# ASP.NET MVC 5, ASP.NET MVC Core 2.2 and Web API

MAX002BD

Copyright © 2019 MAX Technical Training

AUTHOR

Michael T Smith
Microsoft Certified Trainer
SharePoint / Office Servers and Services MVP
Senior Instructor, MAX Technical Training

COPYRIGHT

© 2017, 2018 MAX Technical Training, Inc. All Rights Reserved. This manual and any training materials supplied with it are copyrighted with all rights reserved. No part of this publication may be reproduced, transmitted, transcribed, stored in a retrieval system or translated into any language in any form by any means without the written permission of MAX Technical Training, Inc.

TRADEMARKS

Brand names, company names and product names used herein are trademarks or registered trademarks of their respective companies.

MAX Technical Training, Inc.
4900 Parkway Drive, Suite 160
Mason, OH 45040
513.322.8888

MAX002AD / MA-2009

3/13/2019

Copyright © 2019 MAX Technical Training

# Table of Contents



Copyright © 2019 MAX Technical Training

Copyright © 2019 MAX Technical Training

Copyright © 2019 MAX Technical Training

Copyright © 2019 MAX Technical Training

Copyright © 2019 MAX Technical Training

Copyright © 2019 MAX Technical Training

Copyright © 2019 MAX Technical Training

Copyright © 2019 MAX Technical Training

# Module 1: A Brief History of HTML, MVC and the Web

## Welcome!

Microsoft has built a very complete set of tools to let you quickly create multi-tiered modern web applications using the Model View Controller design pattern. These include .NET classes, Visual Studio templates and wizards, and the Razor View Engine.

This class is for:

- New web site developers needing to learn the full MVC collection of features.
- Experienced developers new to Microsoft MVC.

### Prerequisites

- Strong skills in C#.
- Good HTML5 and CSS3 skills.
- An understanding of how mobile devices impact your web projects.
- Basic SQL Server experience.

## MVC Versions

This class uses the following versions:

- Visual Studio 2015 or 2017
- ASP.Net MVC 5.2.3.0
- Razor 3.0.0.0
- Entity Framework 6.0.0.0
- Web API 2.0
- OData v3
- HTML 5
- CSS 3

Note: DLL versions will have different fractional versions. I.e. MVC 5.2.30128.0.

### Which Version Do I Have?

- Visual Studio
  - Launch Visual Studio, click **Help** and About Microsoft Visual Studio.
  - My version: _____
- ASP.Net MVC

Copyright © 2019 MAX Technical Training

    - Launch Visual Studio, open your ASP.Net Web Application project.
    - In the Solution Explorer expand References and click System.Web.MVC.
    - In the Properties panel note the Version property.
    - My version: _____
- Razor
    - Launch Visual Studio, open your ASP.Net Web Application project.
    - In the Solution Explorer expand References and click System.Web.Razor.
    - In the Properties panel note the Version property.
    - My version: _____
- Entity Framework
    - Launch Visual Studio, open your ASP.Net Web Application project.
    - In the Solution Explorer expand References and click EntityFramework.
    - In the Properties panel note the Version property.
    - My version: _____

# What is MVC

MVC stands for Model, View, Controller, and in one form or another has been around since the 1970s. No one "owns" MVC and many people cannot agree on a single definition! For a history of MVC take a look at the Wikipedia article on Model-View-Controller. For our purposes here, MVC means the set of tools available in ASP.NET and Visual Studio to build multi-tiered and testable web applications.

## Models

A Model represents the application's data domain. The model typically defines the business objects as classes, retrieves and stores data to the database, and maintains the model state. While the most common model framework for MVC is the Entity Framework, you can use other frameworks or create your own model code.

## Controllers

Controllers are the heart of MVC. User requests are routed to Controllers, Controllers access models, Controllers apply business logic, Controllers pass the data to Views and then the HTML generated by the View is returned to the user through the Controller.

Controllers are typical .NET classes that inherit from System.Web.Mvc.Controller. These classes are almost never directly instantiated. Instead they are created from a controller factory called from the router. Like all classes, these controllers have methods. These methods are called Actions. Actions are loosely bound at run time with ASP.NET Model Binding using routing rules.

The routing process does not directly pass data to the Controller. It passes the request to a Controller "factory" that determines which controller is needed, creates an instance of that Controller and then calls the Action (method).

## Views

Views are files that generate the HTML that's returned to the user's browser. A View Engine takes a view file, processes it and then returns the HTML to return to the user's browser. The Visual Studio MVC

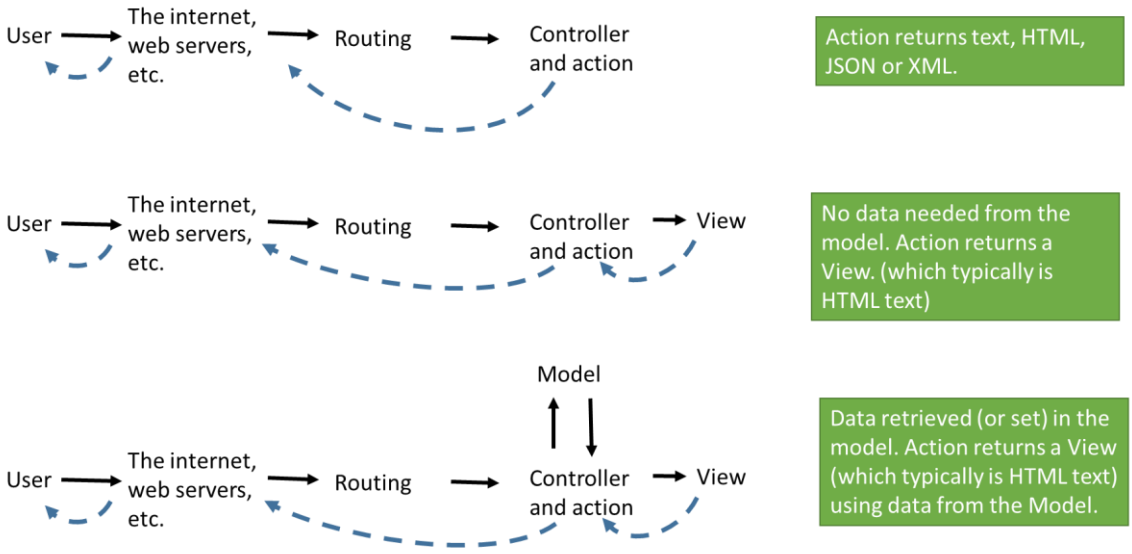
Copyright © 2019 MAX Technical Training

templates include two view engines: ASP.NET Web Forms and Razor. Additional View Engines are available:

- ASP.NET Web Forms
- Razor ← The most popular and the one covered in this course.
- NHAML
- Spark
- And more…
- And… you can write your own!

# Maybe they should have called it RCMV!

In spite of the order of the words, Model, View, Controller, the flow of a request from a user typing a URL to the server returning the page is more in the order of Route, Controller, Model, View. (I guess RCMV was not a very good acronym!) Depending on the request, MVC might not even reference a Model or a View, but it will always use a route and a controller!




For a much more detailed diagram of the flow through an MVC project see the following article and click the "Download PDF Document" link.

- Lifecycle of an ASP.NET MVC 5 Application
  https://docs.microsoft.com/en-us/aspnet/mvc/overview/getting-started/lifecycle-of-an-aspnet-mvc-5-application

# History

The first Community Technology Preview (CTP) of ASP.NET MVC was released in December 2007. Since then there have been seven major releases:

- MVC 1.0 March 2009
- MVC 2.0 March 2010
- MVC 3.0 January 2011

Copyright © 2019 MAX Technical Training

- MVC 4.0 August 2012
- MVC 5.0 October 2013 (MVC 5.2.3 February 2015)
- MVC 6.0.0-rc1 (rc=Release Candidate) November 2015
- ASP.NET Core MVC 1.0.0 August 2016
- ASP.NET Core MVC 2.0.0 August 2017
- ASP.NET Core MVC 2.1.0 May 2018
- ASP.NET Core MVC 2.2.0 November 2018

For the current status of 6.0 and Core 1.0 see https://github.com/aspnet/Mvc/releases

# Dot Net Framework MVC 5 vs. Dot Net Core MVC

| | .Net Framework | .Net Core |
|---|---|---|
| Open source | Only a read-only subset | yes |
| Platform | Windows | Windows, macOS and Linux operating systems |
| Architectures | X86, x64 | x64, x86, and ARM |
| Framework deployment | Framework-dependent deployment | Framework-dependent deployment or Self-contained deployment |
| Framework location | C:\Windows\Microsoft.Net | C:\Program Files\dotnet<br>(find with dotnet --info) |
| Referenced assemblies | *pathToYourProject*\packages<br>(Each project gets its own copy) | C:\Users*yourUserName*\.nuget\packages<br>(Shared with all projects) |
| Hosting | IIS | IIS, self-hosted, Nginx web server on Linux, Kestrel |
| New projects | VS templates, or by hand | VS templates, the "dotnot" CLI or by hand |
| Project structure (many common features and many detail differences) | Web.config, Global.asax, App_Start folder | appsettings.json, Program.cs, Startup.cs |
| Project static content (scripts, icons, etc.) | Site root or folders such as "scripts" or "fonts". | The wwwroot folder. |
| Project start up code | Global.aspx and /App_Start/* | Startup.cs |
| Razor help syntax | HTML helpers<br>( @HTML.helpername ) | Both HTML helpers and HTML-like tags (<a asp-net=""/>) |

The above is a summary. For more details see:

- About .NET Core
  https://docs.microsoft.com/en-us/dotnet/core/about
- Migrate from ASP.NET MVC to ASP.NET Core MVC
  https://docs.microsoft.com/en-us/aspnet/core/migration/mvc?view=aspnetcore-2.2

# MVC and Auto-magic

It may seem that Microsoft's MVC is put together with magic. One of the powers, and the mysteries, of the ASP.NET MVC framework is how often there is a path from "A" to "C" with no "B" in sight. When you don't supply every detail and parameter, the MVC framework applies a set of rules to guess what you want

Copyright © 2019 MAX Technical Training

to do. In this course we will be quite clear as to where “auto-magic” is at work and where you may want to avoid it.

Examples:

- URLs don’t go to pages in MVC. URLs go to a routing process that then picks a Controller class and an Action method. The Action then may return a View that contains the HTML.
- When you don’t explicitly define a route, rules are applied to figure out one.
- When you don’t supply a connection string to the Entity Framework, it looks for a connection string entry in your web.config file with a name that matches the class name.
- When you don’t explicitly specify a view name, MVC uses the Action’s name to find the view.
- You can store View files in multiple locations. MVC will go hunting to find it!

Note: The formal term for “auto-magic” is “conventions”.

# MVC – Highly Customizable

MVC is a programming model and ASP.NET provides a set of templates to help you build an MVC based application. You can customize, replace or ignore every part of MVC.

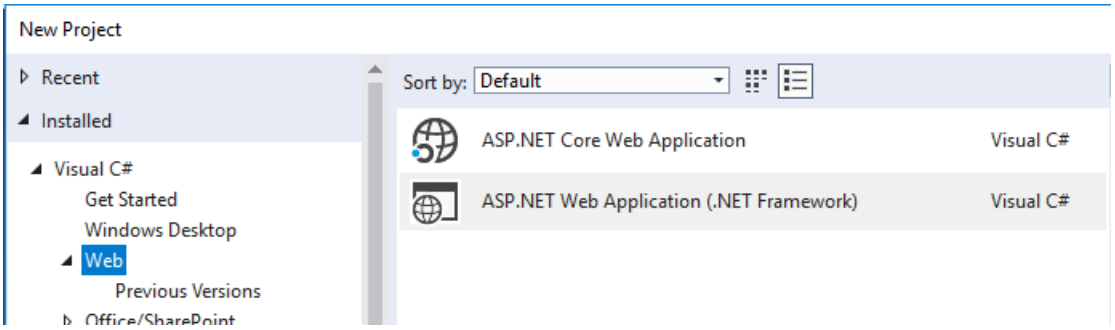
- If you don’t like the Entity Framework, write your own data classes.
- If you don’t like how controllers are auto selected, write your own controller factory class.
- If you don’t like Razor views, select another view model or write your own.

# Creating a New MVC Project

Visual Studio includes templates to get you started with MVC. Pick a template, pick some options, and you have a running project!

## Pick a version

Select .NET Framework or Core:




## Pick the Template:

.NET Framework:

Copyright © 2019 MAX Technical Training

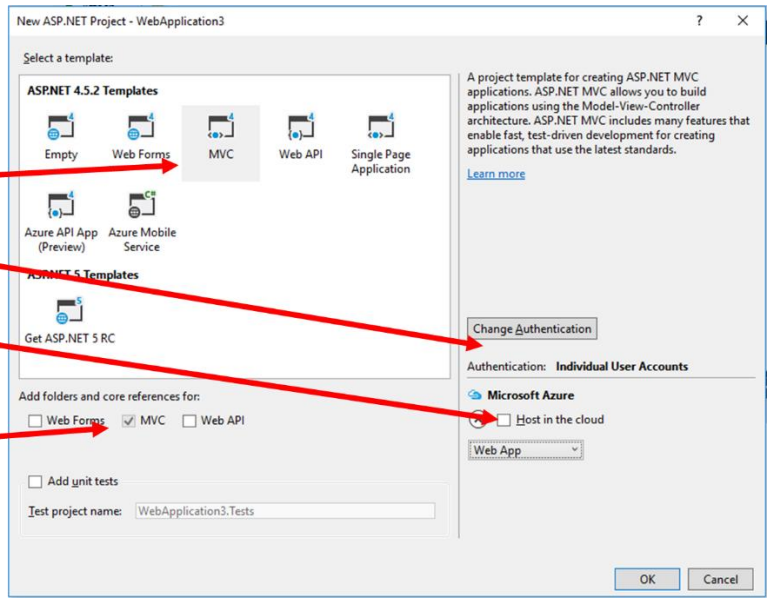
**Important Steps for a new MVC Project**

Click the MVC icon.

Make sure the correct authentication option is selected. (For quick testing change this to “No Authentication”.)

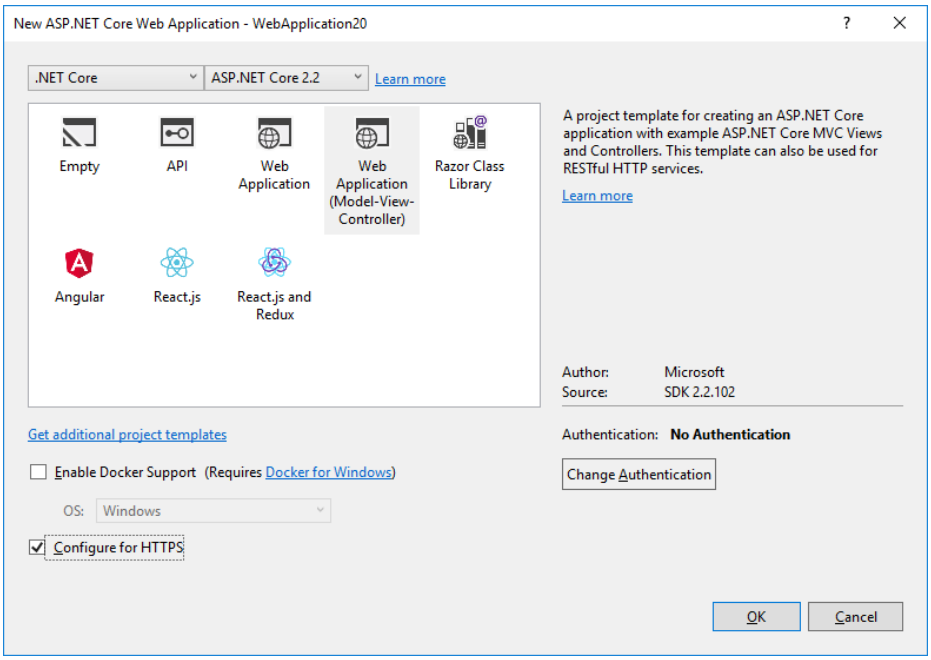
If not planning to deploy this project to Azure, uncheck “Host in the cloud”.

Optionally include support for Web API.




.NET Core

The options are similar, but you can also pick the version of .Net Core, HTTPS support and Docker support.




Note: .NET Core applications can be created from Visual Studio templates or from the CLI (command prompt). The project can then be opened in Visual Studio 2017, Visual Studio Code, Notepad or your favorite code editor.

Examples:

Create a new MVC project in the Projects\myMVC folder with HTTPS support enabled:

```
cd c:\Projects\myMVC
dotnet new mvc
```

Create a new MVC project in the current folder with HTTPS support not enabled:

```
dotnet new mvc --no-https
```

Copyright © 2019 MAX Technical Training

Create a new MVC project in the current folder with HTTPS support enabled and a select authentication method:

```
dotnet new mvc -au none
```
or IndividualB2C or SingleOrg or Windows

List available templates:

```
dotnet -l
```

# References of Note (.Net Framework)

The .Net Framework MVC templates include a set of assembly references. Some of these you may never have seen before. Take a look at the References folder of the project's Solution Explorer. As you might expect, there are a number of "System." and "Microsoft." files.

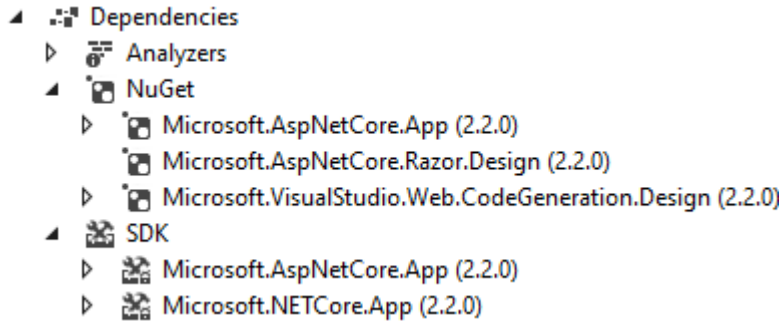
- **Antlr3.Runtime.dll** is used by WebGrease.dll
- **WebGrease.dll** is used by System.Web.Optimization for minification of JavaScript files. Minification removes excess spaces, tabs, comments, long variable names, etc. to produce more compact and faster downloading files. You can remove WebGrease and Antlr3.Runtime from your project and it will still work, but you won't be able to take advantage of the bundling and minification performance improvements.
- **EntityFramework.dll** Note that this does not have the "System" or "Microsoft" prefix. Starting with version 6, the Entity Framework is an open source project licensed under Apache License v2.
    - It is available from https://github.com/aspnet/EntityFramework6 and from NuGet.org.
    - Microsoft code examples and documentation assume you will be using the Entity Framework for your MVC models. EF is not a requirement for MVC, or for models, as you can create your own data classes.
    - Entity Framework is covered in Module 7.
- **OWIN** (Open Web Interface for .NET) This is added when you select "Authentication – Individual User Accounts" when creating a new project.
    - For more on OWIN and Katana see https://msdn.microsoft.com/en-us/magazine/dn451439.aspx
    - If you are creating a Forms Based Authentication project and don't need cloud based authentication, you can strip out OWIN if you like. (You may want to leave it just in case you decide to use Google, Facebook, Microsoft, etc. accounts for logins in the future. And, it will only reduce your deployment size by about 1mb.)
      One example of how to remove OWIN: http://kcshadow.net/aspnet/?q=mvcauthentication
- **Newtonsoft.Json** is a framework to serialize and de-serialize objects to and from the JSON format.
    - Common uses of this library:
        - Serializing and de-serializing JSON.
          ```
          string str = JsonConvert.SerializeObject(someObject);
          var someObject = JsonConvert.DeserializeObject< someObjectType >(str);
          ```
        - LINQ to JSON – Support for Language Integrated Queries.
    - See: http://www.newtonsoft.com/json

# References/Dependencies of Note (.Net Core)

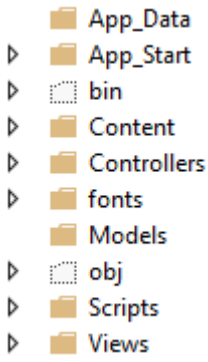
Dependencies are defined in the myCoreMvc.csproj file and displayed in Visual Studio in the Dependencies folder.

The following are built-in:

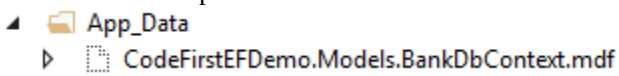
Copyright © 2019 MAX Technical Training




# Common MVC and ASP.NET Project Folders




- **App_Data** – used for XML and other data files. Some developers will store their SQL database files here for their development environment.



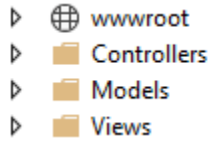

- **App_Code** – Not usually added by default. Used to store code files for shared classes. Files stored here cannot be accessed by IIS / URL.
- **App_Start** – Typically used to store files called from Global.asax such as routing and bundle configuration. (Older versions of ASP.NET templates had this code embedded in the Global.asax file.)
- **Areas** – Not added by default. See Module 8 for more on MVC Areas.
- **bin** – This folder contains all of the files needed for the released project. This folder is usually hidden and can be displayed by clicking Show All Files at the top of the Solution Explorer.
- **Content** – Typically used to store CSS and image files.
- **Controllers** – Controller classes.
- **fonts** – Web embedded fonts. The MVC templates install four versions of a font that is used by Bootstrap.
- **Models** – Classes used to represent Models and View Models including the DbContext file. The Entity Framework will also place its Database First and Model First files here.
- **Scripts** – JavaScript files. The MVC templates will store Bootstrap, jQuery and other libraries here.
- **Views** – This is the default search location for View files. The Views folder will contain a folder for each Controller's Views and a Shared folder.

Notes:

- The App_Code, App_Data, bin and obj folders are not unique to MVC templates. These have had special meaning in ASP.NET projects since ASP.NET 2.0 and Visual Studio 2005.

Copyright © 2019 MAX Technical Training

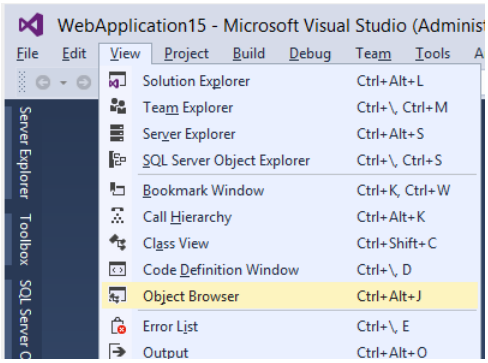
# Common .Net Core MVC Project Folders



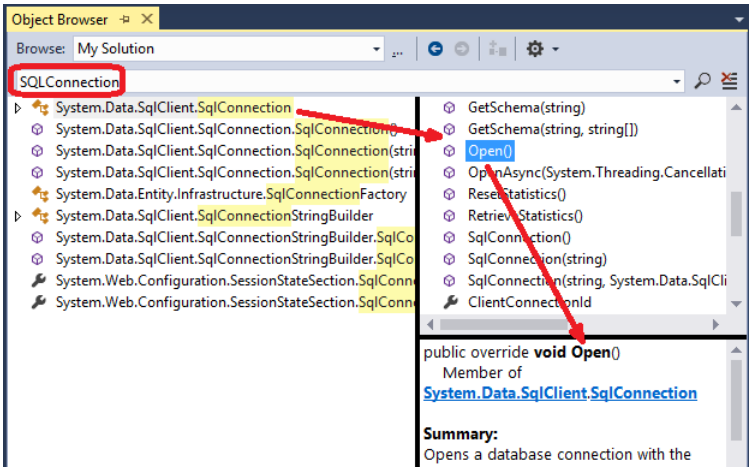

- **wwwroot** – Static code such as JavaScript, CSS and images.
- **Controllers** – Controller classes.
- **Models** – Classes used to represent Models and View Models including the DbContext file. The Entity Framework will also place its Database First and Model First files here.
- **Views** – This is the default search location for View files. The Views folder will contain a folder for each Controller's Views and a Shared folder.

# Use the Object Browser to Explorer Namespaces, Classes and Methods

Visual Studio's Object Browser is used to explore both referenced namespaces and classes, and namespaces and classes created inside of your project. The Object Browser can be launched from the View menu or by pressing Ctrl+Alt+J.




In the Object Browser, you can drill down through the namespaces and classes to find a class or its methods, or you can search for them.




Copyright © 2019 MAX Technical Training

**Object Browser Icons**

See: https://msdn.microsoft.com/en-us/library/y47ychfe.aspx

| Icon | Description | Icon | Description |
|---|---|---|---|
| | Class | | Map |
| | Constant | | Map Item |
| | Delegate | | Method or Function |
| | Enum | | Module |
| | Enum Item | { } | Namespace |
| | Error | | Operator |
| | Event | | Property |
| | Exception | | Structure |
| | Extension Method | ‹T› | Template |
| | External Declaration | | Type Forwarding |
| | Field or Variable | | TypeDef |
| | Interface | | Union |
| | | | Unknown |

Each icon can have a signal icon to indicate their accessibility:

| Icon | Description |
|---|---|
| none | **Public**. Accessible from anywhere in this component and from any component that references it. |
| | **Protected**. Accessible from the containing class or type, or those derived from the containing class or type. |
| | **Private**. Accessible only in the containing class or type. |
| | **Sealed**. |
| | **Friend/Internal**. Accessible only from the project. |
| | **Shortcut**. A shortcut to the object. |

# A Few Notes about Databases for Developers

Microsoft SQL Server is available in numerous versions and editions. For the purposes of this class we are interested in these simple definitions listed below.

- **Full SQL Server**
  - Requires a license.
  - Includes tools and additional features.
- **SQL Server Developer Edition**
  - Free.
  - Same features as the full version, but only licensed for development, test and demonstration purposes only.
- **SQL Express**
  - Free.
  - Limited to 10 GB per database.

Copyright © 2019 MAX Technical Training

  - Does not include Reporting Services or Full Text Search.
  - Also available in an "Advanced Services" edition that does include Reporting Services and Full Text Search.
  - Can be used for in production.
  - Cannot be scaled.
  - Pretty much the full version of SQL Server minus some enterprise level features.
- **LocalDB**
  - Free.
  - Basically, this is the SQL Express engine without tools.
  - LocalDB does not accept remote connections and cannot be administered remotely, so it is used to support only a local application, or for development purposes.
  - Versions and default instances:
    - (LocalDB)\V11.0 - (SQL Express 2012)
    - (LocalDB)\MSSQLLocalDB - (SQL Express 2014)
  - No tools installed, but can be administered using SQL Management Studio and Visual Studio.
  - The system database files for the LocalDB are stored in the user's local AppData path, which is normally hidden by Windows.

