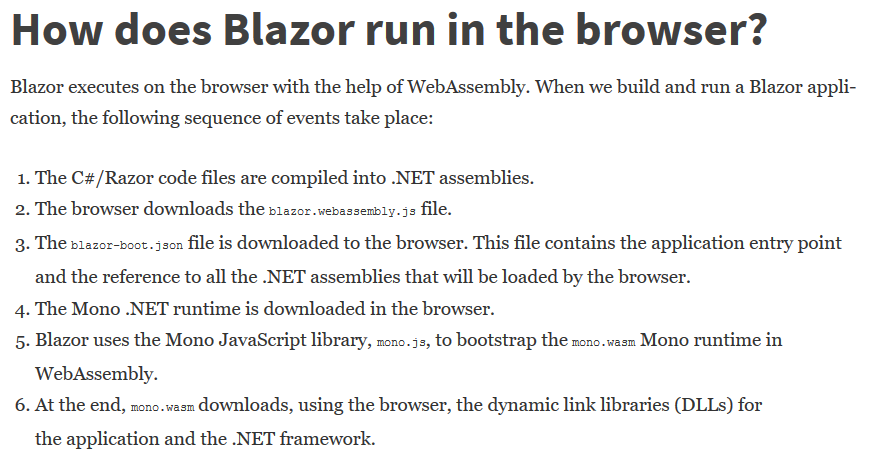
# WebAssembly vs Server

The scaffolded content – outside of pages – is very different. Eg

* Program.cs and Startup.cs – totally different
* Index.html in Webroot
* Etc

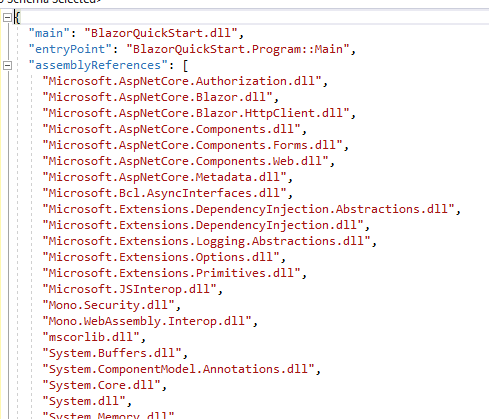
So you’d need a dll with “components” that could be pulled into to different project types

# How It Works



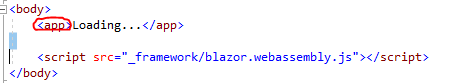
Re # 3

* it is blazor.boot.json
* I found it in the obj\Debug\netstandard2.0\blazor folder
* My project was named BlazorQuickStart – which you can see in Main/Entry point followed by a list of dlls

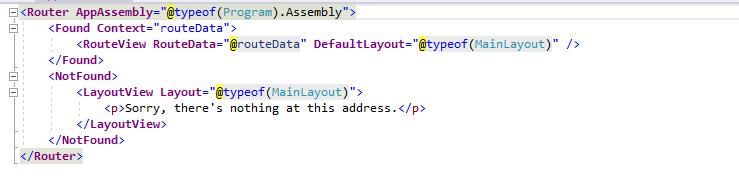


## Booting

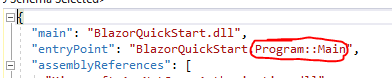
* Wwwroot has index.html – which has
  + The <app> element
  + Runs blazor.webassembly.js

