Developing Cross-Platform Web Apps With Blazor

Wael Kdouh - @waelkdouh

Senior Customer Engineer

v1.0

How to View This Presentation

- To switch to **Notes Page** view:
  - On the ribbon, click the **View** tab, and then click **Notes Page**

- To navigate through notes, use the Page Up and Page Down keys
  - Zoom in or zoom out, if required

- In the **Notes Page** view, you can:
  - Read any supporting text—now or after the delivery
  - Add notes to your copy of the presentation, if required

- Take the presentation files home with you

Module 4: Event Handling and Data Binding

Module Overview

Module 4: Event Handling and Data Binding

Section 1: Data Binding

Lesson: Overview

Life Without Data Binding

- Imagine any application that needs to display data to the user and capture changes made by that user to save the modified data

- One way you could build an application like this is to, once you have the data, iterate over each item of data

- Later, after the user made some changes, you would iterate over your generated elements, and for every one you would inspect the child elements if their data was changed. If so, you would copy the data back into the objects used for saving that data

Life Without Data Binding

- This is an **error-prone process**, and a **lot of work** if you want to do this with something like jQuery (jQuery is a very popular JavaScript framework that allows you to manipulate the browser’s Document Object Model (DOM))

Life With Data Binding

- Modern frameworks like Angular and React have become popular because they simplify this process greatly through data binding
- With data binding most of this work for generating the UI and copying data back into objects is done by the framework

A Quick Look at Razor

- Blazor = Browser + Razor

- So, to understand Blazor you need to understand browsers and Razor

Razor

- Razor is a markup syntax that allows you to embed code in a web page
- In ASP.NET Core MVC the code is executed at the server-side to generate HTML that is sent to the browser
- But in Blazor this code is executed inside your browser (assuming Blazor WebAssembly) and will dynamically update the web page without having to go back to the server

Razor

SurveyPrompt.Razor

```
<div class="alert alert-secondary mt-4" role="alert">
    <span class="oi oi-pencil mr-2" aria-hidden="true"></span>
    <strong>@Title</strong>

    <span class="text-nowrap">
        Please take our
        <a target="_blank" class="font-weight-bold" href="https://bing.com">brief survey</a>
    </span>
    and tell us what you think.
</div>

@code {
    // Demonstrates how a parent component can supply parameters
    [Parameter]
    public string Title { get; set; }
}
```

As you can see, Razor mainly consists of HTML markup

If you want to have some C# properties or methods, you can embed them in the @code section of a Razor file

This works because the Razor file is used to generate a .NET class and everything in @code is embedded in that class

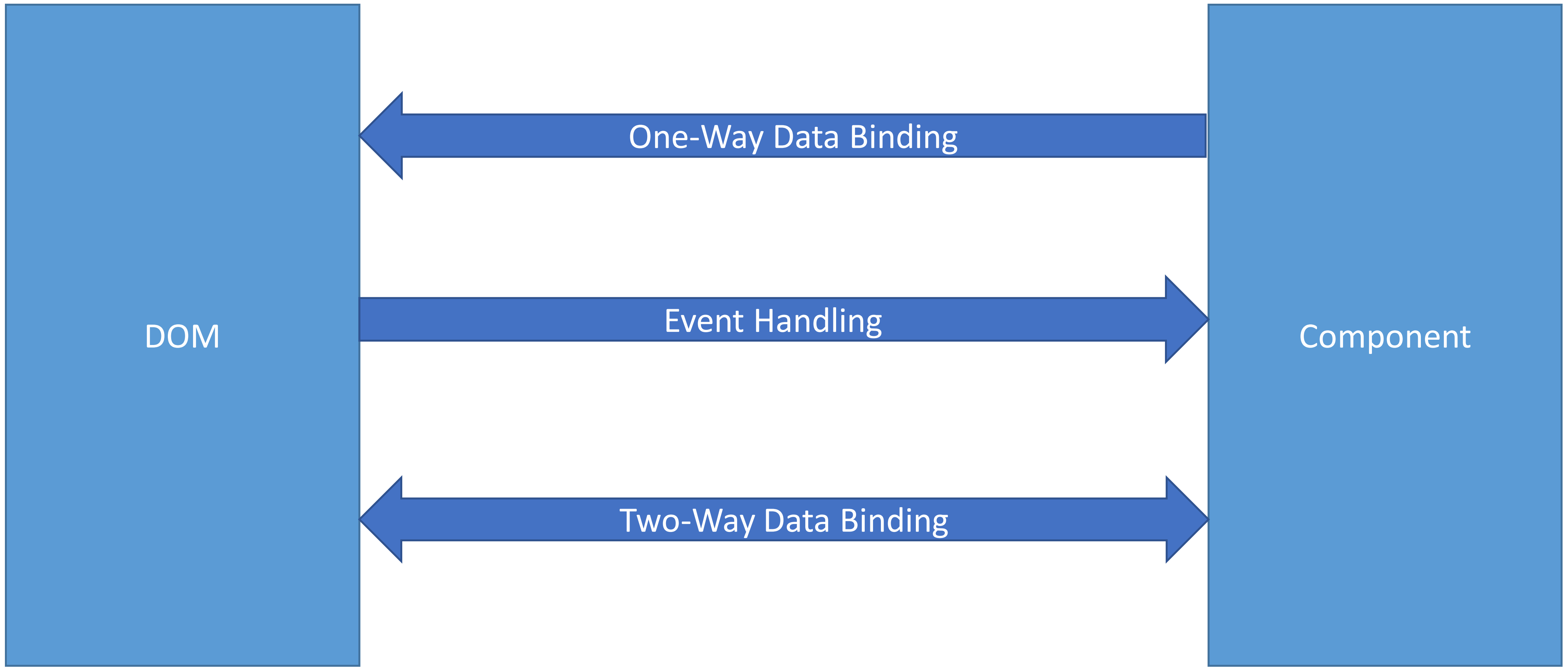
Razor

SurveyPrompt.Razor

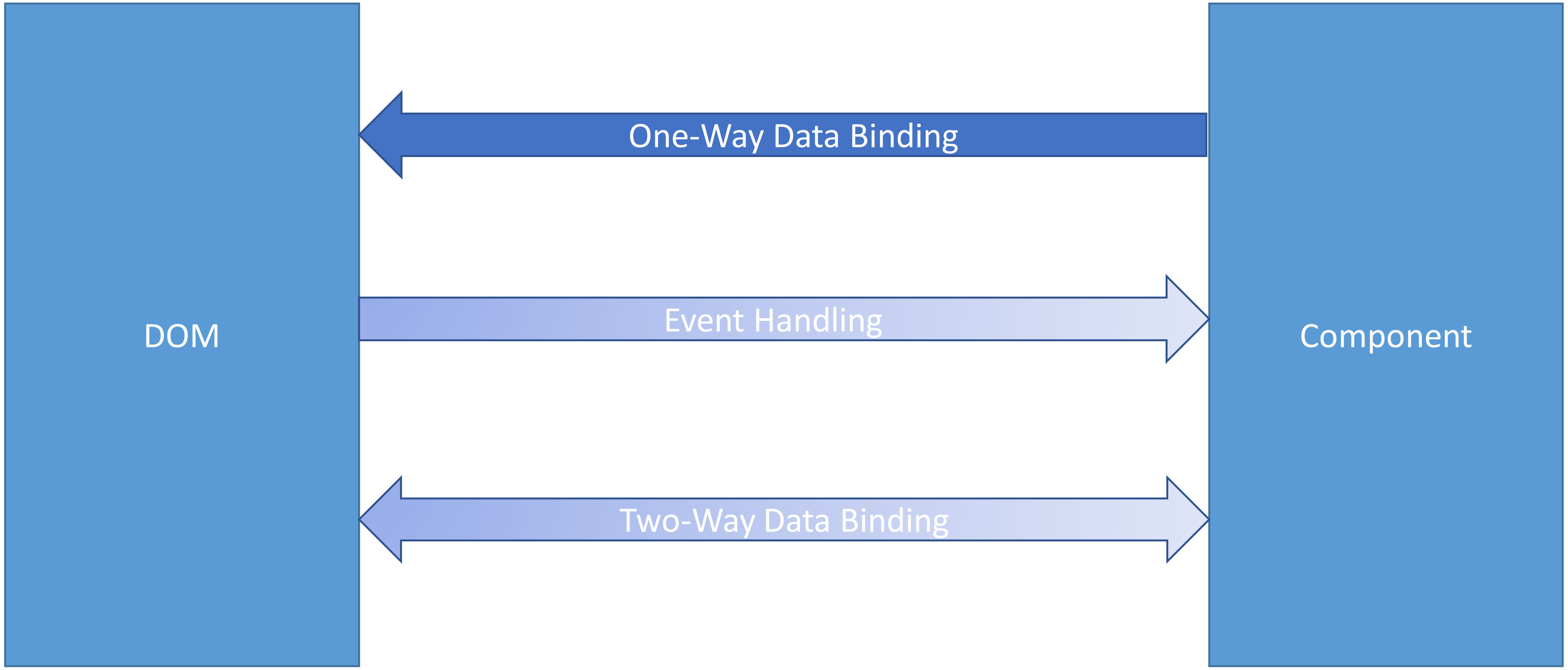
```
<div class="alert alert-secondary mt-4" role="alert">
    <span class="oi oi-pencil mr-2" aria-hidden="true"></span>
    <strong>@Title</strong>

    <span class="text-nowrap">
        Please take our
        <a target="_blank" class="font-weight-bold" href="https://bing.com">brief survey</a>
    </span>
    and tell us what you think.
</div>

@code {
    // Demonstrates how a parent component can supply parameters
    [Parameter]
    public string Title { get; set; }
}
```

SurveyPrompt can embed the contents of the Title property in its HTML markup using the @ syntax. This syntax tells Razor to switch to C#, and this will get the property and embed its value in the markup

Data Binding



Data Binding



Module 4: Event Handling and Data Binding

Section 1: Data Binding

Lesson: One-Way Data Binding

One-Way Data Binding

- One-way data binding is where data flows from the component to the DOM, or vice versa, but only in one direction

