Visual Studio LIVE! | San Diego

EXPERT SOLUTIONS FOR .NET DEVELOPERS

Essential Web Development with ASP.NET Core

Mark Michaelis

Nerd

Level: Intermediate

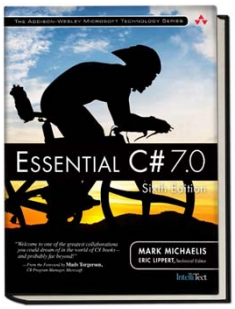
IntelliTect

Code Again for the First Time!

Visual Studio 25 YEARS OF CODING INNOVATION

Mark Michaelis

Chief Technical Architect,
Author, Trainer



MVP Microsoft Most Valuable Professional

Microsoft Regional Director

Professional Scrum Master II

Visual Studio 25 YEARS OF CODING INNOVATION

What Was Wrong With My Cheese?

- ASP.NET (Framework) 1.0 came out almost 16 years ago
  - Massive evolution since then
- Limited support for modern tooling
  - SCSS, TypeScript, gulp, etc.
- Limited to Windows community
- Big bang releases
- No side-by-side framework install

Where is My Cheese?

Open Source	Cross platform	Flexible Hosting
Modular Architecture	Updated Packaging	Unified Stack
Built in Dependency Injection	Modular Tooling	Performance

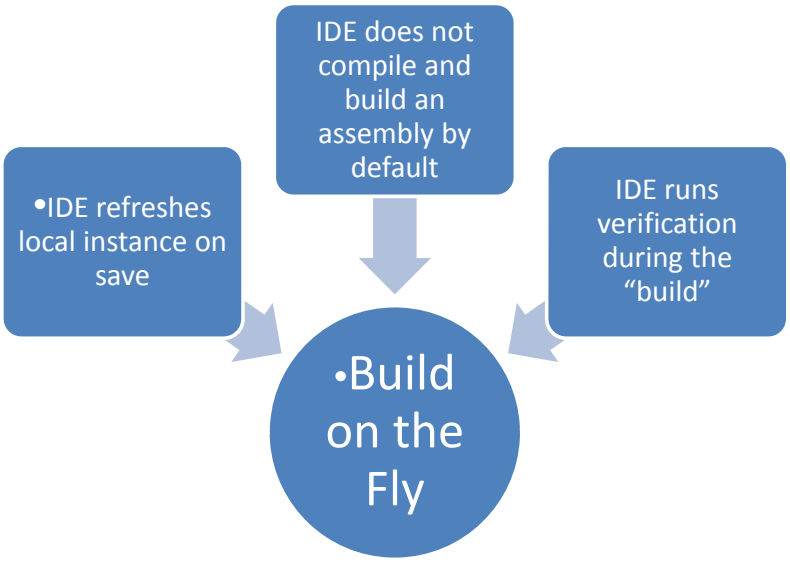
Hosting Flexibility

- IIS/IIS Express
- Self Host
- Kestrel (Node JS/cross platform hosting)