# C# and Visual Studio Notes and Tips

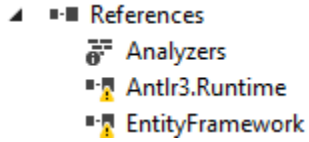
**C#**

- If you have not worked with Lambda Expressions then please read:
  https://msdn.microsoft.com/en-us/library/bb397687.aspx
- Non-nullable types can be made nullable with a question mark.

```
public int? ReorderPoint { get; set; }
```

**Visual Studio**

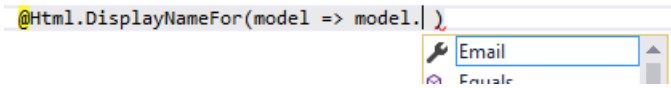
- Yellow flags in the References folder indicates that the referenced files are missing. This is often the case for sample projects where the referenced files have been deleted from the project's packages folder to make the download file smaller.




  - To fix:
    - Right-click the References folder and click Manage NuGet Packages and note the yellow bar across the top of the NuGet tab.
    - Click Restore.
    - In the Solution Explorer click the Refresh button and verify that the yellow flags are now gone.
- When editing a Razor view, IntelliSense may not work as expected:
  - If the view is not strongly typed (@model …) then there will be no IntelliSense for the Model object.
  - IntelliSense may not display, or may pop up and disappear, when there is a closing parentheses just after the cursor. Before typing the "." For a property add a space.
    Does not work:

```
@Html.DisplayNameFor(model => model.)
```

    Works:




Copyright © 2019 MAX Technical Training

Copyright © 2019 MAX Technical Training

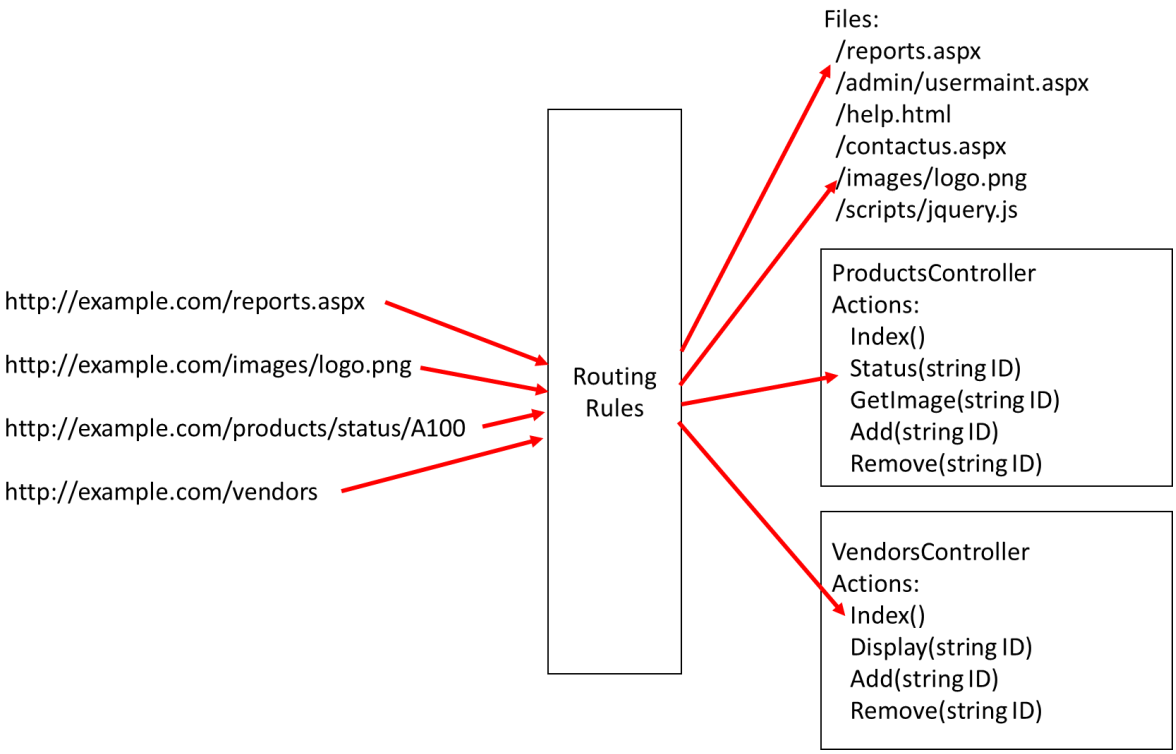
# Module 2 – Routing

MVC URLs don't point to pages or files, they point to controllers. Or rather, they are routed to controllers. When a user enters a URL like http://www.example.com/helicopters, they could be routed to a controller named ProductsController and an action named Category with a parameter of heli. That action could then return a view named "Rotorcraft" or a view named "Specials This Month".

The use of routing:

- Gives you the flexibility to later restructure your application without breaking existing URLs.
- Does not keep you bound to a rigid folder/file structure.
- Separates site navigation from the Models, Controllers and Views.

Although the technology is called "Model View Controller", nothing happens until a route is selected. While MVC relies heavily on routing, routing is part of ASP.NET and is not limited to MVC projects.




## The Routes Collection

The Routes property is a static property (a Shared property in Visual Basic) that represents a collection of all of the objects that are used to specify how a URL request is matched to a class that handles the request. To specify a route, you add the route definition to the Routes property. Typically, you add routes to the Routes property from an event handler for the Application_Start event in the Global.asax file.

You can map routes:

Copyright © 2019 MAX Technical Training

- By doing nothing! (but this only works for non-MVC files such as images, ".html" and ".aspx" pages)
- Using RouteCollection.MapPageRoute to route to non-MVC pages or files such as ".html" and ".aspx" pages.
- Using RouteCollection.MapRoute to route to MVC Controllers and Actions.
- By using Attribute Routing where the routing information is added to the Controller Action.

Routes are added using these .Net Framework methods:

- routes.IgnoreRoute – Specify routes to ignore, or better, routes to forward to ASP.NET (and not MVC).
- routes.MapPageRoute – Specify routes to pages (files on disk) such as report.aspx.
- routes.MapRoute – Specify routes to Controllers and Actions.
- routes.MapMvcAttributeRoutes – Find all attribute based routes and add them to the list of routes.

Routes are added using these .Net Core methods:

- routes.MapRoute – Specify routes to Controllers and Actions.
- routes.MapAreaRoute – Specify routes to Areas, Controllers and Actions.

Processing of Routes:

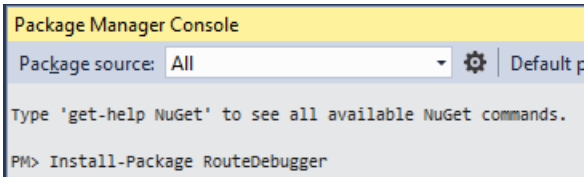
- Files on disk are searched first. I.e. if the URL is to /contactus.aspx, and it is found it will be used.
- The order routes are added is important. They are processed from first to last.
- The last route should be the default or catch all route.
- If no route matches, then the client receives a 404 Not Found error.

# Debugging Routes (.Net Framework)

Routes can often be difficult to debug as you cannot just step through code to explore the routing logic. RouteDebugger is a tool that you can add to your project to test and debug routes. (Search the web for other route debugging tools.)

To add RouteDebugger to your project:

- Right-click References, click Manage NuGet Packages, click Browse and search for "routedebugger" and then install it.
  Or…
- Open the Package manager console (Tools, NuGet Package Manager, Package Manager Console) and type Install-Package RouteDebugger.




This will add one DLL to your project, add RouteDebugger to the list of references, and add the following line to your web.config.

```
<add key="RouteDebugger:Enabled" value="true" /></appSettings>
```

Copyright © 2019 MAX Technical Training

To disable the Route Debugger, change the value above to "false" or remove the RouteDebugger NuGet package.

The next time you run your project you will find data added to the bottom of each and every page. The following example was for a route that uses a constraint to match a phone number. (http://localhost:61660/**123-123-1234**) Notice that three routes were matched ("True" in the first column), and the first one found, top down, was used.

**Route Debugger**

Type in a url in the address bar to see which defined routes match it. A {*catchall} route is added to the list of routes automatically in case none of your routes match.

To generate URLs using routing, supply route values via the query string. example: `http://localhost:14230/?id=123`

**Matched Route**: {phone}

**Route Data**

| Key | Value |
|---|---|
| phone | 123-123-1234 |
| controller | Home |
| action | PhoneNumber |

**Data Tokens**

| Key | Value |
|---|---|

**All Routes**

| Matches Current Request | Url | Defaults | Constraints | DataTokens |
|---|---|---|---|---|
| False | __browserLink/requestData/{requestId} | (null) | (null) | (null) |
| False | {resource}.axd/{*pathInfo} | (null) | (empty) | (null) |
| True | {phone} | controller = Home, action = PhoneNumber, id = UrlParameter.Optional | phone = \d{3}-\d{3}-\d{4} | (empty) |
| False | aboutus.aspx | controller = Home, action = Return404, id = UrlParameter.Optional | (empty) | (empty) |
| False | plane | (null) | (null) | (null) |
| True | {controller}/{action}/{id} | controller = Home, action = Index, id = UrlParameter.Optional | (empty) | (empty) |
| True | {*catchall} | (null) | (null) | (null) |

# Unit Tests for Routes

As true unit testing of routing is not easy, there are several unit test tools available. See here for examples: https://www.nuget.org/packages?q=routing+unit+test

# .Net Framework Routes

Your ASP.NET Web Application can contain either or both of the following site technologies:

- MVC – to create a web site
- Web API – to create web services

Both MVC and Web API use routing to direct requests to Controllers.

## Web API Routes

*(Web API and Web API routing are covered in detail in a later module. It is covered here for completeness. You may want to skip on to "MVC Routes" for now.)*

While not a direct part of MVC, Web API is often part of an MVC project. Like MVC, Web API uses routing and controllers. (You could even call Web API "MC" or "Model Controller".)

Copyright © 2019 MAX Technical Training

**Example Web API URL**: http://www.example.com/**api**/products/123

Visual Studio templates that include Web API add an entry to Global.asax and a file, WebApiConfig.cs, to the App_Start folder.

**Added to Global.asax:** GlobalConfiguration.Configure(WebApiConfig.Register);

The WebApiConfig.cs in a new project adds two entries to the route table by default.

- MapHttpAttributeRoutes – Adds support for routes defined using method attributes (new in MVC5).
- Routes.MapHttpRoute – Adds a route using convention-based routing based on a template.

**WebApiConfig.cs:**

Here is the default routing for Web API.

```
public static class WebApiConfig
{
    public static void Register(HttpConfiguration config)
    {
        // Web API configuration and services

        // Web API routes
        config.MapHttpAttributeRoutes();

        config.Routes.MapHttpRoute(
            name: "DefaultApi",
            routeTemplate: "api/{controller}/{id}",
            defaults: new { id = RouteParameter.Optional }
        );
    }
}
}
```

Note: As the WebApiConfig.Register method receives a WebApiConfig.Register object as a parameter, the following examples have the “config” variable, as seen in the example above, replaced with the object it holds (WebApiConfig.Register).

## WebApiConfig.Register.MapHttpAttributeRoutes();

This code adds support for Web API 2.0 attribute routing and uses attributes on methods in controllers instead of predefined maps to manage routes. If MapHttpAttributeRoutes is enabled, then the Route attribute in the following example says to call this Action when a request is received using the specified URL pattern.

```
[Route("customers/{customerId}/orders")]
public IEnumerable<Order> GetOrdersByCustomer(int customerId) { ... }
```

The example path above with the customerId placeholder in the middle of the route cannot be handled using MapHttpRoute.

For details see: Attribute Routing in ASP.NET Web API 2
https://www.asp.net/web-api/overview/web-api-routing-and-actions/attribute-routing-in-web-api-2

Copyright © 2019 MAX Technical Training

## WebApiConfig.Register.Routes.MapHttpRoute

Routing can also be configured using convention-based routing where you supply a route template. The following is the example found in a new Web API project.

```
config.Routes.MapHttpRoute(
    name: "DefaultApi",
    routeTemplate: "api/{controller}/{id}",
    defaults: new { id = RouteParameter.Optional }
);
```

The "api" in the routeTemplate is a convention and not a requirement. It is recommended that you use something similar to avoid path naming conflicts with MVC routes in the same project.

Note that the above example for MapHttpRoute is using named parameters. You could also have written the above using positional parameters:

```
config.Routes.MapHttpRoute(  "DefaultApi", "api/{controller}/{id}",
   new { id = RouteParameter.Optional }  );
```

Also see: https://www.asp.net/web-api/overview/web-api-routing-and-actions/routing-in-aspnet-web-api

# Page Routes

If you need an MVC-like URL to point to an ASP.NET Web Forms file, or to a traditional HTML web file, you can add a route to the file using MapPageRoute.

Example: Map from /SafetyTips.aspx to /SafetyFirst (i.e. a URL to "/SafetyFirst" will be return the contents of "/SafetyTips.aspx".)

This should be as simple as:

```
routes.MapPageRoute("RouteToLegacyPage", "Home/SafetyFirst", "~/SafetyTips.aspx");
```

Or

```
routes.MapPageRoute(
     routeName: "RouteToLegacyPage",
     routeUrl: "Home/SafetyFirst",
     physicalFile: "~/SafetyTips.aspx");
```

And while this seems to work in simple examples, it fails when the Razor Html.ActionLink tries to find a normal MVC route. All of the MVC routes get mapped to the legacy page. The workaround requires a little trick, we have to add a constraint that almost looks like an "if 2=2" test. In place of the routeUrl we add a place holder. Any name will do. And then add a constraint using that placeholder. As the MapPageRoute method that includes "constraint" has six required parameters, we also need to pass a "false" for "checkPhysicalUrlAccess" and a dummy value for "default" (could be null). So here's what we need:

```
routes.MapPageRoute("RouteToLegacyPage", "{page}", "~/SafetyTips.aspx",
    false,
    new RouteValueDictionary(),
    new RouteValueDictionary { { "page", "SafetyFirst" } });
```

Or with named parameters:

Copyright © 2019 MAX Technical Training

```
routes.MapPageRoute(
    routeName: "RouteToLegacyPage",
    routeUrl: "{page}",
    physicalFile: "~/SafetyTips.aspx",
    checkPhysicalUrlAccess: false,
    defaults: new RouteValueDictionary(),
    constraints: new RouteValueDictionary { { "page", "SafetyFirst" } });
```

# MVC Routes

To add MVC routes you will either define routes using the MapRoute method or add attributes directly to the Action methods of a controller class. Even when using attributes, you will still use MapRoute to define a default route. (*Attributes are described in the next section*.)

## The Default MVC Route

Every new MVC project includes a default route that uses the controller/action/id pattern. This pattern defaults to a controller named "Home" and an Action named "Index" and includes an optional parameter named "id". This is effectively a "catch all" route that will send a request with no path ("http://www.example.com") to the home page of the site. It will also support the default "auto-magic" to direct requests using controller and action names.

The "default" or "catch all" route should be the last one added.

```
routes.MapRoute(
    name: "Default",
    url: "{controller}/{action}/{id}",
    defaults: new { controller = "Home", action = "Index", id = UrlParameter.Optional }
);
```

or

```
routes.MapRoute( "Default", "{controller}/{action}/{id}", new { controller = "Home",
  action = "Index", id = UrlParameter.Optional }  );
```

Any time the routing process reaches this default route, and there is no controller and action that matches, the user will receive a 404 Not Found error.

## Adding Custom Routes

You may want to create friendlier routes for your users or create routes to map to legacy pages in your project.

To create a custom route, you call the MapRoute method with these properties:

- string **name** – A unique name. No two routes can have the same name.
- string **url** – The URL or URL pattern for the route. May include placeholders for controller, view and parameters.
- (optional) object **defaults** – Defaults to use if placeholders are not populated in the URL.
- (optional) object **constraints** – Regular expressions to define placeholders. (I.e. must be numeric, match a phone number, or match a part number pattern.)
- Notes:
    - A route URL cannot start with a slash.
    - You must define a controller, either as a placeholder ({controller}) or as a default.
    - The default route applies the pattern of {controller}/{action}/{id}

Copyright © 2019 MAX Technical Training

- The parameter names like "{id}" are not case sensitive and do not need to match the case of the Action's parameter.
- If you define a default for one parameter, you must supply defaults for all following parameters.

### Example:

Consider a project where you have a Products controller with a Category action that accepts a string of keywords. Using the default routing you could use this URL to access that action:

```
http://www.example.com/Products/Category/helicopters
```

While that works, you might want to supply a shorter URL for our users:

```
http://www.example.com/Products/helicopters
```

If we depended on the default routing, this would attempt to call the "helicopters" action on the "Products" controller. Our user would get a 404 Not Found error. We need to create a custom route for this shorter path.

For the custom route we described above we would create something like this:

```
routes.MapRoute(
   name: "ProductsCategory",
   url: "Products/{id}",
   defaults: new { controller = "Products", action = "Category" }
);
```

Notes:

- Any "name" will do, as long as it is unique within the collection of routes.
- The "url" is not case sensitive. You could use "products" or "Products".
- The "id" must match the parameter name in the action, but is not case sensitive.
  Sample action: `public ActionResult Category(string id) { }`
- "Products" in the name, url and defaults do not have to match. The "products" in the URL is what the user types and the "products" in the defaults is the name of the controller to select. They do not have to be the same.
- You will want to set a default for the "id" if you still want /Products to go to the default Index action.

## Mapping Routes to Actions and Parameters

We will dive deeper into Actions, including path and Query String parameters in the "Controllers" module.

## Route Constraints

You can define constraints, or patterns, for each of the named placeholders listed in the URL property. Constraints are regular expressions.

The following MapRoute is only a match if the URL matches the constraint's regular expression pattern for a phone number.

```
routes.MapRoute(
    name: "phone",
```

Copyright © 2019 MAX Technical Training

```
    url: "{phone}",
    defaults: new { controller = "Home", action = "PhoneNumber" },
    constraints: new { phone = @"\d{3}-\d{3}-\d{4}" } );
```

The following uses a regular expression match for one or more integers:

```
routes.MapRoute(
    name: "part",
    url: "parts/{part}",
    defaults: new { controller = "Parts", action = "PartLookup" },
    constraints: new { part = @"\d+" } );
```

Tip: The "@" in front of the string indicates that it is a C# literal string. Without it, any "\" characters would be treated as escape characters. As an example, "\n" is the new line escape sequence. "Hello\nWorld" displays "Hello" on one line and "World" on the next line. The "@" literal string is frequently used with file paths. As an example, "C:\news\products" would be treated as "C:\<*newline*>ews\products" while @"C:\news\products" will always behave as expected. See: https://msdn.microsoft.com/en-us/library/h21280bw.aspx

## RouteCollection vs. HttpConfiguration.Routes

An obvious question is why the routes for controllers are added to the RouteCollection object and Web API routes are added to HttpConfiguration.Routes. MVC controllers are only designed to work with IIS. Web API controllers can work with either IIS or within Windows Communications Foundation (WCF). *Items added to HttpConfiguration.Routes are also automatically added to RouteCollection.* The templates for Web API use HttpConfiguration.Routes for consistency between web projects and WCF hosted projects.

An interesting discussion on the topic:
http://stackoverflow.com/questions/23381962/routetable-routes-vs-httpconfiguration-routes

# .Net Core Routes

While similar to .Net Framework routes, their a few differences:

- Routes are defined in StartUp.cs instead of Global.asax or /App_Start/RouteConfig.cs. https://docs.microsoft.com/en-us/aspnet/core/fundamentals/routing
- Many of the features missing from .Net Core routes can duplicated by writing Routing Middleware. https://docs.microsoft.com/en-us/aspnet/core/fundamentals/routing?view=aspnetcore-2.2#use-routing-middleware
- These are missing: Page Routes (.MapPageRoute) and .IgnoreRoute.

## MVC Routes

To add MVC routes you will either define routes using the MapRoute method or add attributes directly to the Action methods of a controller class. Even when using attributes, you will still use MapRoute to define a default route. (*Attributes are described in the next section*.)

## StartUp.cs

Routing is added to the Configure method of the Startup class in Startup.cs.

Copyright © 2019 MAX Technical Training

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

## The Default MVC Route

Every new MVC project includes a default route that uses the controller/action/id pattern. This pattern defaults to a controller named “Home” and an Action named “Index” and includes an optional parameter named “id”. This is effectively a “catch all” route that will send a request with no path (“http://www.example.com”) to the home page of the site. It will also support the default “auto-magic” to direct requests using controller and action names.

Note: While the code is similar to .Net Framework routes, there are a few differences.

The “default” or “catch all” route should be the last one added.

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

Any time the routing process reaches this default route, and there is no controller and action that matches, the user will receive a 404 Not Found error.

## Adding Custom Routes

You may want to create friendlier routes for your users or create routes to map to legacy pages in your project.

To create a custom route, you call the MapRoute method with these properties:

- string **name** – A unique name. No two routes can have the same name.
- string **template** – The URL or URL pattern for the route. May include placeholders for controller, view and parameters.
- (optional) object **defaults** – Defaults to use if placeholders are not populated in the URL.
- (optional) object **constraints** – Regular expressions to define placeholders. (I.e. must be numeric, match a phone number, or match a part number pattern.)
- Notes:
    - A route URL cannot start with a slash.
    - You must define a controller, either as a placeholder ({controller}) or as a default.
    - The default route applies the pattern of {controller}/{action}/{id}
    - The parameter names like “{id}” are not case sensitive and do not need to match the case of the Action’s parameter.
    - If you define a default for one parameter, you must supply defaults for all following parameters.

### Example:

Consider a project where you have a Products controller with a Category action that accepts a string of keywords. Using the default routing you could use this URL to access that action:

http://www.example.com/Products/Category/helicopters

Copyright © 2019 MAX Technical Training

While that works, you might want to supply a shorter URL for our users:

http://www.example.com/Products/helicopters

If we depended on the default routing, this would attempt to call the "helicopters" action on the "Products" controller. Our user would get a 404 Not Found error. We need to create a custom route for this shorter path.

For the custom route we described above we would create something like this:

```
routes.MapRoute(
    name: "ProductsCategory",
    template: "Products/{id}",
    defaults: new { controller = "Products", action = "Category" }
    );
```

Notes:

- Any "name" will do, as long as it is unique within the collection of routes.
- The "template" is not case sensitive. You could use "products" or "Products".
- The "id" must match the parameter name in the action, but is not case sensitive.
  Sample action: `public ActionResult Category(string id) { return "Some category " + id; }`
- "Products" in the name, template and defaults do not have to match. The "products" in the URL is what the user types and the "products" in the defaults is the name of the controller to select. They do not have to be the same.
- The example above will generate an error if the "id" is not provided as this new route will look the same as the default route when no parameter is supplied.

  An unhandled exception occurred while processing the request.

  AmbiguousMatchException: The request matched multiple endpoints. Matches:

  DotNetCoreMVC.Controllers.ProductsController.Index (DotNetCoreMVC)
  DotNetCoreMVC.Controllers.ProductsController.Category (DotNetCoreMVC)

  To solve this, add one more route in the following order:

```
routes.MapRoute(
    name: "ProductsCategory",
    template: "Products",
    defaults: new { controller = "Products", action = "Index" }
 );

routes.MapRoute(
    name: "ProductsCategory",
    template: "Products/{id}",
    defaults: new { controller = "Products", action = "Category"}
 );

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

## Mapping Routes to Actions and Parameters

We will dive deeper into Actions, including path and Query String parameters in the "Controllers" module.

## Route Constraints

You can define constraints, or patterns, for each of the named placeholders listed in the URL property. Constraints are regular expressions.

Copyright © 2019 MAX Technical Training

The following MapRoute is only a match if the URL matches the constraint's regular expression pattern for a phone number.

```
routes.MapRoute(
    name: "phone",
    template: "{phone}",
    defaults: new { controller = "Home", action = "PhoneNumber" },
    constraints: new { phone = @"\d{3}-\d{3}-\d{4}" } );
```

The above would map to this controller and action:

```
public class HomeController : Controller
{
    public IActionResult PhoneNumber(string phone)
    {
        return Content(phone);
    }
```

The following uses a regular expression match for one or more integers:

```
routes.MapRoute(
    name: "part",
    template: "parts/{part}",
    defaults: new { controller = "Parts", action = "PartLookup" },
    constraints: new { part = @"\d+" } );
```

Tip: The "@" in front of the string indicates that it is a C# literal string. Without it, any "\" characters would be treated as escape characters. As an example, "\n" is the new line escape sequence. "Hello\nWorld" displays "Hello" on one line and "World" on the next line. The "@" literal string is frequently used with file paths. As an example, "C:\news\products" would be treated as "C:\<*newline*>ews\products" while @"C:\news\products" will always behave as expected. See: https://msdn.microsoft.com/en-us/library/h21280bw.aspx

# Attribute Based Routes

In earlier versions of MVC routing you could only add routes using MapRoute (usually in RouteConfig.cs or Global.asax). Starting with MVC5, you can also define routes directly in the controller class by adding attributes to the controller's Actions. You can define routes using either or both techniques.

If the following action was in the Home controller, then it could be called using: http://www.example.com/home/findproducts.

```
public ActionResult FindProducts(int Id)
{
    return Content("Product search not enabled!" );
}
```

If you wanted to call it using http://www.example.com/productlookup you could either add a route in RouteConfig.cs or add an attribute to the Action.

**Using RouteConfig.cs (.Net Framework):**

```
routes.MapRoute(
    name: "ProductLookup",
    url: "productlookup/{id}",
    defaults: new { controller = "Home", action = "FindProducts" });
```

Copyright © 2019 MAX Technical Training

**Using RouteConfig.cs (.Net Core):**

Same as above, after replacing "url" with "template".

**Using an attribute on the FindProducts Action in the Home controller:**

```
[Route("productlookup/{id}")]
public ActionResult FindProducts( int id )
{
   return Content("Product search not enabled!" );
}
```

## Which Should You Use?

Benefits of convention-based routing (RouteConfig.cs or StartUp.cs)

- Everything about routing is one place.

Benefits of Attribute Routing

- Code is easier to understand as it is clear how the Action is being called.
- Routes and HTTP verbs are both in the same place.
  [Route("productlookup"), HttpVerbs.Post]
- Quicker to find code for a particular route. (I.e. search for "route("productlookup")")
- Additional features such as simple data type constraints.

Best Practices:

- If only using convention-based routing, add comments to each Action to document how the Action is being called.
  // URL: /productlookup
- If only using Attribute Routing, add a comment to either Global.asax or RouteConfig.cs stating so.
  // This site uses Attribute Routing. Visit each controller to discover routing.
- If using both… document everywhere!!!

## Enabling Attribute Routing (.Net Framework)

To enable Attribute Routing add the following line to your list of routes in RouteConfig.cs.