- Data binding from the component to the DOM is where some data, like the counter value, needs to be displayed

- Data binding from the DOM to the component is where some DOM event took place, like the user clicking a button, and you want some code to run

One-Way Data Binding Syntax

```
@page "/counter"

<h1>Counter</h1>

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

<button class="btn btn-primary" @onclick="IncrementCount">Click me</button>

@code {
    private int currentCount = 0;

    private void IncrementCount()
    {
        currentCount++;
    }
}
```

Data Binding from the Component to the DOM

Conditional Attributes

- Sometimes you can control the browser by adding some attributes to DOM elements

- For example, to disable a button you can simply use the disabled attribute
  <button disabled>Click me</button>

- With Blazor you can data-bind an attribute to a Boolean expression (e.g. a property or method of type bool) and Blazor will hide the attribute if the expression evaluates to false (or null) and will show the attribute if it evaluates to true

```
<button @onclick="IncrementCount“
        disabled="@(currentCount >= 10)">Click me</button>
```

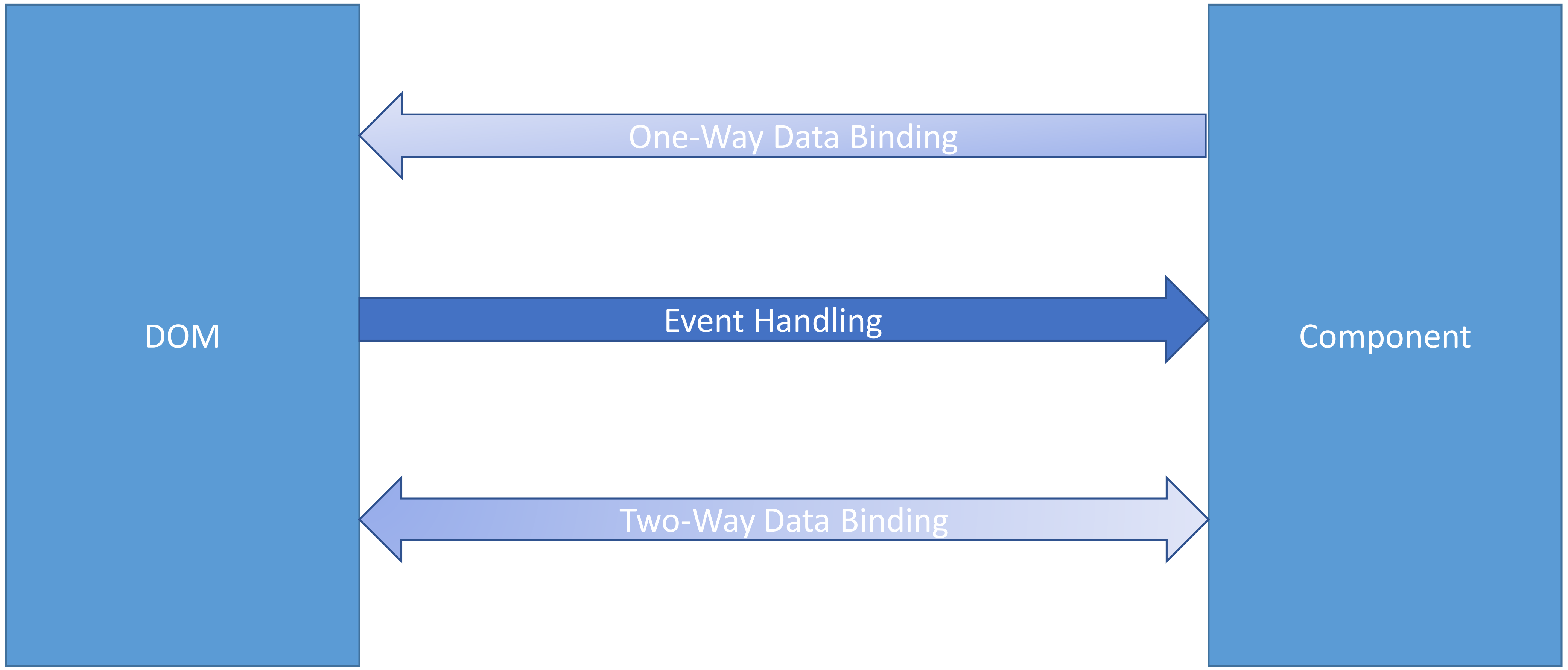
Demo: One-Way Data Binding

Module 4: Event Handling and Data Binding

Section 2: Event Handling

Lesson: Overview

Event Handling



Event Handling

```
@page "/counter"

<h1>Counter</h1>

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

<button class="btn btn-primary" @onclick="IncrementCount">Click me</button>

@code {
    private int currentCount = 0;

    private void IncrementCount()
    {
        currentCount++;
    }
}
```

You update currentCount using the IncrementCount() method

This method gets called by clicking the button. This, again, is a one-way data binding, but in the other direction, from the button to your component

