METAL.JS: JSX

WHAT IS JSX?

- JSX is an interesting alternative to templates developed by the team at Facebook.
- It stands for JavaScript XML, and follows XML-type element declaration for web component development.
- It is an inline markup that looks like HTML and gets transformed into JavaScript.
- A JSX expression starts with an HTML-like open tag, and ends with the corresponding closing tag.
- While Soy Templates is the recommended templating system for Metal.js, JSX is a widely-used technology you may already be familiar with.
- If you already use JSX, or know how to use it, you can reduce some of your development time.

JSX EXAMPLE CODE

- Some example JSX code:

```
import JSXComponent from 'metal-jsx';

class Modal extends JSXComponent {
    render() {
        return <header class="modal-header">
                <button type="button" class="close">
                    <span>×</span>
                </button>
                <h4>{this.config.header}</h4>
            </header>;
    }
}

export default Modal;
```

WHY USE JSX?

- **Familiarity**: Developers are familiar with XML, and JSX provides a similar type of element declaration.
- **Semantics**: JSX is easier to understand, as it follows a declarative type of programming.
- **Ease of development**: JSX provides a clean way to encapsulate all the logic and markup in one definition.

RELATIONSHIP BETWEEN METAL.JS AND JSX

- The only thing you need to do to use JSX in your Metal.js component is to extend from **JSXComponent**, like this:

```
import JSXComponent from 'metal-jsx';

class MyComponent extends JSXComponent {
}

export default MyComponent;
```

JSX RENDER AFTER EXTENDING

- Now that we've extended from **JSXComponent**, we can use JSX in the `render` method to specify what our component should render.

```
import JSXComponent from 'metal-jsx';

class MyComponent extends JSXComponent {
    render() {
        return <div class={this.config.cssClass}>
            Hello {this.name}
        </div>;
    }
}

MyComponent.STATE = {
    name: {
        validator: core.isString,
        value: 'World'
    }
};

export default MyComponent;
```

RENDERING JSX

- JSX components can either be rendered in the usual way (as seen here: *http://metaljs.com/docs/rendering-components.html*), or via the `JSXComponent.render` function, like this:

```
JSXComponent.render(MyComponent, {name: 'my-component'}, parent);
```

JSX COMPILATION

- For the integration between Metal.js and JSX to work, the JSX code needs to be compiled via a babel plugin called `babel-plugin-incremental-dom`.
- Using it directly means you'd need to configure it manually, so the Metal.js team also provides a *babel preset* that you can use instead: *https://www.npmjs.com/package/babel-preset-metal-jsx*.

Notes: