

funcwc

DOM-Native SSR Components

Ultra-lightweight, **type-safe** SSR components with the **DOM as your state container**

Philosophy: DOM as State

🧠 Revolutionary Approach

Instead of JavaScript state objects, funcwc uses the DOM itself as the state container.

💠 State Mapping

CSS Classes → UI states (active, open, loading)

Data Attributes → Component data (data-count="5")

Element Content → Display values (counter numbers, text)

Form Values → Input states (checkboxes, text inputs)

Eliminates state synchronization bugs

```
// Traditional approach  
const state = { count: 5, active: false };  
function updateState(newState) {  
  state = { ...state, ...newState };  
  renderComponent();  
}
```

```
// funcwc approach  
<div data-count="5" class="inactive">  
  <span>5</span>  
</div>
```

```
// Just inspect the DOM to debug!
```

Key Features

funcwc revolutionizes SSR component development with these powerful features:



DOM-Native State

Component state lives in **CSS classes**, **data attributes**, and **element content**



Function-Style Props

Zero duplication between props and render parameters



CSS-Only Format

Auto-generated class names from CSS properties



Type-Safe

Full TypeScript inference with **smart type helpers**



HTMX Ready

Built-in server actions for **dynamic updates**



Zero Runtime

No client-side framework dependencies



SSR-First

Render on server, send **optimized HTML**



JSON Requests, HTML Responses

Standardized JSON-encoded htmx requests; server returns **HTML snippets**

Function-Style Props

Zero Duplication!

- ✦ Define props **directly** in render function parameters
- ✂ **No duplication** between props definition and function parameters
- ↻ **Auto-parsed** from HTML attributes with smart type helpers
- ⚙ **Default values** built directly into the function signature

Maximum developer productivity with minimal boilerplate

```
// ✓ Function-style props - zero duplication!
defineComponent("smart-counter", {
  render: ({
    initialCount = number(0), // Auto-parsed from
    // HTML attributes
    step = number(1), // Default values built-in
    label = string("Counter"), // Type-safe with
    // smart helpers
  }) => (
    <div data-count={initialCount}>
      {label}: {initialCount}
      <button onclick="/* DOM action */">+{step}
    </button>
    </div>
  ),
});
```

CSS-Only Format

Auto-Generated Classes!



Just write **CSS properties** - class names auto-generated!



No selectors needed - just pure CSS properties



Zero duplication between CSS and class names

```
card: `{ border: 2px solid
#ddd; border-radius: 8px;
padding: 1.5rem; }`
```



.card

Simpler styling with automatic class management

```
defineComponent("beautiful-card", {
  styles: {
    // ✨ Just CSS properties - class names auto-generated!
    card: `{ border: 2px solid #ddd; border-radius: 8px; padding: 1.5rem; }`,
    title: `{ font-size: 1.25rem; font-weight: bold; color: #333; }`,
    buttonPrimary:
      `{ background: #007bff; color: white; padding: 0.5rem 1rem; }`,
    // Auto-generates: .card, .title, .button-primary
  },
  render: (props, api, classes) => (
    <div class={classes!.card}>
      <h3 class={classes!.title}>Amazing!</h3>
      <button class={classes!.buttonPrimary}>Click me</button>
    </div>
  ),
});
```


Counter Example

API Updates (JSON in, HTML out)

```
defineComponent("counter", {
  styles: {
    // ✨ CSS-only format - no selectors needed!
    container: `{ display: inline-flex; gap: 0.5rem; padding: 1rem; border: 2px solid #007bff;
border-radius: 6px; align-items: center; background: white; }`,
    counterButton: `{ padding: 0.5rem; background: #007bff; color: white; border: none; border-
radius: 4px; cursor: pointer; min-width: 2rem; font-weight: bold; }`,
    counterButtonHover: `{ background: #0056b3; }`,
    display: `{ font-size: 1.5rem; min-width: 3rem; text-align: center; font-weight: bold; color:
#007bff; }`,
  },
  render: ({ initialCount = number(0), step = number(1) }, api, classes) => (
    <div class={classes!.container} data-count={initialCount}>
      <button
        class={classes!.counterButton}
        {...api.adjust({ current: initialCount, delta: -step, step }, {
          target: `closest .${classes!.container}`
        })}
      >
        -{step}
      </button>
      <span class={classes!.display}>{initialCount}</span>
      <button
        class={classes!.counterButton}
        {...api.adjust({ current: initialCount, delta: step, step }, {
          target: `closest .${classes!.container}`
        })}
      >
        +{step}
      </button>
    </div>
  ),
});
```

