Understanding ASP.NET Core Initialization

Latest Stories in Your Inbox

by Ed Charbeneau | Á July 20, 2016 | .NET, Web, Developer Central | À 2 Comments

É	ċ
á	ö

ASP.NET Core ushers in a new generation of ASP.NET applications. This new generation of application can run on many platforms such as Windows, Mac, Linux and Docker. Running cross platform means dropping the need for Internet Information Services (IIS) to host the application.

In addition, ASP.NET Core has been engineered with dependency injection and a modular HTTP middleware pipeline to facilitate application services. Together these changes affect the way that ASP.NET Core applications are structured and initialized. Let's look at the startup process and get an understanding where all of the pieces fit and why each is important.

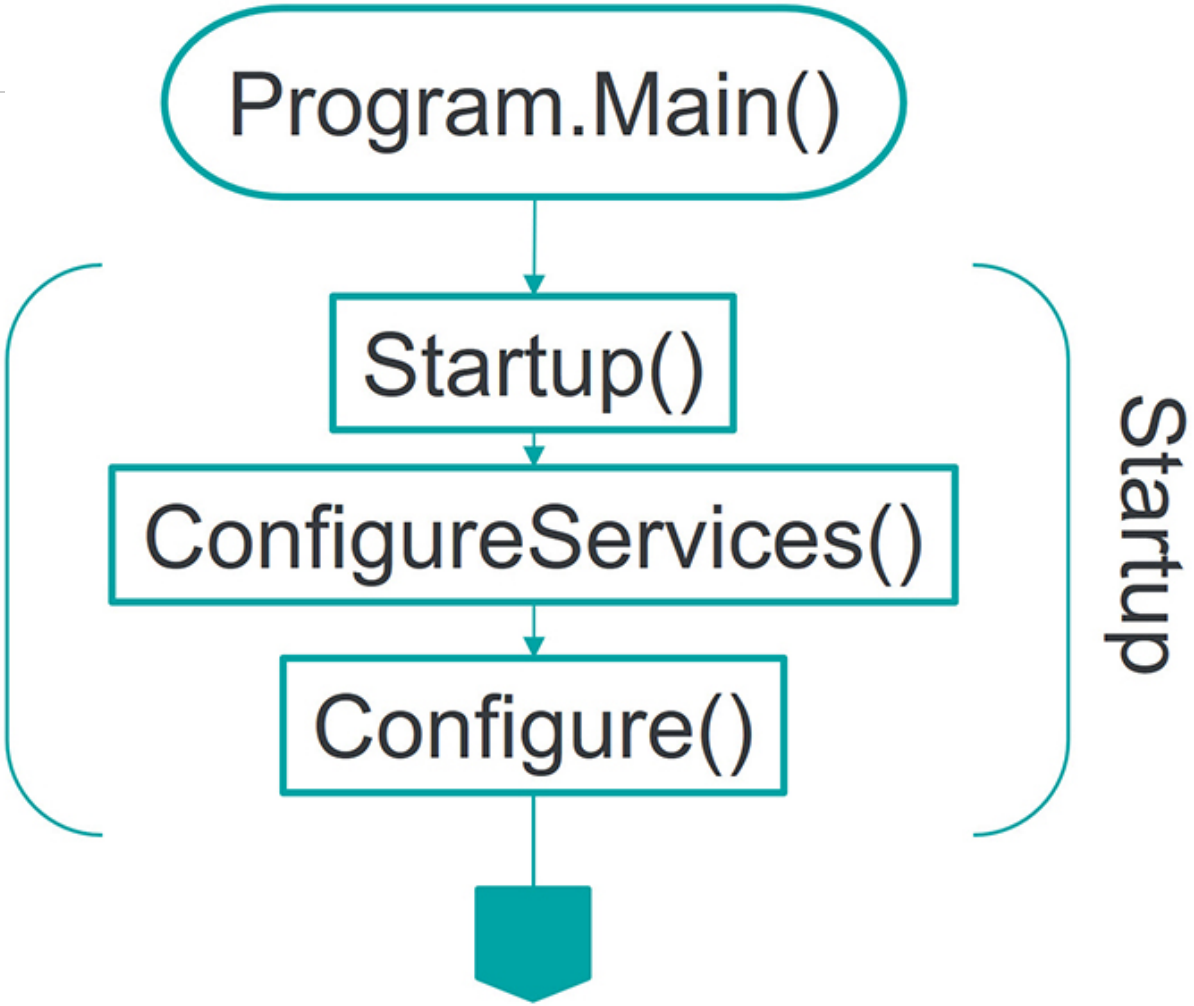
ASP.NET Core as a Console Application

Prior to ASP.NET Core, ASP.NET Framework applications were loaded by IIS. The executable InetMgr.exe creates and calls a managed web application's entry point. The HttpApplication.Application_Start() event is fired during this initialization process. A developer's first chance to execute code was to handle the Application_Start event in Global.asax.

