ASP.NET and Web Tools for Visual Studio 2013 Release Notes

This document describes the release of ASP.NET and Web Tools for Visual Studio 2013.

Contents

* [Installation Notes](http://www.asp.net/visual-studio/overview/2013/release-notes#TOC1)
* [Documentation](http://www.asp.net/visual-studio/overview/2013/release-notes#TOC2)
* [Software Requirements](http://www.asp.net/visual-studio/overview/2013/release-notes#TOC4)

New Features in ASP.NET and Web Tools for Visual Studio 2013

* [One ASP.NET](http://www.asp.net/visual-studio/overview/2013/release-notes#TOC6)
* [New Web Project Experience](http://www.asp.net/visual-studio/overview/2013/release-notes#newproj)
* [ASP.NET Scaffolding](http://www.asp.net/visual-studio/overview/2013/release-notes#scaffold)
* [Browser Link](http://www.asp.net/visual-studio/overview/2013/release-notes#browser-link)
* [Visual Studio Web Editor Enhancements](http://www.asp.net/visual-studio/overview/2013/release-notes#web-editor)
* [Windows Azure Web Sites Support in Visual Studio](http://www.asp.net/visual-studio/overview/2013/release-notes#waws)
* [Web Publish Enhancements](http://www.asp.net/visual-studio/overview/2013/release-notes#publish)
* [NuGet 2.7](http://www.asp.net/visual-studio/overview/2013/release-notes#nuget)
* [ASP.NET Web Forms](http://www.asp.net/visual-studio/overview/2013/release-notes#TOC9)
* [ASP.NET MVC 5](http://www.asp.net/visual-studio/overview/2013/release-notes#TOC10)
* [ASP.NET Web API 2](http://www.asp.net/visual-studio/overview/2013/release-notes#TOC11)
* [ASP.NET SignalR](http://www.asp.net/visual-studio/overview/2013/release-notes#TOC13)
* [ASP.NET Identity](http://www.asp.net/visual-studio/overview/2013/release-notes#TOC8)
* [Microsoft OWIN Components](http://www.asp.net/visual-studio/overview/2013/release-notes#TOC7)
* [Entity Framework 6](http://www.asp.net/visual-studio/overview/2013/release-notes#ef6)
* [ASP.NET Razor 3](http://www.asp.net/visual-studio/overview/2013/release-notes#TOC14)
* [ASP.NET App Suspend](http://www.asp.net/visual-studio/overview/2013/release-notes#TOC15)
* [Known Issues and Breaking Changes](http://www.asp.net/visual-studio/overview/2013/release-notes#knownissues)

Installation Notes

ASP.NET and Web Tools for Visual Studio 2013 are bundled in the main installer and can be downloaded [here](http://www.asp.net/downloads).

Documentation

Tutorials and other information about ASP.NET and Web Tools for Visual Studio 2013 are available from the [ASP.NET web site](http://www.asp.net/).

Software Requirements

ASP.NET and Web Tools requires Visual Studio 2013.

New Features in ASP.NET and Web Tools for Visual Studio 2013

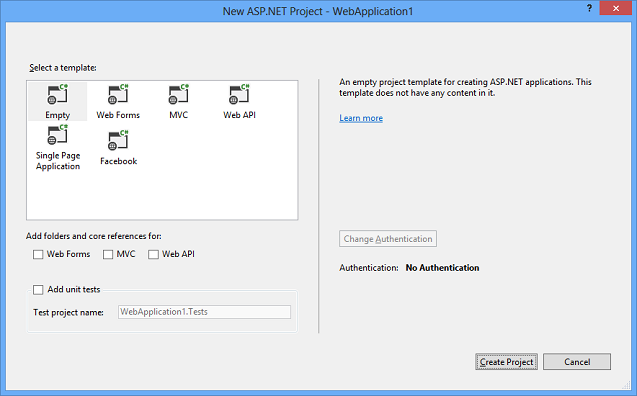
The following sections describe the features that have been introduced in the release.

One ASP.NET

With the release of Visual Studio 2013, we have taken a step towards unifying the experience of using ASP.NET technologies, so that you can easily mix and match the ones you want. For example, you can start a project using MVC and easily add Web Forms pages to the project later, or scaffold Web APIs in a Web Forms project. One ASP.NET is all about making it easier for you as a developer to do the things you love in ASP.NET. No matter what technology you choose, you can have confidence that you are building on the trusted underlying framework of One ASP.NET.

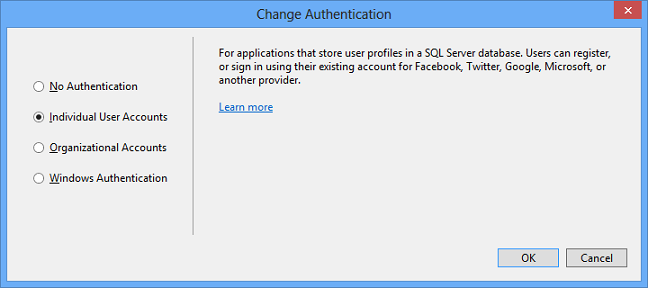
New Web Project Experience

We have enhanced the experience of creating new web projects in Visual Studio 2013. In the **New ASP.NET Web Project** dialog you can select the project type you want, configure any combination of technologies (Web Forms, MVC, Web API), configure authentication options, and add a unit test project.



The new dialog enables you to change the default authentication options for many of the templates. For example, when you create an ASP.NET Web Forms project you can select any of the following options:

* No Authentication
* Individual User Accounts (ASP.NET membership or social provider log in)
* Organizational Accounts (Active Directory in an internet application)
* Windows Authentication (Active Directory in an intranet application)



For more information about the new process for creating web projects, see [Creating ASP.NET Web Projects in Visual Studio 2013](http://www.asp.net/visual-studio/overview/2013/creating-web-projects-in-visual-studio) . For more information about the new authentication options, see [ASP.NET Identity](http://www.asp.net/visual-studio/overview/2013/release-notes#TOC8) later in this document.

ASP.NET Scaffolding

ASP.NET Scaffolding is a code generation framework for ASP.NET Web applications. It makes it easy to add boilerplate code to your project that interacts with a data model.

In previous versions of Visual Studio, scaffolding was limited to ASP.NET MVC projects. With Visual Studio 2013, you can now use scaffolding for any ASP.NET project, including Web Forms. Visual Studio 2013 does not currently support generating pages for a Web Forms project, but you can still use scaffolding with Web Forms by adding MVC dependencies to the project. Support for generating pages for Web Forms will be added in a future update.

