Build a .NET MAUI Blazor Hybrid app with a Blazor Web App

Article • 11/12/2024

This article shows you how to build a .NET MAUI Blazor Hybrid app with a Blazor Web App that uses a shared user interface via a Razor class library (RCL).

Prerequisites and preliminary steps

For prerequisites and preliminary steps, see Build a .NET MAUI Blazor Hybrid app. We recommend using the .NET MAUI Blazor Hybrid tutorial to set up your local system for .NET MAUI development before using the guidance in this article.

.NET MAUI Blazor Hybrid and Web App solution template

The .NET MAUI Blazor Hybrid and Web App solution template sets up a solution that targets Android, iOS, Mac, Windows and Web that reuses UI. You can choose a Blazor interactive render mode for the web app and it creates the appropriate projects for the app, including a Blazor Web App and a .NET MAUI Blazor Hybrid app. A shared Razor class library (RCL) maintains the Razor components for the app's UI. The template also provides sample code to show you how to use dependency injection to provide different interface implementations for the Blazor Hybrid and Blazor Web App, which is covered in the Using interfaces to support different device implementations section of this article.

If you haven't already installed the .NET MAUI workload, install it now. The .NET MAUI workload provides the project template:

.NET CLI

```
dotnet workload install maui
```

Create a solution from the project template with the following .NET CLI command:

.NET CLI

```
dotnet new maui-blazor-web -o MauiBlazorWeb -I Server
```

In the preceding command:

- The `-o|--output` option creates a new folder for the app named `MauiBlazorWeb`.
- The `-I|--InteractivityPlatform` option sets the interactivity render mode to Interactive Server (`InteractiveServer`). All three interactive Blazor render modes (`Server`, `WebAssembly`, and `Auto`) are supported by the project template. For more information, see the Use Blazor render modes section.

The app automatically adopts global interactivity, which is important because MAUI apps always run interactively and throw errors on Razor component pages that explicitly specify a render mode. For more information, see BlazorWebView needs a way to enable overriding ResolveComponentForRenderMode (dotnet/aspnetcore #51235) .

Use Blazor render modes

Use the guidance in one of the following subsections that matches your app's specifications for applying Blazor render modes in the Blazor Web App but ignore the render mode assignments in the MAUI project.

Render mode specification subsections:

- Global Server interactivity
- Global Auto or WebAssembly interactivity

Global Server interactivity

- Interactive render mode: **Server**
- Interactivity location: **Global**
- Solution projects
  - MAUI (`MauiBlazorWeb`)
  - Blazor Web App (`MauiBlazorWeb.Web`)
  - RCL (`MauiBlazorWeb.Shared`): Contains the shared Razor components without setting render modes in each component.

Project references: `MauiBlazorWeb` and `MauiBlazorWeb.Web` have a project reference to `MauiBlazorWeb.Shared`.

Global Auto or WebAssembly interactivity

- Interactive render mode: **Auto** or **WebAssembly**
- Interactivity location: **Global**

- Solution projects
  - MAUI (`MauiBlazorWeb`)
  - Blazor Web App
    - Server project: `MauiBlazorWeb.Web`
    - Client project: `MauiBlazorWeb.Web.Client`
  - RCL (`MauiBlazorWeb.Shared`): Contains the shared Razor components without setting render modes in each component.

Project references:

- `MauiBlazorWeb`, `MauiBlazorWeb.Web`, and `MauiBlazorWeb.Web.Client` projects have a project reference to `MauiBlazorWeb.Shared`.
- `MauiBlazorWeb.Web` has a project reference to `MauiBlazorWeb.Web.Client`.

Per-page/component Server interactivity

- Interactive render mode: **Server**
- Interactivity location: **Per-page/component**
- Solution projects
  - MAUI (`MauiBlazorWeb`): Calls `InteractiveRenderSettings.ConfigureBlazorHybridRenderModes` in `MauiProgram.cs`.
  - Blazor Web App (`MauiBlazorWeb.Web`): Doesn't set an `@rendermode` directive attribute on the `HeadOutlet` and `Routes` components of the `App` component (`Components/App.razor`).
  - RCL (`MauiBlazorWeb.Shared`): Contains the shared Razor components that set the `InteractiveServer` render mode in each component.