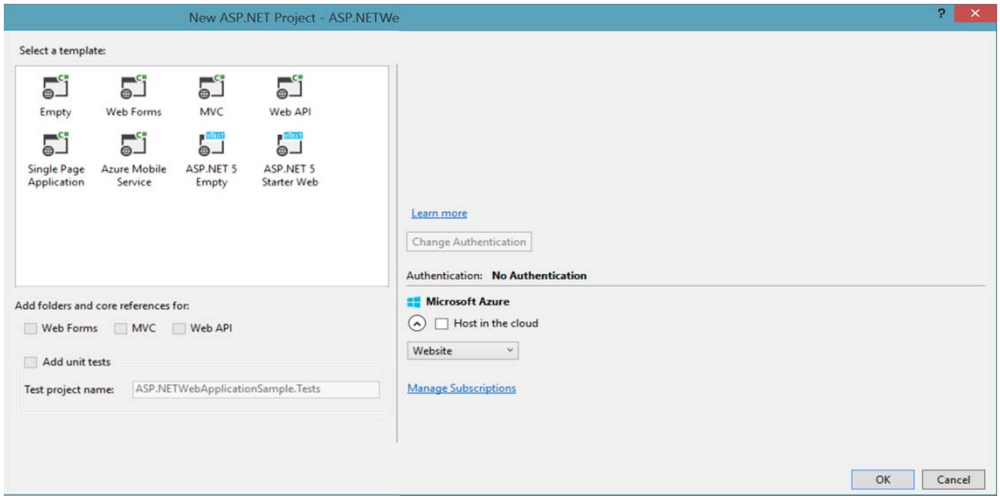
CLR Modularity Options

- Full .NET CLR
- Minimal – Core CLR
- Cross-Platform - CLR

Dynamic Development



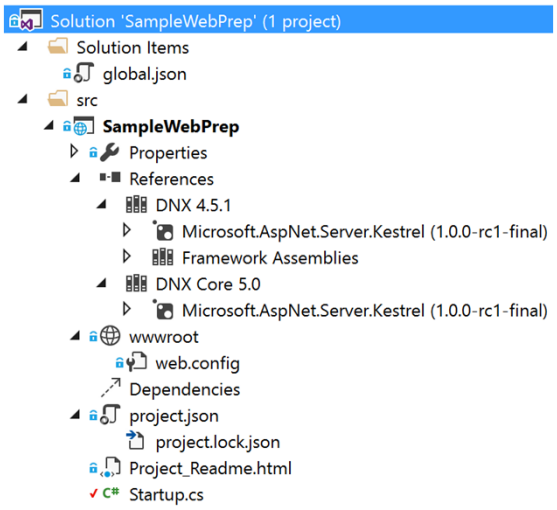
Demo



Empty Project

Notes:

- Project file MSBuild Format
- Minimized down to what is required.
- wwwroot
  - Web.config (for IIS redirect)
- References display the frameworks that are targeted



Web Root (wwwroot) Explained

- The root of your website (http://hostname/)
- Contains static assets
  - HTML
  - CSS
  - Image files
  - JavaScript
- Not build input files:
  - C# files
  - Coffee Script, TypeScript, LESS, or Sass files
  - Source files like pre-minified JS or pre-optimized image files

Visual Studio

Enterprise 2017

Key Web Features

.NET Core 2.1.4

dotnet --info

- dotnet sdk
- Microsoft.NETCore.App
- ASP.NET Core
- *.csproj support

VS Tooling for .NET Core

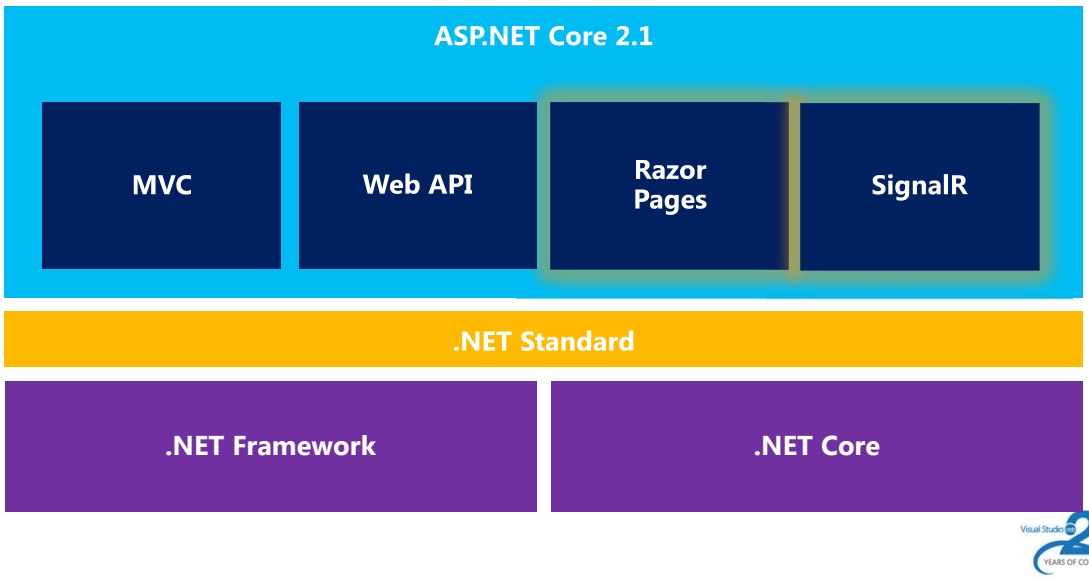
- ASP.NET Core Config UI
- Updated Task Runner
- Upgraded Extensions