Event Binding Syntax

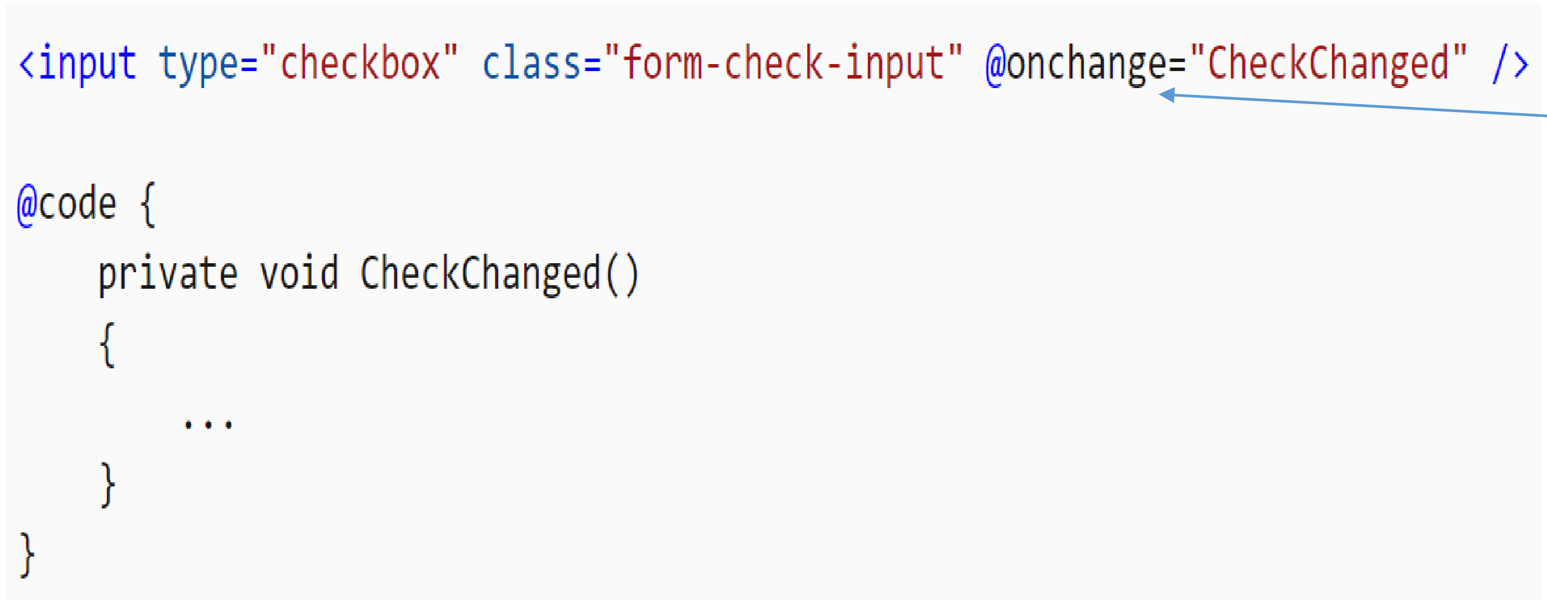
- Blazor components provide event handling features. For an HTML element attribute named **on{*EVENT*}** (for example, onclick and onsubmit) with a delegate-typed value, blazor components treat the attribute's value as an event handler. The attribute's name is always formatted **@on{*EVENT*}**

```
<button class="btn btn-primary" @onclick="UpdateHeading">
    Update heading
</button>

@code {
    private void UpdateHeading(MouseEventArgs e)
    {
        ...
    }
}
```

The following code calls the UpdateHeading method when the button is clicked in the UI

Event Binding Syntax

- Blazor components provide event handling features. For an HTML element attribute named **on{*EVENT*}** (for example, onclick and onsubmit) with a delegate-typed value, blazor components treat the attribute's value as an event handler. The attribute's name is always formatted **@on{*EVENT*}**



```
<input type="checkbox" class="form-check-input" @onchange="CheckChanged" />

@code {
    private void CheckChanged()
    {
        ...
    }
}
```

The following code calls the CheckChanged method when the check box is changed in the UI

Event Binding Syntax

- Event handlers can also be asynchronous and return a Task
- There's no need to manually call StateHasChanged
- Exceptions are logged when they occur

Event Binding Syntax

```
<button class="btn btn-primary" @onclick="UpdateHeading">
    Update heading
</button>

@code {
    private async Task UpdateHeading(MouseEventArgs e)
    {
        ...
    }
}
```

UpdateHeading is called asynchronously when the button is selected

Event Argument Types

Event	Class	DOM events and notes
Clipboard	`ClipboardEventArgs`	`oncut`, `oncopy`, `onpaste`
Drag	`DragEventArgs`	`ondrag`, `ondragstart`, `ondragenter`, `ondragleave`, `ondragover`, `ondrop`, `ondragend` `DataTransfer` and `DataTransferItem` hold dragged item data.
Error	`ErrorEventArgs`	`onerror`

Event	`EventArgs`	*General*
		`onactivate`, `onbeforeactivate`, `onbeforedeactivate`, `ondeactivate`, `onended`, `onfullscreenchange`, `onfullscreenerror`, `onloadeddata`, `onloadedmetadata`, `onpointerlockchange`, `onpointerlockerror`, `onreadystatechange`, `onscroll`
		