```
routes.MapMvcAttributeRoutes();
```

Note that the order of adding routes is important! In the following example, the Attribute based routes would be evaluated first, then the custom MapRoute routes and then the default route.

```
routes.IgnoreRoute("{resource}.axd/{*pathInfo}");

routes.MapMvcAttributeRoutes();

routes.MapRoute(
    name: "ProductLookup",
    url: "productlookup",
    defaults: new { controller = "Home", action = "Phonenumber", id = UrlParameter.Optional });

routes.MapRoute(
    name: "Default",
    url: "{controller}/{action}/{id}",
    defaults: new { controller = "Home", action = "Index", id = UrlParameter.Optional }
);
```

In the next example, the custom MapRoute route will be evaluated first.

Copyright © 2019 MAX Technical Training

```
routes.IgnoreRoute("{resource}.axd/{*pathInfo}");

routes.MapRoute(
    name: "ProductLookup",
    url: "productlookup",
    defaults: new { controller = "Home", action = "Phonenumber", id = UrlParameter.Optional });

routes.MapMvcAttributeRoutes();

routes.MapRoute(
    name: "Default",
    url: "{controller}/{action}/{id}",
    defaults: new { controller = "Home", action = "Index", id = UrlParameter.Optional }
);
```

## Attribute Routing Features

Attribute Routing supports the all the features of route rules, often with more flexibility. As an example, one option not available in convention-based routing is the very simple constraint notation using a colon and a data type. ({partnumber:int})

**Optional parameters are added with a question mark.**

[Route("products/**{partnumber?}**")]

i.e. http://example.com/products or http://example.com/products/ABC123

**Default parameters are added with an equal sign.**

[Route("products/**{partnumber=12345}**")]

**Route constraints are added with a colon.**

Convention-based routing lets you specify a constraint by using a regular expression. Attribute Routing adds support for datatypes. Here are a few samples:

[Route("products/{partnumber**:int**}")] must be an integer

[Route("products/{available**:datetime**}")] must be an DateTime

[Route("products/{partnumber**:range(1000,9999)**}")] must be an integer between the two values.

**Multiple constraints are just chained together.**

[Route("products/{partnumber**:int:min(1000):max(9999)**}")] must be an integer from 1000 to 9999

**Regular expressions are also supported with Attribute Routing.**

[Route("products/{available**: regex(^\d{3}-\d{3}-\d{4}$)}**")] must match a regular expression

Also see:

Copyright © 2019 MAX Technical Training

- Attribute Routing in ASP.NET MVC 5
  https://blogs.msdn.microsoft.com/webdev/2013/10/17/attribute-routing-in-asp-net-mvc-5/

# Routes and Parameters

Parameters can be passed to Actions using either values in the path or as Query String values. Both of the following could be used to access the Bikes controller and the FindBike action with four parameters.

http://www.example.com/Bikes/FindBike/42/red/carbonfiber/A100

http://www.example.com/Bikes/FindBike?size=42&color=red&construction=carbonfiber&model=A100

Advantages of using a path/route:

- Shorter URLs.
- More often used by search engines for keywords extraction.
- Can be redefined by changing the routing rules.

Advantages of using query string:

- Parameters have names and are self-describing.
- Parameters can be listed in any order.
- Parameters can be omitted.

## Path Parameters

The keywords you have seen in routes so far have mapped to a controller and an action.

```
url: "{controller}/{action}/{id}",
```

While “controller” and “action” are built-in keywords, “id” is not. Any additional placeholders you add to your url will also be added to a dictionary of values that is passed to the controller. MVC will attempt to match parameters by name. Notice in the example below that all of the parameters are matched except for “{construction}”, which is ignored.


## Dummy Placeholders

You can add “dummy” placeholders so your URLs can contain extra data only needed for search engine optimization (SEO). For example, the bike’s model name might be added with the hope that it will be picked up by the web’s search engines.

Copyright © 2019 MAX Technical Training

You might define a route like "lookupbike/{SEOtext}/{model}" so you can use a URL like the following example where your action only cares about the "model" ("A100") parameter.

http://www.example.com/lookupbike/Speed%20Racer%20100/A100

## Query String Parameters

If you need to use query string parameters, maybe because the URL is being created by an HTML form, MVC will map from the query string's names to an Action's parameters. An additional advantage of query strings is that unlike routes, the values do not have to be supplied in any particular order.

In the example below note that "&construction=carbonfiber" is be ignored.

```
routes.MapRoute(
    name: "FindBikeQ",
    url: "FindBike",
    defaults: new { controller = "Products", action = "FindBikeQ" }
);
```

localhost:4651/findbike?biketype=racing&size=42&color=red&construction=carbonfiber&model=A100

```
public ActionResult FindBikeQ(string biketype, string size, string color, string model)
{
    return Content("Find this bike:  size=" + size + "  color=" + color + "  model=" + model);
}
```

The HTML form to build the query string above might look like the following and could use either POST or GET as the method.

```
<!DOCTYPE html>
<html>
<head>
    <title></title>
    <meta charset="utf-8" />
</head>
<body>
    <form method="get" action="/FindBike">
        Bike Type <input type="text" name="biketype" /> <br />
        Bike Size <input type="text" name="size" /> <br />
        Bike Color <input type="text" name="color" /> <br />
        Bike Construction <input type="text" name="construction" /> <br />
        Bike Model <input type="text" name="model" /> <br />
        <input type="submit" value="Submit" />
    </form>
</body>
</html>
```

Bike Type racing
Bike Size 44
Bike Color Green
Bike Construction Steel
Bike Model A200
Submit

# ASPX and other Routes (.Net Framework only)

How does routing impact non-MVC (or non-controller based) URLs? Prior to checking for a route, ASP.NET checks to see if there is a physical file, and if it finds one, it passes the request to the appropriate handler for the file type. If the file does not exist it then searches for a route that looks like the page name. This default search for files is why image, CSS and JavaScript files are still loaded even though there are no routing rules defined for them.

Copyright © 2019 MAX Technical Training

## Rerouting ASPX and HTML pages

*Note: .Net Core does not support >ASPX or AXD files.*

If you are rewriting a site and your users still have the legacy pages bookmarked, or search engines have them indexed, you may want to still support the old URLs, but redirect them to MVC controllers. You can easily do this using MapRoute and using the ASPX or HTML file name as the URL pattern.

- User types http://www.example.com/aboutus.aspx
- The routing engine looks for a physical file named "aboutus.aspx"
    - **If found**, the request is passed to the ASP.NET handler for ASPX pages.
    - **If not found**, the routing engine search the routes for an entry that matches "aboutus.aspx". The following will redirect the request to the "Home" controller and the "AboutUs" action:

```
routes.MapRoute(
   name: "aboutusRedirect",
   url: "aboutus.aspx",
   defaults: new { controller = "Home", action = "AboutUs", id = UrlParameter.Optional
});
```

## What if the ASPX or HTML file is still on the server?

If you need to leave legacy pages on the server as you migrate to MVP (or other routed) pages, then add one more line to the routing code:

```
routes.IgnoreRoute("{resource}.axd/{*pathInfo}");
routes.RouteExistingFiles = true;
routes.MapRoute( ……
```

Setting routes.RouteExistingFiles to true forces the routing engine to check routes before checking for physical files. This will let you route what looks like a physical page to a controller. The following example will call the Home controller's AboutUs action instead of running the physical file.

```
routes.MapRoute(
   name: "noaspx",
   url: "aboutus.aspx",
   defaults: new { controller = "Home", action = "AboutUs", id = UrlParameter.Optional });
```

Note that using routes.RouteExistingFiles will have a minor impact on performance as each JavaScript, CSS and image file request will be first checked to see if it is a route, and then it will check for a physical file.

## Ignored Routes

The default MVC templates include the following line as the first item in your route definitions:

```
routes.IgnoreRoute("{resource}.axd/{*pathInfo}");
```

"axd" files are dynamically generated by ASP.NET and never match to physical file on disk, and will be processed as if they are to go to an MVC controller. This IgnoreRoute tells the router to ignore all .axd file requests regardless of their names or what follows in the URL.

routes.IgnoreRoute creates a route mapped to the StopRoutingHandler route handler. This will stop the routing process and let normal ASP.NET handle the request.

**The default "axd" ignored route is probably the only one you need as physical files (such as JavaScript and CSS) are checked for first by default, and then routing rules are checked. The "axd" route is needed as there are no physical "axd" files.**

Copyright © 2019 MAX Technical Training

As an example of a common use, you might want to prevent MVC from trying to process the favicon.ico file.

```
routes.IgnoreRoute("{*favicon}", new {favicon=@"(.*/)?favicon.ico(/.*)?"});
```

In the example the “new” object being created is a regular expression that defines any path that includes “favicon.ico”. The “favicon” property is then used in the IgnoreRoute rule ({*favicon}).

Tip: Like all of the other routing rules, order is important. In general, add you “ignores” before any other route rules.

# Routing Order

## .Net Framework

As has been mentioned a few times already, the order things are added to routing is important. In a typical MVC + Web API project with all of the routing options, you will find the following order of additions and processing. Remember, you can change the order anyway you like:

- ↓ Default processing of files on disk.
  - .html, .aspx, .css, .js, etc.
- ↓ Web API attribute based routes
  - config.MapHttpAttributeRoutes();
- ↓ Web API custom routes
  - config.Routes.MapHttpRoute( name: "myApiRoute", …
- ↓ Web API default route
  - config.Routes.MapHttpRoute( name:"DefaultApi", routeTemplate: "api/{controller}/{id}", defaults: new { id = RouteParameter.Optional })
- ↓ Ignoring of .axd files
  - routes.IgnoreRoute("{resource}.axd/{*pathInfo}");
- ↓ MVC attributes based routes
  - routes.MapMvcAttributeRoutes();
- ↓ MVC custom routes
  - routes.MapRoute( name: "ProductsCategory", url: "Products/{id}", defaults: new { controller="Products", action="Category", id=UrlParameter.Optional });
- ↓ MVC default route
  - routes.MapRoute( name: "Default", url: "{controller}/{action}/{id}", defaults: new { controller = "Home", action = "Index", id = UrlParameter.Optional });
- ↓ 404 Not Found!

Copyright © 2019 MAX Technical Training

# .Net Core

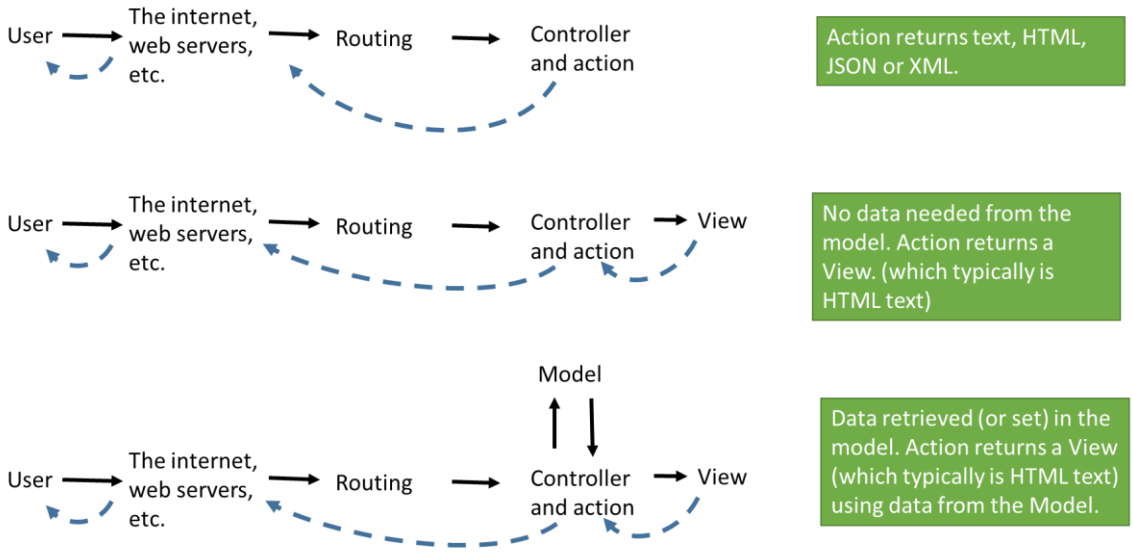
- Default processing of files in the project's wwwroot folder (If app.UseStaticFiles() is in StartUp.cs.)
  - .html, .aspx, .css, .js, etc.
- MVC attributes based routes
- MVC custom routes (if app.UseMvc is in StartUp.cs)
  - routes.MapRoute( name: "ProductsCategory", url: "Products/{id}",
    defaults: new { controller="Products", action="Category", id=UrlParameter.Optional });
- MVC default route (if app.UseMvc is in StartUp.cs)
  - routes.MapRoute( name: "Default", template: "{controller}/{action}/{id}",
    defaults: new { controller = "Home", action = "Index", id = UrlParameter.Optional });
- 404 Not Found!

Copyright © 2019 MAX Technical Training

# Module 3 – Controllers and Actions

## Controllers

Controllers are the heart of MVC. User requests are routed to Controllers, Controllers access models, Controllers apply business logic, Controllers pass the data to Views and finally, the HTML generated by the View is returned to the user.




The routing process does not directly pass data to the Controller. It passes the request to a Controller "factory" that determines which controller is needed, creates an instance of that Controller, and then calls the Action (method).

## Controllers are Classes

Controllers are typical .NET classes that inherit from System.Web.Mvc.Controller. These classes are almost never directly instantiated, instead they are created from a controller factory called from the router. One exception is unit testing, where you will create an instance of the controller for testing purposes.

Like most classes, these controllers have methods. These methods are called Actions. Actions are loosely bound at run time using ASP.NET Model Binding using routing rules. While Controllers can have both properties and methods, the only common publicly exposed methods are methods used as Actions.

You can create custom Model Binders to perform route-to-Action binding when the built-in routing does not do what you need. This is pretty rare as between the routing options and Action code, the built-in Model Binder will usually do the job.

Notes:

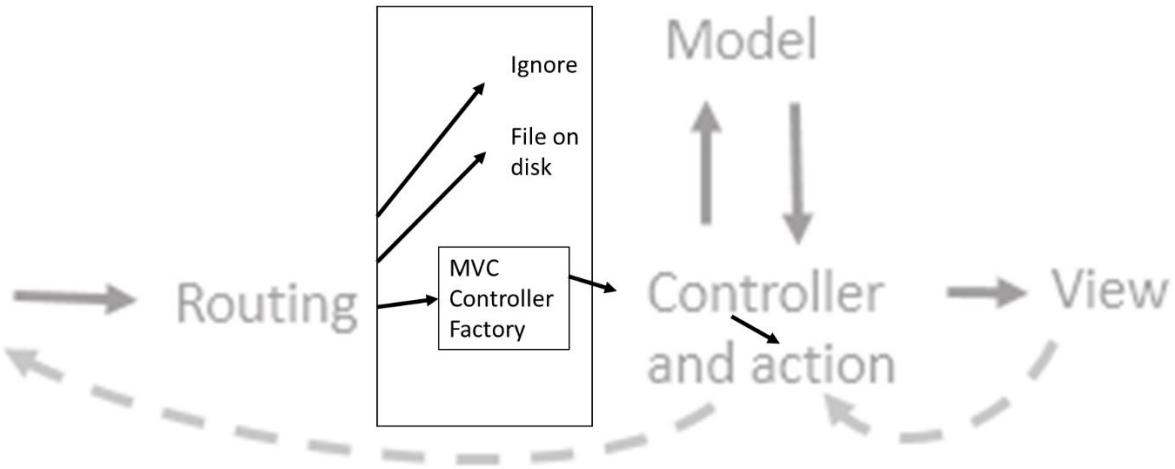
Copyright © 2019 MAX Technical Training

- Mark non-Action public methods so they will not be found by the routing process. [NonAction]

```
[NonAction]
public double CalculatePrice(string partNum)
{
    return 0.0;
}
```

# Actions are Methods

Actions are methods and are created the same as a typical public method. There is a big difference though, methods are bound at compile time and are pre-matched based on the method name and parameter signature while Actions are matched at runtime based on rules.




Tip: An object factory is an object that creates objects. In the case of our controllers, each route could pass to a different controller, and as each is a different class or type, we normally would have to write code to uniquely create each of the controller objects. With a factory, parameters are passed in and an object (controller) of the correct type is returned. See: https://en.wikipedia.org/wiki/Factory_method_pattern#C.23

## Attributes

While classes and methods often use attributes, attributes are very commonly used when creating Controllers and Actions. You will see a number of controller and action attributes used in this module and the next. The following example has three attributes: ActionName, RequireHttps and HandleError.

```
// GET: OnSale/toys
[ActionName("OnSale")]
[RequireHttps]
[HandleError(Order = 1, ExceptionType = typeof(DivideByZeroException), View="SomethingIsBroken")]
public ActionResult GetItemsOnSale(string category)
{
```

## Actions Return Action Result Objects

Actions typically return data that is compatible with ActionResult. The controller's base class includes helper methods to build these objects for you. The most often returned object is a View which contains the HTML to return to the browser.

Copyright © 2019 MAX Technical Training

The following example returns the home page of the site. The default View returned is named the same as the Action, “Index” in this example.

```
public ActionResult Index()
{
    return View();
}
```

Most Actions will pass some data (an object) to the View.

```
// GET: CreditCards
public ActionResult Index()
{
    return View(db.CreditCards.ToList());
}
```

Many Actions will include additional logic.

```
// GET: CreditCards/Edit/5
public ActionResult Edit(int? id)
{
    if (id == null)
    {
        return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
    }
    CreditCard creditCard = db.CreditCards.Find(id);
    if (creditCard == null)
    {
        return HttpNotFound();
    }
    return View(creditCard);
}
```

# Creating Controller Classes

A Controller is just a class that inherits from System.Web.Mvc.Controller. While this class can be stored anywhere in a project, by convention it is stored inside of the Controllers folder of the project or the Controllers folder of an Area folder. (Areas are covered later in this course.)

## Naming Conventions and “Auto-magic”

Note: The formal term for “auto-magic” is “conventions”.

The router assumes that the controller is named *somename*Controller. When a route points to *somename*, an instance of *somename*Controller will be created.

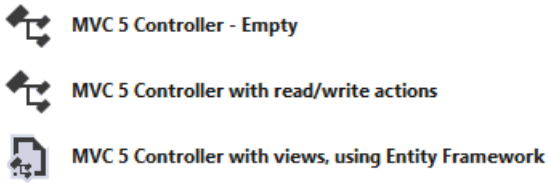
**Singular or Plural?**

- The sample templates use singular names for Controllers. HomeController, AccountController, etc.
- The MVC scaffolding for an Entity Framework model uses plural names. ProductsControllers, etc.
- Best practice? As the Controller’s name is by default used in the URL, use what makes the most sense for your project!

Copyright © 2019 MAX Technical Training

# Options for creating new Controllers:

- **Manually** – Add a new class file, add a using statement for System.Web.Mvc, inherit from System.Web.Mvc.Controller and add at least one Action (public method).
  public class ReportsController : Controller { … }
- **Using Templates** – Right-click the Controllers folder of a project or Area, click Add and then Controller and choose from one of the templates.

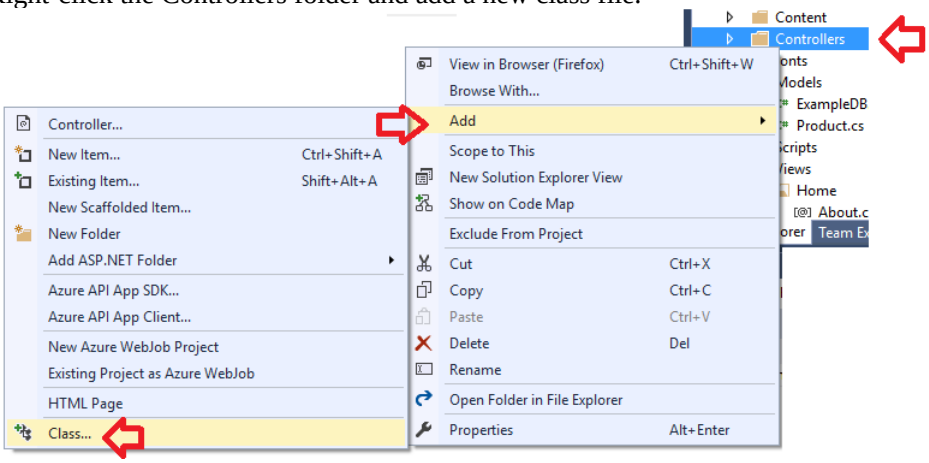



# Manually Add a Controller

The steps below are pretty much the same as for creating any class file. Note that the following steps can be done for you by using the "MVC 5 Controller - Empty" template!

Steps:

1. Right-click the Controllers folder and add a new class file.




   a. While a controller class can be put anywhere in a project, the convention is to place it in the Controllers folder.
2. Name the class and click Add. (for this example: ReportsController.cs)
3. Add a using statement for System.Web.Mvc.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
```

4. Edit the class so it inherits from System.Web.Mvc.Controller.

```
public class ReportsController : Controller
{
```

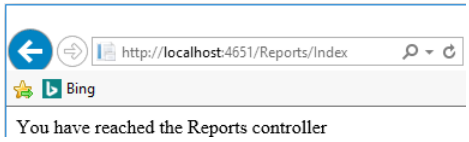
5. Add code for at least one Action.
   Note: Most Actions will return Views. For testing you can return simple text by using the controller's Content() helper method.

```
public ActionResult Index()
{
    return View();
}
```

```
public ActionResult Index()
{
    return Content("You have reached the Reports controller");
}
```

Copyright © 2019 MAX Technical Training

6. Test. Run the project from Visual Studio, edit the URL in the browser to point to /Reports/Index and confirm that the Controller and Action were found.

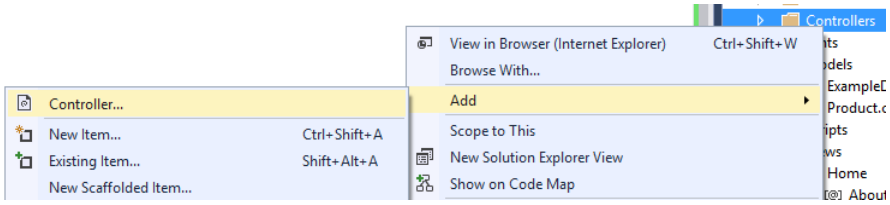



# Add a Controller Using Templates

Visual Studio's templates can do a lot of the work for you.

Adding a Controller from a template:

- Adds necessary using statements.
- Creates the class code
- Writes one or more Actions.
- Creates a folder for your views, and depending on the template, create an initial set of views.



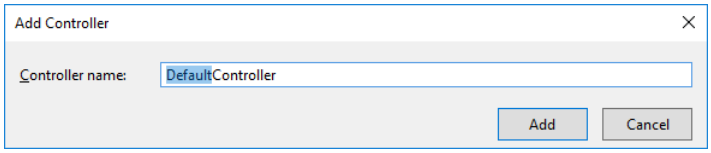

## Creating an "Empty" Controller.


The "Empty" Controller template is not quite empty as it automatically adds a sample Action named Index. This template:

- Adds the System.Web.Mvc using statement.
- Creates the class.
- Adds a sample Action named "Index".

Steps:

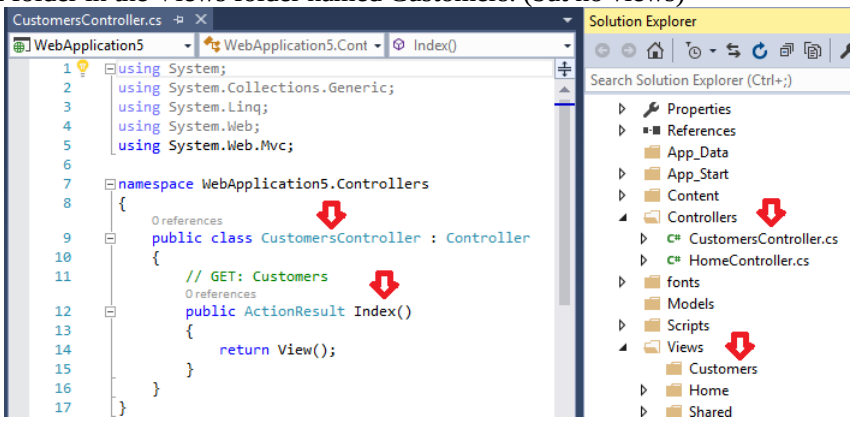
1. Right-click the Controllers folder, click Add and then Controller.
2. Click MVC 5 Controller – Empty and click Add.
3. Enter a name in the form of *controllername*Controller and click Add.




4. Click Add.
5. Note that if you named the Controller "Customers" then the following have been created:
    a. A class named CustomersController in the Controllers folder.
    b. An Action named Index.

Copyright © 2019 MAX Technical Training

c. A folder in the Views folder named Customers. (but no views)




6. Note the code will compile, but at run time you will get an error as you have not yet created a view named "Index".
7. Next steps:
    a. Create additional Actions and business logic code.
    b. Create Views.

## Creating a Controller with "read/write actions"

**MVC 5 Controller with read/write actions**

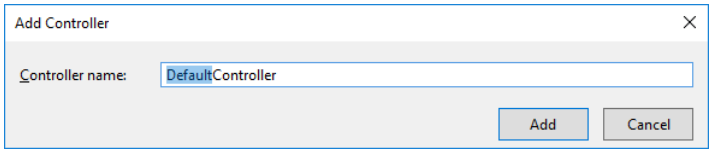
This template creates a Controller with a typical set of Actions for maintaining a collection of some type of object such as customers, products, vendors, etc. These Actions are only shells and have no database code. No Views are created by this template.

Actions are created for:

- Index – for listing all items in the collection.
- Details(int) – for listing all of the properties of a single item.
- Create – (GET) for displaying a form to create a new item.
- Create(FormCollection) – (POST) for receiving data from a "new item" form.
- Edit(int) – (GET) for displaying a form to edit an existing item.
- Edit(int, FormCollection) – (POST) for receiving data from an "edit item" form.
- Delete(int) – for listing all of the properties of a single item to confirm a delete.
- Delete(int, FormCollection) – (POST) for processing the delete.

