

Q

Live weekly coding

- > Introduction
- > Layouts
- Components

Creating a component
One-way binding

- Literals, expressions, and directives
- Component events

#### **Browser DOM events**

- Two-way bindingBinding directives
- Cascading values

Cascading values by name Cascading values by type Overriding cascaded values

Code generated HTML attributes
Capturing unexpected parameters
Replacing attributes on child components

Component lifecycles

- Multi-threaded rendering
  - Thread safety using InvokeAsync
- > Render trees
- > Templating components with RenderFragments
- > Routing
- ✓ Forms

Editing form data

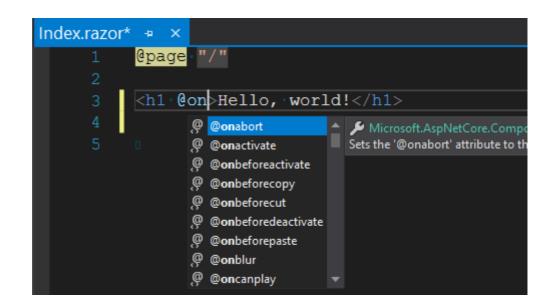
## **Browser DOM events**



When rendering any mark-up, it is possible to assign standard JavaScript events on the rendered HTML elements so that our own Blazor C# methods are executed. For example, we have used the <code>@onclick Directive</code> in many samples elsewhere:

<button @onclick=ButtonClicked>Click me</button>

These event directives have full IntelliSense support within the Visual Studio editor, so starting to type the @ symbol should present us with a comprehensive list of available directives, along with a description identifying which argument class type the event passes us in our event handler. DOM events start with @on.



"Sets the '@onabort' attribute to the provided string or delegate value.

A delegate value should be of type

 $\verb|'Microsoft.AspNetCore.Components.Web.ProgressEventArgs|''$ 

**Warning:** When writing a Blazor app that runs entirely on the server, Blazor will hook events in the browser and send them to server so our C#

Descending from InputBase<T>
Validation
Handling form submission
EditContext, FieldIdentifiers, and
FieldState
Accessing form state
Writing custom validation

Component libraries

- > JavaScript interop
- > Dependency injection

Descending from InputBase<T> methods can be invoked. This can lead to a noticeable slow-down for Validation frequently fired events such as onmousemove.

**Note:** Because JavaScript invocation of C# methods is asynchronous, this means that in C# methods we cannot cancel events as we can in JavaScript. This is because cancelling browser DOM events is a synchronous operation, by the time our C# has been asynchronously invoked it is already too late to cancel the event.

Available DOM events and their parameter types include:

#### **General events**

DOM event	Argument type
onactivate	EventArgs
onbeforeactivate	EventArgs
on <i>b</i> efore <i>d</i> eactivate	EventArgs
on <i>d</i> eactivate	EventArgs
onended	EventArgs
onfullscreenchange	EventArgs
onfullscreen <i>e</i> rror	EventArgs
onloaded <i>d</i> ata	EventArgs
onloaded <i>m</i> etadata	EventArgs
onpointerlockchange	EventArgs
onpointerlockerror	EventArgs
on ready state change	EventArgs
onscroll	EventArgs

#### **Focus events**

5014	
DOM event	Argument type

onfocus	FocusEventArgs
on <i>b</i> lur	FocusEventArgs
onfocusin	FocusEventArgs
onfocusout	FocusEventArgs

### **Mouse events**

DOM event	Argument type
on <i>m</i> ouseover	MouseEventArgs
on <i>m</i> ouseout	MouseEventArgs
on <i>m</i> ouse <i>m</i> ove	MouseEventArgs
on <i>m</i> ouse <i>d</i> own	MouseEventArgs
on <i>m</i> ouse <i>u</i> p	MouseEventArgs
onclick	MouseEventArgs
on <i>d</i> blclick	MouseEventArgs
oncontextmenu	MouseEventArgs
onwheel	WheelEventArgs
on <i>m</i> ousewheel	WheelEventArgs

## **Drag events**

DOM event	Argument type
on <i>d</i> rag	DragEventArgs
on <i>d</i> ragend	DragEventArgs
on <i>d</i> ragenter	DragEventArgs
on <i>d</i> rag <i>l</i> eave	DragEventArgs
on <i>d</i> rag <i>o</i> ver	DragEventArgs
on <i>d</i> ragstart	DragEventArgs

on <i>d</i> rop	DragEventArgs	

# **Keyboard events**

DOM event	Argument type
on <i>k</i> ey <i>d</i> own	KeyboardEventArgs
onkeyup	KeyboardEventArgs
onkeypress	KeyboardEventArgs

# Input events

DOM event	Argument type
onchange	ChangeEventArgs
on <i>i</i> nput	ChangeEventArgs
on <i>i</i> nvalid	EventArgs
on <i>r</i> eset	EventArgs
onselect	EventArgs
onselectstart	EventArgs
onselection change	EventArgs
onsubmit	EventArgs

# **Clipboard events**

DOM event	Argument type
on <i>b</i> efore <i>c</i> opy	EventArgs
on <i>b</i> efore <i>c</i> ut	EventArgs
on before paste	EventArgs
oncopy	ClipboardEventArgs

Onnaste ClinhoardEventArgs	oncut	ClipboardEventArgs
onpuste cirpbon devental gs	on <i>p</i> aste	ClipboardEventArgs

## **Touch events**

DOM event	Argument type
ontouch <i>c</i> ancel	TouchEventArgs
ontouchend	TouchEventArgs
ontouch <i>m</i> ove	TouchEventArgs
on touch start	TouchEventArgs
ontouchenter	TouchEventArgs
ontouch <i>l</i> eave	TouchEventArgs

### **Pointer events**

DOM event	Argument type
ongotpointercapture	PointerEventArgs
onlostpointercapture	PointerEventArgs
onpointercancel	PointerEventArgs
on <i>p</i> ointer <i>d</i> own	PointerEventArgs
on <i>p</i> ointer <i>e</i> nter	PointerEventArgs
on <i>p</i> ointer <i>l</i> eave	PointerEventArgs
on <i>p</i> ointer <i>m</i> ove	PointerEventArgs
onpointerout	PointerEventArgs
onpointerover	PointerEventArgs
on <i>p</i> ointer <i>u</i> p	PointerEventArgs

## **Media events**

DOM event	Argument type
oncanplay	EventArgs
oncan <i>p</i> lay <i>t</i> hrough	EventArgs
oncuechange	EventArgs
on duration change	EventArgs
onemptied	EventArgs
on <i>p</i> ause	EventArgs
on <i>p</i> lay	EventArgs
on <i>p</i> laying	EventArgs
onratechange	EventArgs
onseeked	EventArgs
onseeking	EventArgs
onstalled	EventArgs
onstop	EventArgs
onsuspend	EventArgs
on <i>t</i> ime <i>u</i> pdate	EventArgs
onvolumechange	EventArgs
onwaiting	EventArgs

## **Progress events**

DOM event	Argument type
onloadstart	ProgressEventArgs
on <i>t</i> imeout	ProgressEventArgs
onabort	ProgressEventArgs
onload	ProgressEventArgs

onloadend	ProgressEventArgs
on <i>p</i> rogress	ProgressEventArgs
onerror	ErrorEventArgs