When using scaffolding, we ensure that all required dependencies are installed in the project. For example, if you start with an ASP.NET Web Forms project and then use scaffolding to add a Web API Controller, the required NuGet packages and references are added to your project automatically.

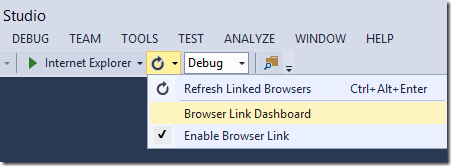
To add MVC scaffolding to a Web Forms project, add a **New Scaffolded Item** and select **MVC 5 Dependencies** in the dialog window. There are two options for scaffolding MVC; Minimal and Full. If you select Minimal, only the NuGet packages and references for ASP.NET MVC are added to your project. If you select the Full option, the Minimal dependencies are added, as well as the required content files for an MVC project.

Support for scaffolding async controllers uses the new async features from Entity Framework 6.

For more information and tutorials, see [ASP.NET Scaffolding Overview](http://www.asp.net/visual-studio/overview/2013/aspnet-scaffolding-overview).

Browser Link – SignalR channel between browser and Visual Studio

The new [Browser Link](http://www.asp.net/visual-studio/overview/2013/using-browser-link) feature lets you connect multiple browsers to Visual Studio and refresh them all by clicking a button in the toolbar. You can connect multiple browsers to your development site, including mobile emulators, and click refresh to refresh all the browsers all at the same time. Browser Link also exposes an API to enable developers to write Browser Link extensions.

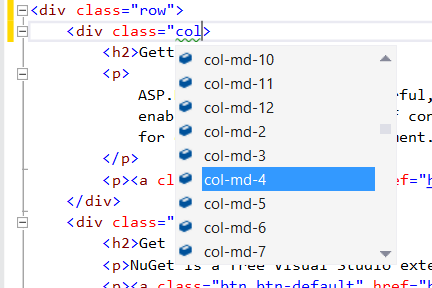


By enabling developers to take advantage of the Browser Link API, it becomes possible to create very advanced scenarios that crosses boundaries between Visual Studio and any browser that’s connected. Web Essentials takes advantage of the API to create an integrated experience between Visual Studio and the browser’s developer tools, remote controlling mobile emulators and a lot more.

Visual Studio Web Editor Enhancements

Visual Studio 2013 includes a new HTML editor for Razor files and HTML files in web applications. The new HTML editor provides a single unified schema based on HTML5. It has automatic brace completion, jQuery UI and AngularJS attribute IntelliSense, attribute IntelliSense Grouping, ID and class name Intellisense, and other improvements including better performance, formatting and SmartTags.

The following screenshot demonstrates using Bootstrap attribute IntelliSense in the HTML editor.



Visual Studio 2013 also comes with both CoffeeScript and LESS editors built in. The LESS editor comes with all the cool features from the CSS editor and has specific Intellisense for variables and mixins across all the LESS documents in the @import chain.

Windows Azure Web Sites Support in Visual Studio

In Visual Studio 2013 with the Windows Azure SDK 2.2 (expected to be available in a few days), you can use **Server Explorer** to interact directly with your remote Windows Azure Web Sites. You can sign in to your Windows Azure account, create new sites, configure existing sites, view real-time logs, and more. Coming soon after SDK 2.2 is released, you'll be able to run in debug mode remotely in Windows Azure. Most of the new features for Windows Azure Web Sites also work in Visual Studio 2012 when you install the current release of the .NET SDK for Windows Azure.

For more information, see the following resources:

* [Getting started with Windows Azure and ASP.NET](http://www.windowsazure.com/en-us/develop/net/tutorials/get-started/)
* [Troubleshooting Windows Azure Web Sites in Visual Studio](http://www.windowsazure.com/en-us/develop/net/tutorials/troubleshoot-web-sites-in-visual-studio/)

Web Publish Enhancements

Visual Studio 2013 includes new and enhanced Web Publish features. Here are a few of them:

* Easily [automate Web.config file encryption](http://go.microsoft.com/fwlink/?LinkId=325529). (This link and the following two point to documentation on MSDN that might not be available until late in the day on 10/17.)
* Easily [automate taking an application offline during deployment](http://go.microsoft.com/fwlink/?LinkId=325530).
* Configure Web Deploy to [use file checksum instead of last-changed date](http://go.microsoft.com/fwlink/?LinkId=325531) for determining which files should be copied to the server.
* Quickly publish individual selected files (including Web.config) when you're using the FTP or file system publish methods as well as with Web Deploy.

For more information about ASP.NET web deployment, see [the ASP.NET site](http://go.microsoft.com/fwlink/?LinkId=322027).

NuGet 2.7

NuGet 2.7 includes a rich set of new features which are described in detail at [NuGet 2.7 Release Notes](http://docs.nuget.org/docs/release-notes/nuget-2.7).

This version of NuGet also removes the need to provide explicit consent for NuGet’s package restore feature to download packages. Consent (and the associated checkbox in NuGet’s preferences dialog) is now granted by installing NuGet. Now package restore simply works by default.

ASP.NET Web Forms

One ASP.NET

The Web Forms project templates integrate seamlessly with the new One ASP.NET experience. You can add MVC and Web API support to your Web Forms project, and you can configure authentication using the One ASP.NET project creation wizard. For more information, see [Creating ASP.NET Web Projects in Visual Studio 2013](http://www.asp.net/visual-studio/overview/2013/creating-web-projects-in-visual-studio) .

ASP.NET Identity

The Web Forms project templates support the new ASP.NET Identity framework. In addition, the templates now support creation of a Web Forms intranet project. For more information, see [Authentication Methods](http://www.asp.net/visual-studio/overview/2013/creating-web-projects-in-visual-studio#auth) in **Creating ASP.NET Web Projects in Visual Studio 2013** .

Bootstrap

The Web Forms templates use [Bootstrap](http://twitter.github.io/bootstrap/) to provide a sleek and responsive look and feel that you can easily customize. For more information, see [Bootstrap in the Visual Studio 2013 web project templates](http://www.asp.net/visual-studio/overview/2013/creating-web-projects-in-visual-studio#bootstrap) .

ASP.NET MVC 5

One ASP.NET

The Web MVC project templates integrate seamlessly with the new One ASP.NET experience. You can customize your MVC project and configure authentication using the One ASP.NET project creation wizard. An introductory tutorial to ASP.NET MVC 5 can be found at [Getting Started with ASP.NET MVC 5](http://www.asp.net/mvc/tutorials/mvc-5/introduction/getting-started) .

