Blazor Control API Reference

Build your own Blazor control as a Razor class library and place it on an HTML5 or Portable display through the BlazorCtrl element.

A custom Blazor control is a Razor component compiled into a Razor class library (DLL). The FrameworX web client loads that DLL and runs the component in the browser. Tags reach the component's parameters through the display's CodeBehind.

This page covers **building** the control: the projects, the rules FrameworX applies, and a template you can download and adapt.

> To **place and configure** the finished control on a display and bind it to tags, see Blazor Control Reference. For a ready solution that uses this template's control, see Blazor Control Example.

Download the template
Requirements
Rules
Build your own control from the template
1. Open the solution and try the test bench
2. Rename it
3. Add your parameters
4. Build
5. Deploy
6. Place it and bind it to tags
Troubleshooting
Source code
Related
In this section...

Download the template

Download BlazorControl-ProgressBar.zip. It contains one solution with two projects:

Project	Role
`BlazorControlExample`	The control: a Razor class library with one component, `ProgressBar`. It shows a production target, the current production and a bar filled in proportion, with a **Restart Simulation** button that sends 0 back to FrameworX. Its DLL is what FrameworX loads.
`WebUIBlazor`	Optional test bench: a small ASP.NET Core web app that shows the control with fixed values, so you can run and debug it without FrameworX. FrameworX never uses it; do not copy its DLL to FrameworX.

Validated with FrameworX 10.1.5.

Requirements

- Visual Studio with the **ASP.NET and web development** workload, and the .NET 8 SDK or later.
- A Razor class library. The template targets **.NET 8**; the FrameworX 10.1.5 web client runs on .NET 10 and loads it.
- The control runs only in HTML5 clients (a web browser).
- The display can be an **HTML5** display or a **Portable** display. On a Portable display the control appears only at runtime in the browser.
- The DLL is deployed to the installation's `BlazorControls` folder, by default `C:\Program Files\Tatsoft\FrameworX\fx-10\HTML5\BlazorControls\`.

Rules

Follow these rules for every custom Blazor control. Breaking one does not always produce an error: the component can be missing from the list, show default values, or ignore changes.

1. **The component is public.** It must be public, not abstract, derive from `Microsoft.AspNetCore.Components.ComponentBase` (every `.razor` component does), and have a public parameterless constructor. Only such types are listed when you pick the component.
2. **The DLL is in the BlazorControls folder before you select it.** The Designer stores the assembly path relative to `fx-10\HTML5\BlazorControls\` and rejects a DLL selected anywhere else with the message "Blazor Control should be located at ...".
3. **The assembly has its own name.** FrameworX ships a `BlazorControlExample.dll` in `BlazorControls`. Building the template unchanged produces a DLL with the same name, which overwrites it.
4. **Every value FrameworX sets or binds is a public property marked `[Parameter]`.**
5. **A value the control changes and FrameworX must receive back has an `EventCallback<T>` parameter.** Invoke it with the new value when the control changes it. In the CodeBehind, `BindEvent` names that callback in its third argument to make the binding two-way.
6. **The component runs in the browser.** FrameworX runs it in Interactive WebAssembly render mode, so it cannot use server-only APIs. The test bench runs it in Interactive Server mode and does not catch this: always test in FrameworX.
7. **Extra stylesheets and scripts are deployed with the DLL.** The FrameworX web client already loads Bootstrap (`BlazorControls\BlazorBootstrap\bootstrap.min.css`). Any other `.css` or `.js` file your control needs goes into `BlazorControls` and is listed in the element's Dependencies field.
8. **Close the Designer before replacing the DLL.** It keeps the DLL loaded, and the file cannot be overwritten.

Build your own control from the template

1. Open the solution and try the test bench

Open `BlazorControlSolution.sln`. To see the control without FrameworX, set `WebUIBlazor` as the startup project and run it: the browser opens `Home.razor`, which renders the component with fixed values.

WebUIBlazor/Components/Pages/Home.razor

```
@page "/"

@rendermode InteractiveServer

<PageTitle>Home</PageTitle>

<ProgressBar TargetProduction="100" CurrentProduction="50" BarColor="blue" @rendermode="InteractiveServer"><
/ProgressBar>
```

2. Rename it

Choose an assembly name and a namespace for your control (rule 3). The component's full type name is the namespace plus the component name, for example `MyCompany.Controls.ProgressBar`.

File	What to change
`BlazorControlExample.csproj`	Add `<AssemblyName>` and `<RootNamespace>` to the `PropertyGroup`, or rename the project. Without them, both default to the project file name, `BlazorControlExample`.
`WebUIBlazor/Components/_Imports.razor`	Change `@using BlazorControlExample` to your namespace.
`WebUIBlazor/WebUIBlazor.csproj`	Update the `ProjectReference` path if you rename the project folder or file.