Steps:

1. Right-click the Controllers folder, click Add and then Controller.
2. Click MVC 5 Controller with read/write actions and click Add.
3. Enter a name in the form of *controllername*Controller and click Add.




4. Click Add.
5. Note that if you named the Controller "Customers" then the following have been created:
    a. A class named CustomersController in the Controllers folder.
    b. Actions named Index, Details, Create, Create (POST), Edit, Edit (POST), Delete, and Delete (POST).
    c. A folder in the Views folder named Customers. (but no views)

Copyright © 2019 MAX Technical Training

6. Note the code will compile, but at run time you will get an error as you have not yet created a view named Index.
7. Next steps:
    a. Create data access and business logic code.
    b. Create Views.

## Creating a Controller with views, using Entity Framework


MVC 5 Controller with views, using Entity Framework

The is the most complete template as it can take an existing Entity Framework entity (a SQL table, etc.) and create a controller and all of the needed Views to maintain the data.

*We will explorer the Entity Framework in the next module.*

Actions are created for: (Same as "MVC 5 Controller with read/write actions" plus Dispose().)

- Index – for listing all items in the collection.
- Details(int) – for listing all of the properties of a single item.
- Create – (GET) for displaying a form to create a new item.
- Create(FormCollection) – (POST) for receiving data from a "new item" form.
- Edit(int) – (GET) for displaying a form to edit an existing item.
- Edit(int, FormCollection) – (POST) for receiving data from an "edit item" form.
- Delete(int) – for listing all of the properties of a single item to confirm a delete.
- Delete(int, FormCollection) – (POST) for processing the delete.
- Dispose(bool disposing) – Used to call the dispose method of the DataEntities object.

Views are created for: (Views/*controllerName*)

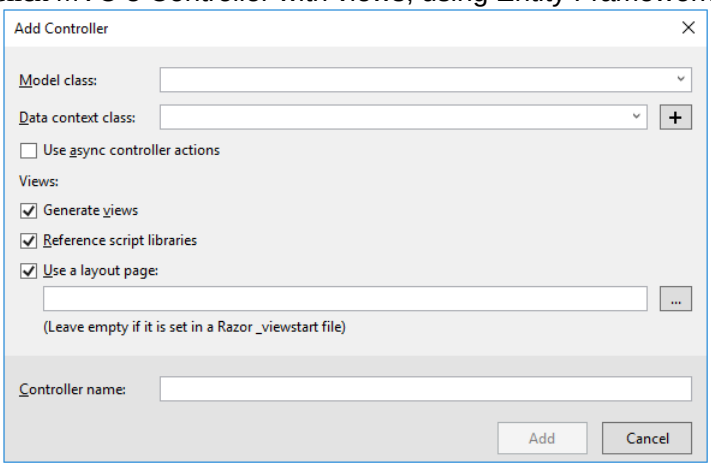
- Create
- Delete
- Details
- Edit
- Index

Steps:

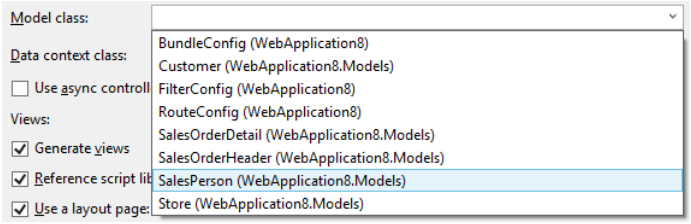
1. Create your project, add the Entity Framework and build your model. ;-)
2. Build your project to make sure the Entity Framework is ready.
3. Right-click the Controllers folder, click Add and then Controller.

Copyright © 2019 MAX Technical Training

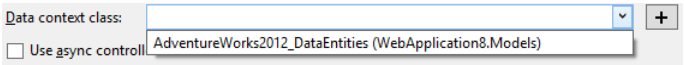
4. Click MVC 5 Controller with views, using Entity Framework and click Add.



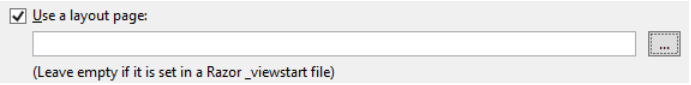

5. Select a Model class. Not all of the classes listed are for models!



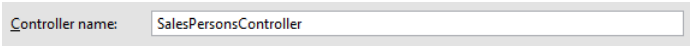

6. Select a Data context class.



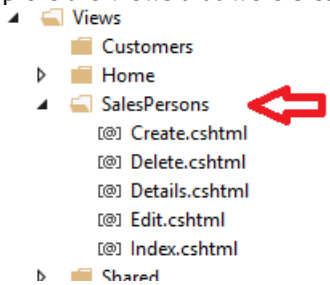

7. If you are using a layout page then select it from the dropdown menu. (The default is Views/Shared/_Layout.cshtml.) Leave this blank if using the layout page specified in the _ViewStart.cshtml file.




8. Enter a name in the form of *controllername*Controller and click Add. The default name displayed here is based on the name of the selected Model class.

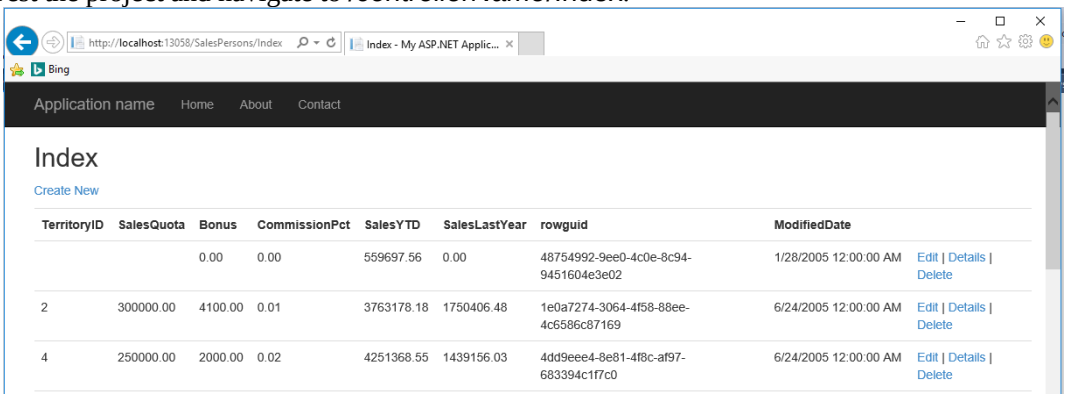



9. Click Add.
10. Click OK for any security messages that are displayed. (The Entity Framework templates were downloaded from the web using NuGet!)
11. Explore the new Controller class and the Actions that were created. Customize the code as needed.
12. Explore the views that were created in Views/*controllerName*.




Copyright © 2019 MAX Technical Training

13. Test the project and navigate to */controllerName*/Index.




14. Looks like we have some work to do to clean up the display of the data!
15. Click the Details link on one of the items.
16. Click the Edit link. Make a trivial change to the data and click Save.
17. You may not be able to test Delete and Insert now without first researching the database's constraints.

# Controller Attributes

MVC includes a number of attributes that can be applied to a Controller class.

```
[OutputCache(Duration = 300)]
[Authorize(Roles = "Managers")]
[RequireHttps]
public class SalesController : Controller
{
```

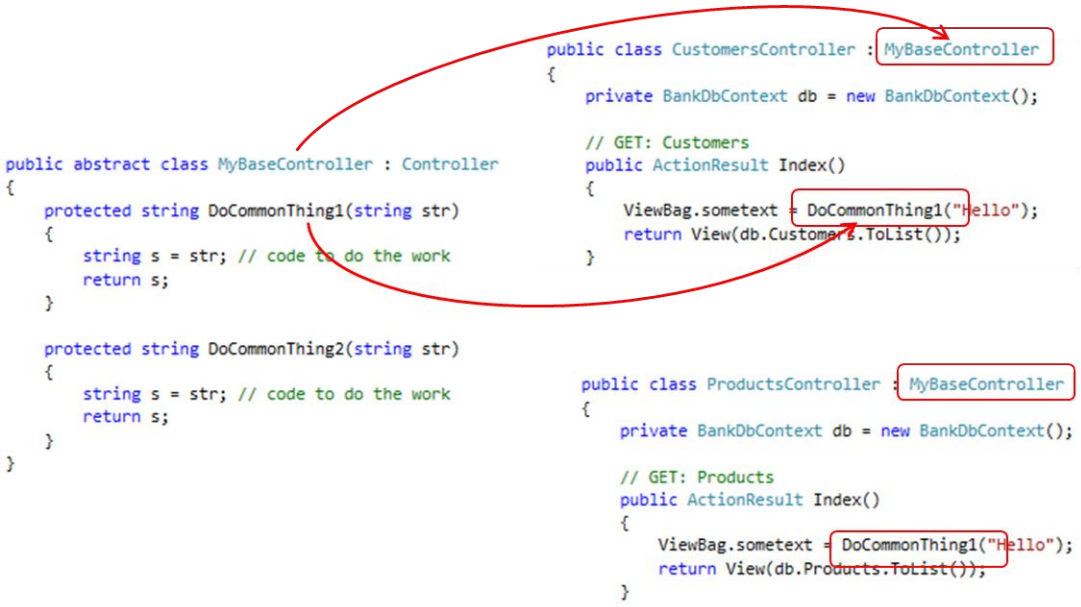
For now we will just list a few samples and wait to cover them in more detail later in this module.

- AllowAnonymous
- Authorize
- HandleError
- OutputCache
- RequireHttps
- SessionState
- ValidateAntiForgeryToken
- ValidateInput

# Controllers and Inheritance

If you have some code that will be used in all of your controllers, maybe code to help with localization and languages, then you can create a controller base class and then inherit from that class when creating other controllers. The "add controller" wizard does not have an option for this, but you can change the inherited class name after the controllers have been created. You should declare the base class as abstract to prevent its accidental use.

Copyright © 2019 MAX Technical Training



```
public abstract class MyBaseController : Controller
{
    protected string DoCommonThing1(string str)
    {
        string s = str; // code to do the work
        return s;
    }

    protected string DoCommonThing2(string str)
    {
        string s = str; // code to do the work
        return s;
    }
}
```

```
public class CustomersController : MyBaseController
{
    private BankDbContext db = new BankDbContext();

    // GET: Customers
    public ActionResult Index()
    {
        ViewBag.sometext = DoCommonThing1("Hello");
        return View(db.Customers.ToList());
    }
```

```
public class ProductsController : MyBaseController
{
    private BankDbContext db = new BankDbContext();

    // GET: Products
    public ActionResult Index()
    {
        ViewBag.sometext = DoCommonThing1("Hello");
        return View(db.Products.ToList());
    }
```

# Actions

Actions are methods that are called based on "rules" defined for routes. Actions typically return an object that inherits from the ActionResult class.

An action is a method that returns data to a browser. In the simplest form an action could return a string or an integer. These actions return the data as a string, and with a Content Type of "text/html". (The ToString() method is called on any type returned via an Action.)

```
public string aString()
{
    return "This is a string.";
}
public int anInt()
{
    return 123;
}
public BankAccount aBankAccount()
{
    var ba = new BankAccount();
    ba.OwnerName = "Fred";
    ba.Balance = 12345.12;
    return ba;
}
```

All three of the above actions will return strings to the browser with a Content Type of "text/html". The results displayed in the browser are:

- This is a string.
- 123
- MyNamespace.BankAccount
  This the default text from the BankAccount class's ToString() method. No error is generated, but only useless text is returned.

Note that you will probably never return objects other than those that inherit from the ActionResult class. If you want to return data other than a view you will generally use one of the controller's helper methods such as Content() or Json().

Copyright © 2019 MAX Technical Training

# Actions Typically Return an ActionResult Compatible Type

The Controller base class includes helper methods that can create ActionResult compatible types for you. While probably 98% of the examples of a returned ActionResult object is the ViewResult type returned by the View() helper function, there's actually a large family of return types available.

## ActionResult Helper Methods

The helper functions listed below are methods found in the System.Web.Mvc.Controller base class. They can be called either as base.*helpername*() or *helpername*().

| | | |
|---|---|---|
| **Content**(String) | Returns the content of the string to the browser | return Content("Hello") |
| **Content**(String, String) | Returns the content of the string to the browser along with a MIME type. | return Content("Hello", "text/html",System.Text) |
| **Content**(String, String, encoding) | Returns the content of the string to the browser along with a MIME type and text encoding. | return Content("Hello", "text/html",System.Text.UTF8Encoding.UTF8) |
| **File**(Byte[], String) | Returns a file (byte array) and a MIME type. Default file name is the same as the action. | string s = "Hello";<br>byte[] myBytes = System.Text.Encoding.UTF8.GetBytes(s);<br>return File(myBytes, "application/octet-stream:"); |
| **File**(Byte[], String, String) | Returns a file (byte array) and a MIME type with a default file name. | string s = "Hello";<br>byte[] myBytes = System.Text.Encoding.UTF8.GetBytes(s);<br>return File(myBytes, "application/octet-stream:","Sample.txt"); |
| **File**(Stream, String) | Returns a file (stream) and a MIME type. Default file name is the same as the action. | string s = "Hello";<br>byte[] myBytes = System.Text.Encoding.UTF8.GetBytes(s);<br>MemoryStream stream = new MemoryStream(myBytes);<br>return File(stream, "application/octet-stream"); |
| **File**(Stream, String) | Returns a file (stream) and a MIME type with a default file name. | string s = "Hello";<br>byte[] myBytes = System.Text.Encoding.UTF8.GetBytes(s);<br>MemoryStream stream = new MemoryStream(myBytes);<br>return File(stream, "application/octet-stream", "SampleFile.txt"); |
| **File**(String, String) | Returns a file from a path on the server. Default file name is the same as the action. | string path = @"C:\test\test.csv";<br>return File(path, "application/octet-stream"); |
| **File**(String, String, String) | Returns a file from a path on the server and supply a file name. | string path = @"C:\test\test.csv";<br>return File(path, "application/octet-stream", "SampleFile.csv"); |
| **HttpNotFound**() | Returns the default 404 error to the browser. | return HttpNotFound(); |
| **HttpNotFound**(String) | Returns the default 404 error to the browser with a custom message. | return HttpNotFound("Customer not found"); |
| **JavaScript**(String) | Sets the ContentType to text/html; charset=utf-8 Typically called from jQuery: $.getScript("/Home/Sho | string s = "alert('hello')";<br>return Content(s); |

Copyright © 2019 MAX Technical Training

| | | |
|---|---|---|
| | wAlert") or from a script block: <script src="home/javascriptDemo"></script> | |
| **Json**(object) | Returns a JSON stringified object. (POST only) | var bankaccount = new { acct = 123, name = "Mike Smith", balance = 1234.56 };<br>return Json(bankaccount); |
| **Json**(object,option) | Returns a JSON stringified object. (GET allowed) | var bankaccount = new { acct = 123, name = "Mike Smith", balance = 1234.56 };<br>return Json(bankaccount,JsonRequestBehavior.AllowGet);<br><br>For secruity reasons, this method will not by default allow returning JSON from a Get requst. |
| **PartialView** | PartialView follows the same pattern as View listed below, but returns PartialViews. | Note: _viewstart.cshtml is not executed for partial views. |
| **Redirect**(String) | Performs a browser redirection (303) to a URL. | return Redirect("http://www.example.com"); |
| **RedirectToRoute**(RouteValueDictionary) | Performs a browser redirection (303) to a route. | return RedirectToRoute( new { controller = "home", action = "index" }); |
| **RedirectToRoute**(String) | Performs a browser redirection (303) to a route (string). | return RedirectToRoute("home/index"); |
| **RedirectToAction**(String) | Performs a browser redirection (303) to an action (in current controller). | return RedirectToAction("index"); |
| **View**() | Returns a rendered view using the action's name and no model. | teturn View(); |
| **View**(String) | Returns a rendered view using the specified view name and no model. | return View("SalesReport"); |
| **View**(Object) | Returns a rendered view using the action's name and a model object. | return View(objCustomer); |
| **View**(String, Object) | Returns a rendered view using the specified view name and a model object. | return View("SalesReport",objCustomer); |

## What about XML?

There's a helper method for JSON and one for JavaScript, what about XML? If the goal is to take a string of XML and send it to the browser with the correct content type, then there are at least two ActionResult Helpers that do the job for us:

**Set the Response.ContentType and then use the Content() helper to return the XML text:**

Copyright © 2019 MAX Technical Training

```
public ActionResult EmployeesAsXML1()
{
    ControllerContext.HttpContext.Response.ContentType = "test/xml";
    return Content("<employees><employee id='1'><name>mike</name></employee></employees>");
}
```

To learn more about the Response object see: https://msdn.microsoft.com/en-us/library/system.web.httpresponse(v=vs.110).aspx

**Covert the XML text to a byte array and return using the File() helper:**

```
public ActionResult EmployeesAsXML2()
{
    string xml = "<employees><employee id='1'><name>mike</name></employee></employees>";
    byte[] xmldata = System.Text.Encoding.UTF8.GetBytes(xml);
    return File(xmldata,"text/xml");
}
```

### Custom ActionResults

Another option for our XML problem is to create a custom ActionResult class. Create a class that inherits from ActionResult and override the ExecuteResult base method. The example below accepts as string and returns as a Content Type of "text/xml". If you wanted this to work like the Json() helper then you could modify the class to accept any object, serialize it to XML and then return the XML.

```
public class XmlResult : ActionResult
{
    string Data;
    0 references
    public XmlResult(string XML) { Data = XML;  }

    2 references
    public override void ExecuteResult(ControllerContext context)
    {
        HttpContextBase ctxBase = context.HttpContext;
        ctxBase.Response.ContentType = "text/xml";
        ctxBase.Response.Buffer = true;
        ctxBase.Response.Clear();
        ctxBase.Response.Write(Data);
    }
}
```

# Creating Actions

Actions are created much like any other method.

- Declare as public.
- Return type is usually ActionResult.
- The data returned is usually from a helper method from the controller's base class.

Unlike most methods, Actions frequently include attributes. These attributes are used to assist the Model Binder with selecting the correct action, to enforce authentication and the use of HTTP verbs.

Example Action:

```
[HttpPost]
public ActionResult Action()
{
    return View();
}
```

This action:

Copyright © 2019 MAX Technical Training

- Has the name of "Action".
- Can only be accessed via a POST.
- Returns a view result named "Action".

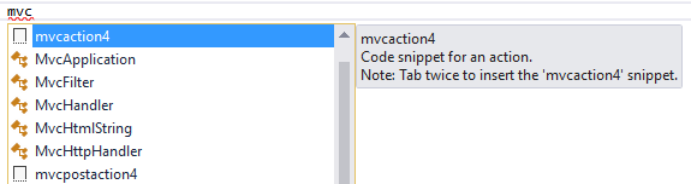
# Options for Creating New Actions

Actions can be manually created, added using Visual Studio snippets, or added with new controllers.

- Manually! Just type the code:

```
public ActionResult MyAction()
{
    return View();
}
```

- Use a Visual Studio snippet:
  - Type "mvcaction4" and press tab.



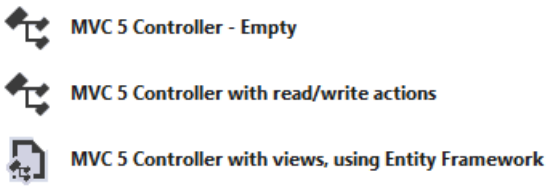

This will auto create:

```
public ActionResult MyAction()
{
    return View();
}
```

  - Type "mvcpostaction4" and press tab to auto create an Action with a POST attribute.

```
[HttpPost]
public ActionResult Action()
{
    return View();
}
```

- Add Actions when adding a new controller. When adding a controller using the wizards you are offered two options to add actions to the new controller.




  - For "with read/write actions", actions are created for:
    - Index – for listing all items in the collection.
    - Details(int) – for listing all of the properties of a single item.
    - Create – (GET) for displaying a form to create a new item.
    - Create(FormCollection) – (POST) for receiving data from a "new item" form.
    - Edit(int) – (GET) for displaying a form to edit an existing item.
    - Edit(int, FormCollection) – (POST) for receiving data from an "edit item" form.
    - Delete(int) – for listing all of the properties of a single item to confirm a delete.
    - Delete(int, FormCollection) – (POST) for processing the delete.
  - For "with views, using Entity Framework", you get the views in the above list plus:
    - Dispose(bool disposing) – Used to call the dispose method of the DataEntities object.

Copyright © 2019 MAX Technical Training

# Common Issues for Actions

Here are a few issues you may run into when creating actions.

- Auto-magic vs. Strings
- Two actions with the same name, but different method signatures, may not be uniquely selected by a route. (Actions are not strongly bound.)
- The "Ambiguous" error.
- Actions return HTML, but not directly.

## Auto-magic vs. Strings

Note: The formal term for "auto-magic" is "conventions".

When you use the View() helper method, MVC "guesses" the view's name using the name of the Action. You can also supply a view's name with the View() helper method as a string: View("index"). As in all cases when using strings instead of object names to identify an object, you don't get compile-time error detection or IntelliSense, and you only discover errors at runtime.

## Actions are not strongly bound

Actions are matched at runtime using routing rules. These rules frequently will select the "first match" instead of an "exact match" when choosing an action. Avoid creating two actions with the same name, but with different signatures. Instead, create two different method names.

For example: you want to be able to handle both of these URLs:

http://www.example.com/products/bikes/42 (42 is a bike size)

http://www.example.com/products/bikes/42/red

So you add these two routes:

```
routes.MapRoute(
    name: "BikesSize",
    url: "Products/Bikes/{size}",
    defaults: new { controller = "Products", action = "FindBike" }
);

routes.MapRoute(
    name: "BikesSizeColor",
    url: "Products/Bikes/{size}/{color}",
    defaults: new { controller = "Products", action = "FindBike" }
);
```

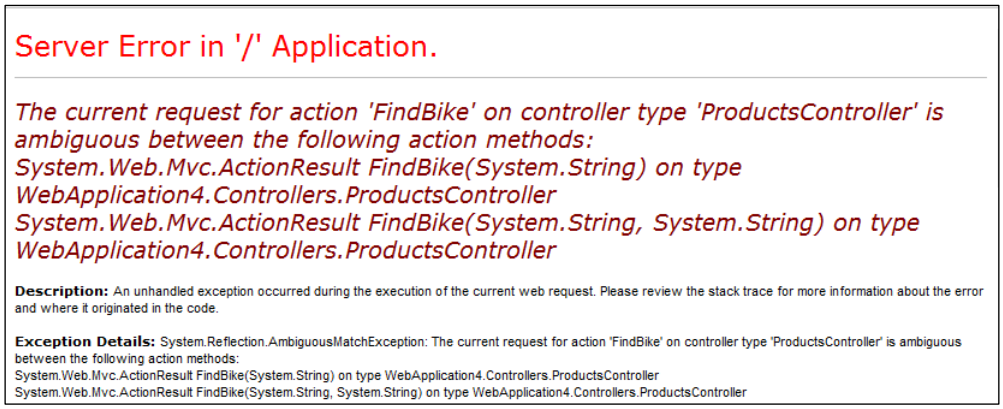
Due to some uniqueness of the code you decide to create two actions with the same name, but with different signatures (number and types of parameters).

```
Public ActionResult FindBike(string size)
{
    return Content("You got the BIKE category!  size=" + size  + "!");
}
```

Copyright © 2019 MAX Technical Training

```
public ActionResult FindBike(string size, string color)
{
    return Content("You got the BIKE category!  size=" + size + "  color=" + color + "!");
}
```

While the above will build without error, you will get a run time error:




# One Solution to the Ambiguous Error

Actions look like normal class methods, but are "looked up" by routing rules instead of being bound at compile time. You cannot depend on a method's signature to uniquely identify it in a controller. To solve this error, each action must have a unique name. You can then add routing rules to pick the correct action.

These are the same actions as above, except now they have unique names:

```
public ActionResult FindBike1(string size)
{
    return Content("You got the BIKE category!  size=" + size  + "!");
}

public ActionResult FindBike2(string size, string color)
{
    return Content("You got the BIKE category!  size=" + size + "  color=" + color + "!");
}
```

We now need to update the routes with the new Action names:

```
routes.MapRoute(
    name: "BikesSize",
    url: "Products/Bikes/{size}",
    defaults: new { controller = "Products", action = "FindBike1" }
);

routes.MapRoute(
    name: "BikesSizeColor",
    url: "Products/Bikes/{size}/{color}",
    defaults: new { controller = "Products", action = "FindBike2" }
);
```

Copyright © 2019 MAX Technical Training

# Another Solution to the Ambiguous Error

In the simple example we have created here it might be better to just have one action and add a little code logic to deal with the two possible ends.

Add one route that collects both parameters:

```
routes.MapRoute(
    name: "BikesSizeColor",
    url: "Products/Bikes/{size}/{color}",
    defaults: new { controller = "Products", action = "FindBike" }
);
```

And add one Action with a little more logic:

```
public ActionResult FindBike(string size, string color)
{
    if ( color != null )
      { return Content("You got the BIKE category!  size=" + size + "  color=" + color + "!"); }
    else
      { return Content("You got the BIKE category!  size=" + size + "!"); }
}
```

Note: While our examples above return data using the Content() helper method, we would normally be returning data using a view and the View() helper method.

# Actions Return HTML, but not Directly

The View() helper method used in Controllers returns an object that describes a View and the data to pass to it, but it does not execute the View or generate the HTML. You do not have direct access to the HTML rendered by the View.

One of the core guidelines of MVC is that Controllers supply the "switching logic" of the project while Views supply the HTML to be returned to the browser. I.e. Controllers should not create or return HTML. But in the real-world of development we sometimes have to break the rules. As an example, your legal department sends out a panic memo that you are to immediately stop using the term "widget" to describe our products and must now use "thingamajig". You know that this will not be quick and easy as "widget" is used in a lot of views, in data stored in SQL tables and in data retrieved from external web services. The code below retrieves the view, passes the ViewData and TempData objects and then calls the Render method to retrieve the HTML. The HTML is then processed as a string and the string returned using the Content() helper method.

```
public ActionResult NewProducts()
{
    string HTML;
    using (StringWriter sw = new StringWriter())
    {
        var myViewResult = ViewEngines.Engines.FindView(ControllerContext, "NewProducts", "");
        var myViewContext = new ViewContext(ControllerContext, myViewResult.View, ViewData, TempData, sw);
        myViewResult.View.Render(myViewContext, sw);
        HTML = sw.ToString();
    }

    string s = HTML.Replace("widget", "thingamajig");
    return Content(s);
}
```

Copyright © 2019 MAX Technical Training

Remember this is not a “best practice” and should be thought of as a “duct tape and bailing wire” quick fix that should be removed as soon as possible!