For information on upgrading MVC 4 projects to MVC 5, see [How to Upgrade an ASP.NET MVC 4 and Web API Project to ASP.NET MVC 5 and Web API 2](http://www.asp.net/mvc/tutorials/mvc-5/how-to-upgrade-an-aspnet-mvc-4-and-web-api-project-to-aspnet-mvc-5-and-web-api-2).

ASP.NET Identity

The MVC project templates have been updated to use ASP.NET Identity for authentication and identity management. A tutorial featuring Facebook and Google authentication and the new membership API can be found at [Create an ASP.NET MVC 5 App with Facebook and Google OAuth2 and OpenID Sign-on](http://www.asp.net/mvc/tutorials/mvc-5/create-an-aspnet-mvc-5-app-with-facebook-and-google-oauth2-and-openid-sign-on) and [Deploy a Secure ASP.NET MVC app with Membership, OAuth, and SQL Database to a Windows Azure Web Site](http://www.windowsazure.com/en-us/develop/net/tutorials/web-site-with-sql-database/).

Bootstrap

The MVC project template has been updated to use [Bootstrap](http://getbootstrap.com/) to provide a sleek and responsive look and feel that you can easily customize. For more information, see [Bootstrap in the Visual Studio 2013 web project templates](http://www.asp.net/visual-studio/overview/2013/creating-web-projects-in-visual-studio#bootstrap) .

Authentication filters

Authentication filters are a new kind of filter in ASP.NET MVC that run prior to authorization filters in the ASP.NET MVC pipeline and allow you to specify authentication logic per-action, per-controller, or globally for all controllers. Authentication filters process credentials in the request and provide a corresponding principal. Authentication filters can also add authentication challenges in response to unauthorized requests.

Filter overrides

You can now override which filters apply to a given action method or controller by specifying an override filter. Override filters specify a set of filter types that should not be run for a given scope (action or controller). This allows you to configure filters that apply globally but then exclude certain global filters from applying to specific actions or controllers.

Attribute routing

ASP.NET MVC now supports attribute routing, thanks to a contribution by Tim McCall, the author of [http://attributerouting.net](http://attributerouting.net/). With attribute routing you can specify your routes by annotating your actions and controllers.

ASP.NET Web API 2

Attribute routing

ASP.NET Web API now supports attribute routing, thanks to a contribution by Tim McCall, the author of [http://attributerouting.net](http://attributerouting.net/). With attribute routing you can specify your Web API routes by annotating your actions and controllers like this:

[RoutePrefix("orders")]

public class OrdersController : ApiController

{

[Route("{id}")]

public Order Get(int id) { }

[Route("{id}/approve")]

public Order Approve(int id) { }

}

Attribute routing gives you more control over the URIs in your web API. For example, you can easily define a resource hierarchy using a single API controller:

public class MoviesController : ApiController

{

[Route("movies")]

public IEnumerable<Movie> Get() { }

[Route("actors/{actorId}/movies")]

public IEnumerable<Movie> GetByActor(int actorId) { }

[Route("directors/{directorId}/movies")]

public IEnumerable<Movie> GetByDirector(int directorId) { }

}

Attribute routing also provides a convenient syntax for specifying optional parameters, default values, and route constraints:

// Optional parameter

[Route("people/{name?}")]

// Default value

[Route("people/{name=Dan}")]

// Constraint: Alphabetic characters only.

[Route("people/{name:alpha}")]

For more information about attribute routing, see [Attribute Routing in Web API 2](http://www.asp.net/web-api/overview/web-api-routing-and-actions/attribute-routing-in-web-api-2).

OAuth 2.0

The Web API and Single Page Application project templates now support authorization using OAuth 2.0. OAuth 2.0 is a framework for authorizing client access to protected resources. It works for a variety of clients including browsers and mobile devices.

Support for OAuth 2.0 is based on new security middleware provided by the Microsoft OWIN Components for bearer authentication and implementing the authorization server role. Alternatively, clients can be authorized using an organizational authorization server, such as Windows Azure Active Directory or ADFS in Windows Server 2012 R2.

OData Improvements

**Support for $select, $expand, $batch, and $value**

ASP.NET Web API OData now has full support for $select, $expand, and $value. You can also use $batch for request batching and processing of change sets.

The $select and $expand options let you change the shape of the data that is returned from an OData endpoint. For more information, see [Introducing $select and $expand support in Web API OData](http://www.asp.net/web-api/overview/odata-support-in-aspnet-web-api/using-$select,-$expand,-and-$value).

**Improved extensibility**

The OData formatters are now extensible. You can add Atom entry metadata, support named stream and media link entries, add instance annotations, and customize how links are generated.

**Type-less support**

You can now build OData services without needing to define CLR types for your entity types. Instead, your OData controllers can take or return instances of **IEdmObject**, which are the OData formatters serialize/deserialize.

**Reuse an existing model**

If you already have an existing entity data model (EDM), you can now reuse it directly, instead of having to build a new one. For example, if you are using Entity Framework, you can use the EDM that EF builds for you.

Request Batching

Request batching combines multiple operations into a single HTTP POST request, to reduce network traffic and provide a smoother, less chatty user interface. ASP.NET Web API now supports several strategies for request batching:

* Use the $batch endpoint of an OData service.
* Package multiple requests into a single MIME multipart request.
* Use a custom batching format.

To enable request batching, simply add a route with a batching handler to your Web API configuration:

public static class WebApiConfig

{

public static void Register(HttpConfiguration config)

{

config.Routes.MapHttpBatchRoute(

routeName: "WebApiBatch",

routeTemplate: "api/batch",

batchHandler: new DefaultHttpBatchHandler(GlobalConfiguration.DefaultServer));

}

}

You can also control whether requests or executed sequentially or in any order.

Portable ASP.NET Web API Client

You can now use the ASP.NET Web API Client to create portable class libraries that work across your Windows Store and Windows Phone 8 applications. You can also create portable formatters that can be shared across client and server.

Improved Testability

Web API 2 makes it much easier to unit test your API controllers. Just instantiate your API controller with your request message and configuration, and then call the action method you wish to test. It is also easy to mock the **UrlHelper** class, for action methods that perform link generation.

IHttpActionResult

You can now implement IHttpActionResult to encapsulate the result of your Web API action methods. An IHttpActionResult returned from a Web API action method is executed by the ASP.NET Web API runtime to produce the resultant response message. An IHttpActionResult can be returned from any Web API action to simplify unit testing of your Web API implementation. For convenience a number of IHttpActionResult implementations are provided out of the box including results for returning specific status codes, formatted content or content-negotiated responses.

HttpRequestContext

The new **HttpRequestContext** tracks any state that is tied to the request but is not immediately available from the request. For example, you can use the **HttpRequestContext** to get route data, the principal associated with the request, the client certificate, the **UrlHelper** and the virtual path root. You can easily create an **HttpRequestContext** for unit testing purposes.

