Introducing ASP.NET 5

Monday, February 23, 2015

[**.NET**](http://weblogs.asp.net/scottgu/Tags/.NET) [**ASP.NET**](http://weblogs.asp.net/scottgu/Tags/ASP.NET) [**Community News**](http://weblogs.asp.net/scottgu/Tags/Community%20News)

The first preview release of ASP.NET 1.0 came out almost 15 years ago.  Since then millions of developers have used it to build and run great web applications, and over the years we have added and evolved many, many capabilities to it.

I'm excited today to post about a new release of ASP.NET that we are working on that we are calling ASP.NET 5.  This new release is one of the most significant architectural updates we've done to ASP.NET.  As part of this release we are making ASP.NET leaner, more modular, cross-platform, and cloud optimized.  The ASP.NET 5 preview is now available as a preview release, and you can start using it today by [downloading](http://go.microsoft.com/fwlink/?LinkId=521794) the latest CTP of Visual Studio 2015 which we just made available.

ASP.NET 5 is an open source web framework for building modern web applications that can be developed and run on Windows, Linux and the Mac. It includes the MVC 6 framework, which now combines the features of MVC and Web API into a single web programming framework.  ASP.NET 5 will also be the basis for SignalR 3 - enabling you to add real time functionality to cloud connected applications. ASP.NET 5 is built on the .NET Core runtime, but it can also be run on the full .NET Framework for maximum compatibility.

With ASP.NET 5 we are making a number of architectural changes that makes the core web framework much leaner (it no longer requires System.Web.dll) and more modular (almost all features are now implemented as NuGet modules - allowing you to optimize your app to have just what you need).  With ASP.NET 5 you gain the following foundational improvements:

* Build and run cross-platform ASP.NET apps on Windows, Mac and Linux
* Built on .NET Core, which supports true side-by-side app versioning
* New tooling that simplifies modern Web development
* Single aligned web stack for Web UI and Web APIs
* Cloud-ready environment-based configuration
* Integrated support for creating and using NuGet packages
* Built-in support for dependency injection
* Ability to host on IIS or self-host in your own process

The end result is an ASP.NET that you'll feel very familiar with, and which is also now even more tuned for modern web development.

Flexible, Cross-Platform Runtime

ASP.NET 5 works with two runtime environments to give you greater flexibility when hosting your app. The two runtime choices are:

**.NET Core**– a new, modular, cross-platform runtime with a smaller footprint.  When you target the .NET Core, you’ll be able to take advantage of some exciting new benefits:

1) You can deploy the .NET Core runtime with your app which means your app will run with this deployed version of the runtime rather than the version of the runtime that is installed on the host operating system. Your version of the runtime runs side-by-side with versions for other apps. You can update that runtime, if needed, without affecting other apps, or you can continue running on the same version even though other apps on the system have been updated.  This makes app deployment and framework updates much easier and less impactful to other apps running on a system.

2) Your app is only dependent on features it really needs. Therefore, you are never prompted to update/service the runtime for features that are not relevant to your app. You will spend less time testing and deploying updates that are perhaps unrelated to the functionality of your app.

3) Your app can now be run cross-platform. We will provide a cross-platform version of .NET Core for Windows, Linux and Mac OS X systems.  Regardless of which operating system you use for development or which operating system you target for deployment, you will be able to use .NET. The cross-platform version of the runtime has not been released yet, but we are working on it on [GitHub](https://github.com/dotnet/coreclr) and plan to have an official preview of it out soon.

**.NET Framework**– The API for .NET Core is currently more limited than the full .NET Framework, so you may need to modify existing apps to target .NET Core. If you don't want to have to update your app you can instead run ASP.NET 5 applications on the full .NET Framework (version 4.5.2 and above).  When doing this you have access to the complete set of .NET Framework APIs. Your existing applications and libraries will work without modification on this runtime.

MVC 6 - a unified programming model

MVC, Web API and Web Pages provide complementary functionality and are frequently used together when developing a solution. However, in past ASP.NET releases, these programming frameworks were implemented separately and therefore contained some duplication and inconsistencies. With MVC 6, we are merging those models into a single programming model. Now, you can create a single web application that handles the Web UI and data services without needing to reconcile differences in these programming frameworks. You will also be able to seamlessly transition a simple site first developed with Web Pages into a more robust MVC application.

