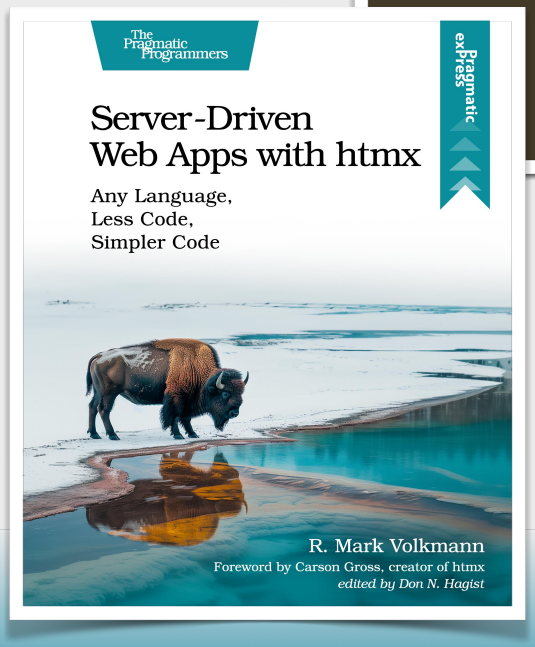
Web REactive Components (wrec)

Slides at https://github.com/mvolkmann/talks/

R. Mark Volkmann
Object Computing, Inc.
https://objectcomputing.com
r.mark.volkmann@gmail.com

About Me

- Partner and Distinguished Software Engineer at Object Computing, Inc. in St. Louis, Missouri USA
- 44 years of professional software development experience
- Writer and speaker
- **Blog** at https://mvolkmann.github.io/blog/
- Author of Manning book "**Svelte ... in Action**"
- Author of Pragmatic Bookshelf book "**Server-Driven Web Apps with htmx**"
- Currently writing a book on web components



What is wrec

- Library that simplifies development of web components
  - https://www.npmjs.com/package/wrec
- Strong focus on simplicity and reactivity
- To define a web component
  - define a class that extends **`Wrec`**
  - define properties with static property named **`property`**
  - specify scoped CSS with static property named **`css`**
  - specify HTML to render with static property named **`html`**

Getting Started ...

- Create project directory and **`cd`** to it
- Create **`package.json`** by entering **`npm init -y`**
- Install Vite by entering **`npm i -D vite`**
- Add **`dev`** script to **`package.json`**
- Install wrec by entering **`npm i wrec`**
- Create **`hello-world.js`**

package.json

```
{
  "name": "hello-world",
  "scripts": {
    "dev": "vite"
  },
  "devDependencies": {
    "vite": "^7.3.1"
  },
  "dependencies": {
    "wrec": "^0.18.5"
  }
}
```

The **`html`** tag function enables syntax highlighing and code completion of HTML inside tagged template literals.

hello-world.js

```
import { html, Wrec } from "wrec";

class HelloWorld extends Wrec {
  static html = html`<h1>Hello, World!</h1>`;
}

HelloWorld.register();
```

... Getting Started

- Create **`index.html`**
- Start local server by entering **`npm run dev`**
- Browse **`localhost:5173`**

index.html

```
<!doctype html>
<html lang="en">
  <head>
    <script type="module" src="hello-world.js"></script>
  </head>
  <body>
    <hello-world></hello-world>
  </body>
</html>
```

Better hello-world

hello-world.js

```
import { css, html, Wrec } from "wrec";

class HelloWorld extends Wrec {
  static properties = {
    name: { type: String, value: "World" },
  };

  static css = css`
    h1 {
      color: blue;
    }
  `;

  static html = html`<h1>Hello, <span>this.name</span>!</h1>`;
}

HelloWorld.register();
```

```
<hello-world name="Earth"></hello-world>
```

Hello, Earth!