		Clipboard
		`onbeforecut`, `onbeforecopy`, `onbeforepaste`
		
		Input
		`oninvalid`, `onreset`, `onselect`, `onselectionchange`, `onselectstart`, `onsubmit`
		
		Media
		`oncanplay`, `oncanplaythrough`, `oncuechange`, `ondurationchange`, `onemptied`, `onpause`, `onplay`, `onplaying`, `onratechange`, `onseeked`, `onseeking`, `onstalled`, `onstop`, `onsuspend`, `ontimeupdate`, `onvolumechange`, `onwaiting`

Event Argument Types

Event	Class	DOM events and notes
Focus	`FocusEventArgs`	`onfocus`, `onblur`, `onfocusin`, `onfocusout` Doesn't include support for `relatedTarget`.
Input	`ChangeEventArgs`	`onchange`, `oninput`
Keyboard	`KeyboardEventArgs`	`onkeydown`, `onkeypress`, `onkeyup`
Mouse	`MouseEventArgs`	`onclick`, `oncontextmenu`, `ondblclick`, `onmousedown`, `onmouseup`, `onmouseover`, `onmousemove`, `onmouseout`
Mouse pointer	`PointerEventArgs`	`onpointerdown`, `onpointerup`, `onpointercancel`, `onpointermove`, `onpointerover`, `onpointerout`, `onpointerenter`, `onpointerleave`, `ongotpointercapture`, `onlostpointercapture`
Mouse wheel	`WheelEventArgs`	`onwheel`, `onmousewheel`
Progress	`ProgressEventArgs`	`onabort`, `onload`, `onloadend`, `onloadstart`, `onprogress`, `ontimeout`

Touch	`TouchEventArgs`	`ontouchstart`, `ontouchend`, `ontouchmove`, `ontouchenter`, `ontouchleave`, `ontouchcancel`
		
		`onstop`, `onsuspend`, `ontimeupdate`, `onvolumechange`, `onwaiting`

Using C# Lambda Expressions

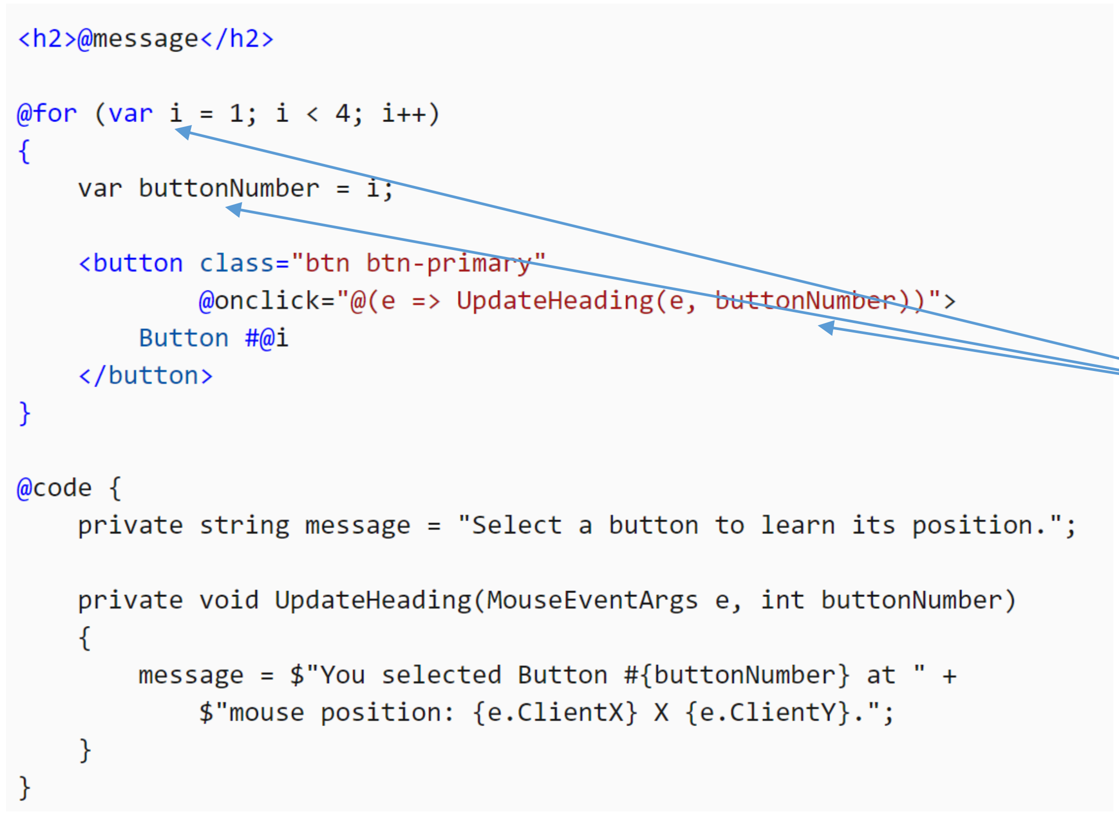
- Lambda expressions can also be used:

```
<button @onclick="@(e => Console.WriteLine("Hello, world!"))">Say hello</button>
```