You can now return Razor views and content-negotiated data from the same controller and using the same MVC filter pipeline.

In addition to unifying the existing frameworks we are also adding new features to make server-side Web development easier, like the new tag helpers feature. Tag helpers let you use HTML helpers in your views by simply extending the semantics of tags in your markup.

So instead of writing this:

@Html.ValidationSummary(true, "", new { @class = "text-danger" })

<div class="form-group">

    @Html.LabelFor(m => m.UserName, new { @class = "col-md-2 control-label" })

    <div class="col-md-10">

        @Html.TextBoxFor(m => m.UserName, new { @class = "form-control" })

        @Html.ValidationMessageFor(m => m.UserName, "", new { @class = "text-danger" })

    </div>

</div>

You can instead write this:

<div asp-validation-summary="ModelOnly" class="text-danger"></div>

<div class="form-group">

    <label asp-for="UserName" class="col-md-2 control-label"></label>

    <div class="col-md-10">

        <input asp-for="UserName" class="form-control" />

        <span asp-validation-for="UserName" class="text-danger"></span>

    </div>

</div>

Tag helpers make authoring your views more natural and readable. They also simplify customizing the output of HTML helpers with additional markup while letting you take full advantage of the HTML editor.

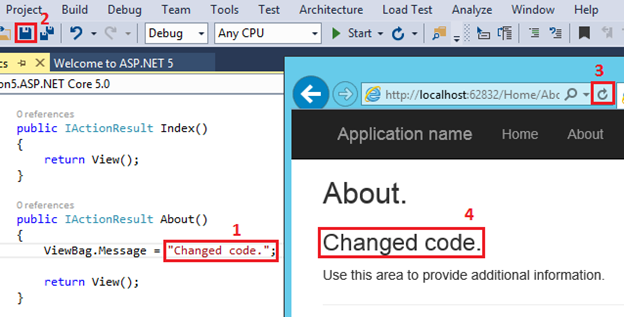
For more examples of creating MVC 6 apps, see these [tutorials](http://www.asp.net/vnext/overview/aspnet-vnext).

Modern web development

This week's ASP.NET 5 preview also includes a number of other great development features that enable you to build even better web applications:

**Dynamic Development**

In Visual Studio 2015, we take advantage of dynamic compilation to provide a streamlined developer experience. You no longer have to compile your application every time you want to see a change. Instead, just (1) edit the code, (2) save your changes, (3) refresh the browser, and then (4) see your change automatically appear.

[](https://mscblogs.blob.core.windows.net/media/scottgu/WindowsLiveWriter/IntroducingASP.NET5_12282/image_2.png)

You enjoy a development experience that is similar to working with an interpreted language without sacrificing the benefits of a compiled language.

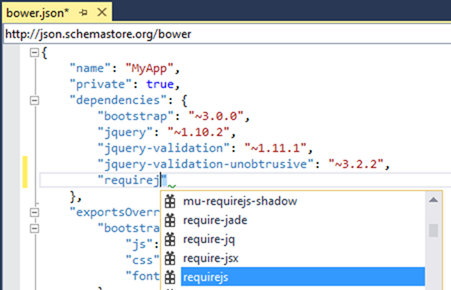
You can also optionally use other code editors to work on your ASP.NET 5 projects. Every function within the Visual Studio user interface is matched with cross-platform command-line operations.

**Integration with Popular Web Development Tools (Bower, Grunt and Gulp)**

Another exciting feature in Visual Studio 2015 is built-in support for Bower, Grunt, and Gulp - popular open source tools that we think should be in every Web developer’s toolkit.

* Bower is a package manager for client-side libraries, including both JavaScript and CSS libraries.
* Grunt and Gulp are task runners, which help you to automate your web development workflow. You can use Grunt or Gulp for tasks like compiling LESS, CoffeeScript, or TypeScript files, running JSLint, or minifying JavaScript files.