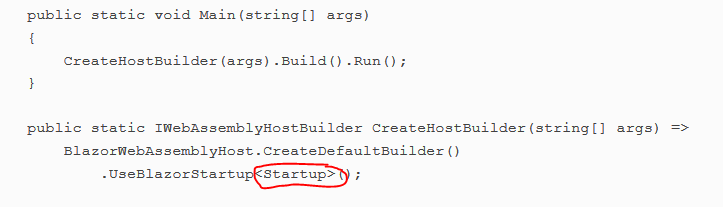


* App.razor – This is scaffolded by VS on the create



* The blazor.boot.json identifies the entry point Program.Main

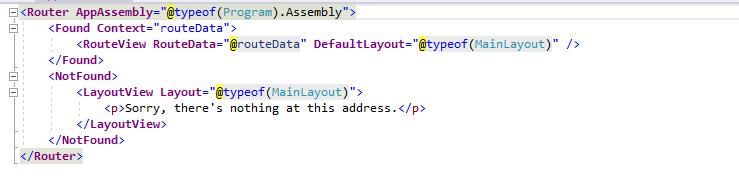




* Startup.Configure has

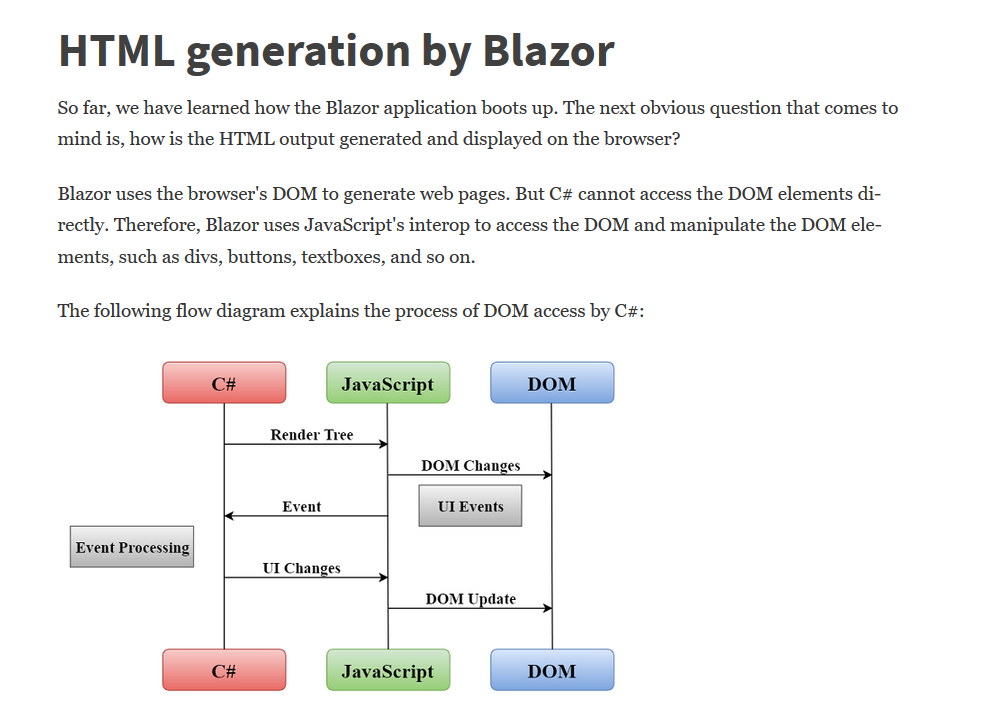


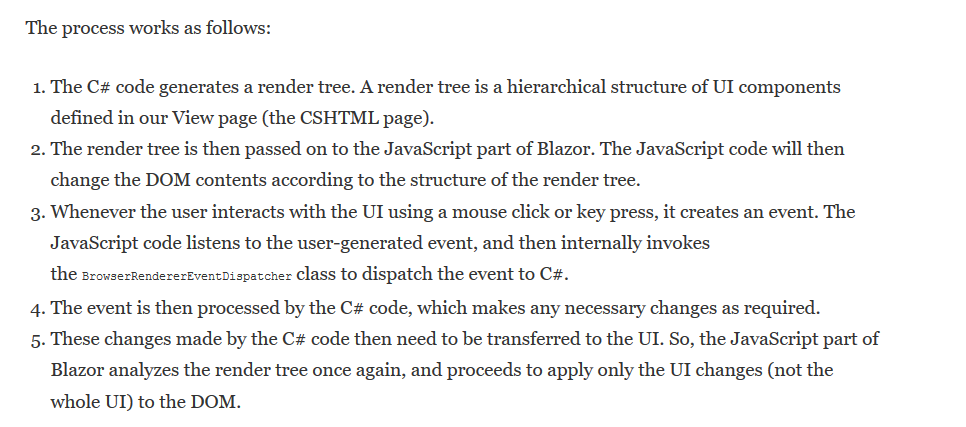
* App.razor – This is scaffolded by VS on the create