Using C# Lambda Expressions

- It's often convenient to pass additional values, such as when iterating over a set of elements

- The following example creates three buttons, each of which calls UpdateHeading passing an event argument (MouseEventArgs) and its button number (buttonNumber) when selected in the UI:



```
<h2>@message</h2>

@for (var i = 1; i < 4; i++)
{
    var buttonNumber = i;

    <button class="btn btn-primary"
            @onclick="@(e => UpdateHeading(e, buttonNumber))">
        Button #@i
    </button>
}

@code {
    private string message = "Select a button to learn its position.";

    private void UpdateHeading(MouseEventArgs e, int buttonNumber)
    {
        message = $"You selected Button #{buttonNumber} at " +
            $"mouse position: {e.ClientX} X {e.ClientY}.";
    }
}
```

Do not use the loop variable (i) in a for loop directly in a lambda expression. Otherwise the same variable is used by all lambda expressions causing i's value to be the same in all lambdas

Always capture its value in a local variable (buttonNumber in this example) and then use it

EventCallback

- A common scenario with nested components is the desire to run a parent component's method when a child component event occurs, for example, when an onclick event occurs in the child

- To expose events across components, use an EventCallback

- A parent component can assign a callback method to a child component's EventCallback

EventCallback

Parent Component

```
@page "/ParentComponent"

<h1>Parent-child example</h1>

<ChildComponent Title="Panel Title from Parent"
                OnClick="@ShowMessage">
    Content of the child component is supplied
    by the parent component.
</ChildComponent>

<p><b>@messageText</b></p>

@code {
    private string messageText;

    private void ShowMessage(MouseEventArgs e)
    {
        messageText = $"Blaze a new trail with Blazor! ({e.ScreenX}, {e.ScreenY})";
    }
}
```

Child Component

```
<div class="panel panel-default">
    <div class="panel-heading">@Title</div>
    <div class="panel-body">@ChildContent</div>

    <button class="btn btn-primary" @onclick="OnClick">
        Trigger a Parent component method
    </button>
</div>

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

    [Parameter]
    public RenderFragment ChildContent { get; set; }

    [Parameter]
    public EventCallback<MouseEventArgs> OnClick { get; set; }
}
```

Button's onclick handler is set up to receive an EventCallback delegate from the ParentComponent

The EventCallback is typed with MouseEventArgs, which is appropriate for an onclick event from a peripheral device

EventCallback

- Prefer the strongly typed EventCallback<T> over EventCallback
- EventCallback<T> provides better error feedback to users of the component
- Similar to other UI event handlers, specifying the event parameter is optional
- Use EventCallback when there's no value passed to the callback

Demo: Event Handling

Prevent Default Actions

- Use the **@on{EVENT}:preventDefault** directive attribute to prevent the default action for an event

- When a key is selected on an input device and the element focus is on a text box, a browser normally displays the key's character in the text box. In the following example, the default behavior is prevented by specifying the **@onkeypress:preventDefault directive attribute**. The counter increments, and the + key isn't captured into the <input> element's value:

```
<input value="@_count" @onkeypress="KeyHandler" @onkeypress:preventDefault />

@code {
    private int _count = 0;

    private void KeyHandler(KeyboardEventArgs e)
    {
        if (e.Key == "+")
        {
            _count++;
        }
    }
}
```

Specifying the @on{EVENT}:preventDefault attribute without a value is equivalent to @on{EVENT}:preventDefault="true"

Prevent Default Actions

- Use the **@on{EVENT}:preventDefault** directive attribute to prevent the default action for an event

- When a key is selected on an input device and the element focus is on a text box, a browser normally displays the key's character in the text box. In the following example, the default behavior is prevented by specifying the **@onkeypress:preventDefault <u>directive</u> attribute**. The counter increments, and the + key isn't captured into the <input> element's value:

```
<input @onkeypress:preventDefault="_shouldPreventDefault" />

@code {
    private int _count = 0;

    private void KeyHandler(KeyboardEventArgs e)
    {
        if (e.Key == "+")
        {
            _count++;
        }
    }
}
```

The value of the attribute can also be an expression. In the following example, _shouldPreventDefault is a bool field set to either true or false

Prevent Default Actions

- Use the **@on{EVENT}:preventDefault** directive attribute to prevent the default action for an event

- When a key is selected on an input device and the element focus is on a text box, a browser normally displays the key's character in the text box. In the following example, the default behavior is prevented by specifying the **@onkeypress:preventDefault <u>directive</u> attribute**. The counter increments, and the + key isn't captured into the <input> element's value:

```
<input value="@_count" @onkeypress="KeyHandler" @onkeypress:preventDefault />

@code {
    private int _count = 0;

    private void KeyHandler(KeyboardEventArgs e)
    {
        if (e.Key == "+")
        {
            _count++;
        }
    }
}
```

An event handler isn't required to prevent the default action. The event handler and prevent default action scenarios can be used independently

Demo: Prevent Default Actions

Stop Event Propagation

- Use the **@on{EVENT}:stopPropagation** directive attribute to stop event propagation