Integrated Docker Support

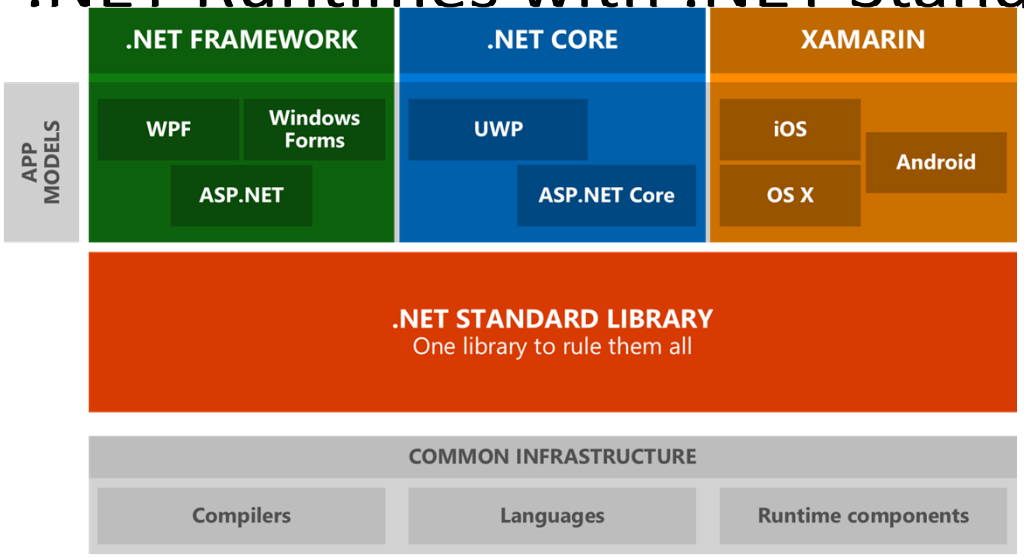
- Yml files
- Debugging
- Deployment

ASP.NET CORE COMPONENTIZATION

ASP.NET Core 2.1



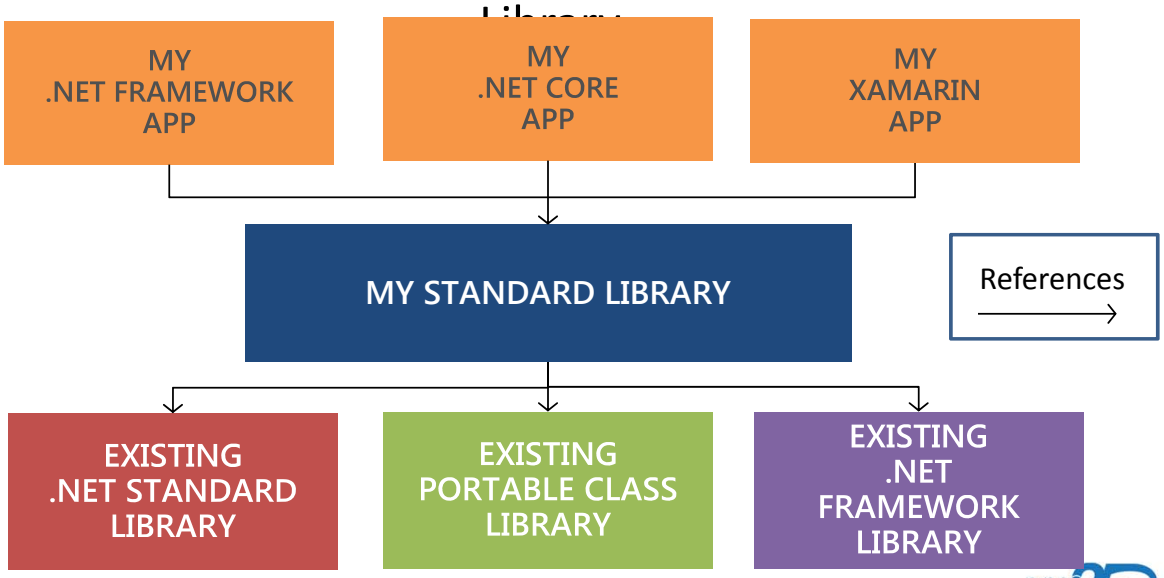
.NET Runtimes with .NET Standard



Before .NET Standard 2.0

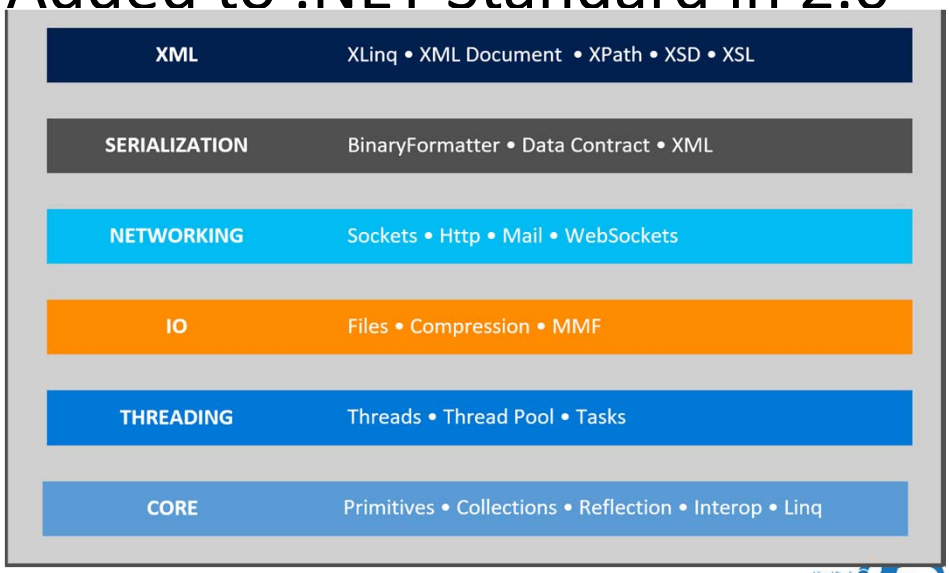
- Portability is relatively limited because .NET Core is relatively small
- .NET Framework libraries (the vast number of libraries that exist today) are only available on Windows
- .NET Core 1.1 is based on a large number of mini-assemblies, thus creating a pain to manage and a pain to upgrade.