Because the principal for the request is flowed with the request instead of relying on **Thread.CurrentPrincipal**, the principal is now available throughout the lifetime of the request while it is in the Web API pipeline.

CORS

Thanks to another great contribution from Brock Allen, ASP.NET now fully supports Cross Origin Request Sharing (CORS).

Browser security prevents a web page from making AJAX requests to another domain. [CORS](http://www.w3.org/TR/cors/) is a W3C standard that allows a server to relax the same-origin policy. Using CORS, a server can explicitly allow some cross-origin requests while rejecting others.

Web API 2 now supports CORS, including automatic handling of preflight requests. For more information, see [Enabling Cross-Origin Requests in ASP.NET Web API](http://www.asp.net/web-api/overview/security/enabling-cross-origin-requests-in-web-api).

Authentication Filters

Authentication filters are a new kind of filter in ASP.NET Web API that run prior to authorization filters in the ASP.NET Web API pipeline and allow you to specify authentication logic per-action, per-controller, or globally for all controllers. Authentication filters process credentials in the request and provide a corresponding principal. Authentication filters can also add authentication challenges in response to unauthorized requests.

Filter Overrides

You can now override which filters apply to a given action method or controller, by specifying an override filter. Override filters specify a set of filter types that should not run for a given scope (action or controller). This allows you to add global filters, but then exclude some from specific actions or controllers.

OWIN Integration

ASP.NET Web API now fully supports OWIN and can be run on any OWIN capable host. Also included is a **HostAuthenticationFilter** that provides integration with the OWIN authentication system.

With OWIN integration, you can self-host Web API in your own process alongside other OWIN middleware, such as SignalR. For more information, see [Use OWIN to Self-Host ASP.NET Web API](http://www.asp.net/signalr/overview/signalr-20/getting-started-with-signalr-20/tutorial-signalr-20-self-host).

ASP.NET SignalR 2.0

The following sections describe features of SignalR 2.0.

* [Built on OWIN](http://www.asp.net/visual-studio/overview/2013/release-notes#builtonowin)
* [MapHubs and MapConnection are now MapSignalR](http://www.asp.net/visual-studio/overview/2013/release-notes#MapSignalR)
* [Cross-Domain Support](http://www.asp.net/visual-studio/overview/2013/release-notes#crossdomain)
* [iOS and Android support via MonoTouch and MonoDroid](http://www.asp.net/visual-studio/overview/2013/release-notes#mobile)
* [Portable .NET Client](http://www.asp.net/visual-studio/overview/2013/release-notes#portable)
* [New Self-Host Package](http://www.asp.net/visual-studio/overview/2013/release-notes#selfhost)
* [Backward-compatible server support](http://www.asp.net/visual-studio/overview/2013/release-notes#backwardcompat)
* [Removed server support for .NET 4.0](http://www.asp.net/visual-studio/overview/2013/release-notes#remove40)
* [Sending a message to a list of clients and groups](http://www.asp.net/visual-studio/overview/2013/release-notes#messagelist)
* [Sending a message to a specific user](http://www.asp.net/visual-studio/overview/2013/release-notes#sendtouser)
* [Better error handling support](http://www.asp.net/visual-studio/overview/2013/release-notes#errorhandling)
* [Easier unit testing of hubs](http://www.asp.net/visual-studio/overview/2013/release-notes#unittesting)
* [JavaScript error handling](http://www.asp.net/visual-studio/overview/2013/release-notes#javascripterror)

For an example of how to upgrade an existing 1.x project to SignalR 2.0, see [Upgrading a SignalR 1.x Project](http://www.asp.net/signalr/overview/signalr-20/getting-started-with-signalr-20/upgrading-signalr-1x-projects-to-20).

Built on OWIN

SignalR 2.0 is built completely on [OWIN (the Open Web Interface for .NET)](http://owin.org/). This change makes the setup process for SignalR much more consistent between web-hosted and self-hosted SignalR applications, but has also required a number of API changes.

MapHubs and MapConnection are now MapSignalR

For compatibility with OWIN standards, these methods have been renamed to MapSignalR. MapSignalR called without parameters will map all hubs (as MapHubs does in version 1.x); to map individual **PersistentConnection** objects, specify the connection type as the type parameter, and the URL extension for the connection as the first argument.

The MapSignalR method is called in an Owin startup class. Visual Studio 2013 contains a new template for an Owin startup class; to use this template, do the following:

1. Right-click on the project
2. Select **Add**, **New Item...**
3. Select **Owin Startup class**. Name the new class **Startup.cs**.

In a **Web application,** the Owin startup class containing the MapSignalR method is then added to Owin's startup process using an entry in the application settings node of the Web.Config file, as shown below.

In a **Self-hosted application**, the Startup class is passed as the type parameter of the WebApp.Start method.

**Mapping hubs and connections in SignalR 1.x (from the global application file in a web application):**

protected void Application\_Start(object sender, EventArgs e)

{

// Map all hubs to "/signalr"

RouteTable.Routes.MapHubs();

// Map the Echo PersistentConnection to "/echo"

RouteTable.Routes.MapConnection<myconnection>("echo", "/echo");

}

**Mapping hubs and connections in SignalR 2.0 (from an Owin Startup class file):**

using Microsoft.AspNet.SignalR;

using Microsoft.Owin;

using Owin;

[assembly: OwinStartup(typeof(MyWebApplication.Startup))]

namespace MyWebApplication

{

public class Startup

{

public void Configuration(IAppBuilder app)

{

// Map all hubs to "/signalr"

app.MapSignalR();

// Map the Echo PersistentConnection to "/echo"

app.MapSignalR<echoconnection>("/echo");

}

}

}

In a **Self-hosted application**, the Startup class is passed as the type parameter for the WebApp.Start method, as shown below.

string url = "http://localhost:8080";

using (WebApp.Start<startup>(url))

{

Console.WriteLine("Server running on {0}", url);

Console.ReadLine();

}

Cross-Domain Support

In SignalR 1.x, cross domain requests were controlled by a single EnableCrossDomain flag. This flag controlled both JSONP and CORS requests. For greater flexibility, all CORS support has been removed from the server component of SignalR (JavaScript lients still use CORS normally if it is detected that the browser supports it), and new OWIN middleware has been made available to support these scenarios.

In SignalR 2.0, If JSONP is required on the client (to support cross-domain requests in older browsers), it will need to be enabled explicitly by setting EnableJSONP on the HubConfiguration object to true, as shown below. JSONP is disabled by default, as it is less secure than CORS.

