
📘 MicroUI Guide: Composing Components

🔹 What is Component Composition?

- Composition means building components out of other components.
- Instead of writing everything in one place, you can split logic/UI into smaller pieces.
- This makes code reusable, cleaner, and easier to test.

🔹 Simple Parent → Child Example

```
import { createComponent } from "@magnumjs/micro-ui";

// Child Component
const Child = createComponent({
  render(props) {
    return `<p>Hello, ${props.name} 👋</p>`;
  }
});

// Parent Component

const Parent = createComponent({
  state: { name: "Alice" },
  render() {
    return `
      <div>
        <h2>I am the Parent</h2>
        ${Child({ name: this.state.name })}
      </div>
    `;
  }
});
```

✅ Child is called inside Parent’s render.

✅ Parent passes props down: name: this.state.name.

✅ Child renders its part independently.

🔹 **Multiple Children Example**

```
const Card = createComponent({
  render(props) {
    return `
      <div class="card">
        <h3>${props.title}</h3>
        <p>${props.content}</p>
      </div>
    `;
  }
});

const Dashboard = createComponent({
  render() {
    return `
      <section>
        ${Card({ title: "Users", content: "123 active" })}
        ${Card({ title: "Sales", content: "$1,230" })}
        ${Card({ title: "Messages", content: "45 new" })}
      </section>
    `;
  }
});
```

✅ Dashboard reuses the same Card multiple times.

✅ Each Card gets its own props.

✅ Keeps layout consistent and reusable.

🔹 **Parent Updating Child with State**

```
const Counter = createComponent({
  state: { count: 0 },
  render() {
    return `
      <div>
        <p>Count: ${this.state.count}</p>
        <button data-ref="inc">+1</button>
```

```
      </div>
    `;
  },
  onMount() {
    this.ref("inc").addEventListener("click", () => {
      this.setState({ count: this.state.count + 1 });
    });
  }
});

const App = createComponent({
  render() {
    return `
      <main>
        <h1>Counter App</h1>
        ${Counter()}
      </main>
    `;
  }
});
```

✅ App hosts the Counter.

✅ Counter manages its own state.

✅ Parent doesn't need to know the details.

🔹 **Why Composition Matters**

- 🔹 Encourages reusable building blocks.
- 🔹 Keeps code organized (separation of concerns).
- 🔹 Allows independent testing of components.
- 🔹 Makes UI scalable as projects grow.

There are various styles for event handling:

1. Using this.addEvent in the render function body
2. Using declarative attributes like data-action-click

3. Refs approach (already in the last example)

👉 "You can choose whichever style fits your component best. All work consistently."

🔹 **Child with Events (Multiple Ways)**

Option 1 – Using this.addEvent in render

```
const Button = createComponent({
  render() {
    this.addEvent("click button", () => alert("Clicked!"));
    return `<button>Click Me</button>`;
  }
});
```

Option 2 – Using data-action-click

```
const Button = createComponent({
  render() {
    return `<button data-action-click="sayHello">Click Me</button>`;
  },
   sayHello() {
     alert("Clicked via data-action!");
   }
});
```

Option 3 – Using ref directly

```
const Button = createComponent({
  render() {
    return `<button data-ref="btn">Click Me</button>`;
  },
  onMount() {
    this.ref("btn").addEventListener("click", () => alert("Clicked via ref!"));
  }
});
```

👉 Tip: Use the style that feels most natural for your use case.

All 3 methods work and can even be mixed across different components.

⚡ **MicroUI** –> https://github.com/magnumjs/micro-ui