Any .NET Standard App Using Any .NET Standard Library



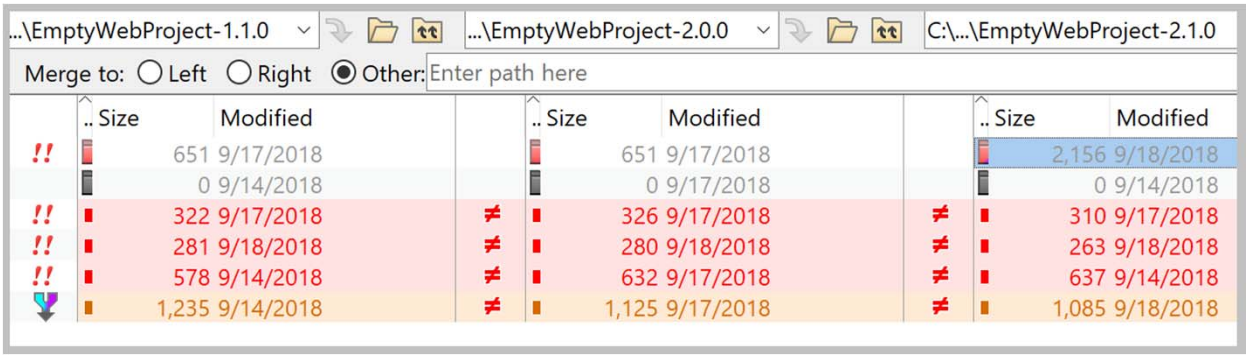
What's Added to .NET Standard in 2.0

Add Support for .NET Framework BCL APIs



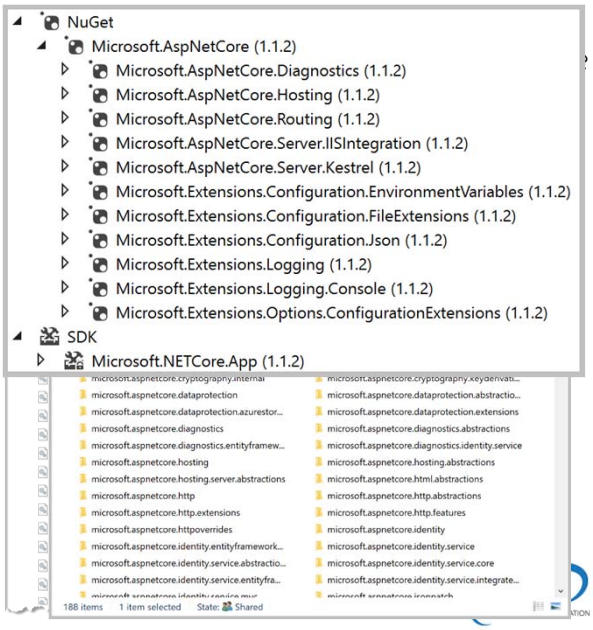
```
>dotnet new web -n EmptyWebApplication
The template "ASP.NET Core Empty" was created successfully.

Processing post-creation actions...
Running 'dotnet restore' on EmptyWebApplication\EmptyWebApplication.csproj...
  Restoring packages for ...\EmptyWebApplication\EmptyWebApplication.csproj...
  Generating MSBuild file ...\EmptyWebApplication\obj\EmptyWebApplication.csproj.nuget.g.props.
  Generating MSBuild file ...\EmptyWebApplication\obj\EmptyWebApplication.csproj.nuget.g.targets.
  Restore completed in 16.88 sec for ...\EmptyWebApplication\EmptyWebApplication.csproj.

Restore succeeded.

>dotnet publish .\EmptyWebApplication\EmptyWebApplication.csproj
Microsoft (R) Build Engine version 15.8.166+gd4e8d81a88 for .NET Core
Copyright (C) Microsoft Corporation. All rights reserved.

  Restore completed in 235.31 ms for ...\EmptyWebApplication\EmptyWebApplication.csproj.
  EmptyWebApplication -> ...\EmptyWebApplication\bin\Debug\netcoreapp2.1\EmptyWebApplication.dll
  EmptyWebApplication -> ...\EmptyWebApplication\bin\Debug\netcoreapp2.1\publish\
>dir publish -Recurse | dir -include *.dll

    Directory: C:\Dropbox\Talks\2018.09.18-Essential Web Development with ASP.NET
    Core\src\temp\EmptyWebApplication\bin\Debug\netcoreapp2.1\publish
```

Comparing ASP.NET Versions



Microsoft.AspNetCore.?