To add the new CORS middleware in SignalR 2.0, add the Microsoft.Owin.Cors library to your project, and call UseCors before your SignalR middleware, as shown in the section below.

**Adding Microsoft.Owin.Cors to your project**: To install this library, run the following command in the Package Manager Console:

Install-Package Microsoft.Owin.Cors

This command will add the 2.0.0 version of the package to your project.

**Calling UseCors**

The following code snippets demonstrate how to implement cross-domain connections in SignalR 1.x and 2.0.

**Implementing cross-domain requests in SignalR 1.x (from the global application file)**

protected void Application\_Start(object sender, EventArgs e)

{

var hubConfiguration = new HubConfiguration();

hubConfiguration.EnableCrossDomain = true;

RouteTable.Routes.MapHubs(hubConfiguration);

}

**Implementing cross-domain requests in SignalR 2.0 (from a C# code file)**

The following code demonstrates how to enable CORS or JSONP in a SignalR 2.0 project. This code sample uses Map and RunSignalR instead of MapSignalR, so that the CORS middleware runs only for the SignalR requests that require CORS support (rather than for all traffic at the path specified in MapSignalR.) Map can also be used for any other middleware that needs to run for a specific URL prefix, rather than for the entire application.

using Microsoft.AspNet.SignalR;

using Microsoft.Owin.Cors;

using Owin;

[assembly: OwinStartup(typeof(MyWebApplication.Startup))]

namespace MyWebApplication

{

public class Startup

{

public void Configuration(IAppBuilder app)

{

// Branch the pipeline here for requests that start with "/signalr"

app.Map("/signalr", map =>

{

// Setup the CORS middleware to run before SignalR.

// By default this will allow all origins. You can

// configure the set of origins and/or http verbs by

// providing a cors options with a different policy.

map.UseCors(CorsOptions.AllowAll);

var hubConfiguration = new HubConfiguration

{

// You can enable JSONP by uncommenting line below.

// JSONP requests are insecure but some older browsers (and some

// versions of IE) require JSONP to work cross domain

// EnableJSONP = true

};

// Run the SignalR pipeline. We're not using MapSignalR

// since this branch already runs under the "/signalr"

// path.

map.RunSignalR(hubConfiguration);

});

}

}

}

iOS and Android support via MonoTouch and MonoDroid

Support has been added for iOS and Android clients using MonoTouch and MonoDroid components from the [Xamarin library](http://xamarin.com/). For more information on how to use them, see [Using Xamarin Components](https://github.com/SignalR/SignalR/wiki/Building-Mono.Mobile.sln). These components will be available in the [Xamarin Store](https://store.xamarin.com/) when the SignalR RTW release is available.

Portable .NET client

To better facilitate cross-platform development, the Silverlight, WinRT and Windows Phone clients have been replaced with a single portable .NET client that supports the following platforms:

* NET 4.5
* Silverlight 5
* WinRT (.NET for Windows Store Apps)
* Windows Phone 8

New Self-Host Package

There is now a NuGet package to make it easier to get started with SignalR Self-Host (SignalR applications that are hosted in a process or other application, rather than being hosted in a web server). To upgrade a self-host project built with SignalR 1.x, remove the Microsoft.AspNet.SignalR.Owin package, and add the Microsoft.AspNet.SignalR.SelfHost package. For more information on getting started with the self-host package, see [Tutorial: SignalR Self-Host](http://www.asp.net/signalr/overview/signalr-20/getting-started-with-signalr-20/tutorial-signalr-20-self-host).

Backward-compatible server support

In previous versions of SignalR, the versions of the SignalR package used in the client and the server needed to be identical. In order to support thick-client applications that would be difficult to update, SignalR 2.0 now supports using a newer server version with an older client. **Note: SignalR 2.0 does not support servers built with older versions with newer clients.**

Removed server support for .NET 4.0

SignalR 2.0 has dropped support for server interoperability with .NET 4.0. .NET 4.5 must be used with SignalR 2.0 servers. There is still a .NET 4.0 client for SignalR 2.0.

Sending a message to a list of clients and groups

In SignalR 2.0, it's possible to send a message using a list of client and group IDs. The following code snippets demonstrate how to do this.

**Sending a message to a list of clients and groups using PersistentConnection**

using Microsoft.AspNet.SignalR;

using System.Collections.Generic;

public class ChatConnection : PersistentConnection

{

static List<string> ConnectionIds = new List<string>();

static List<string> groups = new List<string>{"chatGroup", "chatGroup2"};

protected override System.Threading.Tasks.Task OnReceived(IRequest request, string connectionId, string data)

{

Connection.Send(ConnectionIds, data);

Groups.Send(groups, data);

return base.OnReceived(request, connectionId, data);

}

protected override System.Threading.Tasks.Task OnConnected(IRequest request, string connectionId)

{

ConnectionIds.Add(connectionId);

Groups.Add(connectionId, "chatGroup");

return base.OnConnected(request, connectionId);

}

protected override System.Threading.Tasks.Task OnDisconnected(IRequest request, string connectionId)

{

ConnectionIds.Remove(connectionId);

return base.OnDisconnected(request, connectionId);

}

}

**Sending a message to a list of clients and groups using Hubs**

using Microsoft.AspNet.SignalR;

using System.Collections.Generic;

public class ChatHub : Hub

{

static List<string> ConnectionIds = new List<string>();

static List<string> groups = new List<string> { "chatGroup", "chatGroup2" };

public void Send(string name, string message)

{

// Call the broadcastMessage method to update clients.

Clients.Clients(ConnectionIds).broadcastMessage(name, message);

Clients.Groups(groups).broadcastMessage(name, message);

}

public override System.Threading.Tasks.Task OnConnected()

{

ConnectionIds.Add(Context.ConnectionId);

Groups.Add(Context.ConnectionId, "chatGroup");

return base.OnConnected();

}

public override System.Threading.Tasks.Task OnDisconnected()

{

ConnectionIds.Remove(Context.ConnectionId);

return base.OnDisconnected();

}

}

Sending a message to a specific user

This feature allows users to specify what the userId is based on an IRequest via a new interface IUserIdProvider:

**The IUserIdProvider interface**

public interface IUserIdProvider

{

string GetUserId(IRequest request);

}

By default there will be an implementation that uses the user's IPrincipal.Identity.Name as the user name.

In hubs, you'll be able to send messages to these users via a new API:

**Using the Clients.User API**

public class MyHub : Hub

{

public void Send(string userId, string message)

{

Clients.User(userId).send(message);

}

}

Better Error Handling Support

Users can now throw **HubException** from any hub invocation. The constructor of the **HubException** can take a string message and an object extra error data. SignalR will auto-serialize the exception and send it to the client where it will be used to reject/fail the hub method invocation.