Best hello-world

hello-world.js

```
import { css, html, Wrec } from "wrec";

class HelloWorld extends Wrec {
  static properties = {
    color: { type: String, value: "blue" },
    name: { type: String, value: "World" },
  };

  static css = css`
    h1 {
      color: this.color;
    }
  `;

  static html = html`<h1>Hello, <span>this.name</span>!</h1>`;
}

HelloWorld.register();
```

```
<hello-world color="red" name="Earth"></hello-world>
```

Hello, Earth!

Vanilla Version

requires much more code

hello-world.js

```
class HelloWorld extends HTMLElement {
  static get observedAttributes() {
    return ["color", "name"];
  }

  #color = "black";
  #name = "World";
  #p = document.createElement("p");

  attributeChangedCallback(name, _oldValue, newValue) {
    if (name === "color") {
      if (newValue !== this.#color) this.color = newValue;
    } else if (name === "name") {
      if (newValue !== this.#name) this.name = newValue;
    }
  }

  connectedCallback() {
    this.attachShadow({ mode: "open" });
    this.shadowRoot.replaceChildren(this.#p);
  }

  get color() {
    return this.#color;
  }

  get name() {
    return this.#name;
  }

  set color(value) {
    this.#color = value;
    this.setAttribute("color", value);
    this.#p.style.color = value;
  }

  set name(value) {
    this.#name = value;
    this.setAttribute("name", value);
    this.#p.textContent = `Hello, ${value}!`;
  }
}

customElements.define("hello-world", HelloWorld);
```

`this.` Controversy

- Should JavaScript expressions in element text content, attribute values, and CSS properties be surrounded by some kind of delimiter like curly braces?
- Is searching for **`this.`*`someName`*** in plain text sufficient?
- When would plain text ever contain **`this.`** followed by a letter that is not part of a JavaScript expression?
- Answer: never
- But there is an escape mechanism
  - **`this`** followed by two periods is not treated as part of a JavaScript expression and will be replaced by **`this`** followed by a single period

Property/Attribute Reflection

- It's automatic!
- Open DevTools and click Elements tab
- Change an attribute of a custom element
  - like **`color`** or **`name`** attribute in a **`hello-world`** element
  - updates what is rendered AND corresponding property
- Select a custom element and click Console tab
- Change properties of **`$0`**
  - for example, **`$0.name = 'Moon'`**
  - updates what is rendered AND corresponding attribute

Secret Sauce

- The first time an instance of a given web component class is used,
  wrec searches all attribute values, element text content, and CSS properties
  for expressions matching **`this.{property-name}`** using a regular expression.
- Expressions in CSS properties are replaced by a local variable reference and
  a CSS variable declaration whose value is the expression is added before it.
- A static map named **`propToExprsMap`** is created (one per **`Wrec`** subclass)
  whose keys are property names and
  whose values arrays of the expressions where they are found.
- A map named **`exprToRefsMap`** is created for each **`Wrec`** subclass instance
  whose keys are the expressions and whose values are arrays of references
  to the elements, attributes, and CSS variable declarations where they appear.
- For each property, **`Object.defineProperty`** is called to add getter and setter methods.
- When an instance property change is detected by its setter method,
  a list of the expressions that use the property is retrieved from **`propToExprsMap`**
  and new values are computed.
- Then for each expression whose value was computed, a list of all references to the expression
  is retrieved from **`exprToRefsMap`** and they are updated with the new value.

```
h1 {
  --color: this.color;
  color: var(--color);
}
```

JavaScript Expressions

Lit doesn't do this

- **`input`** & **`select`** element **`value`** attribute gets 2-way binding to a property
- Expressions that use property automatically update when value changes

```
import { css, html, Wrec } from "wrec";

class CircleCalculator extends Wrec {
  static properties = {
    radius: { type: Number, value: 0 },
  };

  static css = css`
    :host {
      font-family: sans-serif;
    }
    label {
      font-weight: bold;
    }
  `;
```

```
  static html = html`
    <div>
      <label for="radius">Radius:</label>
      <input id="radius" type="number" value="this.radius" />
    </div>
    <div>
      <label for="circumference">Circumference:</label>
      <span id="circumference">2 * Math.PI * this.radius</span>
    </div>
    <div>
      <label for="area">Area:</label>
      <span id="area">Math.PI * this.radius</span>
    </div>
  `;
}

CircleCalculator.register();
```