In ASP.NET Core, the Global.asax file is no longer available as has been replaced with a new initialization process.

Latest Stories
in Your Inbox

Ú



Latest Stories in Your Inbox

ASP.NET Core applications are a .NET Core Console application that calls into ASP.NET specific libraries. This is a fundamental change in how ASP.NET Core applications are developed. Instead of the application being hosted by IIS, all of the ASP.NET hosting libraries are executed from within Program.cs. This means that a single .NET tool chain can be used for both .NET Core Console applications and ASP.NET Core applications. The benefits of the new architecture allow for more control by the developer over initialization and the hosting environment.

```
public class Program
    {
        public static void Main(string[] args)
        {
            var host = new WebHostBuilder()
                .UseKestrel()
                .UseContentRoot(Directory.GetCurrentDirectory())
                .UseIISIntegration()
                .UseStartup<Startup>() // Invokes Configure & ConfigureServices on Startup
                .Build();

            host.Run();
        }
    }
```

In this excerpt from the Main method of an ASP.NET Core app, Main is responsible for initializing the web host, invoking startup and running the app. Additional settings and hosting functionality can be configured and added to the WebHostBuilder. The host will invoke the Configure and ConfigureServices methods on the application's Startup class.

Startup

ASP.NET Core provides complete control over the application's life cycle. The Startup class is called by the Main method of the application, this is where configuration takes place and services for the application are specified.

When the Startup class is initialized, the application's settings are fetched from appsettings by the ConfigurationBuilder. After the settings are read, the ConfigureServices method is invoked allowing services to be resolved from the dependency injection container and created before configuration

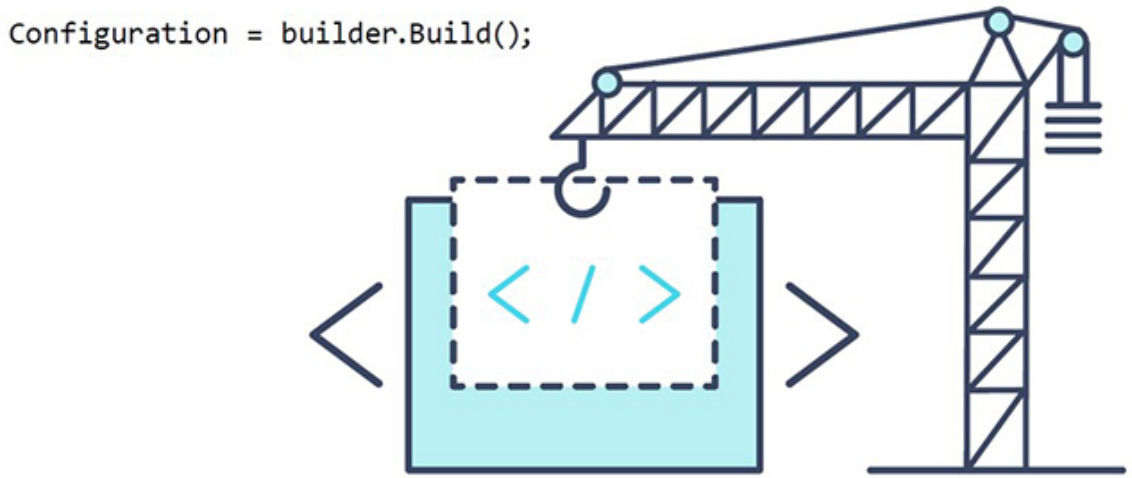
takes place.

Next the Configure method is invoked. Here application settings are applied and middleware components are registered in the application pipeline.

Latest Stories in Your Inbox

Ú

```
public Startup(IHostingEnvironment env)
{
    var builder = new ConfigurationBuilder()
        .SetBasePath(env.ContentRootPath)
        .AddJsonFile("appsettings.json", optional: true, reloadOnChange: true)
        .AddJsonFile($"appsettings.{env.EnvironmentName}.json", optional: true)
        .AddEnvironmentVariables();
    Configuration = builder.Build();
}
```



ConfigureServices

The ConfigureServices method is where services are added to the application's service container. ASP.NET Core uses a dependency injection container to resolve dependencies versus hard-coding concrete implementations. Middleware, frameworks, authentication services and more can be added to the services container.