The **show detailed hub exceptions** setting has no bearing on **HubException** being sent back to the client or not; it is always sent.

**Server-side code demonstrating sending a HubException to the client**

public class MyHub : Hub

{

public void Send(string message)

{

if(message.Contains("<script>"))

{

throw new HubException("This message will flow to the client", new { user = Context.User.Identity.Name, message = message });

}

Clients.All.send(message);

}

}

**JavaScript client code demonstrating responding to a HubException sent from the server**

myHub.server.send("<script>")

.fail(function (e) {

if (e.source === 'HubException') {

console.log(e.message + ' : ' + e.data.user);

}

});

**.NET client code demonstrating responding to a HubException sent from the server**

try

{

await myHub.Invoke("Send", "<script>");

}

catch(HubException ex)

{

Conosle.WriteLine(ex.Message);

}

Easier unit testing of hubs

SignalR 2.0 includes an interface called IHubCallerConnectionContext on Hubs that makes it easier to create mock client side invocations. The following code snippets demonstrate using this interface with popular test harnesses [xUnit.net](http://xunit.codeplex.com/) and [moq](https://code.google.com/p/moq/).

**Unit testing SignalR with xUnit.net**

[Fact]

public void HubsAreMockableViaDynamic()

{

bool sendCalled = false;

var hub = new MyHub();

var mockClients = new Mock<IHubCallerConnectionContext>();

hub.Clients = mockClients.Object;

dynamic all = new ExpandoObject();

all.send = new Action<string>(message =>

{

sendCalled = true;

});

mockClients.Setup(m => m.All).Returns((ExpandoObject)all);

hub.Send("foo");

Assert.True(sendCalled);

}

**Unit testing SignalR with moq**

[Fact]

public interface IClientContract

{

void send(string message);

}

public void HubsAreMockableViaType()

{

var hub = new MyHub();

var mockClients = new Mock<IHubCallerConnectionContext>();

var all = new Mock<IClientContract>();

hub.Clients = mockClients.Object;

all.Setup(m => m.send(It.IsAny<string>())).Verifiable();

mockClients.Setup(m => m.All).Returns(all.Object);

hub.Send("foo");

all.VerifyAll();

JavaScript error handling

In SignalR 2.0, all JavaScript error handling callbacks return JavaScript error objects instead of raw strings. This allows SignalR to flow richer information to your error handlers. You can get the inner exception from the source property of the error.

**JavaScript client code that handles the Start.Fail exception**

connection.start().fail(function(e) {

console.log('The error is: ' + e.message);

});

ASP.NET Identity

New ASP.NET Membership System

ASP.NET Identity is the new membership system for ASP.NET applications. ASP.NET Identity makes it easy to integrate user-specific profile data with application data. ASP.NET Identity also allows you to choose the persistence model for user profiles in your application. You can store the data in a SQL Server database or another data store, including NoSQL data stores such as Windows Azure Storage Tables. For more information, see [Individual User Accounts](http://www.asp.net/visual-studio/overview/2013/creating-web-projects-in-visual-studio#indauth) in **Creating ASP.NET Web Projects in Visual Studio 2013** .

Claims-based authentication

ASP.NET now supports claims-based authentication, where the user’s identity is represented as a set of claims from a trusted issuer. Users can be authenticated using a username and password maintained in an application database, or using social identity providers (for example: Microsoft Accounts, Facebook, Google, Twitter), or using organizational accounts through Windows Azure Active Directory or Active Directory Federation Services (ADFS).

Integration with Windows Azure Active Directory and Windows Server Active Directory

You can now create ASP.NET projects that use Windows Azure Active Directory (WAAD) or Windows Server Active Directory (AD) for authentication. For more information, see [Organizational Accounts](http://www.asp.net/visual-studio/overview/2013/creating-web-projects-in-visual-studio#orgauth) in **Creating ASP.NET Web Projects in Visual Studio 2013** .

OWIN Integration

ASP.NET authentication is now based on OWIN middleware that can be used on any OWIN-based host. For more information about OWIN, see the following [Microsoft OWIN Components](http://www.asp.net/visual-studio/overview/2013/release-notes#TOC7) section.

Microsoft OWIN Components

[Open Web Interface for .NET](http://owin.org/) (OWIN) defines an abstraction between .NET web servers and web applications. OWIN decouples the web application from the server, making web applications host-agnostic. For example, you can host an OWIN-based web application in IIS or self-host it in a custom process.

Changes introduced in the Microsoft OWIN components (also known as the Katana project) include new server and host components, new helper libraries and middleware, and new authentication middleware.

For more information about OWIN and Katana, see [What's new in OWIN and Katana](http://www.asp.net/vnext/overview/owin-and-katana).

**Note:** [**OWIN**](http://www.asp.net/aspnet/overview/owin-and-katana/an-overview-of-project-katana) **applications cannot run in IIS classic mode; they must be run in integrated mode.**

**Note:** [**OWIN**](http://www.asp.net/aspnet/overview/owin-and-katana/an-overview-of-project-katana) **applications must be run in full trust.**

New Servers and Hosts

With this release, new components were added to enable self-host scenarios. These components include the following NuGet packages:

* **Microsoft.Owin.Host.HttpListener**. Provides an OWIN server that uses **HttpListener** to listen for HTTP requests and direct them into the OWIN pipeline.
* **Microsoft.Owin.Hosting** Provides a library for developers who wish to self-host an OWIN pipeline in a custom process, such as a console application or Windows service.
* **OwinHost**. Provides a stand-alone executable that wraps Microsoft.Owin.Hosting and lets you self-host an OWIN pipeline without having to write a custom host application.

In addition, the Microsoft.Owin.Host.SystemWeb package now enables middleware to provide hints to the **SystemWeb** server, indicating that the middleware should be called during a specific ASP.NET pipeline stage. This feature is particularly useful for authentication middleware, which should run early in the ASP.NET pipeline.

Helper Libraries and Middleware

Although you can write OWIN components using only the function and type definitions from the OWIN specification, the new Microsoft.Owin package provides a more user-friendly set of abstractions. This package combines several earlier packages (e.g., Owin.Extensions, Owin.Types) into a single, well-structured object model that can then be easily used by other OWIN components. In fact, the majority of Microsoft OWIN components now use this package.

**Note:**[OWIN](http://www.owin.org/) applications cannot run in IIS classic mode; they must be run in integrated mode.

**Note:**[OWIN](http://www.owin.org/) applications must be run in full trust.

This release also includes the Microsoft.Owin.Diagnostics package, which includes middleware to validate a running OWIN application, plus error-page middleware to help investigate failures.