# Route to Controller to Action

In Module 2 we saw that parameters can be passed to Actions using either values in the path or as Query String values. Once the request has reached the controller, the Model Binder then needs to pick an Action, and then figure out how to map the data found in the URL to the Action/method parameters. Here we will dive a little deeper into working with parameters.

MVC will attempt to bind data to Action (method) parameters in this order:

- Form values from an HTML <form> with method equal to POST. <form method="post" …
- Route values. /products/details/123
- Query Strings /products/details?id=123

MVC will attempt to bind:

- By mapping a property (form, route, query string) by name. I.e. Customer to Customer
- By mapping a property (form, route, query string) by object. I.e. Name to Customer.Name

# Working with Parameters

Both of the following could be used to access the Bikes controller and the FindBike action with four parameters.

```
http://www.example.com/Bikes/FindBike/42/red/carbonfiber/A100
```

```
http://www.example.com/Bikes/FindBike?size=42&color=red&construction=carbonfiber&model=A100
```

Advantages of using a path/route:

- Shorter URLs.
- The text of the URL is more often used by search engines for keywords extraction.
- Can be redefined by changing the routing rules.

Advantages of using query string:

- Parameters have names and are self-describing.
- Parameters can be listed in any order.
- Parameters can be omitted.

## Path Parameters

The keywords you have seen in the sample routes so far have mapped to a Controller, an Action and an ID.

```
url: "{controller}/{action}/{id}",
```

While “controller” and “action” are built-in keywords, “id” is not. Any additional placeholders you add to your URL will be added to a dictionary of values that is then passed to the Controller and the Action. MVC will attempt to match parameters by name. In the example below notice that all of the parameters are matched except for “{construction}”, which is ignored.

Copyright © 2019 MAX Technical Training


# Dummy Placeholders

You can add "dummy" placeholders so your URLs can contain extra data only needed for search engine optimization (SEO). For example, the bike's model name might be added with the hope that it will be picked up by the web's search engines.

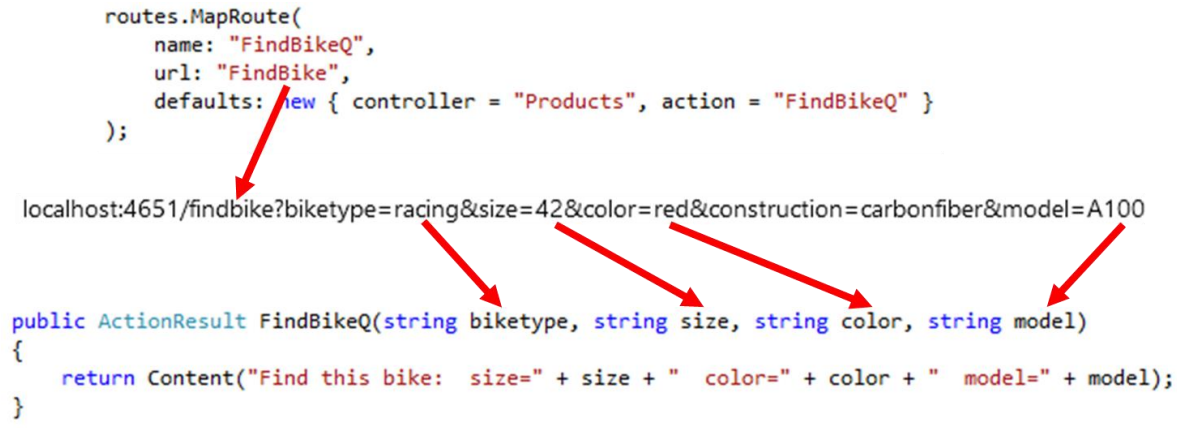
You might define a route like "lookupbike/{SEOtext}/{model}" so you can use a URL like the following example. Your action only cares about the "model" parameter.

http://www.example.com/lookupbike/Speed%20Racer%20100/A100

# Query String Parameters

MVC will map from the query string's names to an Action's parameters. An additional advantage of query strings is that the values do not have to be supplied in any particular order.

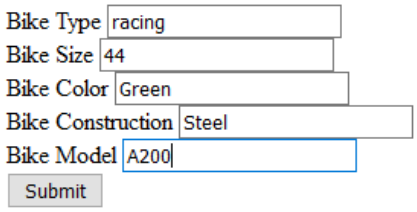
In the example below note that "&construction=carbonfiber" is being ignored.




The HTML form to build the query string above might look like the following:

Copyright © 2019 MAX Technical Training

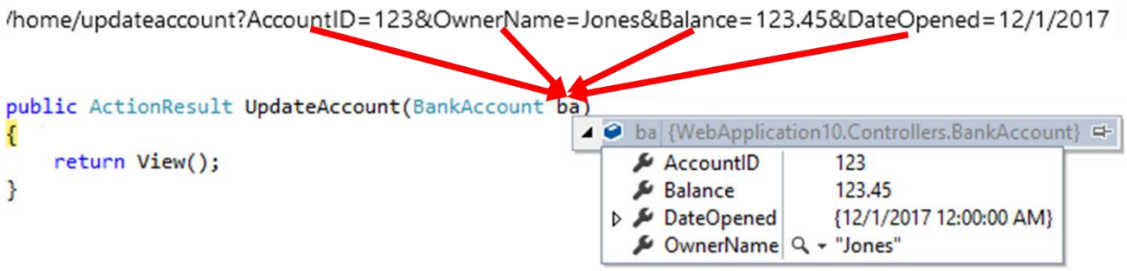
```
<!DOCTYPE html>
<html>
<head>
    <title></title>
    <meta charset="utf-8" />
</head>
<body>
    <form method="get" action="/FindBike">
        Bike Type <input type="text" name="biketype" /> <br />
        Bike Size <input type="text" name="size" /> <br />
        Bike Color <input type="text" name="color" /> <br />
        Bike Construction <input type="text" name="construction" /> <br />
        Bike Model <input type="text" name="model" /> <br />
        <input type="submit" value="Submit" />
    </form>
</body>
</html>
```




Note: If you supply a property using both the path parameter and the Query String parameter, the path parameter will be used.

# Complex Type Binding

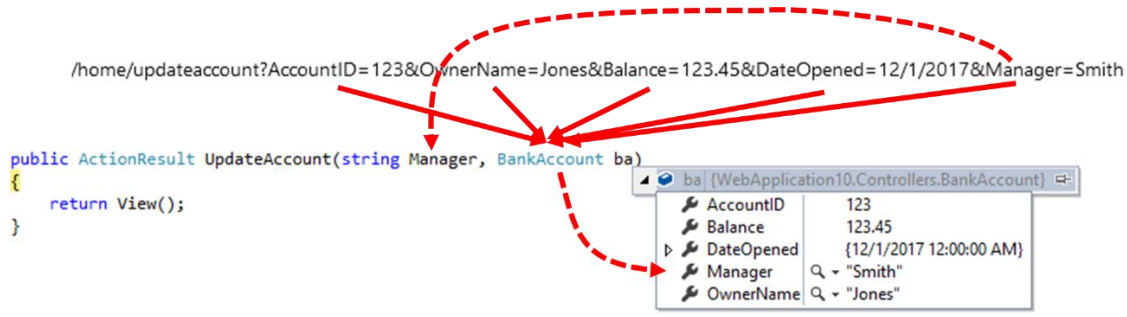
When MVC can't find a parameter with a name that matches an incoming property it will check for a complex type (BankAccount in the example below) to see if it has a property with a matching name.




A Query String was used in this example, but the same applies for path based parameters.

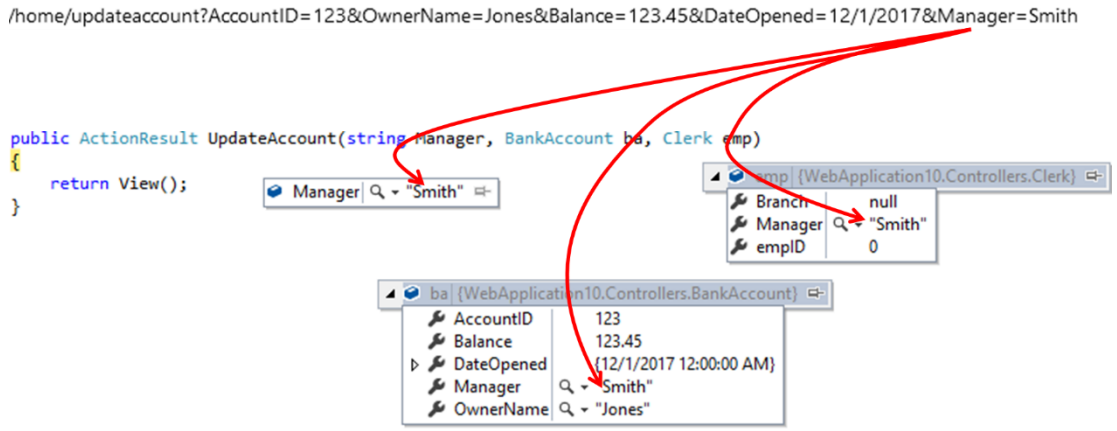
# Multiple Mappings Possible!

In the example below, note that the Manager property in the Query String (or path) maps to both the "ba" BankAccount object and the "Manager" string parameter.




Without careful planning, property mapping can be a challenge!

Copyright © 2019 MAX Technical Training

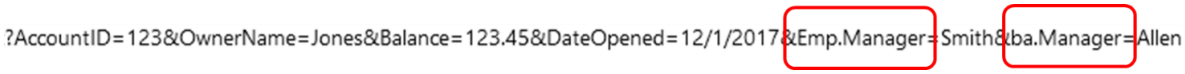



# Adding Parameter Names to Clarify Mapping

When working with complex objects as Action parameters you can prefix each parameter with the object's name. For a route, these prefixes are invisible to a user.

```
routes.MapRoute(
    name: "bank",
    url: "updatebank/{ba.manager}/{emp.manager}",
    defaults: new { controller = "Home", action = "updateaccount" }
);
```

For a Query String, this adds complexity to the URL and may expose useful information to a hacker.




# Reading Query String and Route Data

You can directly read the values from Query Strings and the Route data and not depend on the default parameter mapping by using the RouteData and Request properties of the Controller.

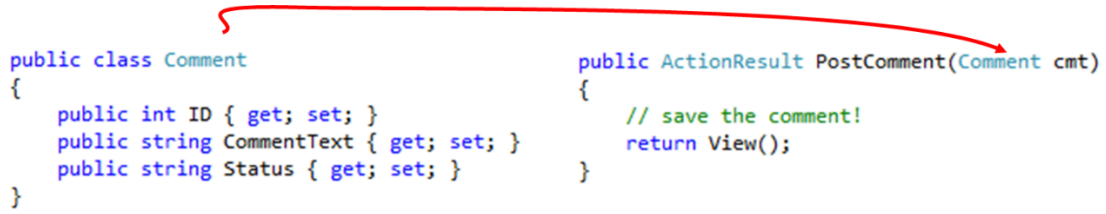
```
public ActionResult UpdateAccount()
{
    var r = RouteData.Values["manager"];
    var q = Request.QueryString["manager"];
    var f = Request.Form["manager"];
```

# Over Posting Attack

As you have seen in the above examples, all properties of a complex object can be accessed using a Query String. While you may have created a view that only lets a user update six out of ten properties of an object, a hacker could create a URL with a Query String, or better… an HTTP Post request, with properties that you don't want changed.

Example: A view to add a new comment:

Copyright © 2019 MAX Technical Training



```
public class Comment
{
    public int ID { get; set; }
    public string CommentText { get; set; }
    public string Status { get; set; }
}
```

```
public ActionResult PostComment(Comment cmt)
{
    // save the comment!
    return View();
}
```

- The post back goes to:
  /comments/PostComment
- The view sends back:
  CommentText=This is a comment!
- The Action updates the cmt.CommentText property.

Now our hacker:

- Creates an HTTP request using their favorite tool. (Such as Fiddler or the Chrome Request Maker plugin.)
  - The request sends back:
    - CommentText=This is bad stuff you don't want posted!
      *and*
    - Status=Approved
  - The Action updates the cmt.CommentText property *and* the cmt.Status property!

## How to prevent Over Posting Attacks

You have several options:

- **Don't use the Entity Framework default scaffolding** where all properties of a table/entity are exposed via the domain model.
  - Don't use the domain model for Comment. Instead, create a View Model that only exposes the properties needed by that view.
  - Add server-side code to only update selected properties.
- **Add an attribute to the Action to only allow Post updates**. (Not a complete solution! But does block Query String attacks.)

```
[HttpPost]
public ActionResult PostComment(Comment cmt)
{
```

- **Add attributes to limit the list of properties that can be updated**.

```
public ActionResult PostComment([Bind(Include = "CommentText")] Comment cmt)
{
```

- **Add attributes to block a list of properties from being updated**.

```
public ActionResult PostComment([Bind(Exclude = "Status")] Comment cmt)
{
```

# Parameter Binding Attributes

Applies to: Actions Parameters

To help manage which properties are bound, and to help block Over Posting Attacks, you can specify which properties of an Action parameter can and cannot be bound. The [Bind] attribute controls if a property, usually of a complex type, can be bound. Options include "Include" and "Exclude". The following insures that the CartType property of the creditCard object cannot be changed by this view.

In this example, all of the properties of the CreditCard object can be bound during routing except for the "CardType" property.

Copyright © 2019 MAX Technical Training

public ActionResult Edit( **[Bind(Exclude = "CardType")]** CreditCard creditCard)

In this example, only the "CommentText" and "CreatedBy" properties of the Comment object can be bound to:

public ActionResult Edit( **[Bind(Include = "CommentText,CreatedBy")]** Comment usercomment)

# The Request Object

Query string values can also be retrieved using the Request object. This avoids any Model Binder parameter mapping confusion, but does not allow for the use of friendly MVC style URLs.

```
// GET: /reports/reportbycategory?cat=ABC
public ActionResult ReportByCategory()
{
    string category = Request["cat"];
    return View();
}
```

The Request object gives you access to:

- Request["key"] Query String and Cookie values
- Request.QueryString["key"] Query String values
- Request.Cookies["key"] Cookie values
- Request.UserAgent Information about the browser
- Request.HttpMethod Returns the method used to make the request (Get, Post, etc.)

The Request object is not unique to MVC. To learn more about the Request object see:
https://msdn.microsoft.com/en-us/library/system.web.httprequest(v=vs.110).aspx

# Catch-all Parameter

If you need to accept an unknown or unlimited number of parameters from a route then add one final parameter placeholder that is prefixed with an asterisk. All of the remaining segments of the URL will be passed to that parameter as a single string.

In this example we need to capture the bike's size, color and a list of options with a URL like this:

http://www.example.com/buildabike/42/red/**basket/bell/waterbottle/repairkit**

The route would look like this:

```
routes.MapRoute(
    name: "BuildABike",
    url: "BuildABike/{size}/{color}/{*options}",          ← note the asterisk here!
    Defaults: new { controller = "Products", action = "BuildABike" }
);
```

And the Action would look like this:

```
public ActionResult BuildABike(string size, string color, string options)
{
    return Content("Your bike!  size=" + size  +
                   " color=" + color  + " and " + options.Split('/').count + " options!");
}
```

Copyright © 2019 MAX Technical Training

Using RouteDebugger for the above example would show how the data was parsed from the route and that the last four segments of the URL were captured into a single value named “options”:

**Route Data**

| Key | Value |
|---|---|
| size | 42 |
| color | red |
| options | basket/bell/waterbottle/repairkit |
| controller | Products |
| action | BuildABike |

Note that the final parameter is just a string and not an array of strings. For the example above you will need to parse the “options” variable using code.

```
String[] bikeOptions;
if (options != null) { bikeOptions = options.Split('/'); }
```

Tip! Notice that Split in the example above uses single quotes as the Split method expects a character type, not a string type.

# Parameter Notes

- All values are passed in as strings.
- Any parameter that is reference type or nullable value type that is not matched will be set to null. Non-nullable parameters that are not matched (example: int) will raise an exception.

  Server Error in '/' Application.

  *The parameters dictionary contains a null entry for parameter 'id' of non-nullable type 'System.Int32' for method 'System.Web.Mvc.ActionResult Details(Int32)' in*

  - Tip: Non-nullable types can be made nullable by adding a question mark to the type.
    - int x = 5; (x can never be null!)
    - int? x = 5; (x can be null!)
    - See: https://msdn.microsoft.com/en-us/library/1t3y8s4s.aspx
- Data type can be validated by adding a constraint to a route definition. The following only maps the route if the {part} is an integer.

```
routes.MapRoute(
    name: "part",
    url: "parts/{part}",
    defaults: new { controller = "Parts", action = "PartLookup" },
    constraints: new { part = @"\d+" } );
```

- Data type will be validated at run time, and will raise an error if the type is not a match.
  - For this route

```
public ActionResult Details(int id)
{
    return View("Details");
}
```

  - If the user types an string where an integer was expected…
    /demo/details/abc instead of /demo/details/123

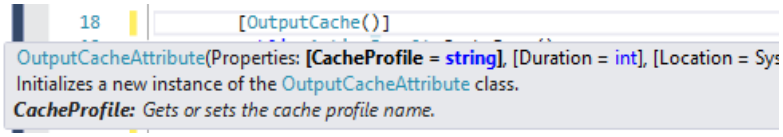
Copyright © 2019 MAX Technical Training

- This error will be raised:

> Server Error in '/' Application.
>
> *The parameters dictionary contains a null entry for parameter 'id' of non-nullable type 'System.Int32' for method 'System.Web.Mvc.ActionResult Details(Int32)' in 'WebApplication10.Controllers.DemoController'. An optional parameter must be a reference type, a nullable type, or be declared as an optional parameter.*
> *Parameter name: parameters*

# Action Attributes

MVC includes a number of attributes that can be applied to Actions. Attributes are placed before the method declaration and are enclosed in square brackets. Many attributes have properties. These can be discovered by typing the attribute's name followed by a left parentheses and noting the IntelliSense dropdown.




When researching attributes note that the attribute's class name includes the word "Attribute".

[RequireHttps] → System.Web.Mvc.RequireHttpsAttribute

## Custom Attributes

You can create your own attributes by creating a class that inherits from System.Attribute. See: https://msdn.microsoft.com/en-us/library/84c42s56(v=vs.110).aspx

## Attribute Based Routes

Routes can be defined by adding the Route attribute to an action. Attribute Based Routes are covered later in Module 3.

```
[Route("products/{partnumber:int}")]
```

## [ActionName]

Multiple methods can have the same name as long as they have the different signatures (number and data types of parameters). When you would like to access an action using a different name than the method's name, then add an ActionName attribute.

A common example is when you want one Action called from a GET and another called from a POST, but both have the same method signatures. As an example, the Controller templates use [ActionName] with the Delete Actions. The first Delete Action retrieves the data for the item and returns a View for user confirmation. The second Delete Action performs the actual delete operation.

Copyright © 2019 MAX Technical Training

```
// GET: CreditCards/Delete/5
public ActionResult Delete(int? id)
{
    if (id == null)
    {
        return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
    }
    CreditCard creditCard = db.CreditCards.Find(id);
    if (creditCard == null)
    {
        return HttpNotFound();
    }
    return View(creditCard);
}

// POST: CreditCards/Delete/5
[HttpPost, ActionName("Delete")]
[ValidateAntiForgeryToken]
public ActionResult DeleteConfirmed(int id)
{
    CreditCard creditCard = db.CreditCards.Find(id);
    db.BillingDetails.Remove(creditCard);
    db.SaveChanges();
    return RedirectToAction("Index");
}
```

# Restricting Access to Actions and Controllers

MVC includes a number of attributes that can be applied to Controllers and Actions to limit access.

## [RequireHttps]

Applies to: Controllers, Actions

The RequireHttps attribute can be applied to the Controller to set the default behavior for all of the Controller's actions. The default is no SSL required.

```
[RequireHttps]
public class SalesController : Controller
{
```

The RequireSSL attribute can also be applied at the Action level to override the setting or default found at the Controller level.

Note: This attribute does not enable SSL support in your web server. An SSL certificate still must be installed on the web server. If you do not have SSL available on your development server then you can wrap the attribute in a conditional compilation statement. Also, search the web for "SSL on IIS 7.0 Using Self-Signed Certificates" to see how to setup IIS with SSL on your development server.

```
#if !DEBUG
    [RequireHttps]
#endif
public ActionResult GetProducts()
{
    return View();
}
```

## [AllowAnonymous]

Applies to: Controllers, Actions

Allows anonymous access. When used at the Action level, this overrides [Authorize] at the Controller.

Copyright © 2019 MAX Technical Training

# [Authorize]

Applies to: Controllers, Actions

Denies anonymous access. The user must be authenticated (logged on). When used at the Action level, this overrides [AllowAnonymous] at the Controller.

Users who are not authorized with get an error ("HTTP Error 401.0 – Unauthorized") or may be directed to a login page.

## User must be authenticated

```
[Authorize]
```

## User must be one of the listed users

This is not commonly used as we don't want to hardcode user names! The name format must match what is returned from the controller's User method. (User.Identity.Name or can also be accessed from System.Security.Principal.WindowsIdentity.GetCurrent().Name)

```
[Authorize(Users = @"example\asmith, example\bjones, example\cjackson")]
```

> Note: The "@" is needed to treat the string as literal text and to not treat the backslash as an escape character.

## User must be one of the listed roles

Roles are groups of users such as administrator or managers. Your choice of authentication may not support roles and may have another name for them such as "groups".

```
[Authorize(Roles = @"Administrator, Manager")]
```

# [ValidateAntiForgeryToken]

Applies to: Actions, Controllers

When working with forms that update data, you want to make sure that a hacker does not "replay" a form post or try a direct attack. The ValidateAntiForgeryToken attribute verifies that the anti-forgery token value has not expired. This token is used to prevent cross site request forgery attacks and is stored both in an HTTP-only cookie and an <input type="hidden"> tag. To use, add the ValidateAntiForgeryToken attribute to your Action and add an HTML helper, @Html.AntiForgeryToken(), to your View's <form> section.

```
[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult Delete(int id, FormCollection collection)
{
```

Also see:

- XSRF/CSRF Prevention in ASP.NET MVC and Web Pages
  https://docs.microsoft.com/en-us/aspnet/mvc/overview/security/xsrfcsrf-prevention-in-aspnet-mvc-and-web-pages

# [ChildActionOnly]

Applies to: Actions

Copyright © 2019 MAX Technical Training

(See the Views module.) This Action can only be called from a View using the Action or RenderAction methods. It cannot be called from a GET (direct browser URL). Note that a child action does not have to be marked as ChildActionOnly to be used as a child action.

## [Http*VerbName*]

[HttpDelete], [HttpGet], [HttpHead], [HttpOptions], [HttpPatch], [HttpPost], [HttpPut]

Applies to: Actions

These attributes only allow access to the Action when requested using the specified verb. Also see AcceptVerbs.

## [AcceptVerbs]

Applies to: Actions

Multiple [Http*VerbName*] attributes cannot be applied to the same Action. The AcceptVerbs attribute is used to specify multiple acceptable verbs for the same Actions. Multiple verb types are OR'd together (|).

Examples:

[AcceptVerbs(HttpVerbs.Post)] (Equivalent to [HttpPost])

[AcceptVerbs(HttpVerbs.Post | HttpVerbs.Get)]

## [NonAction]

Applies to: Actions

By default, MVC treats all public controller methods as Actions. NonAction marks a Controller method as not being an Action and prevents accidental route binding to the method.

```
[NonAction]
public double PriceHelper(string partNumber)
{
```

# Improving Performance with Caching

Caching creates a temporary copy of data, typically stored in RAM, to avoid recreating commonly used data. The life cycle of a request is usually less than a second and consists of the creation of a Controller, the Model data and an Action, all of which are disposed of when the data is returned to the user.

A simple request for a list of products:

1. Our user clicks a link for the Products page.
2. MVC creates an instance of the Products Controller.
3. The Controller creates as instance of the Products Model.
4. The Products Model calls SQL Server to retrieve the 100 products.
5. The Controller passes the products collection to a View.
6. A View object is created and rendered and returns HTML to the user's browser.
7. All of the above objects are then disposed of.

Copyright © 2019 MAX Technical Training

8. All of the above is then repeated for each user visit to this page. (And our list of products rarely changes!)

Examples of caching:

- The entire "Sale of the Day" page. (Only updated one every night.)
- The data for a dropdown list that contains a list of retail stores that carry our product.
- The partial view for the "Tip of the day" that's displayed on ten different pages.

Keep in Mind…

- Caching stores data on the server. If you have multiple load balanced servers the users could potentially be viewing data cached at different times depending on the server they were directed to.
- If the data is stored in SQL Server then consider using the SqlDependency option.
- You may want to investigate third party caching options like Redis (available from NuGet).

Caching options:

- ASP.NET
  - Session and Application variables.
  - System.Web.HttpContext.Current.Cache
- MVC
  - The OutputCache attribute
- Third party solutions:
  - Redis http://redis.io/
  - Azure Redis Cache https://azure.microsoft.com/en-us/services/cache/
- Custom solutions using SQL Server or a dedicated cache server.

# [OutputCache]

Applies to: Actions, Controllers

MVC provides an OutputCache attribute that can be applied at both the Controller and Action level that specifies that an Action's output will be cached. If the Action returns a View, then the View will be cached. (Similar to the "@OutputCache" directive in ASP.NET Web Forms.) Duration is in seconds.

Cache output for five minutes: (VaryByParam is required.)

```
[OutputCache(Duration=300, VaryByParam="none")]
```

## Location

By default, output is cached in three locations: the web server, any proxy servers, and the web browser. You can choose the location: Any, Client Downstream, None, Server, ServerAndClient.

```
[OutputCache(Duration = 300, VaryByParam = "none",
Location = System.Web.UI.OutputCacheLocation.Server)]
```

Copyright © 2019 MAX Technical Training

## VaryByParam

Multiple versions of Action output can be cached based on either a path or Query String parameter. This example caches a different version for each state.

```
[OutputCache(Duration = 300, VaryByParam = "state")]
public ActionResult GetStores( string state ) { … }
```

## SqlDependency

Often you will want to cache content until the source of the content changes. With the SqlDependency option you can establish a link between changes in a Microsoft SQL Server table and the automatic expiring of a cached object.