After the <app> element in rendered blazor.webassembly.js is loaded / run (which loads the dlls etc)

## HTML Handling



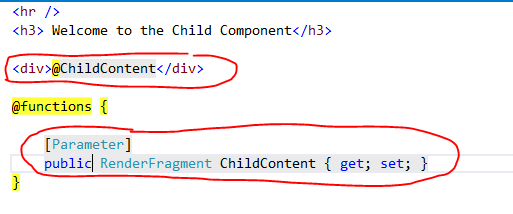


# Components

<https://docs.microsoft.com/en-us/aspnet/core/blazor/components?view=aspnetcore-3.0#event-handling>

## Parent/Child

* Child
  + has no @page
  + Has some of it’s own HTML / Model stuff – but also has
    - Literally <div>@ChildContent</div>
    - A RenderFragment named ChildContent decorated with [Parameter]



* Parent passes content to child

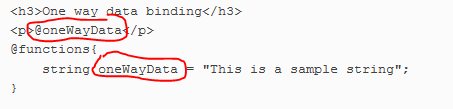
The RenderFragment parameter is provided between the tags of the child component.

See "Parent-child example" commit

## Data Binding

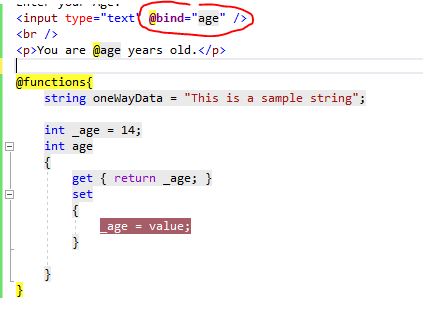
### One Way

Value of Field / Prop from Model is inserted into HTML via @PropertyName

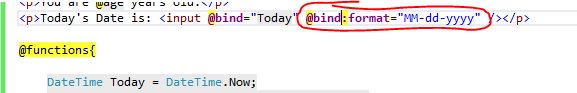


### Two Way

* Model to HTML and HTML to model
* Use the @bind=”PropertyName” attribute

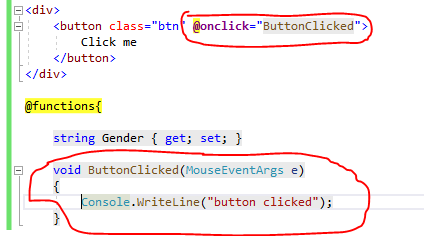


Also – you can format a Date. Whether you’d need to do this with Data Annotations I don’t know. Hopefully not.



## Events

<https://docs.microsoft.com/en-us/aspnet/core/blazor/components?view=aspnetcore-3.0#event-handling>



## LifeCycle Methods

<https://docs.microsoft.com/en-us/aspnet/core/blazor/components?view=aspnetcore-3.0#lifecycle-methods>

# Layouts

<https://docs.microsoft.com/en-us/aspnet/core/blazor/layouts?view=aspnetcore-3.0>

A layout is a Component that

* Inherits from LayoutComponentBase
* Has a @Body that tells where a page is rendered

You can specify which Layout to use via @layout <>

Default layout I s specified in App.razor

Each folder can contain an \_Imports.razor file – which can contain – which apply to all in the folder and subfolder. This @layout overrides the default

* @import
* @layout

# Routing

## @page directive

* Use the @page “/wtf” directive
* There can be multiple @page in which case multiple routes go to the same page
* Route Parameters
  + Enclose the parm name in {} – Eg @page “/route/{routeparm}
  + Add a property in the model
    - With the same name – routeparm
    - Decorated with [Parameter]

## Code

One the services injected by the framework is



* @inject NavigationManager
* Call it’s NavigateTo() method

See “Added Code-based Navigation”

