Animation Add-Ons

> **Note:**
> `ReactTransitionGroup` and `ReactCSSTransitionGroup` have been moved to the `react-transition-group` package that is maintained by the community. Its 1.x branch is completely API-compatible with the existing addons. Please file bugs and feature requests in the new repository.

The `ReactTransitionGroup` add-on component is a low-level API for animation, and `ReactCSSTransitionGroup` is an add-on component for easily implementing basic CSS animations and transitions.

High-level API: ReactCSSTransitionGroup

`ReactCSSTransitionGroup` is a high-level API based on `ReactTransitionGroup` and is an easy way to perform CSS transitions and animations when a React component enters or leaves the DOM. It's inspired by the excellent ng-animate library.

Importing

```
import ReactCSSTransitionGroup from 'react-transition-group'; // ES6
var ReactCSSTransitionGroup = require('react-transition-group'); // ES5 with
npm
```

```
class TodoList extends React.Component {
  constructor(props) {
    super(props);
    this.state = {items: ['hello', 'world', 'click', 'me']};
```

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

  }

  handleAdd() {
    const newItems = this.state.items.concat([
      prompt('Enter some text')
    ]);
    this.setState({items: newItems});
  }

  handleRemove(i) {
    let newItems = this.state.items.slice();
    newItems.splice(i, 1);
    this.setState({items: newItems});
  }

  render() {
    const items = this.state.items.map((item, i) => (
      <div key={i} onClick={() => this.handleRemove(i)}>
        {item}
      </div>
    ));

    return (
      <div>
        <button onClick={this.handleAdd}>Add Item</button>
        <ReactCSSTransitionGroup
          transitionName="example"
          transitionEnterTimeout={500}
          transitionLeaveTimeout={300}>
          {items}
        </ReactCSSTransitionGroup>
      </div>
    );
  }
}
```

Note:

You must provide the `key` attribute for all children of `ReactCSSTransitionGroup`, even when only rendering a single item. This is how React will determine which children have entered, left, or stayed.

In this component, when a new item is added to `ReactCSSTransitionGroup` it will get `example-enter` CSS class and the `example-enter-active` CSS class added in the next

tick. This is a convention based on the `transitionName` prop.

You can use these classes to trigger a CSS animation or transition. For example, try adding this CSS and adding a new list item:

```
.example-enter {
  opacity: 0.01;
}

.example-enter.example-enter-active {
  opacity: 1;
  transition: opacity 500ms ease-in;
}

.example-leave {
  opacity: 1;
}

.example-leave.example-leave-active {
  opacity: 0.01;
  transition: opacity 300ms ease-in;
}
```

You'll notice that animation durations need to be specified in both the CSS and the render method; this tells React when to remove the animation classes from the element and — if it's leaving — when to remove the element from the DOM.

Animate Initial Mounting

`ReactCSSTransitionGroup` provides the optional prop `transitionAppear`, to add an extra transition phase at the initial mount of the component. There is generally no transition phase at the initial mount as the default value of `transitionAppear` is `false`. The following is an example which passes the prop `transitionAppear` with the value `true`.

```
render() {
  return (
    <ReactCSSTransitionGroup
      transitionName="example"
      transitionAppear={true}
      transitionAppearTimeout={500}
      transitionEnter={false}
      transitionLeave={false}>
      <h1>Fading at Initial Mount</h1>
    </ReactCSSTransitionGroup>
```

```
  );
}
```

During the initial mount `ReactCSSTransitionGroup` will get the `example-appear` CSS class and the `example-appear-active` CSS class added in the next tick.

```
.example-appear {
  opacity: 0.01;
}

.example-appear.example-appear-active {
  opacity: 1;
  transition: opacity .5s ease-in;
}
```

At the initial mount, all children of the `ReactCSSTransitionGroup` will `appear` but not `enter`. However, all children later added to an existing `ReactCSSTransitionGroup` will `enter` but not `appear`.

> **Note:**
> The prop `transitionAppear` was added to `ReactCSSTransitionGroup` in version `0.13`. To maintain backwards compatibility, the default value is set to `false`.
>
> However, the default values of `transitionEnter` and `transitionLeave` are `true` so you must specify `transitionEnterTimeout` and `transitionLeaveTimeout` by default. If you don't need either enter or leave animations, pass `transitionEnter={false}` or `transitionLeave={false}`.

Custom Classes

It is also possible to use custom class names for each of the steps in your transitions. Instead of passing a string into transitionName you can pass an object containing either the `enter` and `leave` class names, or an object containing the `enter`, `enter-active`, `leave-active`, and `leave` class names. If only the enter and leave classes are provided, the enter-active and leave-active classes will be determined by appending '-active' to end of the class name. Here are two examples using custom classes:

```
// ...
<ReactCSSTransitionGroup
  transitionName={ {
    enter: 'enter',
    enterActive: 'enterActive',
    leave: 'leave',
    leaveActive: 'leaveActive',
    appear: 'appear',
    appearActive: 'appearActive'
  } }>
  {item}
</ReactCSSTransitionGroup>

<ReactCSSTransitionGroup
  transitionName={ {
    enter: 'enter',
    leave: 'leave',
    appear: 'appear'
  } }>
  {item2}
</ReactCSSTransitionGroup>
// ...
```

Animation Group Must Be Mounted To Work

In order for it to apply transitions to its children, the `ReactCSSTransitionGroup` must already be mounted in the DOM or the prop `transitionAppear` must be set to `true`.

The example below would **not** work, because the `ReactCSSTransitionGroup` is being mounted along with the new item, instead of the new item being mounted within it. Compare this to the Getting Started section above to see the difference.