For example, the following Action will be cached on the first request. The Action's data returned from the cache for all future requests until the SQL table is updated. On the next visit the View will be recreated.

```
[OutputCache(SqlDependency = "ProductListDependency:Products",Duration = 300)]
public ActionResult Index()
{
```

The setup for SqlDependency includes:

- Configure the SQL database to support:
  - Run: aspnet_regsql.exe -S *servername* -E -ed -d *databasename* -et -t *tablename*
  - Note: aspnet_regsql.exe is in the C:\Windows\Microsoft.Net\Framework64*version* folder.
- Add a section to the project's web.config file for <caching><sqlCacheDependency>. Note that the pollTime is in milliseconds.

```
<caching>
  <sqlCacheDependency enabled="true">
    <databases>
      <add name="ProductListDependency" connectionStringName="MyConnectionString"/>
    </databases>
  </sqlCacheDependency>
```

- If already present, add a connection string to the project's web.config file. This is probably the same connection used for your existing SQL connection. <connectionStrings><add …
- The addition of the OutputCache attribute to the Action.

```
[OutputCache(SqlDependency = "ProductListDependency:Products",Duration = 300)]
public ActionResult Index()
{
```

Notes:

- pollTime is in milliseconds.
- Duration is in seconds.
- The cached View will expire either when the Duration has passed or the last poll of the SQL server has detected a table change.
- Do not poll too frequently. Each poll is a round trip to the database server.
- The table specified for the SqlDependency does not have the be the same as the table(s) used to populate the View.
- Cached data may be automatically expired to free up space when the server is running low on resources.

See:

- https://msdn.microsoft.com/en-us/library/system.web.caching.sqlcachedependency(v=vs.110).aspx

Copyright © 2019 MAX Technical Training

## Reusable Cache Profiles (CacheProfile)

If you have common pattern of caching, and you would like to store it in a central and editable location, you can create cache profiles in the project's web.config file. The profile defines the same options that can be applied to the OutputCache attribute.

An example web.config entry:

```
  <caching>
    <outputCacheSettings>
      <outputCacheProfiles>
        <add name="CachePublicData" duration="300" varyByParam="none"/>
      </outputCacheProfiles>
    </outputCacheSettings>
  </caching>
</system.web>
```

An example use of the cache profile:

```
[OutputCache(CacheProfile="CachePublicData")]
```

## VaryByHeader

On each request to a site a browser sends a collection of key/value pairs as part of the header.

| Key | Value |
|---|---|
| Request | GET /en-us/default.aspx HTTP/1.1 |
| Accept | text/html, application/xhtml+xml, */* |
| Accept-Language | en-US |
| User-Agent | Mozilla/5.0 (Windows NT 6.1; WOW64; Trident/7.0; rv:11.0) like Gecko/20100101 Firefox/12.0 |
| Accept-Encoding | gzip, deflate |
| Host | msdn.microsoft.com |
| DNT | 1 |
| Connection | Keep-Alive |
| Cookie | MC0=1488160020924; AmbientContext=%7b%22layout.limit_max_width%22%3afalse%2c%22 |

Some of these choices are good examples for VaryByHeader as they only return one value. Some like Accept-Language can return multiple values. If you used VaryByHeader for this you would not have a cached copy for each language, you would have one for each language combination!

| | |
|---|---|
| Accept-Language | en-US,fr-FR;q=0.7,de;q=0.3 |

VaryByHeader example:

You might have two different domains (www.example1.com and www.example2.com) pointing to your site and you deliver different results based on the domain. You could cache unique results for each domain by using:

```
[OutputCache(Duration = 300, VaryByHeader = "host")]
```

| | |
|---|---|
| Accept-Encoding | gzip, deflate |
| Host | msdn.microsoft.com |

## VaryByCustom

You can cache from a custom value, typically something returned from the browser, by overriding Global.asax's GetVaryByCustomString method. You might detect the browser type, the language preferences sent by the browser or maybe an IP address range. The method accepts a string from VaryByCustom and returns a string that is used as a key for a cached item.

Copyright © 2019 MAX Technical Training

As an example, our site returns English, French and German versions of a page and we would like to cache these three versions.

Steps:

1. Override Global.asax's GetVaryByCustomString method.
2. Write code to:
    a. Check to see if the "by" string is "AcceptLanguage". (Our choice of names!)
    b. Check each language listed in the header sent by the browser to see if any start with "en", "fr" or "de". If there's a match , return the first two letters, otherwise return "en".

```
public override string GetVaryByCustomString(HttpContext context, string custom)
{
    if (custom == "AcceptLanguage")
    {
        string accept = context.Request.Headers["Accept-Language"];
        string[] langs = accept.Split(',');
        foreach (string lang in langs)
        {
            if (lang.StartsWith("en") || lang.StartsWith("fr") || lang.StartsWith("de"))
            {
                return lang.Substring(0, 2);
            }
        }
        return "en";
    }
    // if (other custom things to check) {}

    // otherwise return the default
    return base.GetVaryByCustomString(context, custom);
}
```

3. Add an OutputCache attribute to the Action with VaryByCustom="AcceptLanguage". (The same name used in step 2a above.)

```
[OutputCache(VaryByCustom = "AcceptLanguage", Duration = 300)]
public ActionResult MyAction()
{
    return View();
}
```

Notes:

- A single override of Global.asax's GetVaryByCustomString can handle multiple "custom" tests.
- On the first call to an Action, or after the cached item has expired, GetVaryByCustomString is called after the "return" statement to get the key to use for the cached item.
- Each call to the Action after it has been cached first calls GetVaryByCustomString to get the key to use to see if there is a cached version of the View.
- GetVaryByCustomString can be used from [OutputCache] and from the System.Web.HttpRuntime.Cache object.

# Caching Objects

Objects can be cached using many techniques that are part of the .Net Framework or from third party solutions.

Notes about object caching:

- Data is "boxed" and stored as an object type. I.e. a string is cast as an object and then stored.
- Data retrieved from a cache must be "unboxed" and cast back to the correct type.
- Cache objects expire.
- You must always check that the cache still contains the object before trying to access it.
- Caching technologies may have a helper method to test for existence before retrieving the data.

Copyright © 2019 MAX Technical Training

- Caching technologies may have callback features to automatically repopulate an expired item.

Classes and methods to choose from:

- System.Web.HttpContext.Current.Cache & System.Web.HttpRuntime.Cache
  - Both System.Web.HttpContext.Current.Cache and System.Web.HttpRuntime.Cache store data in the same cache. System.Web.HttpRuntime.Cache is preferred as it can be used outside of the current web request context and may be a little faster.
  - Is not available to asynchronous processes running on a background thread.

  - System.Web.HttpContext.Current.Cache
    - System.Web.HttpContext.Current.Cache["FromHttpContext.Current"] = "123";
      or from the Controller class helper method:
    - HttpContext.Cache["FromHttpContext"] = "123";
  - System.Web.HttpRuntime.Cache
    - System.Web.HttpRuntime.Cache["FromHttpRuntime"] = "123";
  - *controller*.HttpContext (MVC Controllers)
    - This is a helper method from the Controller base class and is a shortcut to System.Web.HttpContext.Current.Cache.
  - *page*.HttpContext (ASP.NET Web Forms pages)
    - This is a helper method from the Controller base class and is a shortcut to System.Web.HttpContext.Current.Cache.

- System.Runtime.Caching
  - No dependency on System.Web (so it works with non-web projects and .NET Core.
  - Requires adding an additional reference to an MVC project.
  - Available to asynchronous processes running on a background thread.
  - Includes a convenient "AddOrGetExisting" method.

# Adding Items to a Cache

You have already seen that adding the [OutputCache] attribute to an Action will cache an entire page. You can also cache business objects to an in-memory (on the server) cache. In the examples below we will be using System.Runtime.Caching, but most of the examples also apply to System.Web.HttpRuntime.Cache.

For details on System.Runtime.Caching see
https://msdn.microsoft.com/en-us/library/system.runtime.caching.memorycache(v=vs.110).aspx

Items can be added to the cache:

- With no expiration options.
- With an expiration.
- With an expiration and options.

## Adding using an indexer

Objects can be quickly added to the cache with the following shortcut assignments. Note that these do not automatically expire, but may be expired as system resources run low.

```
System.Runtime.Caching.MemoryCache.Default["LevelOneDiscount"]="0.15";
System.Runtime.Caching.MemoryCache.Default["ProductList"]=products;
```

Or:

Copyright © 2019 MAX Technical Training

```
using System.Runtime.Caching;
…
MemoryCache.Default["LevelOneDiscount"]="0.15";
MemoryCache.Default["ProductList"]=products;
```

Or:

```
At the class level add:
  var mycache = MemoryCache.Default;

And in your class methods:
  mycache["LevelOneDiscount"]="0.15";
```

## Adding with the Add method

The Add method has several overloads to add a new cache item and options. The Add method returns a Boolean to indicate success.

```
bool status = mycache.Add( string, object, DateTimeOffset, string);
```

The first string is the name of the cached item. The last string is optional and represents a "regionName", and as it is not implemented yet in the .NET Framework, it can be ignored!)

This example sets a 60 minute expiration. The parameters are the name of the item (the key), the object be cached and an absolute expire time.

```
string[] states = { "Ohio", "Kentucky", "New York" };
mycache.Add("StateList", states, DateTimeOffset.Now.AddMinutes(60));
```

# Reading Items from a Cache

As items are stored into the cache they are cast as object. When an object is returned from the cache it will need to be cast back to the original data type.

In one of the earlier examples we stored an array of strings that represented states. The example below returns this list.

```
string[] states2 = (string[])mycache["StateList"];
```

If the item named "StateList" is no longer in the cache, then a null value is returned. As the item could have already expired, or was never added to the cache, you will need to check to see if the returned value is null, and if it is, reload the value to the cache.

# Cache Notes

- The cache classes described here are memory based. Each time your application is restarted the cache content will be lost.
- You must always check to see if the item in the cache exists (or if it returns null). It may have expired.
- The key name in System.Runtime.Cache is case sensitive!

Copyright © 2019 MAX Technical Training

# Dealing with Controller and Action Exceptions

You have several options to deal with runtime exceptions.

- Traditional error handling using try-catch-finally.
- Action level attributes. ( [HandleError()] )
- Controller level attributes. ( [HandleError()] )
- Web.config <customErrors>

## [HandleError]

MVC includes Action and Controller attributes to trap errors not otherwise handled in code.

Steps:

1. Custom Errors must be enabled in web.config. The following example redirects all 404 errors to a View named Error404 and all otherwise unhandled errors to a view named "Error". These Views are typically stored in the Views/Shared folder.

```
<system.web>
  <compilation debug="true" targetFramework="4.5.2" />
  <httpRuntime targetFramework="4.5.2" />
  <customErrors mode="On" defaultRedirect="Error">
    <error statusCode="404" redirect="~/Home/Error404"/>
  </customErrors>
</system.web>
```

2. Add a HandleError attribute to the Controller class or to an Action.
   a. Redirect all errors to the default error view: (Error.cshtml in the Views/Shared folder):
      [HandleError]
   b. Redirect all errors to a specified view:
      [HandleError(View="SomethingIsBroken")]
   c. Redirect a selected exception to a View:
      [HandleError(ExceptionType = typeof(DivideByZeroException),
      View="SomethingIsBroken")]
3. Create a View to report the error:
   a. Create a View in Views/Shared.
   b. To report the error data:
      i. On the first line of the file set the model:
         @model System.Web.Mvc.HandleErrorInfo
      ii. In the code, access the Model.ControllerName, Model.ActionName and Model.Exception values.

```
@model System.Web.Mvc.HandleErrorInfo
...
Action: @Model.ActionName <br />
Controller: @Model.ControllerName <br />
Exception: @Model.Exception <br />
```

## Setting the Order for Applying [HandleError]

Setting an order will determine with class level and which action level HandleError's will apply. The default, and highest, order number is -1. Larger numbers are lower priority.

[HandleError(Order = 1, ExceptionType = …

Copyright © 2019 MAX Technical Training

# Passing Data to Views

Most of the example Actions you have seen so far just ended with "return View();". These did not pass any data to the View. For most Actions we will need to collect some data and forward it to the View. Some of the possibilities area:

- Pass a Model (an object): `View(domainobject);`
- Pass a View Model:
```
var object = CreateCustomerView(domainobject);
View(object);
```
- Pass ViewData / ViewBag
```
ViewBag.username="Fred";
View();
```
- Pass a model and view data:
```
ViewBag.username="Fred";
View(someobject);
```

## "Model 101"

For the purposes of this discussion, a Model is just an object that is passed to a View. The object could be a string, BankAccount or a List<T>. We will dive deeper into Models in later modules of this course.

## Domain Model vs. View Model

The Domain Model describes our primary business objects and frequently represent data from a SQL table. The model includes all of the properties of the customer, bankaccount or product.

A View Model is subset or superset of the Domain Model. For example, the domain model for Customer includes a DateOfFirstOrder property. This value is never changed and we would not want accidental changes to be made or for a hacker to get access to the value. The View Model may also include non-domain model properties that are needed in the view but are not stored with the primary object. For example, our view may need the number of orders created by the customer and that value is not a property of the Domain Model.

| Domain Model Properties | View Model Properties |
|---|---|
| CustomerID<br>CompanyName<br>Address<br>City<br>State<br>PostalCode<br>DateOfFirstOrder<br>SalesRep | CustomerID<br>CompanyName<br>Address<br>City<br>State<br>PostalCode<br>NumberOfOrders |

## View Model Examples

- Hide the accidental exposure of security related fields like "IsAdministrator" and "Password".
- Filter data: `db.CustomerTypes.Where(ct => ct.Status == "Active")`
- Add derived properties: `customer.FirstName + " " + customer.LastName`
- Add data for a dropdown list: `public List<CustomerType> custTypes { get; set; }`

Copyright © 2019 MAX Technical Training

# ViewBag and ViewData

The ViewData object is obsolete, but still supported for backwards compatibility for older versions of MVC. It has been replaced with the ViewBag object. The ViewBag is a dynamic object and properties can be added at any time.

```
public ActionResult Index()
{
    ViewBag.UserName = User.Identity.Name;
    if (ViewBag.UserName == "")
    {
        ViewBag.UserName = "Guest";
    }
    return View();
}
```

ViewBag Notes:

- Data is shared between ViewBag and ViewData. Data stored as ViewBag.somedata =123 can be retrieved using ViewData["somedata"].
- All data stored to a ViewData are cast to object and must be recast to the original type on retrieval. Recasting is not needed for ViewBag.
- Data is forwarded from a Controller to a view, never from a View to a Controller. (i.e. ViewBag data is lost on a redirect.) The only way for a View to send data to a Controller is through a <form> submission or a Query String.
- As ViewBag is dynamic and has no predefined properties, simple typos will not be caught by the editor or the compiler and can be a challenge to debug.
- Life Cycle: The ViewBag is initially created in the Controller and then is passed to a View where it is visible to the View, the layout file and any child views.

# TempData

The TempData object is a dictionary (name/value pairs) similar to ViewData, but is used to pass data between Controllers and not from a Controller to a View. TempData is commonly used to forward data from an Action when it redirects to another Action, for example using RedirectToAction().

```
public ActionResult Index()
{
    ViewBag.UserName = "Guest";
    TempData["UserSource"] = "internal";

    return RedirectToAction("Index2");
}
public ActionResult Index2()
{
    // this data has been lost!
    string s1 = ViewBag.UserName;

    // this data visible to destination action
    string s2 = (string)TempData["UserSource"];

    return View();
}
```

Notes:

- Data from TempData is not accessible from within a View().
- All data stored to TempData are cast to object and must be recast to the original type on retrieval.

Copyright © 2019 MAX Technical Training

# Action Filters

Action Filters are a type of attribute that may run before or after an action has run. As an example, the OutputCache attribute is called before an Action is executed to see if there is still a cached copy available. If there is still a cached copy, then OutputCache returns data from the Action without re-executing the Action, otherwise, the Action is executed. OutputCache is called after the action has run so it can store the result in the cache.

## Built-in Action Filters

MVC includes three out of the box Action Filters:

- Authorize
- Handle Error
- OutputCache

When multiple filters are applied they are run in this order: Authorization, Action, Result and Exception.

## Custom Action Filters

You can create your own custom Action Filters by creating a class inherits from ActionFilterAttribute.

Each interface defines four override-able methods:

- OnActionExecuting – called before the action is executed.
- OnActionExecuted – called after the action is executed and before the result is executed.
- OnResultExecuting – called before the action result is executed.
- OnResultExecuted – called after the action result is executed.

Each of the override-able methods receives a filterContext parameter which provides access to data about the controller and the request.

Also see:

- Creating Custom Action Filters
  https://msdn.microsoft.com/en-us/library/dd381609(v=vs.100).aspx
- Understanding Action Filters (C#)
  https://docs.microsoft.com/en-us/aspnet/mvc/overview/older-versions-1/controllers-and-routing/understanding-action-filters-cs

# Best Practices

- Keep controllers light weight. Business logic typically belongs in Models.
- Don’t create HTML in a Controller, that’s what Views are for!
- Don’t directly access a database server from a Controller, that’s what Models are for!
- Lock down all Actions. Where appropriate, use attributes to require authentication and authorization, limit the properties that can be changed and the HTTP methods that can be used. Write code to do server-side validation. Also see:
  - https://www.codeproject.com/tips/845612/developing-secure-web-applications-xss-attack-the

Copyright © 2019 MAX Technical Training

    - https://www.codeproject.com/articles/288631/secure-asp-net-mvc-applications#HTTPS
    - https://msdn.microsoft.com/en-us/library/hh404095.aspx
    - https://docs.microsoft.com/en-us/aspnet/mvc/overview/security/create-an-aspnet-mvc-5-web-app-with-email-confirmation-and-password-reset
    - What not to do in ASP.NET, and what to do instead https://docs.microsoft.com/en-us/aspnet/aspnet/overview/web-development-best-practices/what-not-to-do-in-aspnet-and-what-to-do-instead
- Require HTTPS to ensure that all data is encrypted across the network and the internet.
- Only allow GETs to be used to read data, never to update data.
- Validate data everywhere… in the client, in the Action, in the model.
- Log security violations.
- Controllers should not contain direct data access. (That's why we have models!)
- Do not include HTML string manipulation. HTML generally belongs in Views.
- Consider using Areas in large and complex projects.
- Keep search engines in mind when designing routes.
- Use caching where possible. Cache content that is frequently accessed, but infrequently updated.

```
[OutputCache(Duration = 300, VaryByParam = "none")]
public ActionResult Index()
{
    return View();
```

- Create one controller per business entity. (Not necessarily per table or Entity Framework entity.)
- Consider using a View Model in place of a Model plus ViewData/ViewBag.
- Mark non-Action public methods. [NonAction]

```
[NonAction]
public double CalculatePrice(string partNum)
{
    return 0.0;
}
```

# Attributes for Controllers and Actions

The following is for reference and lists most of the MVC Controller and Action attributes.

To find the MSDN documentation for each of the following navigate to:

https://msdn.microsoft.com/en-us/library/system.web.mvc.***AttributeName***attribute

For example to find the documentation for AcceptVerbs go to:

https://msdn.microsoft.com/en-us/library/system.web.mvc.***AcceptVerbs***attribute

Where the attribute is used:

- C = Controller
- A = Action
- P = Action Parameter
- MC = Model Class
- MP = Model Property

Copyright © 2019 MAX Technical Training

| Attribute | Where | |
|---|---|---|
| AcceptVerbs | A | [AcceptVerbs(HttpVerbs.Post)] *(equivalent to [HttpPost])*<br>[AcceptVerbs(HttpVerbs.Post \| HttpVerbs.Get)]<br><br>Verbs: Get, Post, Put, Delete, head, Patch, Options.<br>Note: While [HttpGet] and [HttpPost] cannot both be used on the same Action, AcceptVerbs lets you specify any number of verbs. |
| ActionFilter | | Abstract base class for filter attributes. Never directly used. |
| ActionMethodSelector | | Never directly used. You can inherit from this class to customize how Actions are select. |
| ActionName | A | Used to supply an alternate name for an Action.<br><br>[ActionName("Categories")] |
| ActionNameSelector | | Never directly used. |
| AdditionalMetadata *(for a model class)* | MC | Adds additional metadata about a model property that can be accessed from a view using ViewData.ModelMetadata.AdditionalValues.<br><br>Code for the Model:<br>[System.Web.Mvc.AdditionalMetadata("testMetadata","ABC")]<br>public class BankAccount : BillingDetail<br><br>Code for the View:<br>ViewData.ModelMetadata.AdditionalValues["testMetadata"] |
| AllowAnonymous | A,C | Allows anonymous access. When used at the Action level, this overrides [Authorize] at the Controller. |
| AsyncTimeout | A | Used to set the timeout (in milliseconds) for asynchronous Actions. (Default is 45 seconds or 45000.) Also see [NoAsyncTimeout].<br><br>`[AsyncTimeout(60000)]`<br>`public async Task<ActionResult> IndexAsync()`<br>`{`<br>`  // ...`<br>`  return View();`<br>`}` |
| Authorize | A,C | Denies anonymous access. When used at the Action level, this overrides [AllowAnonymous] at the Controller.<br><br>Can limit access to selected Roles:<br>[Authorize(Roles="Admins, SalesManagers")] |
| Bind | P | Controls if a property, usually of a complex type, can be bound. Options include "Include" and "Exclude". The following insures that the CartType property of the creditCard object cannot be changed by this view. (This is one way to deal with the "Over Posting Attack".)<br><br>public ActionResult Edit( **[Bind(Exclude = "CardType")]** CreditCard creditCard) |
| ChildActionOnly | A | This Action can only be called from a View using the Action or RenderAction methods. It cannot be called from a GET. A child action does not have to be marked as ChildActionOnly to be used as a child action. |

Copyright © 2019 MAX Technical Training

| | | |
|---|---|---|
| Compare | | System.Web.Mvc.Compare is obsolete. Use System.ComponentModel.DataAnnotations.Compare instead.<br><br>[System.ComponentModel.DataAnnotations.Compare("Password", ErrorMessage = "Passwords must match!")]<br>public string Password2 { get; set; } |
| Filter | | Never directly used. Base class for Action filters. |
| HandleError | A,C | Used to redirect to an error page or view. Custom Errors must be enabled in web.config!<br>(<customErrors mode="On"></customErrors>)<br><br>Redirect all errors to the default view (Error.cshtml in the Views/Shared folder):<br>[HandleError]<br><br>Redirect all errors to a view:<br>[HandleError(View="SomethingIsBroken")]<br><br>Redirect a selected exception to a View:<br>[HandleError(ExceptionType = typeof(DivideByZeroException), View="SomethingIsBroken")]<br><br>Setting an order will determine with class level and which action level HandleError's will apply. The default, and highest, order number is -1. Larger numbers are lower priority.<br>[HandleError(Order = 1, ExceptionType = …<br><br>A view can access error related data:<br>• On the first line set the model: @model System.Web.Mvc.HandleErrorInfo<br>• In the code access Model.ControllerName, Model.ActionName and Model.Exception.<br><br>**@model System.Web.Mvc.HandleErrorInfo**<br>...<br>Action: **@Model.ActionName** <br /><br>Controller: **@Model.ControllerName** <br /><br>Exception: **@Model.Exception** <br /> |
| HttpDelete | A | Only permits the Action to be called from the DELETE method. For multiple methods see AcceptVerbs. |
| HttpGet | A | Only permits the Action to be called from the GET method. For multiple methods see AcceptVerbs. |
| HttpHead | A | Only permits the Action to be called from the HEAD method. For multiple methods see AcceptVerbs. |
| HttpOptions | A | Only permits the Action to be called from the OPTIONS method. For multiple methods see AcceptVerbs. |
| HttpPatch | A | Only permits the Action to be called from the PATCH method. For multiple methods see AcceptVerbs. |
| HttpPost | A | Only permits the Action to be called from the POST method. For multiple methods see AcceptVerbs. |

Copyright © 2019 MAX Technical Training

| | | |
|---|---|---|
| HttpPut | A | Only permits the Action to be called from the PUT method. For multiple methods see AcceptVerbs. |
| ModelBinder | P | Used to specify a custom model binder class. |
| NoAsyncTimeout | A | Sets that asynchronous timeout to infinite. Also see [NoAsyncTimeout].<br>`[NoAsyncTimeout]`<br>`public async Task<ActionResult> IndexAsync()`<br>`{`<br>`  // ...`<br>`  return View();`<br>`}` |
| NonAction | | Marks a Controller method as not being an Action. (By default, MVC treats all public controller methods as Actions. |
| OutputCache | A, C | Specifies that an Action's output will be cached. If the Action returns a View, then the View will be cached. (Similar to a view's "@OutputCache" directive. Duration is in seconds.<br>Cache output for five minutes: (VaryByParam is required.)<br>[OutputCache(Duration=300, VaryByParam="none")]<br>By default, output is cached in three locations: the web server, any proxy servers, and the web browser. You can choose the location: Any, Client Downstream, None, Server, ServerAndClient.<br>[OutputCache(Duration = 300, VaryByParam = "none", **Location = System.Web.UI.OutputCacheLocation.Server)**]<br>Multiple versions of Action output can be cached. This example caches a different version for each state.<br>[OutputCache(Duration = 300, **VaryByParam = "state")**]<br>public ActionResult GetStores( string state ) { … }<br>Cache profiles can be created in the project's web.config and then be referenced by OutputCache.<br>[OutputCache(CacheProfile="CachePublicData")]<br>`<caching>`<br>`<outputCacheSettings>`<br>`<outputCacheProfiles>`<br>`<add name="CachePublicData" duration="300" varyByParam="none"/>`<br>`</outputCacheProfiles>`<br>`</outputCacheSettings>`<br>`</caching>`<br>`</system.web>` |
| Remote | A | Enables client side validation by allowing jQuery to call this Action on the server. (Requires additional configuration. See: https://msdn.microsoft.com/en-us/library/gg508808(VS.98).aspx) |
| RequireHttps | A, C | Limits access to HTTPS requests. (SSL must be configured on the web servers.)<br>**https**://www.examples.com/products/details/123 |
| SessionState | C | Sets the session state behavior for a Controller. Options are Default, Disabled, ReadOnly, Required<br>[SessionState(System.Web.SessionState. |

Copyright © 2019 MAX Technical Training

| | | |
|---|---|---|
| | | SessionStateBehavior.Required)] |
| ValidateAntiForgeryToken | A,C | Verifies that the anti-forgery token value has not expired. Used to prevent cross site request forgery attacks. The token is stored both in an HTTP-only cookie and an <input type="hidden"> tag. To use, add the attribute to your Action and add @Html.AntiForgeryToken() to your View's <form> section.<br>`[HttpPost]`<br>`[ValidateAntiForgeryToken]`<br>`public ActionResult Delete(int id, FormCollection collection)`<br>`{`<br>Also see: https://docs.microsoft.com/en-us/aspnet/web-api/overview/security/preventing-cross-site-request-forgery-csrf-attacks |
| ValidateInput | A,C | When set to false, will allow (potentially dangerous) HTML & JavaScript to be posted from a form.<br>`[HttpPost]`<br>`[ValidateInput(false)]`<br>`public ActionResult Create(FormCollection collection)`<br>`{`<br>Note: [AllowHTML] can be more granular as it can be applied to individual properties in a model. When ValidateInput(false) is applied to a Controller or Action, every parameter of the Action will accept all HTML text. See: https://www.codeproject.com/articles/995931/preventing-xss-attacks-in-asp-net-mvc-using-valida |

Copyright © 2019 MAX Technical Training

# Module 4 – Entity Framework

This module provides an introduction to the Entity Framework so you can you get started with MVC Models without having to write a lot of database code. The Entity Frame is not part of MVC or required by MVC, but is one of the quickest ways of build model classes for an MVC project. Also, the MVC wizards are MVC aware and can create Controllers and Views for you. The Entity Framework is not limited to Microsoft SQL Server… there's also support for DB2, MySQL, Oracle, PostgreSQL and SQLite.

