoint 2 Passed

Hint

The generic syntax for binding a method in the constructor is:

```
this.methodName = this.methodName.bind(this);
```

2.

Pass `.changeName()` down to `Child`!

In **Parent.js**, inside of `Parent`'s render function, add a second attribute to `<Child />`. Give this attribute a *name* of `onChange`, and a *value* of the `changeName` method.

Checkpoint 3 Passed

Hint

`<Child>` starts with a single `name` property, like this:

```
<Child name={this.state.name} />
```

If you wanted to pass another property called `myProperty` with a value of `this.changeName`, you might do something like this:

```
<Child name={this.state.name} myProperty={this.changeName} />
```

Your solution will look similar.

Parent.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import { Child } from './Child';
```

```
class Parent extends React.Component {
  constructor(props) {
    super(props);

    this.state = { name: 'Frarthur' };
    this.changeName = this.changeName.bind(this);
  }

  changeName(newName) {
    this.setState({
      name: newName
    });
  }

  render() {
    return <Child name={this.state.name} onChange={this.chan
geName} />
  }
}

ReactDOM.render(
  <Parent />,
  document.getElementById('app')
);
```

Child.js

```
import React from 'react';

export class Child extends React.Component {
  render() {
    return (
      <div>
        <h1>
          Hey my name is {this.props.name}!
        </h1>
        <select id="great-names">
          <option value="Frarthur">
            Frarthur
          </option>
```

```
          <option value="Gromulus">
            Gromulus
          </option>

          <option value="Thinkpiece">
            Thinkpiece
          </option>
        </select>
      </div>
    );
  }
}
```

Automatic Binding

Great work! *Stateless components updating their parents' state* is a React pattern that you'll see more and more. Learning to recognize it will help you understand how React apps are organized.

Click Next to move on to the final version of our programming pattern!

Parent.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import { Child } from './Child';

class Parent extends React.Component {
  constructor(props) {
    super(props);

    this.state = { name: 'Frarthur' };
    this.changeName = this.changeName.bind(this);
  }

  changeName(newName) {
    this.setState({
```

```
      name: newName
    });
  }

  render() {
    return <Child name={this.state.name} onChange={this.chan
geName} />
  }
}

ReactDOM.render(
  <Parent />,
  document.getElementById('app')
);
```

Child.js

```
import React from 'react';

export class Child extends React.Component {
  constructor(props) {
    super(props);
    this.handleChange = this.handleChange.bind(this);
  }

  handleChange(e) {
    const name = e.target.value;
    this.props.onChange(name);
  }

  render() {
    return (
      <div>
        <h1>
          Hey my name is {this.props.name}!
        </h1>
        <select id="great-
names" onChange={this.handleChange}>
          <option value="Frarthur">
            Frarthur
```

```
          </option>

          <option value="Gromulus">
            Gromulus
          </option>

          <option value="Thinkpiece">
            Thinkpiece
          </option>
        </select>
      </div>
    );
  }
}
```