Latest Stories in Your Inbox

Ú

```
public void ConfigureServices(IServiceCollection services)
{
    // Add framework services.
    services.AddDbContext<ApplicationDbContext>(options =>
        options.UseSqlServer(Configuration.GetConnectionString("DefaultConnection")));

    services.AddIdentity<ApplicationUser, IdentityRole>()
        .AddEntityFrameworkStores<ApplicationDbContext>()
        .AddDefaultTokenProviders();

    services.AddMvc();

    // Add application services.
    services.AddTransient<IEmailSender, AuthMessageSender>();
    services.AddTransient<ISmsSender, AuthMessageSender>();
}
```

In the example above, the call to AddMvc will add the necessary services for MVC to the services container. Individual application services are added with the Add[Transient|Scoped|Singleton] methods. These methods map an abstract type to a concrete service for a specific lifetime. Choosing the right lifetime - Transient, Scoped, Singleton - will depend on how your service is intended to interact with the application.

```
// method signature services.AddTransient<TService, TImplementation>();
services.AddTransient<TService, TImplementation>();
```

When adding services, the types are specified where TService is the Type, typically an interface, and TImplementation is the concrete class that will be used to satisfy the interface.

Configure

In the request pipeline, middleware components are initialized. The Configure method is used to specify how the ASP.NET application will respond to individual HTTP requests in the form of middleware.

Middleware in ASP.NET Core are built from extension methods on IApplicationBuilder and give developers direct access to HTTP requests. Typical Use*Action* middleware will handle requests and responses and pass the request on to the next component in the pipeline for further action. In addition, Run and Map middleware delegates are used for less common scenarios - these delegates will terminate or branch the request pipeline. Additional services like IHostingEnvironment and ILoggerFactory may also be configured, these services are initialized in the ConfigureServices method.

Latest Stories in Your Inbox

Ú

```
public void Configure(IApplicationBuilder app, IHostingEnvironment env, ILoggerFactory loggerFact
{
    loggerFactory.AddConsole(Configuration.GetSection("Logging"));
    loggerFactory.AddDebug();

    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
        app.UseDatabaseErrorPage();
        app.UseBrowserLink();
    }
    else
    {
        app.UseExceptionHandler("/Home/Error");
    }

    app.UseStaticFiles();

    app.UseIdentity();

    // Add external authentication middleware below. To configure them please see http://go.microsof

    app.UseMvc(routes =>
    {
        routes.MapRoute(
            name: "default",
            template: "{controller=Home}/{action=Index}/{id?}");
    });
}
```

Latest Stories in Your Inbox

Ú

Each Use extension method adds middleware to the request pipeline. If we look at the Configure method generated by the ASP.NET Core project template, we'll see several pieces of middleware have already been defined for us. Because of the modular design of IApplicationBuilder, middleware can be added for specific deployment scenarios such as development and production. In the default project template, alternative exception handling is added with UseDeveloperExceptionPage and UseDatabaseErrorPage middleware.

Other middleware used by default includes:

- UseStaticFiles allows the application to serve static resources.
- UseIdentity enables ASP.NET authentication for the application.
- UseMvc adds MVC to the pipeline and allows configuration of routing.