The Entity Framework is an object-relational mapper (O/RM) that maps between .NET CLR objects and relational databases. Instead of tables you work with entity objects, and instead of writing SQL queries you write LINQ queries.

**The Entity Framework:**

- Eliminates most SQL statements.
- Is not string based as SQL statements are, and help avoid errors with the help of object level IntelliSense and compile time code validation.
- Also supports Language Integrated Queries (LINQ) with object level IntelliSense and compile time code validation reducing the number of errors and hacking opportunities of string based SQL statements.
- Provides an in-memory model of the database objects to support caching and batching of updates.

Note that the Entity Framework is not part of the .NET Framework and must be downloaded and installed. The easiest way to install it is from the NuGet Package manager.

The Entity Framework has been around for a while and continues to evolve.

- Version 1 – 2008
- Version 4 – 2010 (never was a version 2 or 3!)
- Version 5 – 2012
- Version 6 – 2013
- Core – 2017

## Other Object-Relational Mapper Tools

From simple projects like Dapper (https://github.com/StackExchange/Dapper) to full featured tools like NHibernate (https://nhibernate.info/). For a more complete list see:
https://en.wikipedia.org/wiki/List_of_object-relational_mapping_software

# Entities?

While an entity often matches to a SQL table, it could also represent a subset of that table, maybe 12 of the 30 fields, or even a combination of multiple tables. A Customer entity could map to a Customer table, or could map to a combination of Customer, SalesRep, Region, CustCategory, etc.

Copyright © 2019 MAX Technical Training

# Context?

The DbContext object is an in-memory representation of entities. Data is read from the database into the context; inserts, updates and deletes are made into the context; and the changes to items in the context are written back to the database. (Microsoft.EntityFramework.DbContext or Microsoft.EntityFrameworkCore.DbContext)

# Getting the Entity Framework

The Entity Framework is not part of MVC or installed with the .NET Framework and must be separately downloaded and installed. *A few Visual Studio templates, such as those for MVC Core, automatically include the Entity Framework.*

- Step 1: Know which version you need! Your team may have standardized on a version of the Entity Framework, and may even specify the exact release.
- Step 2: Download the correct version.
    - You could download EF from Github and then manually install it.
        - EF6: https://github.com/aspnet/EntityFramework6
        - EF Core: https://github.com/aspnet/EntityFramework
    - (Easy way!) You could use NuGet and the Visual Studio NuGet tools to download, install and configure EF into your project.

**To get the Entity Framework using NuGet:**

1. Open your project and then perform one of the following:
    a. Right-click References in the Solution Explorer and select Manage NuGet Packages.
    b. Click the Browse tab.
    c. Search for the "Entity Framework", select a version and click Install.

    Or

    a. In the ribbon click Tools, NuGet Package Manager and Package Manager Console.
    b. Type the following PowerShell command:
    ```
    Install-Package entityframework
    ```

**To get a specific version using the NuGet Package Manager Console:**

1. Open your project.
2. In the ribbon click Tools, NuGet Package Manager and Package Manager Console.
3. Optional: In the console use Find-Package to a see a list of available packages.
    ```
    Find-Package entityframework -ExactMatch -AllVersions |
      Select -ExpandProperty versions |
      Select Version, IslegacyVersion, IsPrerelease
    ```
4. Type the following PowerShell command:
    ```
    Install-Package entityframework -Version 6.0.1.0
    ```

**After installing:**

1. Expand References in the Solution Explorer and verify that the Entity Framework assemblies are there.
2. Click the EntityFramework assembly and click Properties. Verify the version number.

# To Check the Version of Entity Framework in a Core Project

In Visual Studio expand Dependencies, NuGet, Microsoft.AspNetCoreApp and find "Microsoft.EntityFrameworkCore".

Copyright © 2019 MAX Technical Training

# Resources

- **Introduction to Entity Framework 6**
  https://docs.microsoft.com/en-us/ef/ef6/index
- **Entity Framework Glossary**
  https://docs.microsoft.com/en-us/ef/ef6/resources/glossary
- **Entity Framework 6**
  https://github.com/aspnet/EntityFramework6
- **Entity Framework Core**
  https://github.com/aspnet/EntityFrameworkCore

# Database First, Model First and Code First

The Entity Framework supplies three strategies for creating entities and databases.

- **Database First**
  - The SQL database already exists, or someone else owns and maintains it.
  - Entity Framework:
    - Uses a wizard to select database objects and display them in a diagram that is stored as XML (*.edmx).
    - We can customize the entities using a graphical designer.
  - Entity Framework Core uses a PowerShell cmdlet (https://docs.microsoft.com/en-us/ef/core/get-started/aspnetcore/existing-db#reverse-engineer-your-model) or from the CLI (dotnet ef dbcontext scaffold "yourConnectionString" Microsoft.EntityFrameworkCore.SqlServer -o Models)
- **Model First**
  - The SQL database typically does not already exist.
  - We design (draw) the database using a graphical designer.
  - On first run, EF creates the database and tables for us.
  - Not available with Entity Framework Core. (But there are third party tools like https://www.devart.com/entitydeveloper/entity-framework-core-designer.html).
- **Code First**
  - The SQL database typically does not already exist.
  - We write plain old CLR Object (POCO) classes.
  - We define relationships.
  - On first run, EF creates the database and tables for us.
- **Code First from Database**
  - Creates a Code First model from an existing database.
    (This is a good way to study model attributes!)
  - Entity Framework Core uses a PowerShell cmdlet (same as for Database First above)

# Primary Entity Framework Objects

Regardless of which option you use to create your entities, you will end up with a DbContext object and a collection of entity objects that are “plain ole C# objects” (POCO).

- **DbContext Properties:**
  - .Database – access to the underlying SQL database. You an access properties like Connection for the connection string and timeouts, and methods like Delete that can delete the entire database!
  - *.entityName* – access to the entity (BankAccount, Customer, etc.) and its properties and contents.

Copyright © 2019 MAX Technical Training

- .SaveChanges() – A method that saves all pending entity changes.

- **An entity:**
  - Is a member of a DbContext.
    - An example of retrieving data with a context and two entities:

```
BankDbContext db = new BankDbContext();
int CustCount = db.Customers.Count();
int AcctCount = db.CheckingAccounts.Count();
```

  - Includes methods to:
    - .Add() to add new objects (Product, Vendors, etc.) to the database.
    - .Find()
    - .Remove()
    - Why no "Edit()"? When you use the Find() method to find an object you get… well an object, and your variable just contains a pointer to the original object. Change a property of the found object and you've changed the property of the in-memory instance of that object. Call the DbContext's SaveChanges() method and the edit is written back to the database!
    - And through extension methods, the full family of LINQ methods.

# Database First - .Net Framework MVC 5

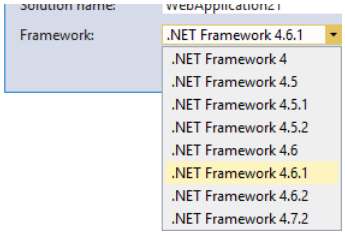
If you are rewriting an application and already have a SQL database that will not change, or will change very little, then you would consider using the Database First approach to creating the entities for your project. With Database First the Entity Framework wizards study your database and then create the entities for you.

Note: The following steps will download the most recent version of the Entity Framework. See "Getting the Entity Framework" earlier in this module for the steps to download any specific version.

## Create an MVC project

Remember MVC is not required for the Entity Framework and the Entity Framework is not required for MVC. They just work well together!

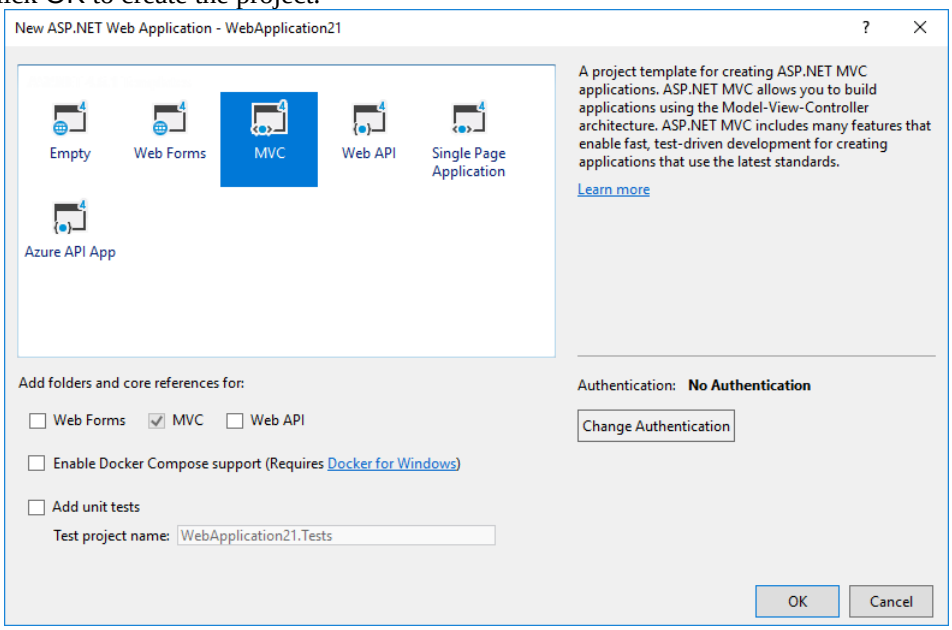
1. Launch Visual Studio 2017.
2. Click New, Project.
3. Expand the templates to Visual C# and Web.
4. Click ASP.NET Web Application (.NET Framework).
5. Select the desired Framework version.




6. Name the project and click OK.
7. In the New ASP.NET Project dialog box click Change Authentication, select No Authentication and OK. (For this demo.)

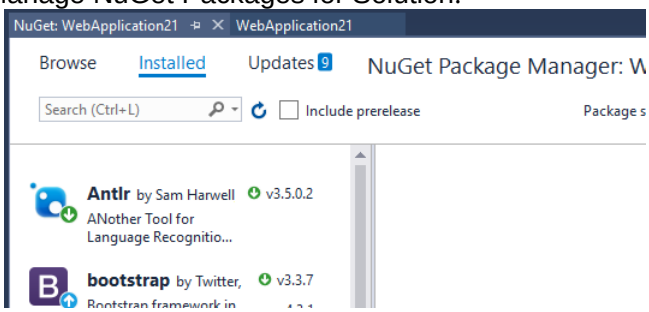
Copyright © 2019 MAX Technical Training

8. Click OK to create the project.

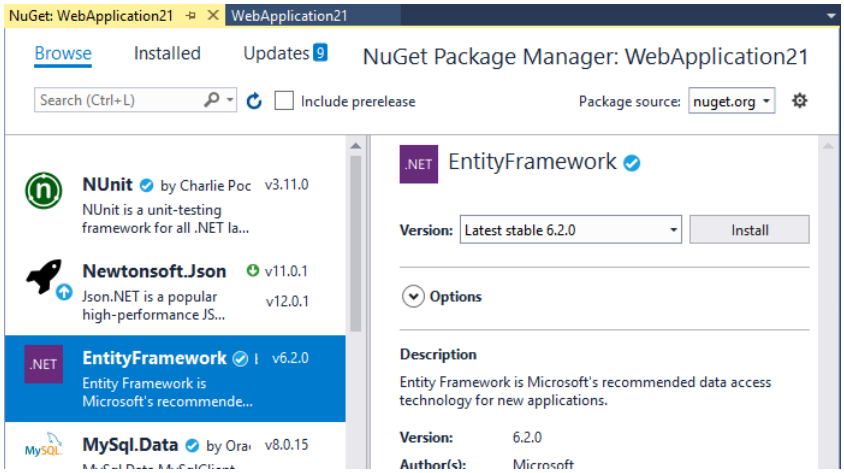



# Use NuGet to download and install the Entity Framework

1. Right-click References in the Solution Explorer pane, or click Tools, NuGet Package Manager, Manage NuGet Packages for Solution.



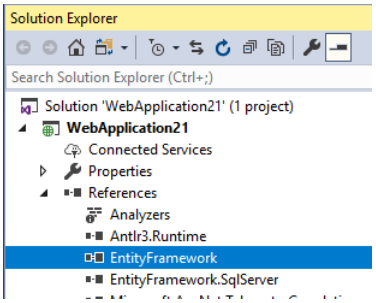

2. Click the Browse tab.




3. If displayed, click the EntityFramework package. Otherwise, search for the Entity Framework. (If you see both Core and EF6, choose EF6.)
4. Click the Install button.
5. Click OK and accept the license agreement to complete the install.

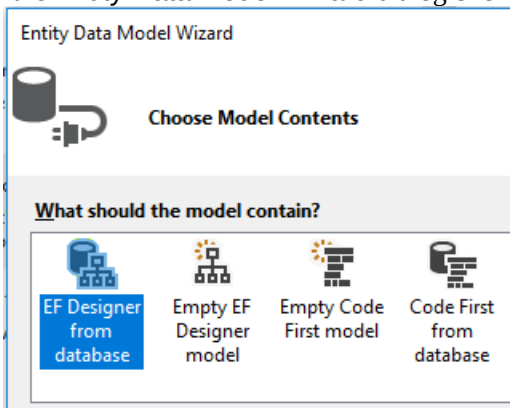
Copyright © 2019 MAX Technical Training

6. Review the list of references and note that the EntityFramework assemblies have been installed.




# Create the Data Model

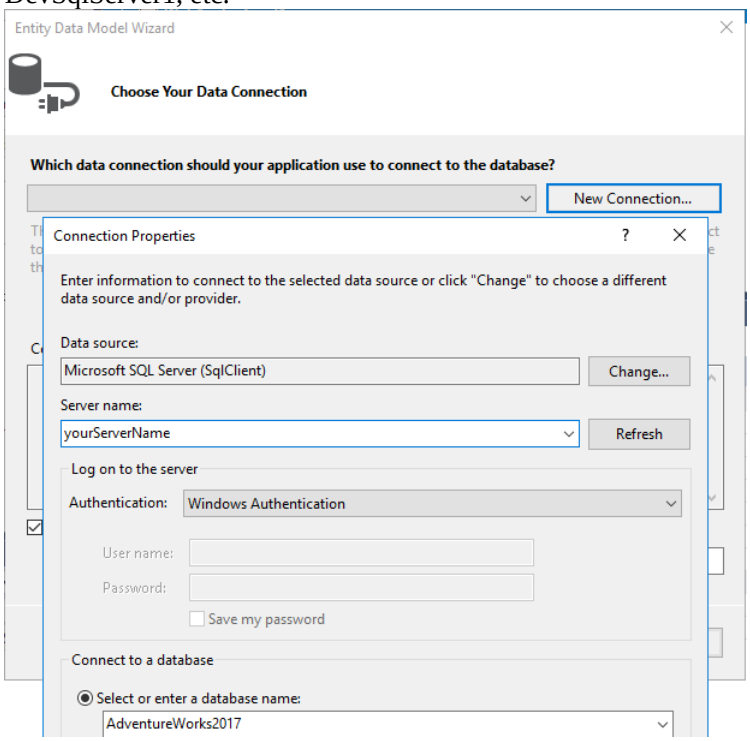
1. In the Solution Explorer, right-click your project, click Add, New Item and then in the Visual C#, Data templates click ADO.NET Entity Data Model, give the model a name ("AdventureWorks" for this example) and click Add.
   or
   In the Solution Explorer, right-click your project, click Add, find and click ADO.NET Entity Data Model, give the model a name ("AdventureWorksDB" for this example) and click OK. (This name is will be the name of your "context".)
2. In the Entity Data Model Wizard dialog click EF Designer from database and click Next.



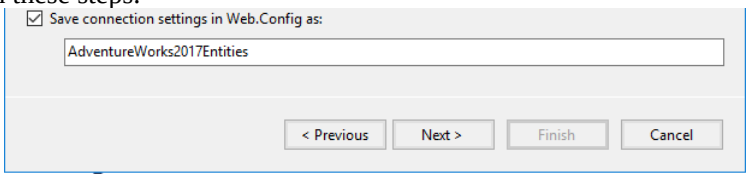

3. Click New Connection.
4. In the Server Name box type the name of your SQL Server and instance.
   Check with your instructor. Possible options for a typical local Visual Studio install:
   (localdb)\ProjectsV13
   (localdb)\MSSQLLocalDB

Copyright © 2019 MAX Technical Training

DevSqlServer1, etc.

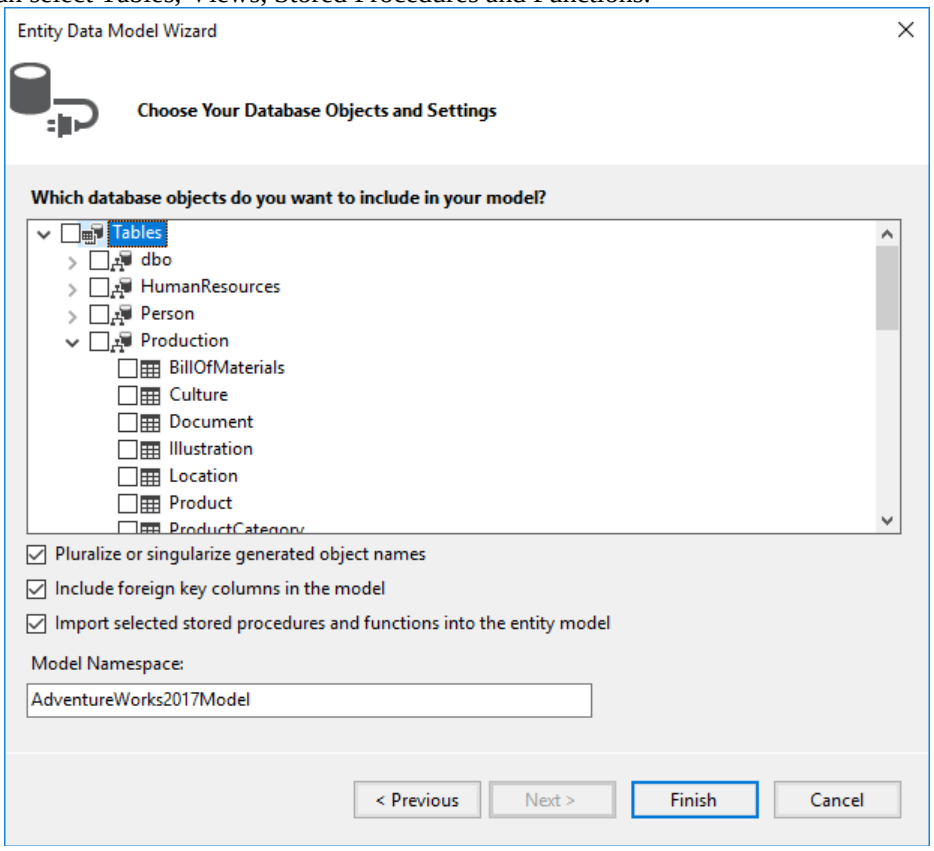



5. From the Select or enter a database name dropdown select your database and click OK.
6. At the bottom of the Entity Data Model Wizard dialog, note the option to save the connection string to your web.config file. This is a default name, you can change it. We will check for this later in these steps.




Copyright © 2019 MAX Technical Training

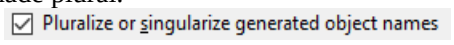
7. Click Next. We are now ready to select the database objects needed for our project. Note that you can select Tables, Views, Stored Procedures and Functions.



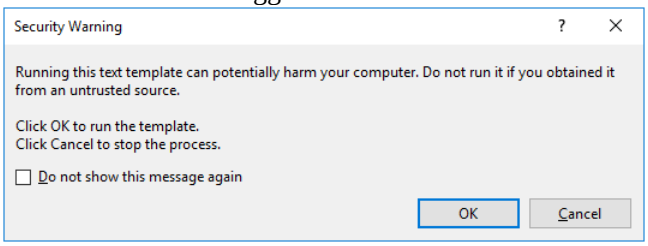

8. For this demo, checkmark the following tables:
   Product
   ProductCategory
   ProductSubcategory
9. Expand Views and note that SQL views can be added as entities. They may be updateable, partially updatable or not updateable depending on their design.
10. Expand Stored Procedures and Functions and note that these can be added as methods to the entity model. Stored procedures will appear as a new method on the DbContext:

```
db.uspUpdateEmployeeHireInfo(123, "Manager",DateTime.Now,DateTime.Now,12345,1,true);
```

11. Note the first check box. When checked, entity names are made singular and Entity Set names are made plural.

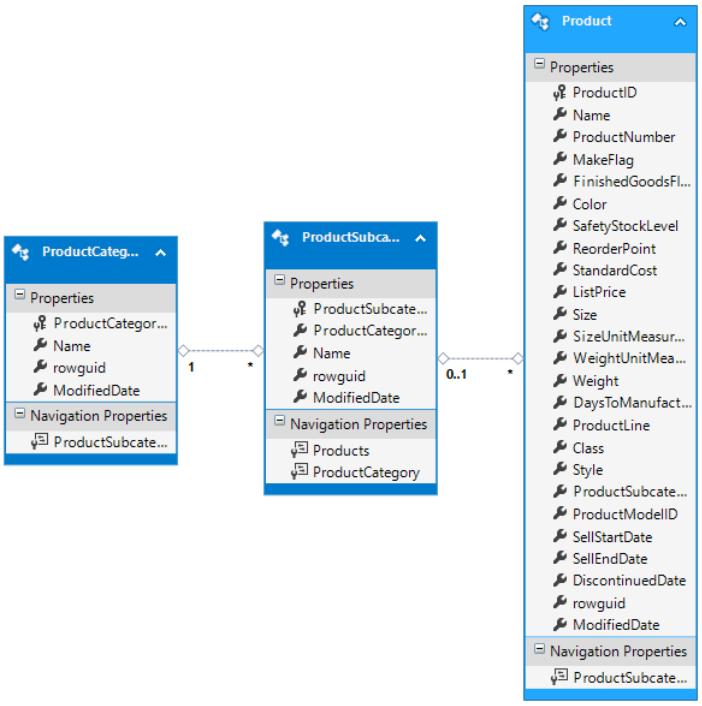



12. Note the Model Namespace. You can rename this to anything you like. The default name is the name of the database plus "Model".
13. Click Finish.
14. Click OK for the security warnings. The Entity Framework was downloaded from the web using NuGet so its files are flagged as downloaded content.



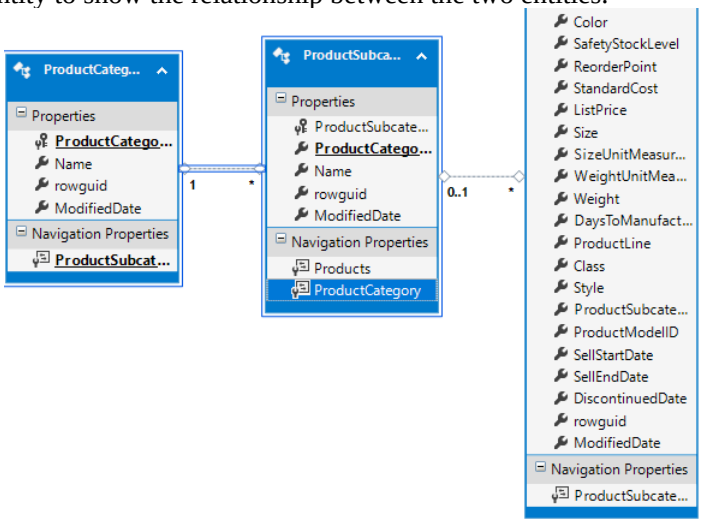

Copyright © 2019 MAX Technical Training

15. The designer will now be displayed with your “Conceptual Entity Model”.




# Exploring the Designer

1. Each box represents an Entity that has been mapped to a table in SQL Server.
2. Note the lines that indicate relationships. These were guessed from the table’s foreign keys. The foreign keys are located in the Navigation Properties section of each entity.
3. In the ProductSubcategory entity, click the ProductCategory Navigation property. Note that the relation line is highlighted along with the related Navigation Property in the ProductCategory entity to show the relationship between the two entities.



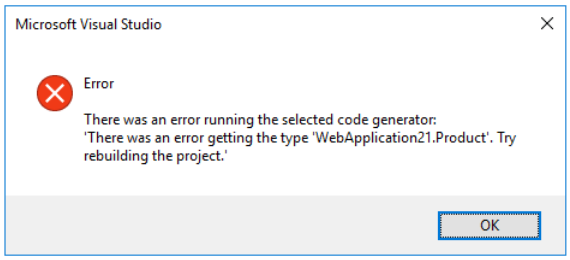

4. Click the relationship line between ProductSubCategory and Products. Note that the Navigation Properties in both entities are now underlined to show the relationship.
5. Click on each of the following and review the properties in the Properties panel: an Entity heading, a field, a relationship line and a Navigation Property.