**Bower:**To add a JavaScript library to your ASP.NET project add it directly in the bower.json config file:

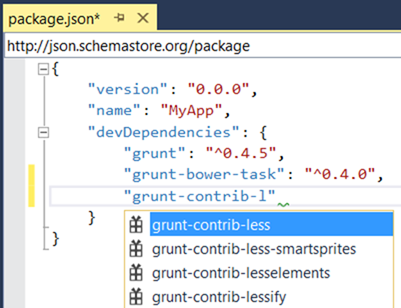
[](https://mscblogs.blob.core.windows.net/media/scottgu/WindowsLiveWriter/IntroducingASP.NET5_12282/image_14.png)

Notice that Visual Studio gives you IntelliSense with a list of available packages. The next time you open the solution, Visual Studio automatically restores any missing packages, so you don’t need to check the packages into source control.

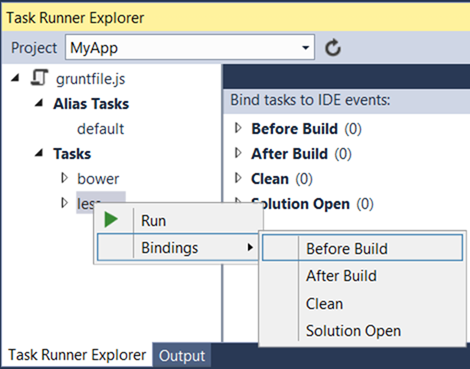
For server-side packages, you’ll still use NuGet Package Manager.

**Grunt:**In modern web development, you can find yourself managing a lot of tasks, just to build your app: Compiling LESS, TypeScript, or CoffeeScript files, linting, JavaScript minification, running JS unit tests, and so on. Every team will have its own set of requirements, depending on the particular tools that you use. Task runners make it easier to manage and coordinate these tasks. Visual Studio 2015 will support two popular task runners, Grunt and Gulp.

For example, let’s say you want to use Grunt to compile LESS files. Just go into package.json and add the [grunt-contrib-less](https://github.com/gruntjs/grunt-contrib-less) package, which is a third-party Grunt plugin.

[](https://mscblogs.blob.core.windows.net/media/scottgu/WindowsLiveWriter/IntroducingASP.NET5_12282/image_16.png)

Use the new Task Runner Explorer in Visual Studio 2015 to bind the task to a build step (pre-build, post-build, clean, or when the solution is opened).

[](https://mscblogs.blob.core.windows.net/media/scottgu/WindowsLiveWriter/IntroducingASP.NET5_12282/image_18.png)

This makes it incredibly easy to automate common tasks within your projects - and have them work both for you, as well as across a team wide project.

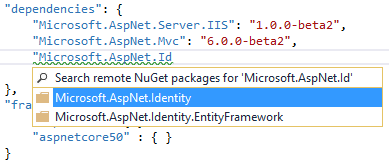
**Simplified dependency management**

In ASP.NET 5 you manage dependencies by adding NuGet packages. You can use the NuGet Package Manager or simply edit the JSON file (project.json) that lists the NuGet packages and versions used in your project. The project.json file is easy to work with and you can edit it with any text editor, which enables you to update dependencies even when the app has been deployed to the cloud.

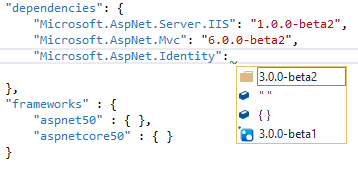
The project.json file looks like:

[](https://mscblogs.blob.core.windows.net/media/scottgu/WindowsLiveWriter/IntroducingASP.NET5_12282/image_4.png)

In Visual Studio 2015, IntelliSense assists you with finding the available NuGet packages that you can add as dependencies.

[](https://mscblogs.blob.core.windows.net/media/scottgu/WindowsLiveWriter/IntroducingASP.NET5_12282/image_6.png)

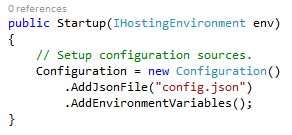
And, Intellisense can even help you with the available versions:

[](https://mscblogs.blob.core.windows.net/media/scottgu/WindowsLiveWriter/IntroducingASP.NET5_12282/image_8.png)

**Cloud-ready configuration**

In ASP.NET 5, we eliminated the need to use Web.config file for configuration values. We wanted to make it easier for you to deploy your app to the cloud and have the app automatically read the correct configuration values for that environment. The new system enables you to request named values from a variety of sources (such as JSON, XML, or environment variables). You can decide which formats work best in your situation.