1.*	• Microsoft.AspNetCore
2.0:	• Microsoft.AspNetCore.All
2.1:	• Microsoft.AspNetCore.App

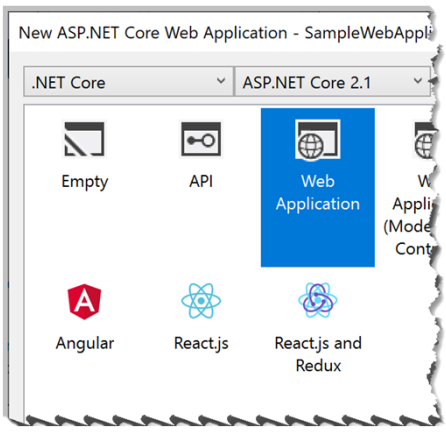


Program.cs

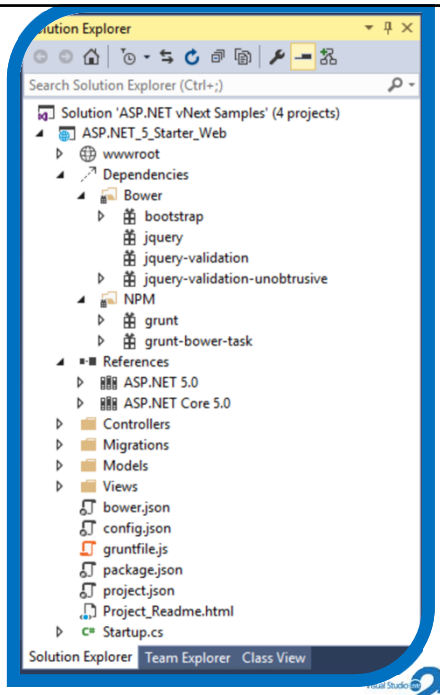
```
public class Program
{
    public static void Main(string[] args)
    {
        CreateWebHostBuilder(args).Build().Run();
    }

    public static IWebHostBuilder CreateWebHostBuilder(string[] args) =>
        WebHost.CreateDefaultBuilder(args)
            .UseStartup<Startup>();
}
```

Startup.cs

```
public class Startup
{
    // This method gets called by the runtime. Use this method to add services to the container.
    public void ConfigureServices(IServiceCollection services)
    {
    }

    // This method gets called by the runtime. Use this method to configure the HTTP request pipeline.
    public void Configure(IApplicationBuilder app, IHostingEnvironment env)
    {
        if (env.IsDevelopment())
        {
            app.UseDeveloperExceptionPage();
        }

        app.Run(async (context) =>
        {
            await context.Response.WriteAsync("Hello World!");
        });
    }
}
```



PROJECT STRUCTURE



CONFIGURATION

ASP.NET Config

```
var config = new Configuration()
  .AddIniFile("App_Data\\config.ini")
  .AddJsonFile("App_Data\\config.json")
  .AddXmlFile("App_Data\\config.xml")
  //  .AddCommandLine(args)
  .AddEnvironmentVariables();

string display =
  $"size:{{ {
    config.Get("Display:Font:Size")
      } }}   color:{{ {
    config.Get("Display:Font:Color")
      } }} background:{{ {
    config.Get("Display:Font:Background")
```

App_Data\config.json

```
{
    "display":{
        "font":{
            "color": "Yellow"
        }
    }
}
```

App_Data\config.xml

```
<config>
    <display>
        <font background="Blue"/>
    </display>
</config>
```

```
Command Prompt -
>dotnet run ConfigMessage="Hello, My Name is Inigo Montoya"
```

LOGGING

DEPENDENCY INJECTION

Unified Controller

```
[Route("api/[controller]")]
public class ValuesController :
    Controller
{
    // ...
}
```