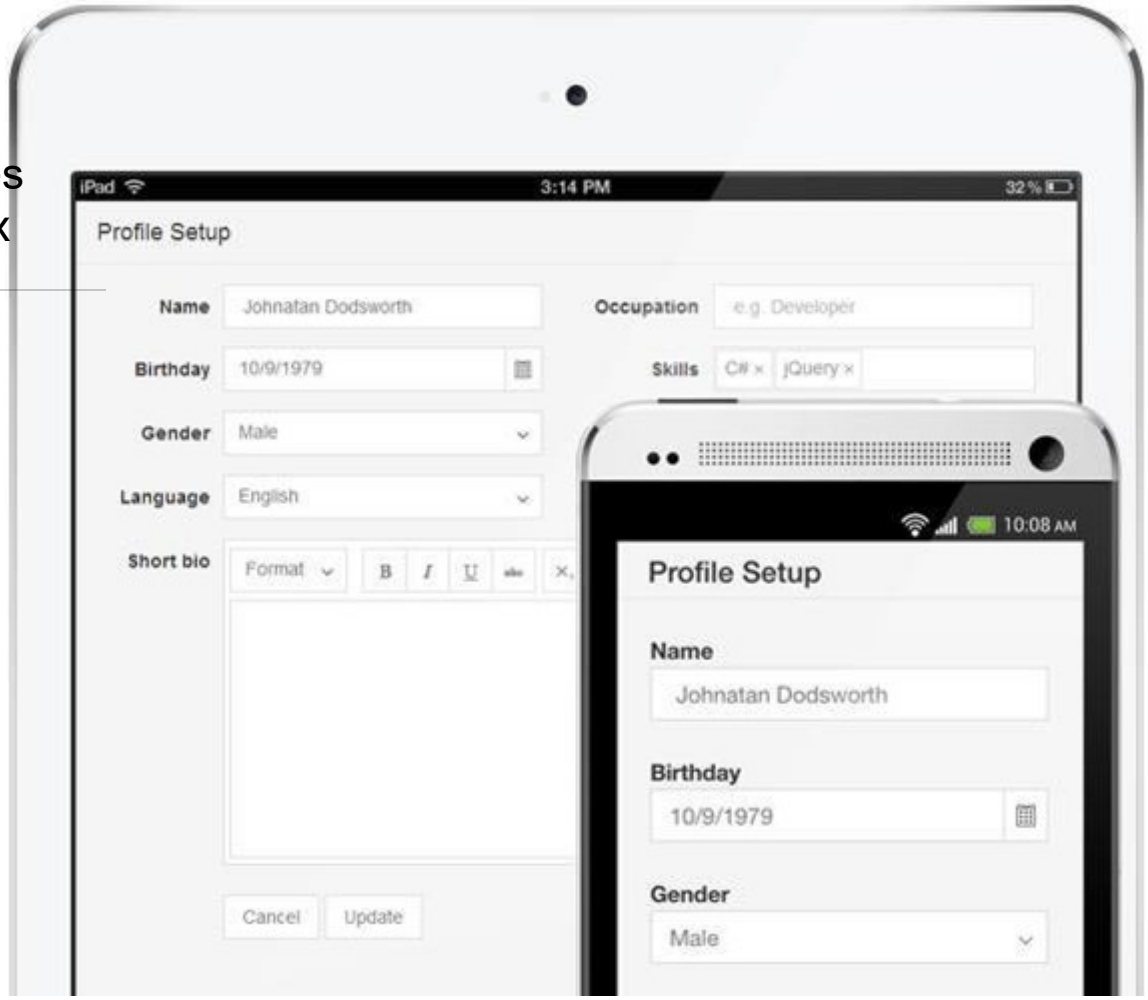
This ability to add application features ad-hoc gives developers the flexibility to adapt the application to fit their needs.

Telerik UI for ASP.NET MVC

Now Compatable with Cross Platform Development on ASP.NET Core

Latest Stories
in Your Inbox

Ú



Telerik UI for ASP.NET MVC is a comprehensive collection of over 70 user interface components built with Kendo UI. Just as ASP.NET Core can run on Windows, Mac, Linux, and Docker, so can UI for MVC. In addition to supporting the cross-platform deployment model of .NET Core, UI For MVC is also able to render for any screen size, on any device using responsive and adaptive rendering techniques.

Even though UI for MVC is primarily a user interface library, it still needs access to application configurations and the HTTP pipeline for activities like making calls from JavaScript to the server endpoints.

Plugging Telerik controls into the application is straight forward, just like other ASP.NET Core components. Start by adding the Kendo UI services to the services container.

Latest Stories in Your Inbox

Ú

```
public void ConfigureServices(IServiceCollection services)
{
    ...

    // Add Kendo UI services to the services container
    services.AddKendo();
}
```

Next, the Kendo UI configuration is set.

```
public void Configure(IApplicationBuilder app, IHostingEnvironment env, ILoggerFactory loggerFact
{
    ...

    // Configure Kendo UI
    app.UseKendo(env);
}
```

Wrapping up

ASP.NET Core is a new generation of ASP.NET applications built to run everywhere. It allows greater flexibility and control over the initialization, configuration and customization. Each part of the application's initialization is designed to be intuitive with a clear separation of responsibility.

For more details on all aspects of ASP.NET Core development, download the free whitepaper "ASP.NET Core MVC Changes Every Developer Should Know".