Radius: 1
Circumference: 6.283185307179586
Area: 3.141592653589793

Parent-Child 2-Way Bindings

```
import { css, html, Wrec } from "wrec";

class ToggleButtons extends Wrec {
  static properties = {
    labels: { type: String },
    value: { type: String, dispatch: true },
  };
  static css = css`...`;
  static html = "this.makeButtons(this.labels)";

  handleClick(event) {
    this.value = event.target.textContent.trim();
  }

  makeButtons(labels) {
    const labelArray = labels.split(",");
    return labelArray.map((label) => {
      const classExpr = `this.value === '${label}' ? 'selected' : ''`;
      return html`
        <button class="${classExpr}" onClick="handleClick">${label}</button>
      `;
    });
  }
}
ToggleButtons.register();
```

details omitted

```
import { html, Wrec } from "wrec";
import "./hello-world";
import "./toggle-buttons";

class MyApp extends Wrec {
  static properties = {
    color: { type: String, value: "red" },
  };
  static html = html`
    <hello-world color="this.color"></hello-world>
    <toggle-buttons
      labels="red,green,blue"
      value="this.color"
    ></toggle-buttons>
  `;
}
MyApp.register();
```

Hello, World!

red | green | blue

number-slider

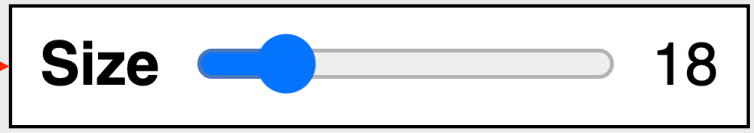
```
import { css, html, Wrec } from "wrec";

class NumberSlider extends Wrec {
  static properties = {
    label: { type: String },
    labelWidth: { type: String },
    max: { type: Number, value: 100 },
    min: { type: Number, value: 0 },
    value: { type: Number },
  };

  static css = css`...`;
```

details omitted

```
  static html = html`
    <label>this.label</label>
    <input
      type="range"
      min="this.min"
      max="this.max"
      value:input="this.value"
    />
    <span>this.value</span>
  `;
}

NumberSlider.register();
```

`:input` causes the `value` property to be updated on every mouse drag, not just when focus moves away

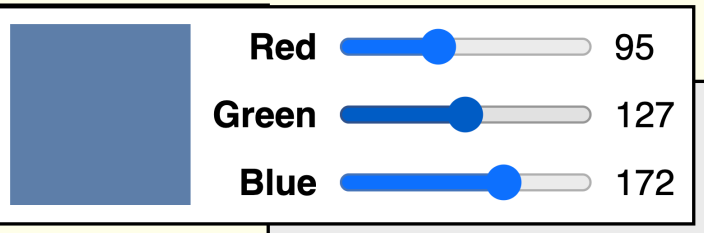
```
<number-slider
  label="Size"
  max="48"
  min="12"
  name="size"
  value="this.size"
></number-slider>
```



property in parent component

color-picker

```
import { css, html, Wrec } from "wrec";
import "./number-slider";

class ColorPicker extends Wrec {
  static properties = {
    labelWidth: { type: String, value: "3rem" },
    red: { type: Number },
    green: { type: Number },
    blue: { type: Number },
    color: {
      type: String,
      computed: "`rgb(${this.red},${this.green},${this.blue})`",
    },
  };

  static css = css`
    :host {
      display: flex;
      gap: 0.5rem;
    }
    #sliders {
      display: flex;
      flex-direction: column;
      justify-content: space-between;
    }
    #swatch {
      background-color: this.color;
      height: 5rem;
      width: 5rem;
    }
  `;

  static html = html`
    <div id="swatch"></div>
    <div id="sliders">
      ${this.makeSlider("Red")}
      ${this.makeSlider("Green")}
      ${this.makeSlider("Blue")}
    </div>
  `;

  static makeSlider(label) {
    return html`
      <number-slider
        label=${label}
        label-width="this.labelWidth"
        max="255"
        value="this.${label.toLowerCase()}"
      ></number-slider>
    `;
  }
}

ColorPicker.register();
```

