Blazor Control Reference

Embed Blazor components in displays.

Reference **Controls** **Viewer** Blazor

A BlazorControl (or Blazor Component) is a reusable UI element that can contain HTML, CSS and C# logic. It is similar to web components in front end frameworks like React or Angular.
Blazor Controls are defined as *".razor"* files and can be used throughout your HTML5 displays. See a solution example using Blazor at BlazorControl Example.

Requirements
Configuration
CodeBehind
Installing Blazor Controls

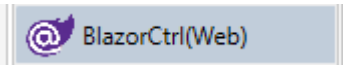
To build your own Blazor control, see Blazor Control API Reference.

Requirements

This component runs only on HTML5 displays, targeting web browsers.

The component need use Interactive WebAssembly Render Mode.

Configuration

1. Go to **Displays / Draw**.
2. On the Components Panel, select **Viewer**, then **BlazorCtrl(Web)**. 

3. Click or drag-and-drop it on the Drawing area to use it.
4. Once you drop it, a File Explorer will open, here you must choose the control DLL file that will be used.
5. Select the file you want to use and click Open
6. Choose which component you want to use from the selected DLL and press OK, then click on the display to insert your control.
7. Double-click your control and configure the required fields (if different than default).

Blazor Control Settings	
Field	**Description**
Control Name	Defines a name for the control so it can be accessed in the CodeBehind script. See Display CodeBehind.
Assembly	The name of the DLL that will be used. E.g.: *BlazorControlExample.dll*
Type	The full name of the control that will be used. E.g.: *BlazorControlExample.ProgressBar*
Dependencies	If the control does not contains css/javascript embedded and requires external files, add them all here by line.

CodeBehind

A Blazor Control usually requires input parameters and/or returns data from it. To link tags to them, we must bind the tags directly in the CodeBehind, and the right place to do it is inside the EventCallback that happens once the controls are fully loaded:

BlazorControlLoaded

```
public async Task BlazorControlLoaded(TBlazorControl control)
{
}
```

To do so, we must use the command *BindEvent* , that will create a link in between the Tag and the Parameter inside the control. There are two different cases and it requires some knowledge on the control itself, they are:

1. The bind is one-way only, unidirectional.
2. The bind is two-way, bidirectional, and has an EventCallback to raise the value when it changes inside the control

It is important to know when to use one or another, to make the right choice of how to bind them correctly. Below is a full example of the two bindings and their respective explanation.

Binding Values to Control

```
public async Task BlazorControlLoaded(TBlazorControl control)
{
        //The BindEvent can be used to bind bidirectional Tags to Blazor properties. It requires an
EventCallback on the Blazor control to receive the value back into the Tag.
    // Parameter 1 - CurrentProdution - The name of the parameter inside the control
    // Parameter 2 - The name of the tag to link with the control
    // Parameter 3 - The name of the EventCallback that will raise the changes
        control.BindEvent("CurrentProduction", @Tag.CurrentProduction.GetName(),
"CurrentProductionRestarted");

        //Using "null" in the third property of the BindEvent function will set the Tag into the property in
a one-way, so the tag will only send values to the control, it will never receive values back.
        control.BindEvent("TargetProduction", @Tag.TargetProduction.GetName(), null);
        control.BindEvent("BarColor", @Tag.BarColor.GetName(), null);
}
```

If you want to set a value directly, without bind any tag into it, you can use:

Set Value

```
public async Task BlazorControlLoaded(TBlazorControl control)
{
        //This event will be triggered for each Blazor Control in the display.
        //Using the TBlazorControl to access the desired Blazor control ProgressBar
        BlazorControlExample.ProgressBar myProgressBarComponent = control.BlazorComponent as
BlazorControlExample.ProgressBar;

        //Once inside the control, you can access any parameter / property / method that has visibility
privileges
        myProgressBarComponent.CurrentProduction = 150;
}
```

Once a command to close the display is triggered, the system will automatically *Unbind* all the tags that were *Binded* before. If at any moment you need to *Unbind* them manually, you can use the following code:

Manual Unbind

```
object eventBind;
TBlazorControl blazorControl;

public async Task BlazorControlLoaded(TBlazorControl control)
{
        //Saves the control in memory for future reference
        blazorControl = control;

        //Bind the tag to the control and receives an object as return to be used later for a manual unbind
        eventBind = control.BindEvent("TargetProduction", @Tag.TargetProduction.GetName(), null);
}

public async Task ManualUnbind()
{
    //Manually Unbind the event
        blazorControl.UnbindEvent(eventBind);
}
```

If at some point you have multiple controls on the same display, you can use the Uid property to decide what actions you will take based on the control you have:

Multiple Controls

```
public async Task BlazorControlLoaded(TBlazorControl control)
{
        if(control.Uid == "ProgressBar")
        {
                //do this
        }
        else if(control.Uid == "LinearGauge")
        {
                //do that
        }
}
```

Installing Blazor Controls

To install a Blazor Control, some steps must be followed, in order to achieve the right results. Below is a step by step on how to do it.

1. Open the installation folder and navigate to *"FrameworX/fx-10/HTML5/BlazorControls"*. This folder is the main folder used for all Blazor Controls, and the system will always use it as a base folder to look for them.
2. In this folder, paste the DLL that you want to use and any *.css* and/or *.js* (javascript) file that are required.
3. Some controls (like Syncfusion or Mudblazor) require to be added in the Dependency Container, if that is the case, you can use the file *"initializer.json"* to do it (same apply for any licensing, css or javascript that you need to add). The file should look like this after the changes:

Initializer.json

```
{
  "builder.Services.1": "Mudblazor.Core.dll;MudBlazor.Services.ServiceCollectionExtensions;AddMudServices",
  "Initializer.1": "<assembly>;<type>;<method>",
  "Licensing.1": "<assembly>;<type>;<method>;<argument>",
  "LoadAssembly.1": "<assembly>",
  "css.1": "MudBlazorCSS.css",
  "js.1": "MudBlazorJS.js"
}
```

Where:

Initializer.json Settings

Field	Description
builder.Services.1	A three parameters configuration: 1. The name of the DLL; 2. The full name of the service, including *"namespace"* + *"class name"*; 3. The name of the method that add the control into the Dependency Container
Initializer.1	The method from the *"builder."* to be called. E.g.: *builder.Build();*
Licensing.1	The full path to the registration of the license. E.g.: *Syncfusion.Licensing.SyncfusionLicenseProvider.RegisterLicense("key");*
LoadAssembly.1	The full path to any external dependencies that are required to the control to work properly.
css.1	The *.css* file full path (starting at the root folder *"BlazorControls"*).
js.1	The *.js* file full path (starting at the root folder *"BlazorControls"*).

In this section...