3. Add your parameters

Each value FrameworX sets is a public `[Parameter]` property. `CurrentProduction` also goes back to FrameworX: the **Restart Simulation** button sets it to 0 and invokes `CurrentProductionRestarted` with the new value (rule 5).

ProgressBar.razor: parameters and the callback

```
@code {

    // Parameter to receive the target production value from the parent component
    [Parameter]
    public int TargetProduction { get; set; }

    // Parameter to receive the current production value from the parent component
    [Parameter]
    public int CurrentProduction { get; set; }

    // Parameter for setting the color of the progress bar (default is blue)
    [Parameter]
    public string BarColor { get; set; } = "blue";

    // Event callback to notify the parent component when the production is restarted
    [Parameter]
    public EventCallback<int> CurrentProductionRestarted { get; set; }

    // Method to reset the production progress
    private void Restart()
    {
        // Logs a message in the browser console using JavaScript (JSInterop)
        _ = JSRuntime.InvokeVoidAsync("console.log", "Restart command requested!");

        // Reset the current production to zero
        CurrentProduction = 0;

        // Trigger the event callback to inform the parent component
        CurrentProductionRestarted.InvokeAsync(0);
    }
}
```

4. Build

Build the solution in Release. The control's DLL is `BlazorControlExample\bin\Release\net8.0\<AssemblyName>.dll`.

5. Deploy

Close the Designer (rule 8), then copy the DLL, and any `.css` or `.js` file it needs (rule 7), to `C:\Program Files\Tatsoft\FrameworX\fx-10\HTML5\BlazorControls\`.

6. Place it and bind it to tags

On an HTML5 or Portable display, open **Displays / Draw**, select **Viewer**, then **BlazorCtrl(Web)**, choose your DLL from the `BlazorControls` folder and pick the component. Then bind its parameters in the display's CodeBehind, in `BlazorControlLoaded`. Both steps are described in Blazor Control Reference. For the template's component, the parameters map as follows:

Parameter	Type	Direction	`BindEvent` third argument
`TargetProduction`	`int`	Tag to control	`null`
`CurrentProduction`	`int`	Both ways	`"CurrentProductionRestarted"`
`BarColor`	`string` (any CSS color)	Tag to control	`null`

Display CodeBehind: bind the template's parameters to tags

```
public async Task BlazorControlLoaded(TBlazorControl control)
{
    // Runs once for each Blazor control on the display, after it has loaded.
    // Two-way: the third argument names the component's EventCallback that sends changes back to the tag.
    control.BindEvent("CurrentProduction", @Tag.CurrentProduction.GetName(), "CurrentProductionRestarted");
    // One-way (third argument null): the tag only sends values to the control.
    control.BindEvent("TargetProduction", @Tag.TargetProduction.GetName(), null);
    control.BindEvent("BarColor", @Tag.BarColor.GetName(), null);
}
```

The tags in this example are `TargetProduction` and `CurrentProduction` (Integer) and `BarColor` (Text).

Troubleshooting

Symptom	Cause	Fix
Message: "Blazor Control should be located at ...".	The DLL was selected from a folder other than `BlazorControls`.	Copy it to `fx-10\HTML5\BlazorControls\` and select it there (rule 2).
Your component is not in the list.	It is not public, is abstract, has no public parameterless constructor, or its `.razor` file is excluded from the build.	Check rule 1, and that the file is compiled into the DLL.
The control shows only its default values.	No `BindEvent` call for those parameters.	Bind them in `BlazorControlLoaded` (step 6).
A change made in the control does not reach the tag.	The binding is one-way, the callback name is wrong, or the component does not invoke its `EventCallback`.	Pass the callback name as the third argument of `BindEvent` and invoke it in the component (rule 5).
The control is visible in the browser but not in the Windows client.	Blazor controls run only in HTML5 clients.	Expected. Open the display in a web browser.
Styles are missing.	A stylesheet or script is not in `BlazorControls`, or not listed in Dependencies.	Deploy it and list it (rule 7).
Copying the DLL fails: "being used by another process".	The Designer has the DLL loaded.	Close the Designer and copy again (rule 8).
It works in the test bench but not in FrameworX.	The test bench runs the component on the server; FrameworX runs it in the browser.	Remove server-only code (rule 6).

Source code

The complete source of this template is in `BlazorControl-ProgressBar.zip`, available under **Download the template** at the top of this page. That zip is the single, authoritative copy of the code. The code on this page is excerpts only, to explain how the template works.

Related

- Blazor Control Reference: place the BlazorCtrl element on a display and bind its parameters in the CodeBehind.
- Blazor Control Example: a ready solution that uses this template's `ProgressBar`.
- Custom Control API Reference: choose between WPF, Blazor and Portable controls.
- WPF Control API Reference: a WPF-only control placed with the WPF Control element.
- Portable API Reference: one control for both the WPF clients and the browser.

In this section...