- In the following example, selecting the check box prevents click events from the second child <div> from propagating to the parent <div>:

```
<label>
    <input @bind="_stopPropagation" type="checkbox" />
    Stop Propagation
</label>

<div @onclick="OnSelectParentDiv">
    <h3>Parent div</h3>

    <div @onclick="OnSelectChildDiv">
        Child div that doesn't stop propagation when selected.
    </div>

    <div @onclick="OnSelectChildDiv" @onclick:stopPropagation="_stopPropagation">
        Child div that stops propagation when selected.
    </div>
</div>

@code {
    private bool _stopPropagation = false;

    private void OnSelectParentDiv() =>
        Console.WriteLine($"The parent div was selected. {DateTime.Now}");
    private void OnSelectChildDiv() =>
        Console.WriteLine($"A child div was selected. {DateTime.Now}");
}
```

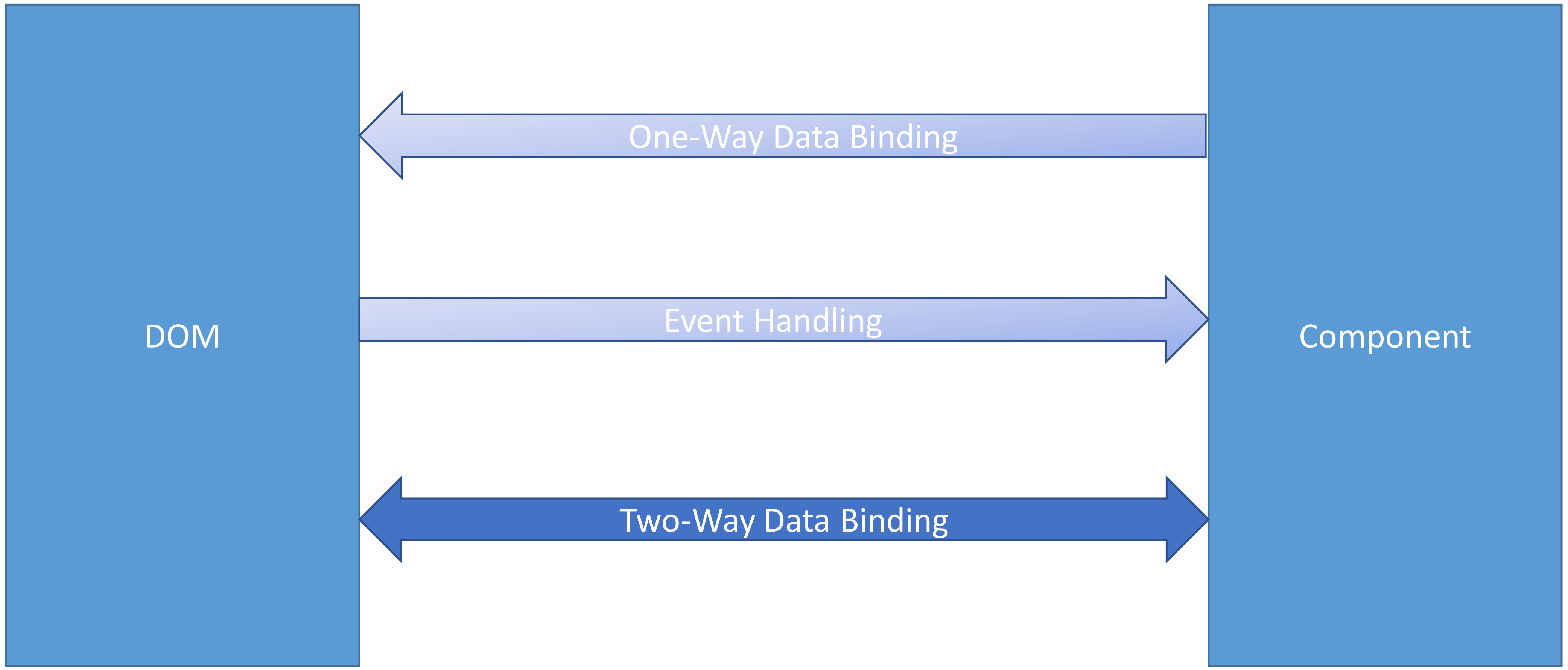
Demo: Stop Event propagation

Module 4: Event Handling and Data Binding

Section 3: Data Binding

Lesson: Two-Way Data Binding

Data Binding



Two-Way Data Binding

- Two-Way data binding (binding to both components and DOM elements) is accomplished with the @bind attribute

- The following example binds a CurrentValue property to the text box's value:

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

@code {
    private string CurrentValue { get; set; }
}
```

When the text box loses focus, the property's value is updated

Two-Way Data Binding

- The text box **is updated in the UI only when the component is rendered, not in response to changing the property's value**

- Since components render themselves after event handler code executes, property updates are usually reflected in the UI immediately after an event handler is triggered

- **Using @bind with the CurrentValue property (<input @bind="CurrentValue" />) is essentially equivalent to the following**:

