Integrating Blazor Web Components

---

1. Introducing Web Components

What are Web Components?

Web Components are a set of browser standards that allow you to build **framework-agnostic, reusable UI elements** that work in any HTML page.

They are based on **three core standards**:

Standard	**Purpose**
Custom Elements	Define your own HTML tags
Shadow DOM	Encapsulated DOM + CSS
HTML Templates	Reusable markup

Why Web Components Matter

- Work with **Angular, React, MVC, plain HTML**
- No framework lock-in
- Long-term browser-native solution

Blazor leverages **Custom Elements** to expose Razor components as standard HTML tags.

---

2. Exploring Custom Elements

Basic Custom Element (Pure JavaScript)

```
<script>
class HelloWorld extends HTMLElement {
  connectedCallback() {
    this.innerHTML = `<h3>Hello from Custom Element</h3>`;
  }
```

```
}

customElements.define('hello-world', HelloWorld);

</script>


<hello-world></hello-world>
```

Key Lifecycle Hooks

Method	Trigger
connectedCallback()	Element added to DOM
disconnectedCallback()	Element removed
attributeChangedCallback()	Attribute change

Blazor **hooks Razor components into this lifecycle**.

---

3. Exploring the Blazor Component (as Web Component)

Blazor allows Razor components to be **compiled into custom elements** using:

Microsoft.AspNetCore.Components.CustomElements

Step 1: Create a Blazor WebAssembly Project

```
dotnet new blazorwasm -n BlazorWebComponentsDemo
```

Step 2: Create a Razor Component

Counter.razor

```
<h3>Counter</h3>


<p>Current count: @count</p>

<button @onclick="Increment">+</button>
```

```
@code {
    int count = 0;
    void Increment() => count++;
}
```

Step 3: Register as Custom Element

Program.cs

```
using Microsoft.AspNetCore.Components.WebAssembly.Hosting;
using Microsoft.AspNetCore.Components.CustomElements;

var builder = WebAssemblyHostBuilder.CreateDefault(args);

builder.RootComponents.RegisterCustomElement<Counter>("blazor-counter");

await builder.Build().RunAsync();
```

This creates:

```
<blazor-counter></blazor-counter>
```

---

4. Adding Blazor to an Angular Site

Architecture

Angular app hosts:

- Blazor WASM runtime
- Blazor custom elements

Step-by-Step

1. Build Blazor Project

dotnet publish -c Release

2. Copy Output

Copy /wwwroot files into Angular assets/blazor/

3. Load Blazor Runtime (index.html)

<script src="assets/blazor/_framework/blazor.webassembly.js"></script>

4. Use Blazor Component

<blazor-counter></blazor-counter>

Result

Angular renders a **Blazor Razor component natively**.

No Angular bridge required.

---

5. Adding Blazor to a React Site

React Integration Pattern

React treats Blazor components as **native HTML elements**.

Load Blazor Runtime

public/index.html

<script src="/blazor/_framework/blazor.webassembly.js"></script>

Use Inside JSX

```
function App() {
  return (
    <div>
      <h2>React App</h2>
      <blazor-counter></blazor-counter>
    </div>
  );
}
```

Data Passing (Attributes)

<blazor-counter start="10"></blazor-counter>

[Parameter] public int Start { get; set; }

---

6. Adding Blazor to MVC / Razor Pages

Why This Is Powerful

- Modern UI inside **legacy MVC apps**
- Gradual migration strategy

Steps

1. Reference Blazor Assets

_Layout.cshtml

<script src="~/blazor/_framework/blazor.webassembly.js"></script>

2. Use in Razor View

<blazor-counter></blazor-counter>

Use Case

- Legacy MVC app
- New features built in Blazor
- Zero rewrite

---

7. Adding Web Components to a Blazor Site

Blazor can **consume external web components** (not only export them).

Example: Using a JS Web Component in Blazor

JS Component

<script>

class FancyBox extends HTMLElement {

  connectedCallback() {

```
    this.innerHTML = `<div style="border:2px solid blue">Fancy Box</div>`;
  }
}
customElements.define('fancy-box', FancyBox);
</script>
```

Blazor Usage

```
<fancy-box></fancy-box>
```

Blazor treats it as **valid HTML**.

---

9. When Should You Use Blazor Web Components?

Scenario	**Recommended**
Mixed tech stacks	Yes
Gradual modernization	Yes
Reusable enterprise UI	Yes
Pure Blazor app	Optional

Key Takeaways

- Blazor Web Components are **first-class Custom Elements**
- Framework-agnostic integration
- Ideal for **enterprise hybrid architectures**
- Enables long-term UI reuse strategy backed by **Microsoft**