In the Startup.cs file, you can now add or remove the sources for configuration values.

[](https://mscblogs.blob.core.windows.net/media/scottgu/WindowsLiveWriter/IntroducingASP.NET5_12282/image_10.png)

The above code snippet shows a project that is set up to retrieve configuration values from a JSON file and environmental variables. You can change this code if you need to specify other sources. In the specified config.json file, you could provide the values.

[](https://mscblogs.blob.core.windows.net/media/scottgu/WindowsLiveWriter/IntroducingASP.NET5_12282/image_12.png)

In your host environment, such as Azure, you can set the environmental variables and those values are automatically used instead of local configuration values after the application is deployed. You can deploy your application without worrying about publishing test values.

**Dependency injection (DI)**

Dependency Injection (DI) is supported in existing ASP.NET frameworks, like MVC, Web API and SignalR, but not in a consistent and holistic way. ASP.NET 5 provides a built-in DI abstraction that is available in a consistent way throughout the entire web stack. You can access services at startup, in middleware, in filters, in controllers, in model binding and virtually any part of the pipeline where you want to use your services. ASP.NET 5 includes a minimalistic DI container to bootstrap the system, but you can easily replace the default container with your container of choice (Autofac, Ninject, etc). Services can be singleton, scoped to the request or transient.

For example, to see how to use constructor injection with ASP.NET MVC 6, create a new ASP.NET 5 Starter Web project and add a simple time service:

using System;

namespace WebApplication1

{

    public class TimeService

    {

        public TimeService()

        {

            Ticks = DateTime.Now.Ticks.ToString();

        }

        public String Ticks { get; set; }

    }

}

The simple service class sets the current Ticks when the constructor is called.

Next, register the time service as a transient service in the ConfigureServices method of the Startup class:

public void ConfigureServices(IServiceCollection services)

{

    services.AddMvc();

    services.AddTransient<TimeService>();

}

Then, update the HomeController to use constructor injection and to write the Ticks when the TimeService object was created.

public class HomeController : Controller

{

    public TimeService TimeService { get; set; }

    public HomeController(TimeService timeService)

    {

        TimeService = timeService;

    }

    public IActionResult About()

    {

        ViewBag.Message = TimeService.Ticks + " From Controller";

        System.Threading.Thread.Sleep(1);

        return View();

    }

    // Code removed for brevity

}

Notice the controller doesn't create a TimeService. It's injected when the controller is instantiated.

In MVC 6 you can use the [Activate] attribute to inject services via properties. You can use [Activate] not just on controllers but also on filters, and view components. This means you can simplify your controller code like this:

public class HomeController : Controller

{

    [Activate]

    public TimeService TimeService { get; set; }

    // Code removed for brevity

}

MVC 6 also supports DI into Razor views via the @inject keyword. In the code below, I’ve injected the time service into the about view directly and defined a TimeSvc property by which it can be accessed:

@using WebApplication23

@inject TimeService TimeSvc

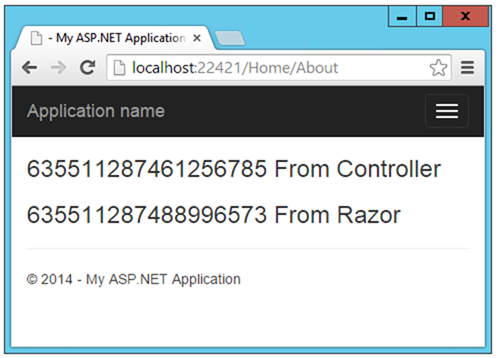
<h3>@ViewBag.Message</h3>

<h3>

    @TimeSvc.Ticks From Razor

</h3>

When you run the app, you can see different ticks values from the controller and the view.