`MauiBlazorWeb` and `MauiBlazorWeb.Web` have a project reference to `MauiBlazorWeb.Shared`.

Add the following `InteractiveRenderSettings` class to the RCL. The class properties are used to set component render modes.

The MAUI project is interactive by default, so no action is taken at the project level in the MAUI project other than calling `InteractiveRenderSettings.ConfigureBlazorHybridRenderModes`.

For the Blazor Web App on the web client, the property values are assigned from RenderMode. When the components are loaded into a BlazorWebView for the MAUI project's native client, the render modes are unassigned (`null`) because the MAUI

project explicitly sets the render mode properties to `null` when `ConfigureBlazorHybridRenderModes` is called.

`InteractiveRenderSettings.cs`:

C#

```
using Microsoft.AspNetCore.Components;
using Microsoft.AspNetCore.Components.Web;

namespace MauiBlazorWeb.Shared;

public static class InteractiveRenderSettings
{
    public static IComponentRenderMode? InteractiveServer { get; set;
} =
        RenderMode.InteractiveServer;
    public static IComponentRenderMode? InteractiveAuto { get; set; }
=
        RenderMode.InteractiveAuto;
    public static IComponentRenderMode? InteractiveWebAssembly { get;
set; } =
        RenderMode.InteractiveWebAssembly;

    public static void ConfigureBlazorHybridRenderModes()
    {
        InteractiveServer = null;
        InteractiveAuto = null;
        InteractiveWebAssembly = null;
    }
}
```

In `MauiProgram.CreateMauiApp` of `MauiProgram.cs`, call `ConfigureBlazorHybridRenderModes`:

C#

```
InteractiveRenderSettings.ConfigureBlazorHybridRenderModes();
```

In the `_Imports.razor` file of the `.Shared` RCL, replace the `@using` statement for Microsoft.AspNetCore.Components.Web.RenderMode with an `@using` statement for `InteractiveRenderSettings` to make the properties of the `InteractiveRenderSettings` class available to components:

diff

```
- @using static Microsoft.AspNetCore.Components.Web.RenderMode
```

```
+ @using static InteractiveRenderSettings
```

> **Note**
>
> The assignment of render modes via the RCL's `InteractiveRenderSettings` class properties differs from a typical standalone Blazor Web App. In a Blazor Web App, the render modes are normally provided by RenderMode in the Blazor Web App's `_Import` file.

Per-page/component Auto interactivity

- Interactive render mode: **Auto**
- Interactivity location: **Per-page/component**
- Solution projects
  - MAUI (`MauiBlazorWeb`): Calls `InteractiveRenderSettings.ConfigureBlazorHybridRenderModes` in `MauiProgram.cs`.
  - Blazor Web App
    - Server project: `MauiBlazorWeb.Web`: Doesn't set an `@rendermode` directive attribute on the `HeadOutlet` and `Routes` components of the `App` component (`Components/App.razor`).
    - Client project: `MauiBlazorWeb.Web.Client`
  - RCL (`MauiBlazorWeb.Shared`): Contains the shared Razor components that set the `InteractiveAuto` render mode in each component.

Project references:

- `MauiBlazorWeb`, `MauiBlazorWeb.Web`, and `MauiBlazorWeb.Web.Client` have a project reference to `MauiBlazorWeb.Shared`.
- `MauiBlazorWeb.Web` has a project reference to `MauiBlazorWeb.Web.Client`.

Add the following `InteractiveRenderSettings` class is added to the RCL. The class properties are used to set component render modes.

The MAUI project is interactive by default, so no action is taken at the project level in the MAUI project other than calling `InteractiveRenderSettings.ConfigureBlazorHybridRenderModes`.