Authentication Components

The following authentication components are available.

* **Microsoft.Owin.Security.ActiveDirectory**. Enables authentication using on-premise or cloud-based directory services.
* **Microsoft.Owin.Security.Cookies** Enables authentication using cookies. This package was previously named Microsoft.Owin.Security.Forms.
* **Microsoft.Owin.Security.Facebook** Enables authentication using Facebook’s OAuth-based service.
* **Microsoft.Owin.Security.Google** Enables authentication using Google’s OpenID-based service.
* **Microsoft.Owin.Security.Jwt** Enables authentication using JWT tokens.
* **Microsoft.Owin.Security.MicrosoftAccount** Enables authentication using Microsoft accounts.
* **Microsoft.Owin.Security.OAuth**. Provides an OAuth authorization server as well as middleware for authenticating bearer tokens.
* **Microsoft.Owin.Security.Twitter** Enables authentication using Twitter’s OAuth-based service.

This release also includes the Microsoft.Owin.Cors package, which contains middleware for processing cross-origin HTTP requests.

**Note:** Support for JWT signing has been removed in the final version of Visual Studio 2013.

Entity Framework 6

For a list of new features and other changes in Entity Framework 6, see [Entity Framework Version History](http://msdn.com/data/jj574253).

ASP.NET Razor 3

ASP.NET Razor 3 includes the following new features:

* Support for Tab editing. Preivously, the **Format Document** command, auto indenting, and auto formatting in Visual Studio did not work correctly when using the **Keep Tabs** option. This change corrects Visual Studio formatting for Razor code for tab formatting.
* Support for URL Rewrite rules when generating links.
* Removal of security transparent attribute. **Note:** This is a breaking change, and makes Razor 3 incompatible with MVC4 and earlier, while Razor 2 is incompatible with MVC5 or assemblies compiled against MVC5.

Razor 3 issues fixed in Visual Studio 2013 from pre-release versions can be found [here](https://aspnetwebstack.codeplex.com/workitem/list/advanced?keyword=&status=Resolved%7cClosed&type=All&priority=All&release=All%7cv5.0%2bPreview%7cv5.0%2bRC%7cv5.0%2bRTM&assignedTo=All&component=Web%2bPages%252fRazor&reasonClosed=Fixed&sortField=LastUpdatedDate&sortDirection=Descending&page=0).

ASP.NET App Suspend

ASP.NET App Suspend is a game-changing feature in the .NET Framework 4.5.1 that radically changes the user experience and economic model for hosting large numbers of ASP.NET sites on a single machine. For more information, see [ASP.NET App Suspend – responsive shared .NET web hosting](http://blogs.msdn.com/b/dotnet/archive/2013/10/09/asp-net-app-suspend-responsive-shared-net-web-hosting.aspx).

Known Issues and Breaking Changes

This section describes known issues and breaking changes in the ASP.NET and Web Tools for Visual Studio 2013.

NuGet

* [New package restore doesn't work on Mono when using SLN file](https://nuget.codeplex.com/workitem/3596) – will be fixed in an upcoming nuget.exe download and [NuGet.CommandLine package](http://www.nuget.org/packages/NuGet.CommandLine/) update.
* [New package restore doesn't work with Wix projects](https://nuget.codeplex.com/workitem/3598) – will be fixed in an upcoming nuget.exe download and [NuGet.CommandLine package](http://www.nuget.org/packages/NuGet.CommandLine/) update.
* [Automatic Package restore doesn't work for projects under a solution folder](https://nuget.codeplex.com/workitem/3625) – will be fixed in NuGet 2.8.

ASP.NET Web API

1. ODataQueryOptions<T>.ApplyTo(IQueryable) doesn’t return IQueryable<T> always, as we added support for $select and $expand.

Our earlier samples for ODataQueryOptions<T> always casted the return value from ApplyTo to IQueryable<T>. This worked earlier because the query options that we supported earlier ($filter, $orderby, $skip, $top) do not change the shape of the query. Now that we support $select and $expand the return value from ApplyTo will not be IQueryable<T> always.

// Sample ODataQueryOptions<T> usage from earlier

public IQueryable<Customer> Get(ODataQueryOptions<Customer> query)

{

IQueryable<customer> result="query.ApplyTo(\_customers)" as iqueryable<customer>; return result;

}

If you are using the sample code from earlier, it will continue working if the client does not send $select and $expand. However, if you wish to support $select and $expand you have to change that code to this.

public IHttpActionResult Get(ODataQueryOptions<Customer> query)

{

IQueryable result = query.ApplyTo(\_customers);

return Ok(result, result.GetType());

}

private IHttpActionResult Ok(object content, Type type)

{

Type resultType = typeof(OkNegotiatedContentResult<>).MakeGenericType(type);

return Activator.CreateInstance(resultType, content, this) as IHttpActionResult;

}

1. **Request.Url or RequestContext.Url is null during a batch request**

In a batching scenario, **UrlHelper** is null when accessed from **Request.Url** or **RequestContext.Url**.

This issue is currently tracked here: [BatchRequestContext.Url is null for batching request](http://aspnetwebstack.codeplex.com/workitem/1301).

The workaround for this issue is to create a new instance of **UrlHelper**, as in the following example:

**Creating a new instance of UrlHelper**

if (RequestContext.Url == null)

{

RequestContext.Url = new UrlHelper(Request);

}

ASP.NET MVC

1. When using MVC5 and OrgAuth, if you have views which do AntiForgerToken validation, you might come across the following error when you post data to the view:

**Error**:

*Server Error in '/' Application.*

*A claim of type 'http://schemas.xmlsoap.org/ws/2005/05/identity/claims/nameidentifier' or 'http://schemas.microsoft.com/accesscontrolservice/2010/07/claims/identityprovider' was not present on the provided ClaimsIdentity. To enable anti-forgery token support with claims-based authentication, please verify that the configured claims provider is providing both of these claims on the ClaimsIdentity instances it generates. If the configured claims provider instead uses a different claim type as a unique identifier, it can be configured by setting the static property AntiForgeryConfig.UniqueClaimTypeIdentifier.*

**Workaround**:

Add the following line in Global.asax to fix it:

AntiForgeryConfig.UniqueClaimTypeIdentifier = ClaimTypes.Name;

This will be fixed for the next release.