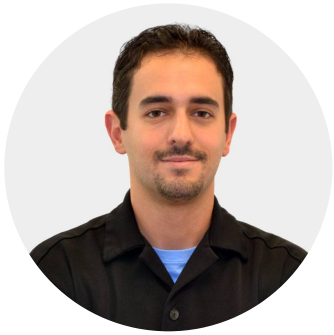
If you'd like to learn more about using the .NET CLI for cross-platform development with .NET, check out our whitepaper, The Command Line: Reinvented for Modern Developers.

Image by Norlando Pobre https://flic.kr/p/4XTMLu, modified by Ed Charbeneau

Latest Stories in Your Inbox

a .NET, ASP.NET MVC, Product Topics, Tutorials, Developer Network

Ú



ABOUT THE AUTHOR

Ed Charbeneau

Ed Charbeneau is a web enthusiast, speaker, writer, design admirer, and Developer Advocate for Telerik. He has designed and developed web based applications for business, manufacturing, systems integration as well as customer facing websites. Ed enjoys geeking out to cool new tech, brainstorming about future technology, and admiring great design. Ed's latest projects can be found on GitHub.

RELATED POSTS

.NET WEB DEVELOPER CENTRAL

Migrating from MVC to Blazor

Latest Stories in Your Inbox

DEVELOPER CENTRAL .NET

New FREE eBook: Unit Testing in .NET—The Complete Guide

.NET MOBILE DEVELOPER CENTRAL

Phone Dialer and Sending Emails & SMS in Xamarin Forms

COMMENTS

ALSO ON **TELERIK BLOGS**

Getting Started Text-To-Speech Xamarin …

11 days ago • 2 comments

Let's learn how to work with the built-in text-to-speech

A Closer Look at the Fiddler Everywhere …

2 months ago • 1 comment

Learn how to use the Fiddler Everywhere Composer

How to Add gRPC to Your Blazor App

a month ago • 5 comments

With Microsoft's announcement of gRPC-

Zone Proximal Development Build …

25 days ago • 6 comments

The best way to increase a new software engineer's

Optimizing Website w

2 months ago

Fiddler Every used to inspe

engines in Xamarin Forms. | feature to create and … | Web, they have extended … | productivity is by … | web traffic, b

2 Comments Telerik Blogs Disqus' Privacy Policy Login
Latest Stories in Your Inbox
Recommend Tweet Share Sort by Best
Join the discussion…
LOG IN WITH
OR SIGN UP WITH DISQUS
Name

Alex Dresko • 4 years ago

Good afternoon!

I really want to enjoy this article, but I'm having too much trouble understanding a few things. Maybe I should stop installing SQL server while I read it... :)

Still.. Here are some thoughts for consideration...

First, you say "In this excerpt from the Main method of an ASP.NET Core app, Main is responsible for configuring and running the app."

Then you follow up with "this [Startup] is where configuration takes place and services for the application are specified."

Seems like conflicting information.

Later... "When the Startup class is initialized, the application's settings are resolved by the ConfigurationBuilder. After the settings are resolved, the ConfigureServices method is invoked allowing services to be resolved and created before configuration takes place."

What??? How is it initialized, specifically? "the application's settings are resolved by the ConfgurationBuild" leaves a lot to the imagination. What are "services" in this context? And, again, what is "resolved" here? A pic/graphic would be SUPER helpful in this paragraph.

That's honestly as far as I got.. I'm extremely well versed in ASP.NET WebForms and MVC.... I very much understand all of the high level concepts around this Core stuff, but I really want to understand it's initialization. Hoping you can clarify this article a bit more so I can continue reading..

Latest Stories in Your Inbox

And I appreciate the comment helps...

Reply · Share ›

Ed Charbeneau ➚ Alex Dresko • 4 years ago

Great feedback **@Alex Dresko**

I tried to clear things up, there's a lot of the word "Configure" used in the initialization process, things get to be a mouth full really quick.

TECHNOLOGIES

- .NET
- JavaScript

TOPICS

- Developer Central
- Web
- Mobile
- Desktop
- Testing
- Reporting

Fiddler

Latest Stories
in Your Inbox

search blogs...

Ú

Progress

Telerik and Kendo UI are part of Progress product portfolio. Progress is the leading provider of application development and digital experience technologies.

Latest Stories In Your Inbox

Company Technology Awards Press Releases Media Coverage Careers Offices

Copyright © 2020, Progress Software Corporation and/or its subsidiaries or affiliates. All Rights Reserved.

Progress, Telerik, Ipswitch, and certain product names used herein are trademarks or registered trademarks of Progress Software Corporation and/or one of its subsidiaries or affiliates in the U.S. and/or other countries. See Trademarks for appropriate markings.