Data Binding in Blazor

---

1. What Is Data Binding in Blazor?

Data Binding is the mechanism that connects **UI elements (Razor components)** with **C# data (fields, properties, or models)** so that:

- Data flows **from the component to the UI**
- User interactions flow **from the UI back to the component**

Blazor uses **component-based rendering**, and data binding ensures the UI automatically re-renders when state changes.

---

2. Why Data Binding Is Important

In Blazor applications, data binding enables:

- Dynamic UI updates without manual DOM manipulation
- Clean separation of UI and business logic
- Reactive user interfaces
- Simplified form handling and validation

Without data binding, every UI update would require explicit JavaScript or manual refresh logic.

---

3. Types of Data Binding in Blazor

Blazor primarily supports the following binding styles:

1. **One-Way Data Binding**
2. **Two-Way Data Binding**
3. **Event-Based (Manual) Binding**
4. **Parameter Binding (Parent → Child)**

---

4. One-Way Data Binding

Concept

Data flows **only from the component (C#) to the UI**.

Example

```
<h3>Hello, @UserName</h3>


@code {
   string UserName = "Blazor Learner";
}
```

Key Points

- UI reflects the value of UserName
- Changes in UI do **not** update UserName
- Re-render happens automatically when UserName changes in code

---

5. Two-Way Data Binding (@bind)

Concept

Data flows **both ways**:

- Component → UI
- UI → Component

This is widely used in **forms and inputs**.

```
@page "/binding"

<h1>@Title</h1>

<p>
    <span>Title changes when the input box loses focus (onChange).</span>
    <input type="text" @bind="Title" />
</p>

<p>
    <span>Title changes when the input box receives input (onInput).</span>
    <input type="text" @bind="Title" @bind:event="oninput" />
</p>

@code {
    private string Title = "Data Binding gives you wings!";
}
```

Example

<input @bind="Name" />

You entered: @Name

@code {

string Name = "";

}

How @bind Works Internally

@bind is syntactic sugar for:

- value
- onchange (or another event)

Equivalent expanded form:

```
<input value="@Name"
    @onchange="e => Name = e.Value.ToString()" />
```

---

6. Binding with Different Input Controls

Control Type	Binding Example
TextBox	<input @bind="Text" />
Checkbox	<input type="checkbox" @bind="IsActive" />
Select	<select @bind="SelectedValue">
Number	<input type="number" @bind="Age" />

Numeric Binding Example

```
<input type="number" @bind="Age" />


@code {
  int Age;
}
```

Blazor automatically handles type conversion.

---

7. Event-Based (Manual) Data Binding

Concept

You manually handle events and update state.

Example

```
<input @oninput="OnInputChanged" />

<p>@Message</p>


@code {
   string Message;


   void OnInputChanged(ChangeEventArgs e)
   {
      Message = e.Value?.ToString();
   }
}
```

When to Use

- Custom validation logic
- High-frequency events (oninput)

---

8. Binding with Component Parameters (Parent → Child)

Parent Component

```
<ChildComponent Title="Dashboard" />
```

Child Component

```
<h3>@Title</h3>


@code {
   [Parameter]
   public string Title { get; set; }
}
```

Key Characteristics

- One-way binding by default
- Parent controls the data
- Child receives data via [Parameter]

---

9. Two-Way Binding Between Components

Blazor supports two-way binding between components using:

- [Parameter]
- EventCallback<T>

Child Component

```
<input @bind="Value" />


@code {
   [Parameter] public string Value { get; set; }
   [Parameter] public EventCallback<string> ValueChanged { get; set; }
}
```

Parent Component

```
<ChildComponent @bind-Value="UserName" />
```

---

10. Data Binding with Models (Forms)

Model Binding Example

```
<EditForm Model="@user">
   <InputText @bind-Value="user.Name" />
   <InputNumber @bind-Value="user.Age" />
</EditForm>
```

```
@code {
    User user = new();


    class User
    {
        public string Name { get; set; }
        public int Age { get; set; }
    }
}
```

Benefits

- Strong typing
- Validation support
- Clean form structure

---

11. Rendering and State Change

Blazor automatically re-renders a component when:

- A bound value changes
- An event handler completes
- StateHasChanged() is called explicitly

This makes data binding **reactive by design**.