```
render() {
  const items = this.state.items.map((item, i) => (
    <div key={item} onClick={() => this.handleRemove(i)}>
      <ReactCSSTransitionGroup transitionName="example">
        {item}
      </ReactCSSTransitionGroup>
    </div>
  ));

  return (
    <div>
      <button onClick={this.handleAdd}>Add Item</button>
      {items}
    </div>
```

```
    );
  }
```

Animating One or Zero Items

In the example above, we rendered a list of items into `ReactCSSTransitionGroup`. However, the children of `ReactCSSTransitionGroup` can also be one or zero items. This makes it possible to animate a single element entering or leaving. Similarly, you can animate a new element replacing the current element. For example, we can implement a simple image carousel like this:

```
import ReactCSSTransitionGroup from 'react-transition-group';

function ImageCarousel(props) {
  return (
    <div>
      <ReactCSSTransitionGroup
        transitionName="carousel"
        transitionEnterTimeout={300}
        transitionLeaveTimeout={300}>
        <img src={props.imageSrc} key={props.imageSrc} />
      </ReactCSSTransitionGroup>
    </div>
  );
}
```

Disabling Animations

You can disable animating `enter` or `leave` animations if you want. For example, sometimes you may want an `enter` animation and no `leave` animation, but `ReactCSSTransitionGroup` waits for an animation to complete before removing your DOM node. You can add `transitionEnter={false}` or `transitionLeave={false}` props to `ReactCSSTransitionGroup` to disable these animations.

Note:

When using `ReactCSSTransitionGroup`, there's no way for your components to be notified when a transition has ended or to perform any more complex logic around

animation. If you want more fine-grained control, you can use the lower-level `ReactTransitionGroup` API which provides the hooks you need to do custom transitions.

Low-level API: ReactTransitionGroup

Importing

```
import ReactTransitionGroup from 'react-addons-transition-group' // ES6
var ReactTransitionGroup = require('react-addons-transition-group') // ES5
with npm
```

`ReactTransitionGroup` is the basis for animations. When children are declaratively added or removed from it (as in the example above), special lifecycle methods are called on them.

- `componentWillAppear()`
- `componentDidAppear()`
- `componentWillEnter()`
- `componentDidEnter()`
- `componentWillLeave()`
- `componentDidLeave()`

Rendering a Different Component

`ReactTransitionGroup` renders as a `span` by default. You can change this behavior by providing a `component` prop. For example, here's how you would render a `<ul>`:

```
<ReactTransitionGroup component="ul">
  {/* ... */}
</ReactTransitionGroup>
```

Any additional, user-defined, properties will become properties of the rendered component

Any additional, user-defined, properties will become properties of the rendered component. For example, here's how you would render a `<ul>` with CSS class:

```
<ReactTransitionGroup component="ul" className="animated-list">
  {/* ... */}
</ReactTransitionGroup>
```

Every DOM component that React can render is available for use. However, `component` does not need to be a DOM component. It can be any React component you want; even ones you've written yourself! Just write `component={List}` and your component will receive `this.props.children`.

Rendering a Single Child

People often use `ReactTransitionGroup` to animate mounting and unmounting of a single child such as a collapsible panel. Normally `ReactTransitionGroup` wraps all its children in a `span` (or a custom `component` as described above). This is because any React component has to return a single root element, and `ReactTransitionGroup` is no exception to this rule.

However if you only need to render a single child inside `ReactTransitionGroup`, you can completely avoid wrapping it in a `<span>` or any other DOM component. To do this, create a custom component that renders the first child passed to it directly:

```
function FirstChild(props) {
  const childrenArray = React.Children.toArray(props.children);
  return childrenArray[0] || null;
}
```

Now you can specify `FirstChild` as the `component` prop in `<ReactTransitionGroup>` props and avoid any wrappers in the result DOM:

```
<ReactTransitionGroup component={FirstChild}>
  {someCondition ? <MyComponent /> : null}
</ReactTransitionGroup>
```

This only works when you are animating a single child in and out, such as a collapsible This approach wouldn't work when animating multiple children or replacing the single c with another child, such as an image carousel. For an image carousel, while the current image is animating out, another image will animate in, so `<ReactTransitionGroup>` needs

image is animating out, another image will animate in, so `<ReactTransitionGroup>` needs to give them a common DOM parent. You can't avoid the wrapper for multiple children, but you can customize the wrapper with the `component` prop as described above.

Reference

`componentWillAppear()`

```
componentWillAppear(callback)
```

This is called at the same time as `componentDidMount()` for components that are initially mounted in a `TransitionGroup`. It will block other animations from occurring until `callback` is called. It is only called on the initial render of a `TransitionGroup`.

`componentDidAppear()`

```
componentDidAppear()
```

This is called after the `callback` function that was passed to `componentWillAppear` is called.

`componentWillEnter()`

```
componentWillEnter(callback)
```

This is called at the same time as `componentDidMount()` for components added to an existing `TransitionGroup`. It will block other animations from occurring until `callback` is called. It will not be called on the initial render of a `TransitionGroup`.

`componentDidEnter()`

```
componentDidEnter()
```

This is called after the `callback` function that was passed to `componentWillEnter()` is called.

`componentWillLeave()`

```
componentWillLeave(callback)
```

This is called when the child has been removed from the `ReactTransitionGroup`. Though the child has been removed, `ReactTransitionGroup` will keep it in the DOM until `callback` is called.

`componentDidLeave()`

```
componentDidLeave()
```

This is called when the `willLeave` `callback` is called (at the same time as `componentWillUnmount()`).