## NavLink Component

* Can be used instead of <a>
* Toggles the “active” css class based on whether or not the current url “matches” href
* Has a Match attribute
  + NavLinkMatch.All – must match the entire URL
  + NavLinkMatch.Prefix – This is a “starts with”

One the services injected by the framework is

# JSInterop

## .NET Calling JS

* Create a JS File
* Reference the JS fiile in index.html
* Use IJSRuntime.Invoke

See the “.NET Calling JS” commit

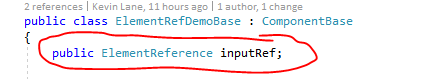
## Element References

<https://docs.microsoft.com/en-us/aspnet/core/blazor/javascript-interop?view=aspnetcore-3.1#capture-references-to-elements>

* HTML has an @ref=”propertyName” attribute



* CS has ElementReference propertyName property

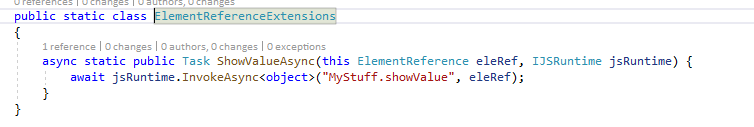


* The JS can “act on” the element reference. In this example the “element” in <input> which has a property name *value*

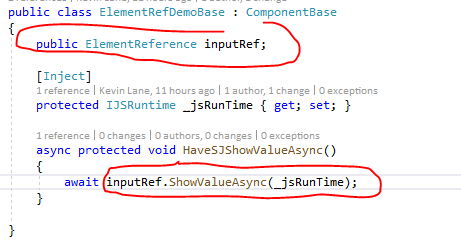


Since in .NET we are working with ElementReference we can create extension methods on ElementReference.

* Extension definition



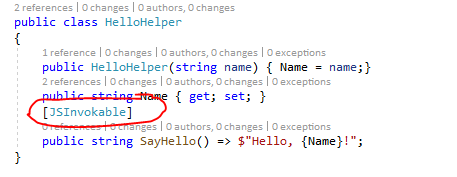
* Extension usage – Component has



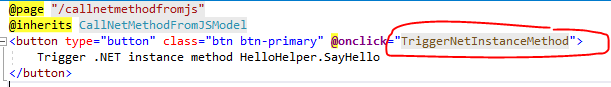
## JS Calling .NET

### Instance

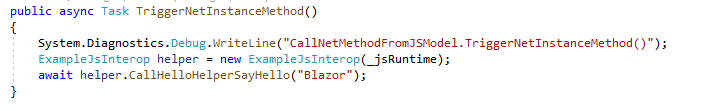
* When all is said and done, the JS code will invoke SayHello on an HelloHelper instance



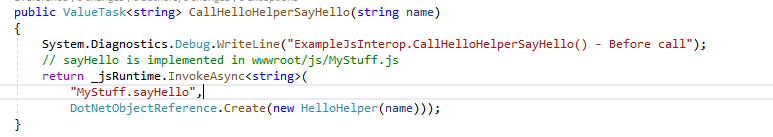
* HelloHelper.SayHello() returns a string
* .NET methods invoked by JS must be decorated with [JSInvokable]
* View calls a method on .NET Component



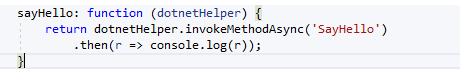
* .NET Component in this case uses a “helper”



* The helper – ExampleJSInterop –
  + Creates the HelloHelper instance (injecting a value)
  + Converts the HelloHelper to a help –DotNetObjectReference.Create()
  + Uses the JSRuntime to invoke the “MyStuff.sayHello” function passing the HelloHelper instance



* JS function receives the “helper” and invokes the SayHello method on the HelloHelper instance



There is nothing “magical” about the fact that both JS and NET have a function/method named SayHello (case insensitive)

It could be jsSayHello and DotNetSayHello. I just don’t want to go back and redo it…

You run it and you get