For the Blazor Web App on the web client, the property values are assigned from RenderMode. When the components are loaded into a BlazorWebView for the MAUI

project's native client, the render modes are unassigned (`null`) because the MAUI project explicitly sets the render mode properties to `null` when `ConfigureBlazorHybridRenderModes` is called.

`InteractiveRenderSettings.cs`:

C#

```
using Microsoft.AspNetCore.Components;
using Microsoft.AspNetCore.Components.Web;

namespace MauiBlazorWeb.Shared;

public static class InteractiveRenderSettings
{
    public static IComponentRenderMode? InteractiveServer { get; set;
} =
        RenderMode.InteractiveServer;
    public static IComponentRenderMode? InteractiveAuto { get; set; }
=
        RenderMode.InteractiveAuto;
    public static IComponentRenderMode? InteractiveWebAssembly { get;
set; } =
        RenderMode.InteractiveWebAssembly;

    public static void ConfigureBlazorHybridRenderModes()
    {
        InteractiveServer = null;
        InteractiveAuto = null;
        InteractiveWebAssembly = null;
    }
}
```

In `MauiProgram.CreateMauiApp` of `MauiProgram.cs`, call `ConfigureBlazorHybridRenderModes`:

C#

```
InteractiveRenderSettings.ConfigureBlazorHybridRenderModes();
```

In the `_Imports.razor` file of the `.Shared.Client` RCL, replace the `@using` statement for Microsoft.AspNetCore.Components.Web.RenderMode with an `@using` statement for `InteractiveRenderSettings` to make the properties of the `InteractiveRenderSettings` class available to components:

diff

```
- @using static Microsoft.AspNetCore.Components.Web.RenderMode
+ @using static InteractiveRenderSettings
```

> **Note**
>
> The assignment of render modes via the RCL's `InteractiveRenderSettings` class properties differs from a typical standalone Blazor Web App. In a Blazor Web App, the render modes are normally provided by RenderMode in the Blazor Web App's `_Import` file.

Per-page/component WebAssembly interactivity

- Interactive render mode: **WebAssembly**
- Interactivity location: **Per-page/component**
- Solution projects
  - MAUI (`MauiBlazorWeb`)
  - Blazor Web App
    - Server project: `MauiBlazorWeb.Web`: Doesn't set an `@rendermode` directive attribute on the `HeadOutlet` and `Routes` components of the `App` component (`Components/App.razor`).
    - Client project: `MauiBlazorWeb.Web.Client`
  - RCLs
    - `MauiBlazorWeb.Shared`
    - `MauiBlazorWeb.Shared.Client`: Contains the shared Razor components that set the `InteractiveWebAssembly` render mode in each component. The `.Shared.Client` RCL is maintained separately from the `.Shared` RCL because the app should maintain the components that are required to run on WebAssembly separately from the components that run on server and that stay on the server.

Project references:

- `MauiBlazorWeb` and `MauiBlazorWeb.Web` have project references to `MauiBlazorWeb.Shared`.
- `MauiBlazorWeb.Web` has a project reference to `MauiBlazorWeb.Web.Client`.
- `MauiBlazorWeb.Web.Client` and `MauiBlazorWeb.Shared` have a project reference to `MauiBlazorWeb.Shared.Client`.

Add the following AdditionalAssemblies parameter to the `Router` component instance for the `MauiBlazorWeb.Shared.Client` project assembly (via its `_Imports` file) in the `MauiBlazorWeb.Shared` project's `Routes.razor` file:

razor

```
<Router AppAssembly="@typeof(Routes).Assembly"
        AdditionalAssemblies="new [] { typeof(MauiBlazorWeb.Shared.-
Client._Imports).Assembly }">
    <Found Context="routeData">
        <RouteView RouteData="@routeData" DefaultLayout="@typeof(Com-
ponents.Layout.MainLayout)" />
        <FocusOnNavigate RouteData="@routeData" Selector="h1" />
    </Found>
</Router>
```