Tag Helpers

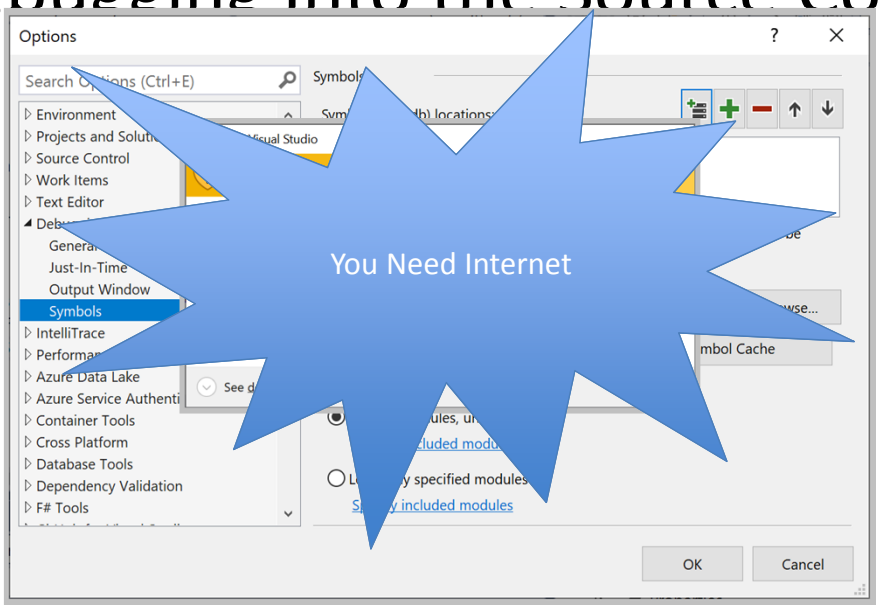
```
@Html.TextBoxFor(
    p => p.FirstName, new { @class = "" })
```

```
<input type="text"
    asp-for="FirstName" class="" />
```

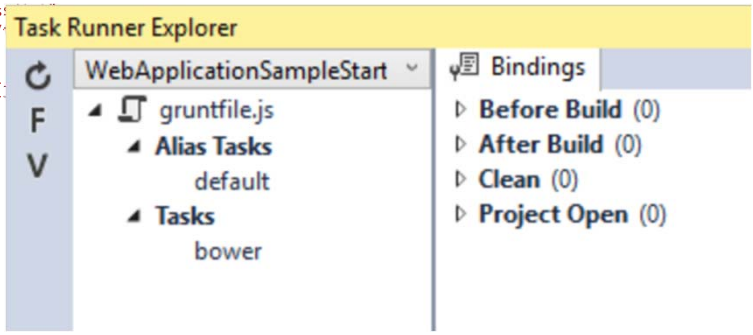
Step Into .NET Core Source Code

DEBUGGING

Debugging Into the Source Code



```
{
    "name": "bower",
    "license": "Apache-2.0",
    "private": true,
    "dependencies": {

    },
    "exportsOverride": {
        "bootstrap": {
            "js": "dist/js/*.*",
            "css": "dist/cs
            "fonts": "dist/
        },
        "jquery": {
            "js": "jquery.{
        }
    }
}
```



DEMO

Links

- Presentation Source Code
  - https://github.com/IntelliTect/Articles/
- ASP.NET Source Code
  - https://github.com/aspnet

MIDDLEWARE

Componentization via App.Use()

- Adds a middleware delegate defined inline to the request pipeline.
- Generally include a call to Invoke() within all App.Use() delegates: await next.Invoke();

```
app.Use(async (context, next) =>
{
    await context.Response.WriteAsync(
        @"Use Block 1\Step 1<br/>");
    await context.Response.WriteAsync(
        @"\t next.Invoke()<br/>");
    await next.Invoke();
    await context.Response.WriteAsync(
        @"Use Block 1\Step 2<br/>");
});
```

App.Map(), App.MapWhen()

- Used for branching the pipeline
- Matches request delegates based on a request path in the URL
- Generally includes a AppBuilder.Run() or AppBuilder.Use()

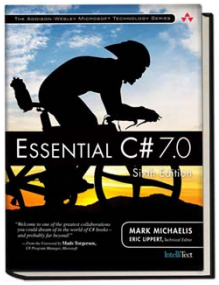
```
app.Map("/Hello", helloAppBuilder =>
{
  helloAppBuilder.Use(
     async (context, next) =>
  {
     await context.Response.WriteAsync(
         "Hello World from Hello");
     await next.Invoke();
  });
});
```

Mark Michaelis

Chief Technical Architect,
Author, Trainer

mark@IntelliTect.com

Twitter: @MarkMichaelis,
fb.com/MarkMichaelis



Microsoft® Most Valuable Professional

Microsoft Regional Director

Professional Scrum Master II

@IntelliTect, fb.com/IntelliTect