DOM State Management



Counter value stored in **data-count** attribute

```
data-count={initialCount}
```

JSON in, HTML out



API helpers generate **HTMX attributes** for dynamic updates

```
api.adjust({...})
```

CSS-Only Format



Auto-generated **class names** from CSS properties

```
classes!.container
```

Todo Item Example

HTMX + Function-Style Props

```
defineComponent("todo-item", {
  api: {
    toggle: patch("/api/todos/:id/toggle", async (req,
params) => {
      const body = await req.json() as { done?: boolean };
      const isDone = !!body.done;
      return new Response(
        renderComponent("todo-item", {
          id: params.id,
          text: "Task updated!",
          done: !isDone,
        }),
        { headers: { "content-type": "text/html;
charset=utf-8" } } },
      );
    },
    remove: del("/api/todos/:id", () => new Response(null,
{ status: 200 })),
  },
  styles: {
    // ✨ CSS-only format for todo items!
    item: `{ display: flex; align-items: center; gap:
0.5rem; padding: 0.75rem; border: 1px solid #ddd; border-
radius: 4px; margin-bottom: 0.5rem; background: white;
transition: background-color 0.2s; }`,
    itemDone: `{ background: #f8f9fa; opacity: 0.8; }`,
    textDone: `{ text-decoration: line-through; color:
#6c757d; }`,
  },
  render: (
    { id = string("1"), text = string("Todo item"), done =
boolean(false) },
    api,
    classes,
  ) => {
    const itemClass = `${classes!.item} ${done ?
classes!.itemDone : ""}`;
    const textClass = `${classes!.text} ${done ?
classes!.textDone : ""}`;

    return (
      <div class={itemClass} data-id={id}>
        <input
          type="checkbox"
          class={classes!.checkbox}
          checked={done}
          {...api.toggle(id, { done: !done })}
        >
        <span class={textClass}>{text}</span>
        <button class={classes!.deleteBtn}
          {...api.remove(id)}>x</button>
      </div>
    );
  },
});
```

↔ Hybrid State Management

- ✓ **Local UI state:** Checkbox syncs to CSS class instantly
- ☁ **Server persistence:** HTMX handles data updates
- Σ **Function-style props:** Zero duplication
- 🛠 **Auto-generated classes:** From CSS-only format

👁 Visual Result







☐ Complete presentation slides

☒ Research funcwc documentation




☐ Prepare demo examples

Architecture & Request Flow

System Components

-  **Server:** Deno.serve serves demo, imports components, renders HTML
-  **Routing:** Component API handlers register with internal router
-  **HTMX:** Loads HTMX and json-enc extension for JSON requests
-  **Headers:** Generated HTMX attributes include Accept and X-Requested-With
-  **Swap/Target:** Non-GET actions default to hx-swap="outerHTML"
-  **Scoping:** Components inject data-component="" on root element

Request Flow

- 1 Browser triggers HTMX action**
User interaction sends JSON payload to server endpoint

- 2 Server processes request**
Handler processes JSON and renders HTML with renderComponent

- 3 Server returns response**
Response returns as text/html with proper content-type

- 4 HTMX swaps HTML**
HTMX replaces target element with new HTML content

Performance Benefits

funcwc delivers significant performance advantages over traditional component frameworks:



Faster

No client-side state management overhead



Smaller

Zero runtime dependencies, minimal JavaScript



Simpler

DOM inspector shows all state



Instant

Direct DOM manipulation, no virtual DOM



Reliable

No state synchronization bugs



Ergonomic

Function-style props + CSS-only format = maximum productivity



Revolutionary DOM-native approach delivers superior performance and developer experience

Conclusion



defineComponent API

Clean object-based configuration for intuitive component definition



CSS-Only Format

Auto-generated class names from CSS property blocks



Function-Style Props

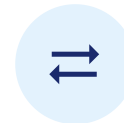
Props inferred from render parameters and defaults



DOM-Native State



Zero Runtime



JSON in, HTML out

Try funcwc today!

Experience the revolutionary approach to component development

Get Started →

Built with ♥ for the modern web. Deno + TypeScript + DOM-native state management + Revolutionary ergonomics.