MicroUI Developer Guide: State, Refs, Events & Shared Context

1. setState — Managing Local Component State

Purpose: Update a component's internal state and trigger a re-render.

- Merges new state with existing state
- Triggers a re-render asynchronously
- Works with `update()` for props changes

```
this.setState({ key: value });
```

```
const Counter = createComponent({
  state: { count: 0 },
  render() {
    return `<button data-action-click="inc">${this.state.count}</but
  },
  on: {
    'click button': () => this.setState({ count: this.state.count +
  }
});
```

2. refs — Accessing DOM Elements

Purpose: Reference specific DOM elements inside a component.

- Returns `null` if element doesn’t exist or component is unmounted
- Lazy resolution: element must exist in the DOM

```
this.ref('elementName');
```

```
const InputFocus = createComponent({
  render() {
    return `<input data-ref="nameInput" />`;
  },
  onMount() {
    this.ref('nameInput').focus();
  }
});
```

3. Events — Declarative Interaction

Two ways to handle events:

1. `on` object
2. `data-action-*` attributes

Advanced: Use `*:` wildcard for global handlers.

```
on: { 'click button': () => console.log('Clicked!') }
```

```
<button data-action-click="increment">+</button>
```

```
on: {
  'click button': () => this.setState({ count: this.state.count + 1
}
```

4. Shared Context — Cross-Component State

Purpose: Share reactive state across components without prop drilling.

```
import { context } from '@magnumjs/micro-ui';
```

```
context.subscribe('user:name', (value) => console.log('New name:', v
context.publish('user:name', 'Tova');
```

5. Built-in Emits

Purpose: Components can emit events directly to parent or global listeners.

```
this.emit('custom:event', payload);

const Child = createComponent({
  render() { return `<button>Click me</button>`; },
  on: {
    'click button': () => this.emit('child:clicked', { time: Date.no
  }
});

const Parent = createComponent({
  render() { return `${Child()}`; },
  on: {
    'child:clicked': (data) => console.log('Child clicked at', data.
  }
});
```

6. Channels — Scoped Reactive Pub/Sub

Purpose: Reactive pub/sub system for multiple components.

```
// Publisher
context.publish('chat:newMessage', 'Hello!');

// Subscriber
context.subscribe('chat:newMessage', (msg) => console.log('New messa
```

7. Putting it Together

Example: Counter with `setState`, Shared Context, Emits, and Channel

```
const Counter = createComponent({
```

```
  state: { count: 0 },
  render() { return `<button data-action-click="inc">+</button>`; },
  on: {
    'click button': () => {
      this.setState({ count: this.state.count + 1 });
      this.emit('counter:updated', this.state.count);
      context.publish('counter:value', this.state.count);
    }
  }
});

context.subscribe('counter:value', val => console.log('Counter value
```

Key Takeaways

- `setState()` → Local reactive state updates
- `refs` → Access component DOM elements safely
- Events → Declarative via `on` object or `data-action-*` attributes
- Shared `context` → Cross-component reactive state
- `emit` → Direct parent/subscriber communication
- Channels → Scoped pub/sub for multi-component apps

✨ MicroUI —> http://github.com/magnumjs/micro-ui