1. After upgrading an MVC4 app to MVC5, build the solution and launch it. You should see the following error:

[A]System.Web.WebPages.Razor.Configuration.HostSection cannot be cast to [B]System.Web.WebPages.Razor.Configuration.HostSection. Type A originates from 'System.Web.WebPages.Razor, Version=2.0.0.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35' in the context 'Default' at location 'C:\windows\Microsoft.Net\assembly\GAC\_MSIL\System.Web.WebPages.Razor\v4.0\_2.0.0.0\_\_31bf3856ad364e35\System.Web.WebPages.Razor.dll'. Type B originates from 'System.Web.WebPages.Razor, Version=3.0.0.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35' in the context 'Default' at location 'C:\Windows\Microsoft.NET\Framework\v4.0.30319\Temporary ASP.NET Files\root\6d05bbd0\e8b5908e\assembly\dl3\c9cbca63\f8910382\_6273ce01\System.Web.WebPages.Razor.dll'.

To fix the above error, open *all* the Web.config files (including the ones in the Views folder) in your project and do the following:

* 1. Update all occurrences of version “4.0.0.0” of “System.Web.Mvc” to “5.0.0.0”.
  2. Update all occurrences of version “2.0.0.0” of “System.Web.Helpers”, "System.Web.WebPages" and "System.Web.WebPages.Razor" to “3.0.0.0”

For example, after you make the above changes, the assembly bindings should look like this:

<dependentAssembly>

<assemblyIdentity name="System.Web.Helpers" publicKeyToken="31bf3856ad364e35" />

<bindingRedirect oldVersion="1.0.0.0-2.0.0.0" newVersion="3.0.0.0" />

</dependentAssembly>

<dependentAssembly>

<assemblyIdentity name="System.Web.WebPages" publicKeyToken="31bf3856ad364e35" />

<bindingRedirect oldVersion="1.0.0.0-2.0.0.0" newVersion="3.0.0.0" />

</dependentAssembly>

<dependentAssembly>

<assemblyIdentity name="System.Web.WebPages.Razor" publicKeyToken="31bf3856ad364e35" />

<bindingRedirect oldVersion="1.0.0.0-3.0.0.0" newVersion="3.0.0.0" />

</dependentAssembly>

<dependentAssembly>

<assemblyIdentity name="System.Web.Mvc" publicKeyToken="31bf3856ad364e35" />

<bindingRedirect oldVersion="1.0.0.0-4.0.0.0" newVersion="5.0.0.0" />

</dependentAssembly>

For information on upgrading MVC 4 projects to MVC 5, see [How to Upgrade an ASP.NET MVC 4 and Web API Project to ASP.NET MVC 5 and Web API 2](http://www.asp.net/mvc/tutorials/mvc-5/how-to-upgrade-an-aspnet-mvc-4-and-web-api-project-to-aspnet-mvc-5-and-web-api-2).

1. When using client-side validation with jQuery Unobtrusive Validation, the validation message is sometimes incorrect for an HTML input element with type=’number’. The validation error for a required value (“The Age field is required”) is shown when an invalid number is entered instead of the correct message that a valid number is required.

This issue is commonly found with scaffolded code for a model with an integer property on the Create and Edit views.

To work around this issue, change the editor helper from:

@Html.EditorFor(person => person.Age)

To:

@Html.TextBoxFor(person => person.Age)

1. ASP.NET MVC 5 no longer supports partial trust. Projects linking to the MVC or WebAPI binaries should remove the [SecurityTransparent](http://msdn.microsoft.com/en-us/library/system.security.securitytransparentattribute.aspx) attribute and the [AllowPartiallyTrustedCallers](http://msdn.microsoft.com/en-us/library/system.security.allowpartiallytrustedcallersattribute.aspx) attribute. Removing these attributes will eliminate compiler errors such as the following.

Attempt by security transparent method ‘MyComponent' to access security critical type 'System.Web.Mvc.MvcHtmlString' failed. Assembly 'PagedList.Mvc, Version=4.3.0.0, Culture=neutral, PublicKeyToken=abbb863e9397c5e1' is marked with the AllowPartiallyTrustedCallersAttribute, and uses the level 2 security transparency model. Level 2 transparency causes all methods in AllowPartiallyTrustedCallers assemblies to become security transparent by default, which may be the cause of this exception.

**SPA Template with Facebook authorization may cause instability in IE while the web site is hosted in intranet zone**

The SPA template provides external log in with Facebook. When the project created with the template is running locally, signing in may cause IE to crash.

Solution:

1.  Host the web site in internet zone; or

2.  Test the scenario in a browser other than IE.

Web Forms Scaffolding

Web Forms Scaffolding has been removed from VS2013 and will be available in a future update to Visual Studio. However, you can still use scaffolding within a Web Forms project by adding MVC dependencies and generating scaffolding for MVC. Your project will contain a combination of Web Forms and MVC.

To add MVC to your Web Forms project, add a new scaffolded item and select **MVC 5 Dependencies**. Select either Minimal or Full depending on whether you need all of the content files, such as scripts. Then, add a scaffolded item for MVC, which will create views and a controller in your project.

MVC and Web API Scaffolding - HTTP 404, Not Found error

If an error is encountered when adding a scaffolded item to a project, it is possible your project will be left in an inconsistent state. Some of the changes made be scaffolding will be rolled back but other changes, such as the installed NuGet packages, will not be rolled back. If the routing configuration changes are rolled back, users will receive an HTTP 404 error when navigating to scaffolded items.

Workaround:

* To fix this error for MVC, add a new scaffolded item and select MVC 5 Dependencies (either Minimal or Full). This process will add all of the required changes to your project.
* To fix this error for Web API:
  1. Add the WebApiConfig class to your project.

**C#**

public static class WebApiConfig

{

public static void Register(HttpConfiguration config)

{

     config.MapHttpAttributeRoutes();

config.Routes.MapHttpRoute(

name: "DefaultApi",

routeTemplate: "api/{controller}/{id}",

defaults: new { id = RouteParameter.Optional }

);

}

}

**Visual Basic**

Public Module WebApiConfig

Public Sub Register(ByVal config As HttpConfiguration)

config.MapHttpAttributeRoutes()

config.Routes.MapHttpRoute(

name:="DefaultApi",

routeTemplate:="api/{controller}/{id}",

defaults:=New With {.id = RouteParameter.Optional}

)

End Sub

End Module

* 1. Configure WebApiConfig.Register in the Application\_Start method in Global.asax as follows:

**C#**

public class WebApiApplication : System.Web.HttpApplication

{

protected void Application\_Start()

{

GlobalConfiguration.Configure(WebApiConfig.Register);

}

}

**Visual Basic**

Public Class WebApiApplication

Inherits System.Web.HttpApplication

Sub Application\_Start()

GlobalConfiguration.Configure(AddressOf WebApiConfig.Register)

End Sub

End Class