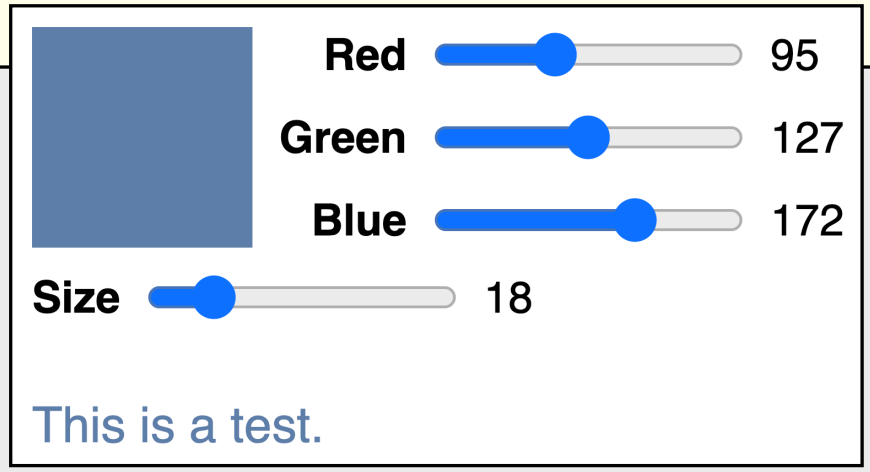


color-demo ...

```
import { css, html, Wrec } from "wrec";
import "./color-picker";
import "./number-slider";

class ColorDemo extends Wrec {
  static properties = {
    color: { type: String },
    size: { type: Number, value: 18 },
  };

  static css = css`
    :host {
      display: flex;
      flex-direction: column;
      gap: 0.5rem;
      font-family: sans-serif;
    }
    p {
      color: this.color;
      font-size: this.size + "px";
    }
  `;
  static html = html`
    <color-picker color="this.color"></color-picker>
    <number-slider
      label="Size"
      max="48"
      min="12"
      name="size"
      value="this.size"
    ></number-slider>
    <p>This is a test.</p>
  `;
}

ColorDemo.register();
```

... color-demo

```
<!doctype html>
<html lang="en">
  <head>
    <script src="color-demo.js" type="module"></script>
  </head>
  <body>
    <color-demo></color-demo>
  </body>
</html>
```



Forms

- Cover formAssociated

State

- Cover the wrec State class

Resources

- **Decorators proposal**
  - https://github.com/tc39/proposal-decorators
- **Decorators in TypeScript 5.0**
  - https://devblogs.microsoft.com/typescript/announcing-typescript-5-0/#decorators
- **ECMAScript specification**
  - https://ecma-international.org/publications-and-standards/standards/ecma-262/

Wrap Up

- Decorators reduce boilerplate code
  and make code more declarative
- Arguments cannot be specified when applying a decorator,
  but they can be specified when applying a decorator factory,
  which returns a configured decorator
- Decorators will be an official JavaScript feature soon,
  but they can be used today with TypeScript and Vite

My Latest Effort

- See npm package **wrec** which greatly simplifies creating web components
  - https://www.npmjs.com/package/wrec
- Name is acronym for **W**eb **RE**active **C**omponents

```
import {html, Wrec} from 'wrec';

class HelloWorld extends Wrec {
  static properties = {
    name: {type: String, value: 'World'}
  };

  static html = html`<div>Hello, <span>this.name</span>!</div>`;
}

HelloWorld.register();
```

```
<hello-world name="Mark"></hello-world>
```