[](https://mscblogs.blob.core.windows.net/media/scottgu/WindowsLiveWriter/IntroducingASP.NET5_12282/image_20.png)

Fast HTTP performance

ASP.NET 5 introduces a new HTTP request pipeline that is modular so you can add only the components that you need. The pipeline is also no longer dependent on System.Web. By reducing the overhead in the pipeline, your app can experience better throughput and a more tuned HTTP stack. The new pipeline is based on many of the learnings from the Katana project and also supports [OWIN](http://owin.org/).

To customize which components are used in the pipeline, use the Configure method in your Startup class. The Configure method is used to specify which middleware you want to “use” in your request pipeline. ASP.NET 5 already includes ported versions of many of the middleware from the Katana project, like middleware for static files, authentication and diagnostics. The following image shows some of the features you can add or remove to the pipeline for your project.

public void Configure(IApplicationBuilder app)

{

    // Add static files to the request pipeline.

    app.UseStaticFiles();

    // Add cookie-based authentication to the request pipeline.

    app.UseIdentity();

    // Add MVC and routing to the request pipeline.

    app.UseMvc(routes =>

    {

    routes.MapRoute(

        name: "default",

        template: "{controller}/{action}/{id?}",

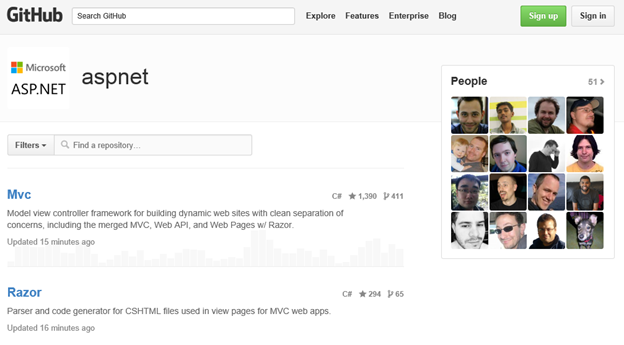
        defaults: new { controller = "Home", action = "Index" });

});

You can also write your own middleware components and add them to the pipeline.

Open source

We are developing ASP.NET 5 as an open source project on [GitHub](https://github.com/aspnet/home). You can view the code, see when changes were made, download the code, and submit changes. We believe making ASP.NET 5 open source will we make it easier for you to understand the code, understand our intended direction, and contribute to the project.

[](https://mscblogs.blob.core.windows.net/media/scottgu/WindowsLiveWriter/IntroducingASP.NET5_12282/image_22.png)

Docs and tutorials

To get started with ASP.NET 5 you can find docs and tutorials on the ASP.NET site at <http://asp.net/vnext>. The following tutorials will guide you through the steps of creating your first ASP.NET 5 project.

* [Manage Client-Side Web Development in Visual Studio 2015, Using Grunt and Bower](http://www.asp.net/vnext/overview/aspnet-vnext/grunt-and-bower-in-visual-studio-2015)
* [View components and Inject in ASP.NET MVC 6](http://www.asp.net/vnext/overview/aspnet-vnext/vc)

Also read [this article](http://blogs.msdn.com/b/webdev/archive/2015/02/23/aspnet-5-updates-for-feb-2015.aspx) for even more ASP.NET and Web Development improvements coming this week.

Hope this help,

Scott

At TechEd, Microsoft took the wraps off of the new ASP.NET server-side components in a package they’re calling ASP.NET vNext. Formerly, this was called Project K, and you’ll see some insiders use both of these terms for the next few months. This project contains some very exciting capabilties for web developers, and the promise within the coming changes is impressive.

Introducing OWIN

OWIN, [the Open Web Interface for .NET](http://www.owin.org/) is taking over the ASP.NET runtime. Microsoft is embracing the open standard of constructing and assembling web components, and allowing you to assemble as opinionated or flexible a web stack as you need. Web projects now have access to an immense collection of building blocks, and you can assemble them any way you would like.

ASP.NET MVC vNext is now OWIN compliant. That means you can run MVC applications on a very thin Katana server, or on a Project Helios managed IIS instance. These are amazing choices that you can make, as you no longer will be obligated to load packages and assemblies that you don’t need for your application.