Add the `MauiBlazorWeb.Shared.Client` project assembly (via its `_Imports` file) with the following AddAdditionalAssemblies call in the `MauiBlazorWeb.Web` project's `Program.cs` file:

C#

```
app.MapRazorComponents<App>()
    .AddInteractiveWebAssemblyRenderMode()
    .AddAdditionalAssemblies(type-
of(MauiBlazorWeb.Shared._Imports).Assembly)
    .AddAdditionalAssemblies(typeof(MauiBlazorWeb.Shared.Client._Im-
ports).Assembly);
```

Add the following `InteractiveRenderSettings` class is added to the `.Shared.Client` RCL. The class properties are used to set component render modes for server-based components.

The MAUI project is interactive by default, so no action is taken at the project level in the MAUI project other than calling `InteractiveRenderSettings.ConfigureBlazorHybridRenderModes`.

For the Blazor Web App on the web client, the property values are assigned from RenderMode. When the components are loaded into a BlazorWebView for the MAUI project's native client, the render modes are unassigned (`null`) because the MAUI project explicitly sets the render mode properties to `null` when `ConfigureBlazorHybridRenderModes` is called.

`InteractiveRenderSettings.cs` (`.Shared.Client` RCL):

C#

```
using Microsoft.AspNetCore.Components;
using Microsoft.AspNetCore.Components.Web;

namespace MauiBlazorWeb.Shared;

public static class InteractiveRenderSettings
{
    public static IComponentRenderMode? InteractiveServer { get; set;
} =
        RenderMode.InteractiveServer;
    public static IComponentRenderMode? InteractiveAuto { get; set; }
=
        RenderMode.InteractiveAuto;
    public static IComponentRenderMode? InteractiveWebAssembly { get;
set; } =
        RenderMode.InteractiveWebAssembly;

    public static void ConfigureBlazorHybridRenderModes()
    {
        InteractiveServer = null;
        InteractiveAuto = null;
        InteractiveWebAssembly = null;
    }
}
```

A slightly different version of the `InteractiveRenderSettings` class is added to the `.Shared` RCL. In the class added to the `.Shared` RCL, `InteractiveRenderSettings.ConfigureBlazorHybridRenderModes` of the `.Shared.Client` RCL is called. This ensures that the render mode of WebAssembly components rendered on the MAUI client are unassigned (`null`) because they're interactive by default on the native client.

`InteractiveRenderSettings.cs` (`.Shared` RCL):

C#

```
using Microsoft.AspNetCore.Components;
using Microsoft.AspNetCore.Components.Web;

namespace MauiBlazorWeb.Shared
{
    public static class InteractiveRenderSettings
    {
        public static IComponentRenderMode? InteractiveServer { get;
set; } =
            RenderMode.InteractiveServer;
        public static IComponentRenderMode? InteractiveAuto { get;
set; } =
```

```
            RenderMode.InteractiveAuto;
        public static IComponentRenderMode? InteractiveWebAssembly {
get; set; } =
            RenderMode.InteractiveWebAssembly;

        public static void ConfigureBlazorHybridRenderModes()
        {
            InteractiveServer = null;
            InteractiveAuto = null;
            InteractiveWebAssembly = null;
            MauiBlazorWeb.Shared.Client.InteractiveRenderSettings
                .ConfigureBlazorHybridRenderModes();
        }
    }
}
```

In `MauiProgram.CreateMauiApp` of `MauiProgram.cs`, call `ConfigureBlazorHybridRenderModes`:

C#

```
InteractiveRenderSettings.ConfigureBlazorHybridRenderModes();
```

In the `_Imports.razor` file of the `.Shared.Client` RCL, replace the `@using` statement for Microsoft.AspNetCore.Components.Web.RenderMode with an `@using` statement for `InteractiveRenderSettings` to make the properties of the `InteractiveRenderSettings` class available to components:

diff

```
- @using static Microsoft.AspNetCore.Components.Web.RenderMode
+ @using static InteractiveRenderSettings
```

> **Note**
>
> The assignment of render modes via the RCL's `InteractiveRenderSettings` class properties differs from a typical standalone Blazor Web App. In a Blazor Web App, the render modes are normally provided by RenderMode in the Blazor Web App's `_Import` file.