```
<input value="@CurrentValue"
    @onchange="@((ChangeEventArgs __e) => CurrentValue =
        __e.Value.ToString())" />

@code {
    private string CurrentValue { get; set; }
}
```

Two-Way Data Binding

- When the component is rendered, the value of the input element comes from the CurrentValue property

- When the user types in the text box and changes element focus, the onchange event is fired and the CurrentValue property is set to the changed value

- **In reality**, the code generation is more complex because @bind handles cases where type conversions are performed. **In principle**, @bind associates the current value of an expression with a value attribute and handles changes using the registered handler

Two-Way Data Binding

- In addition to handling onchange events with @bind syntax, **a property or field can be bound using other events by specifying an @bind-value attribute with an event parameter (@bind-value:event)**. The following example binds the CurrentValue property for the oninput event:

```
<input @bind-value="CurrentValue" @bind-value:event="oninput" />

@code {
    private string CurrentValue { get; set; }
}
```

- **Unlike onchange, which fires when the element loses focus, oninput fires when the value of the text box changes**

Demo: Two-Way Data Binding

Two-Way Data Binding – Component Parameters

- Binding recognizes component parameters, where **@bind-{property}** can bind a property value across components

The Child Component has a Year component parameter and YearChanged callback

```
<h2>Child Component</h2>

<p>Year: @Year</p>

@code {
    [Parameter]
    public int Year { get; set; }

    [Parameter]
    public EventCallback<int> YearChanged { get; set; }
}
```

The parent component uses ChildComponent and binds the ParentYear parameter from the parent to the Year parameter on the child component

```
@page "/ParentComponent"

<h1>Parent Component</h1>

<p>ParentYear: @ParentYear</p>

<ChildComponent @bind-Year="ParentYear" />

<button class="btn btn-primary" @onclick="ChangeTheYear">
    Change Year to 1986
</button>

@code {
    [Parameter]
    public int ParentYear { get; set; } = 1978;

    private void ChangeTheYear()
    {
        ParentYear = 1986;
    }
}
```

Two-Way Data Binding – Component Parameters

- If the value of the ParentYear property is changed by selecting the button in the ParentComponent, the Year property of the ChildComponent is updated. The new value of Year is rendered in the UI when the ParentComponent is rerendered

```
<h2>Child Component</h2>

<p>Year: @Year</p>

@code {
    [Parameter]
    public int Year { get; set; }

    [Parameter]
    public EventCallback<int> YearChanged { get; set; }
}
```

```
@page "/ParentComponent"

<h1>Parent Component</h1>

<p>ParentYear: @ParentYear</p>

<ChildComponent @bind-Year="ParentYear" />

<button class="btn btn-primary" @onclick="ChangeTheYear">
    Change Year to 1986
</button>

@code {
    [Parameter]
    public int ParentYear { get; set; } = 1978;

    private void ChangeTheYear()
    {
        ParentYear = 1986;
    }
}
```

Two-Way Data Binding – Component Parameters

- The Year parameter is bindable because it has a companion YearChanged event that matches the type of the Year parameter.

- By convention, <ChildComponent @bind-Year="ParentYear" /> is essentially equivalent to writing:

```
<ChildComponent @bind-Year="ParentYear" @bind-Year:event="YearChanged" />
```

```
<h2>Child Component</h2>

<p>Year: @Year</p>

@code {
    [Parameter]
    public int Year { get; set; }

    [Parameter]
    public EventCallback<int> YearChanged { get; set; }
}
```

```
@page "/ParentComponent"

<h1>Parent Component</h1>

<p>ParentYear: @ParentYear</p>

<ChildComponent @bind-Year="ParentYear" />

<button class="btn btn-primary" @onclick="ChangeTheYear">
    Change Year to 1986
</button>

@code {
    [Parameter]
    public int ParentYear { get; set; } = 1978;

    private void ChangeTheYear()
    {
        ParentYear = 1986;
    }
}
```

Two-Way Data Binding – Component Parameters

- In general, a property can be bound to a corresponding event handler using @bind-property:event attribute. For example, the property MyProp can be bound to MyEventHandler using the following two attributes:

```
<MyComponent @bind-MyProp="MyValue" @bind-MyProp:event="MyEventHandler" />
```

```
<h2>Child Component</h2>

<p>Year: @Year</p>

@code {
    [Parameter]
    public int Year { get; set; }

    [Parameter]
    public EventCallback<int> YearChanged { get; set; }
}
```

```
@page "/ParentComponent"

<h1>Parent Component</h1>

<p>ParentYear: @ParentYear</p>

<ChildComponent @bind-Year="ParentYear" />

<button class="btn btn-primary" @onclick="ChangeTheYear">
    Change Year to 1986
</button>

@code {
    [Parameter]
    public int ParentYear { get; set; } = 1978;

    private void ChangeTheYear()
    {
        ParentYear = 1986;
    }
}
```

Demo: Two-Way Data Binding – Component Parameters

Module Summary

- In this module, you learned about:
  - One-Way Data Binding
  - Event Handling
  - Two-Way Data Binding

Lab 4: Event Handling and Binding

References

- Microsoft Docs
- Blazor University

Microsoft