Web Applications will now be delivered with 4 distinct layers:

1. *The host application* – in the past this was ALWAYS Microsoft Internet Information Server (IIS). This could be IIS, IIS with Helios, Katana, or some other OWIN capable host.
2. *The CLR framework* – previously you would choose which version of the Microsoft.NET framework you wanted to use. This may be .NET 4.5.2, K runtime, or Mono.
3. *The Application Host* – previously, this would be the W3C executable. Now, this can be a host for the Visual Studio designer, or the new Microsoft.Host.Runtime.
4. *Your Application* – Your application now loads the former ASP.NET frameworks (MVC, WebAPI, or SignalR) as part of its dependencies.

MVC – Meet WebAPI, WebAPI meet MVC

If you’ve been working with ASP.NET for a few years, you’ve no doubt noticed that the MVC and WebAPI frameworks are very similar. In fact, they use the same Controller syntax, with WebAPI responding to controllers actions that are named like HTTP Verbs. In the next version of ASP.NET, this mix-and-match scenario goes away. There will be one framework that provides this Controller-centric server-side capabilities. All of these capabiltes are housed in the new Microsoft.AspNet.Mvc framework.

Bye Bye System.Web

Alas after 15 years, we say goodbye to System.Web. We knew you so well….

To some, this is a welcome departure as the System.Web assembly is the second largest assembly in the base class library (BCL). This 5 megabyte monster was very difficult for the ASP.NET team to manage, and any changes to it required an update to the .NET BCL. Those changes are not deployed easily, and typically only ship with a new Visual Studio or a new version of Windows. In order to keep the ASP.NET run-time more nimble and adapting at the speed of the Internet, the ASP.NET team decided to move all new development out of System.Web.

This also means that ASP.NET web forms is not coming along for the ride into ASP.NET vNext. This does not mean that they will stop maintaining it. It does mean that the changes to it will not be as quick, and that you can still use Microsoft IIS to host your web forms application. None of that has changed or will change for a very long time.

Blistering Performance

The demonstrations on stage at TechEd were nothing short of impressive. Granted, the demos that are currently being shown are very small. The compile and execution time of the code is amazing. In fact, you don’t have to compile at all. The Roslyn compiler will automatically make those changes and deploy them for you when the code files change!

Where is web.config?

Configuration files? Who needs configuration files! The .config file architecture was an artifact of the System.Web library. At this point, why write a configuration file when you can hard-code your configuration in C# code and Roslyn will reload it faster than you can say “I saved a change to the configuration.” Look for a new configuration standard to come along soon, and it will probably be based on JSON, given the new project.json file structure. Which brings us to…

Welcome Project.JSON!

No more XML .csproj nonsense with build artifacts and configuration defined. In this next version of ASP.NET, we will have a JSON file that lists our dependencies, the configurations we want the application to run in, and the commands that we can use against the web application.

Dependencies are kicked up a whole new level, as they no longer require a compiled DLL to reside on disk. You can now add references to NuGet packages, and ASP.NET will fetch that dependency when you first deploy and run your application. This allows you to compose your application from more NuGet building blocks, and adhere to those [SOLID principles](http://en.wikipedia.org/wiki/Solid_(object-oriented_design)).

Project.json has a syntax structure similar to the following:

{

"version": "0.1",

"dependencies": {

},

"commands": {

"run": "",

"web": ""

},

"configurations": {

"net45": {}

"k10": {}

}

}

You can force your project to download and run with the NuGet packages specified in your project.json file with this syntax:

kpm restore

What does this mean for Telerik UI Customers?

At this time, we are eagerly watching the evolution of ASP.NET, and think this is going to be an amazing product for our customers. We will make whatever changes we need in order to support our [MVC helpers](http://www.telerik.com/aspnet-mvc) in this new ASP.NET. Additionally, we will continue to support and add cool, modern and mobile functionality to our [UI for ASP.NET](http://www.telerik.com/products/aspnet-ajax.aspx) controls.

You can continue to build server-side APIs and deliver single-page-application structures that host the KendoUI JavaScript framework, even in this preview build from Microsoft.

Summary

At Telerik, we are very excited about the changes being introduced by Microsoft in this next generation of ASP.NET. We will be there for you as this project from Microsoft moves towards completion. Its all part of Telerik’s commitment to support any software development in any development environment and targeting any screen.