Using interfaces to support different device implementations

The following example demonstrates how to use an interface to call into different implementations across the web app and the native (MAUI) app. The following example creates a component that displays the device form factor. Use the MAUI abstraction layer for native apps and provide an implementation for the web app.

In the Razor class library (RCL), a `Services` folder contains an `IFormFactor` interface.

`Services/IFormFactor.cs`:

C#

```
namespace MauiBlazorWeb.Shared.Services
{
    public interface IFormFactor
    {
        public string GetFormFactor();
        public string GetPlatform();
    }
}
```

The `Home` component (`Components/Pages/Home.razor`) of the RCL displays the form factor and platform.

`Components/Pages/Home.razor`:

razor

```
@page "/"
@using MauiBlazorWeb.Shared.Services
@inject IFormFactor FormFactor

<PageTitle>Home</PageTitle>

<h1>Hello, world!</h1>

Welcome to your new app running on <em>@factor</em> using <em>@plat-
form</em>.

@code {
    private string factor => FormFactor.GetFormFactor();
    private string platform => FormFactor.GetPlatform();
}
```

The web and native apps contain the implementations for `IFormFactor`.

In the Blazor Web App, a folder named `Services` contains the following `FormFactor.cs` file with the `FormFactor` implementation for web app use.

`Services/FormFactor.cs` (`MauiBlazorWeb.Web` project):

C#

```
using MauiBlazorWeb.Shared.Services;

namespace MauiBlazorWeb.Web.Services
{
    public class FormFactor : IFormFactor
    {
        public string GetFormFactor()
        {
            return "Web";
        }

        public string GetPlatform()
        {
            return Environment.OSVersion.ToString();
        }
    }
}
```

In the MAUI project, a folder named `Services` contains the following `FormFactor.cs` file with the `FormFactor` implementation for native use. The MAUI abstractions layer is used to write code that works on all native device platforms.

`Services/FormFactor.cs` (`MauiBlazorWeb` project):

C#

```
using MauiBlazorWeb.Shared.Services;

namespace MauiBlazorWeb.Services
{
    public class FormFactor : IFormFactor
    {
        public string GetFormFactor()
        {
            return DeviceInfo.Idiom.ToString();
        }

        public string GetPlatform()
        {
            return DeviceInfo.Platform.ToString() + " - " +
DeviceInfo.VersionString;
        }
    }
}
```

Dependency injection is used to obtain the implementations of these services.

In the MAUI project, the `MauiProgram.cs` file has following `using` statements at the top of the file:

```
C#

using MauiBlazorWeb.Services;
using MauiBlazorWeb.Shared.Interfaces;
```

Immediately before the call to `builder.Build()`, `FormFactor` is registered to add device-specific services used by the RCL:

```
C#

builder.Services.AddSingleton<IFormFactor, FormFactor>();
```

In the Blazor Web App, the `Program` file has the following `using` statements at the top of the file:

```
C#

using MauiBlazorWeb.Shared.Interfaces;
using MauiBlazorWeb.Web.Services;
```

Immediately before the call to `builder.Build()`, `FormFactor` is registered to add device-specific services used by the Blazor Web App:

```
C#

builder.Services.AddScoped<IFormFactor, FormFactor>();
```

If the solution also targets WebAssembly via a `.Web.Client` project, an implementation of the preceding API is also required in the `.Web.Client` project.

You can also use compiler preprocessor directives in your RCL to implement different UI depending on the device the app is running on. For this scenario, the app must multi-target the RCL just like the MAUI app does. For an example, see the BethMassi/BethTimeUntil GitHub repository .

Additional resources

- ASP.NET Core Blazor Hybrid authentication and authorization
- ASP.NET Core Blazor render modes

- Reuse Razor components in ASP.NET Core Blazor Hybrid apps