Copyright © 2019 MAX Technical Training

At this point there is a lot of customization possible. You could rename an entity, a field or add additional entities. We will leave this design as-is for now.

# Make the Entities Available

Before writing any code that accesses the entities, you will need to build the project, otherwise you may get this error:




1. Build the project. (Ctrl+Shift+B or from the Build menu.)
2. Expand the "edmx" branch of your project. The edmx file is an XML file that defines the entities. These XML files are used to rebuild the C# files on each code build.
   - AdventureWorks.edmx
     - AdventureWorks.Context.tt
       - AdventureWorks.Context.cs
       - AdventureWorks.Designer.cs
       - AdventureWorks.edmx.diagram
     - AdventureWorks.tt
       - AdventureWorks.cs
       - Product.cs
       - ProductCategory.cs
       - ProductSubcategory.cs
3. Click the *.Context.cs file. (* = the name provided when you ran the wizard, and is by default the name of the database.)
4. Note the comment at the beginning of this file, and all of the other files in this folder. After each edit and rebuild of the model, these files will be deleted and rebuilt! Don't edit them!

```
//------------------------------------------------------------------------------
// <auto-generated>
//     This code was generated from a template.
//
//     Manual changes to this file may cause unexpected behavior in your application.
//     Manual changes to this file will be overwritten if the code is regenerated.
// </auto-generated>
//------------------------------------------------------------------------------
```

5. Note the "partial" keyword in the class definition. We can create our own partial classes to extend these auto generated classes. A rebuild will not delete or modify our partial classes. We will take advantage of this C# feature in the Models module of this course. For more about partial classes see: https://msdn.microsoft.com/en-us/library/wa80x488.aspx

```
public partial class AdventureWorks2012Entities : DbContext
{
```

Copyright © 2019 MAX Technical Training

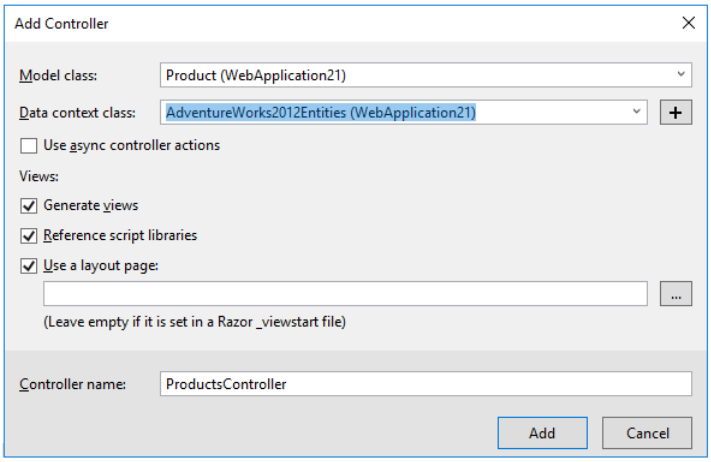
6. The *.Context.cs file defines the DbContext class for the Entity Model. Note that the class inherits from System.Data.Entity.DbContext.


7. Note that the class's constructor receives the name of the connection string stored in web.config.
8. Note that each entity is defined as a property of the DbContext of a type of DbSet<T>.

We will explore the DbContext and DbSet types in the Code First section later in this module.

# Create an MVC Controller and Views

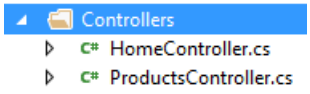
1. Right-click the Controllers folder in the Solution Explorer, click Add and Controller.
2. Click MVC 5 Controller with views, using Entity Framework and click Add.
3. Click the Model class dropdown and click Products. (Some of the classes listed here are not models!)
4. Click the Data context class dropdown and click AdventureWorks2012Entities.
5. Make sure Generate views is checked.



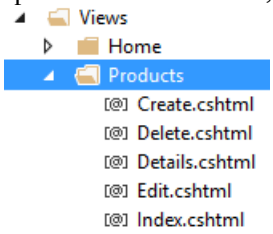

6. Note the Controller's name. "Products" in a path will by default map to a controller named ProductsController.
7. Click Add. (If you get an error, it may be because you have not built the project since the last edit in the Entity Designer.)

Copyright © 2019 MAX Technical Training

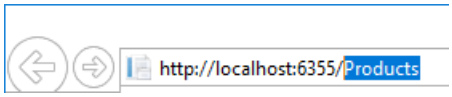
8. Expand the Controllers folder and note the new ProductsController.



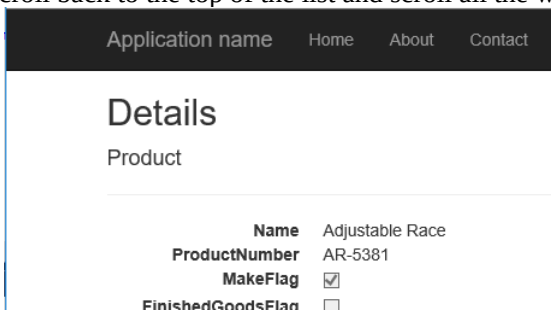

9. Expand the Views folder, expand the Products folder and note the new views.



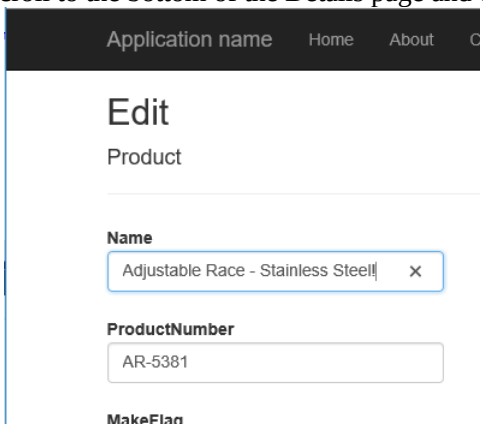

10. Click the ProductsController.cs file and review the code.
    a. Note the Actions that have been created, the attributes above them and the code inside of them.
11. Click the Index view in the Views/Products folder and note the HTML and the Razor "@" prefixed code.
12. Start your project. (Press F5 or click the Debug menu and click Start Debugging.)
13. Edit the URL of the browser and add /Products.




14. Scroll down through the list of products.
15. Scroll back to the top of the list and scroll all the way to the right and click the Details button.




16. Scroll to the bottom of the Details page and click Edit.




17. Modify the product's name and click Save.

While these views may not be very pretty, they were "free"!

You did not create:

- any SQL statements

Copyright © 2019 MAX Technical Training

- any ASP.NET SQL objects (SqlConnection, SqlCommand, SqlAdaptor, etc.)
- any C# code in the Controller (yet!)
- any pages, HTML or Razor code (yet!)

Next steps… make it pretty!

- Remove fields that your users don't need to know about. (rowguid for example)
- Create a few View Models to only display and edit what the user needs.
- Add sortable columns and paging.
- Add your company's design, logos and art.
- To add more tables to your design, display the designer (the .edmx file), right-click any empty space and click Update Model from Database. Select tables, views or stored procedures to add and click Finish.
    - Remember to rebuild the project to update the Entity Framework code.
    - Changes that impact existing Controllers or Views will require manual code changes, or the deletion and recreation of the Controller.

# Database First using "Code First from Database" - .Net Framework MVC 5

The "Database First" above creates XML files (.edmx) that describe the entities. These are used during a build to create the C# (.cs) files. I.e these code files are overwritten on each build, making it a bit hard to customize!

Note: "Code First from Database" was added with Entity Framework 6.1.

The steps to create the project, add the Entity Framework are almost identical to the above demo. The difference is after you select your database and tables. Instead of generating XML that is used to recreate the C# files after each build, the wizards create pure C# code.

Advantages over "EF Designer from database" (the original Database first tool)

- The end result is the same as "Code first", but you did not have to write the code… and the code will work the first time!
- Manual changes to the generated code files will not be overwritten from the XML design files.
- No ".edmx" files in the project.
- Consistency with design process of Entity Framework Core where there is no "Designer from database" feature.

Disadvantages:

- No visual designer to browse or customize the entity design. All changes must be made in code.

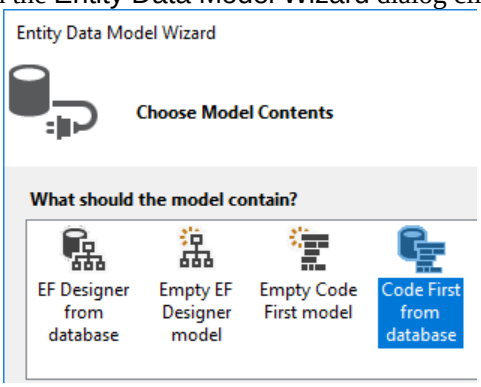
## Create an MVC Project

Use the steps from the previous demo to create the project and use Nuget to install the Entity Framework (6.1 or later).

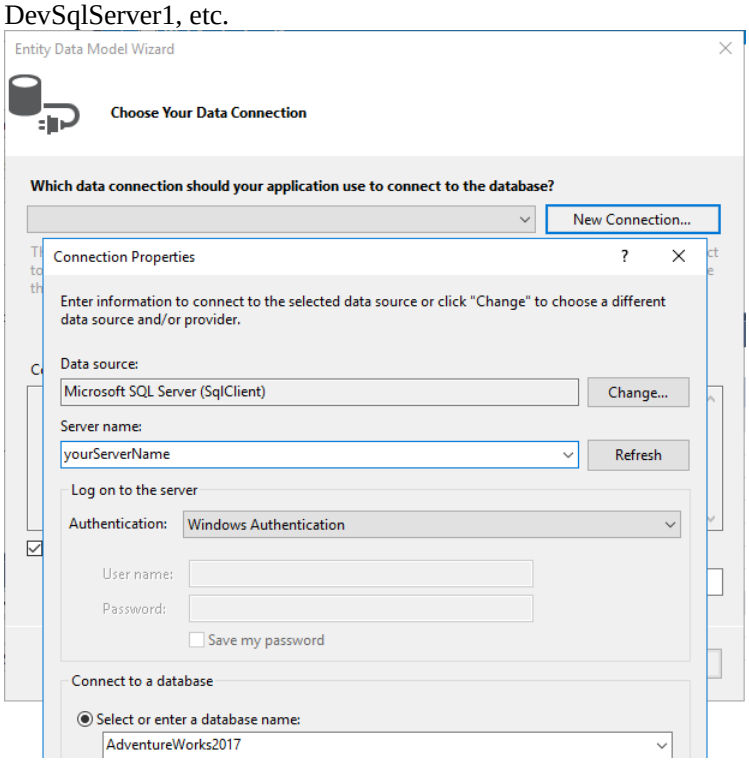
Copyright © 2019 MAX Technical Training

# Create the Data Model

1. In the Solution Explorer, right-click your Model folder, click Add, find and click ADO.NET Entity Data Model, give the model a name ("AdventureWorksDB" for this example) and click OK. (This name is will be the name of your "context".) Note, if you don't right-click the Model folder, the generated files will be placed in the root of your project, or in some other folder. You can just move these to the Model folder if you like.
2. In the Entity Data Model Wizard dialog click EF Designer from database and click Next.



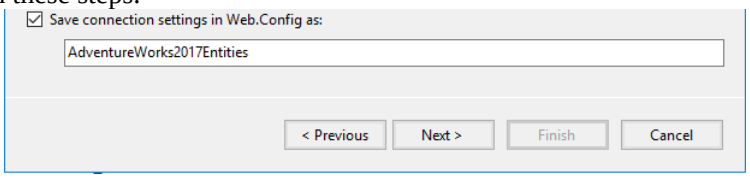

3. Click New Connection.
4. In the Server Name box type the name of your SQL Server and instance.
   Check with your instructor. Possible options for a typical local Visual Studio install:
   (localdb)\ProjectsV13
   (localdb)\MSSQLLocalDB
   DevSqlServer1, etc.



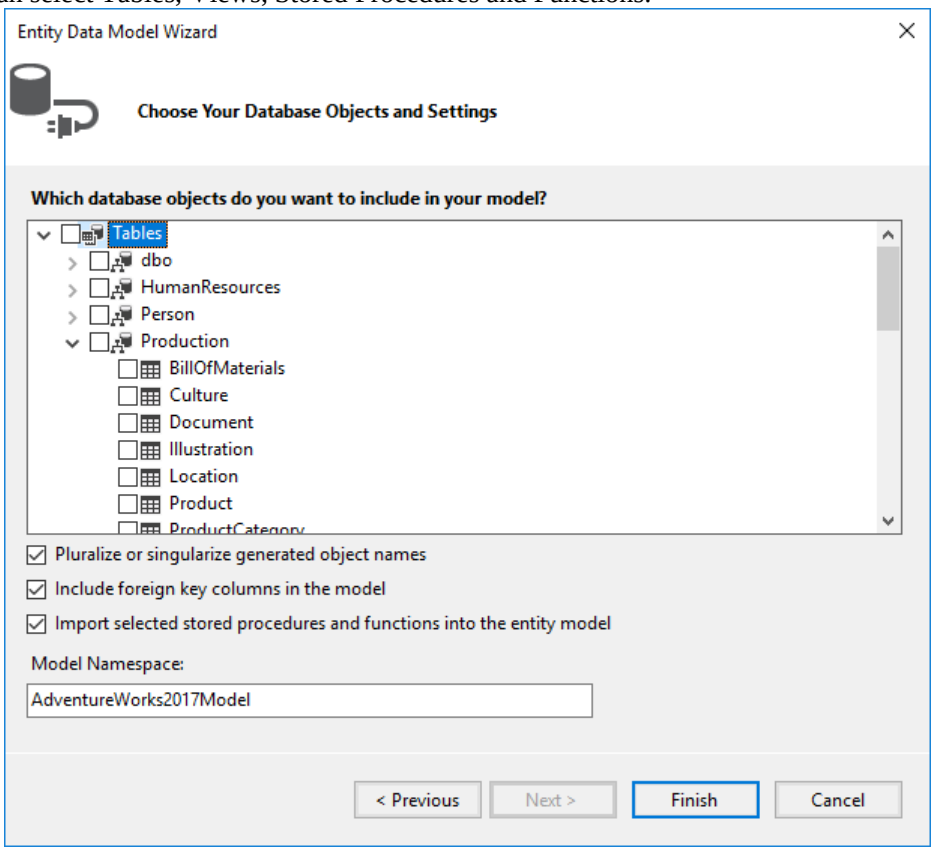

5. From the Select or enter a database name dropdown select your database and click OK.
6. At the bottom of the Entity Data Model Wizard dialog, note the option to save the connection string to your web.config file. This is a default name, you can change it. We will check for this later

Copyright © 2019 MAX Technical Training

in these steps.



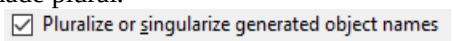

7. Click Next. We are now ready to select the database objects needed for our project. Note that you can select Tables, Views, Stored Procedures and Functions.



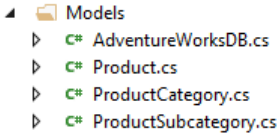

8. For this demo, checkmark the following tables:
   Product
   ProductCategory
   ProductSubcategory
9. Expand Views and note that SQL views can also be added as entities. They may be updateable, partially updatable or not updateable depending on their design.
10. Expand Stored Procedures and Functions and note that these also can be added as methods to the entity model. Stored procedures will appear as a new method on the DbContext:

```
db.uspUpdateEmployeeHireInfo(123, "Manager",DateTime.Now,DateTime.Now,12345,1,true);
```

11. Note the first check box. When checked, entity names are made singular and Entity Set names are made plural.




12. Note the Model Namespace. You can rename this to anything you like. The default name is the name of the database plus "Model".
13. Click Finish.
14. Expand the Models folder and explore the created files.




Copyright © 2019 MAX Technical Training

## Create an MVC Controller and Views

Follow the steps from the previous demo. There are no changes in the workflow.

# Database First - .Net Core 2.x MVC

.Net Core has two major changes that impact the Database First process:

- The .Net Core MVC template pre-loads the Entity Framework Core and Dependencies. (No need to install EF.)
- There is no wizard to automate the process. You instead use a PowerShell cmdlet.

## Create an MVC project

Remember MVC is not required for the Entity Framework and the Entity Framework is not required for MVC. They just work well together! And, in the case of the .Net Core templates, already built-in!

1. Launch Visual Studio 2017.
2. Click New, Project.
3. Expand the templates to Visual C# and Web.
4. Click ASP.NET Web Application (.NET Framework).

## Create the Data Model

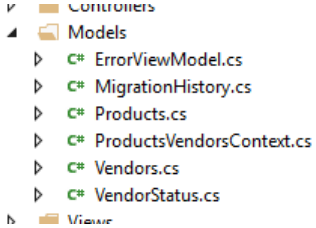
1. PowerShell is used to build the EF classes from your existing database design.

   ```
   Scaffold-DbContext "YourConnectionString" Microsoft.EntityFrameworkCore.SqlServer
   -OutputDir Models
   ```

You can add -Tables to specify selected tables.

```
Scaffold-DbContext "YourConnectionString" Microsoft.EntityFrameworkCore.SqlServer
-Tables Products, -OutputDir Models
```

2. Review the created files. Notice that support for migrations has been enabled.




Notes:

- For more see: Getting Started with EF Core on ASP.NET Core with an Existing Database https://docs.microsoft.com/en-us/ef/core/get-started/aspnetcore/existing-db

## Create an MVC Controller and Views

Follow the steps from the “Database First - .Net Framework MVC 5”. There are no changes in the workflow.

Copyright © 2019 MAX Technical Training

# Model First - .Net Framework Only

Model First is used when you are creating a new database and would like to use the graphical user interface to "draw" the database design and to set the entity and field properties. The EF Wizards then create the SQL statements needed to build the database tables.

## Create an MVC project

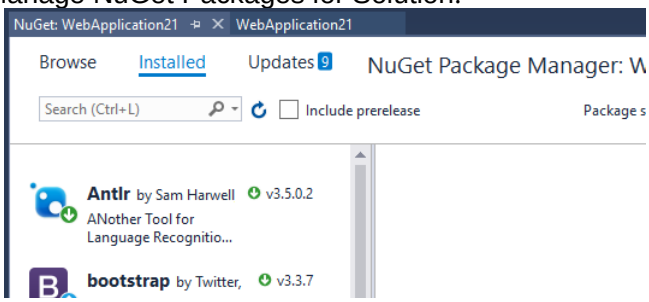
Remember MVC is not required for the Entity Framework and the Entity Framework is not required for MVC. They just work well together!

Note: The steps through the install of the Entity Framework are identical to the Database First example above.

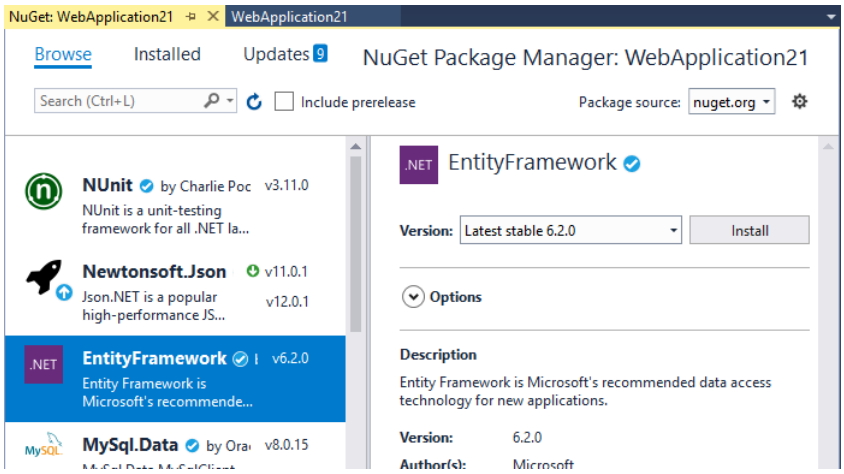
1. Launch Visual Studio 2015.
2. Click New, Project.
3. Expand the templates to Visual C#, Web.
4. Click ASP.NET Web Application.
5. Name the project and click OK.
6. In the New ASP.NET Project dialog box click Change Authentication, select No Authentication and OK.
7. Click OK.

## Use NuGet to download and install the Entity Framework

1. Right-click References in the Solution Explorer pane, or click Tools, NuGet Package Manager, Manage NuGet Packages for Solution.

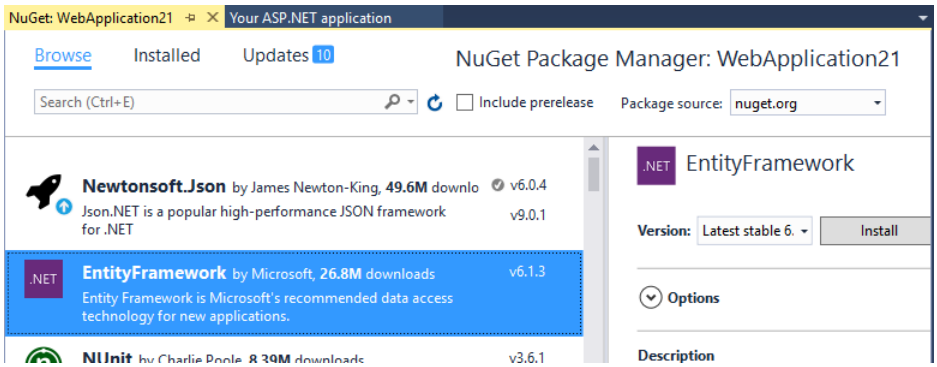



2. Click the Browse tab.

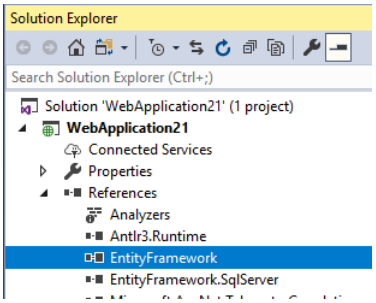



Copyright © 2019 MAX Technical Training

3. If displayed, click the EntityFramework package. Otherwise, search for the Entity Framework. (If you see both Core and EF6, choose EF6.)
4. Click the Install button.




5. Click OK and accept the license agreement to complete the install.
6. Review the list of references and note that the EntityFramework assembly has been installed.

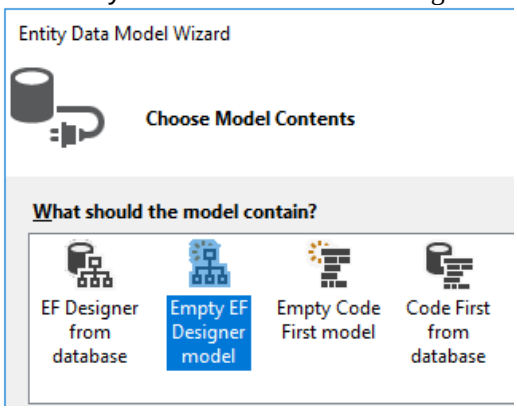



# Create the Data Model

1. In the Solution Explorer, right-click your project, click Add, New Item and then in the Visual C#, Data templates click ADO.NET Entity Data Model, give the model a name ("AdventureWorks" for this example) and click Add.

   or

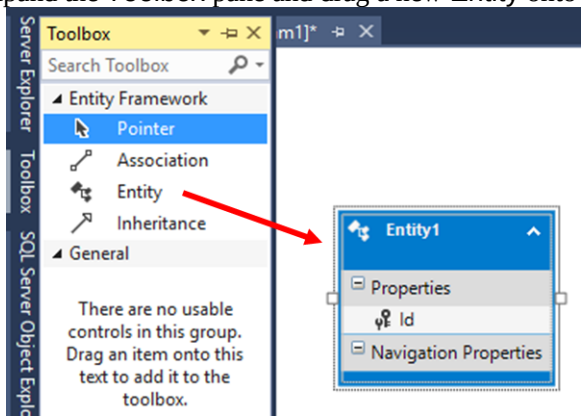
   In the Solution Explorer, right-click your project, click Add, click ADO.NET Entity Data Model, give the model a name ("AdventureWorks" for this example) and click OK.
2. In the Entity Data Model Wizard dialog click Empty EF Designer model and click Finish.




3. Either:

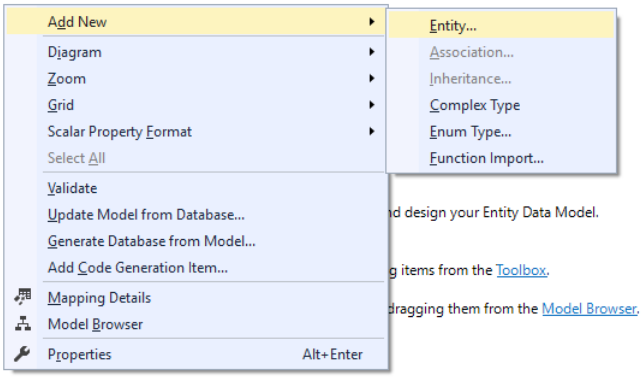
Copyright © 2019 MAX Technical Training

a. Expand the Toolbox pane and drag a new Entity onto the designer surface.




or

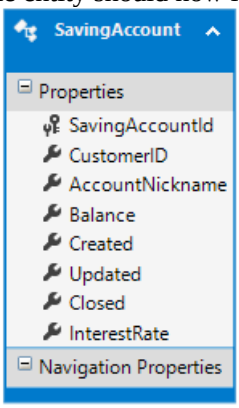
b. Right-click the designer surface, click Add New and select Entity. Enter a name and click OK.



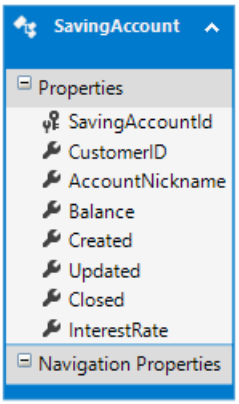

4. Right-click the entity and then you can:
   a. Rename the entity.
   b. Click a property and rename it.
   c. Right-click the entity, click properties and edit the entity's properties.
   d. Right-click the entity, click Add New and add new scalar, navigation or complex properties.
5. Make the following changes to change the current default entity into a SavingAccount entity.
   a. Entity name: SavingAccount
   b. Scalar Properties:
      i. int SavingAccountID
      ii. int32 CustomerID
      iii. string AccountNickname
      iv. double Balance
      v. DateTime Created
      vi. DateTime Updated
      vii. DateTime Closed
      viii. double InterestRate

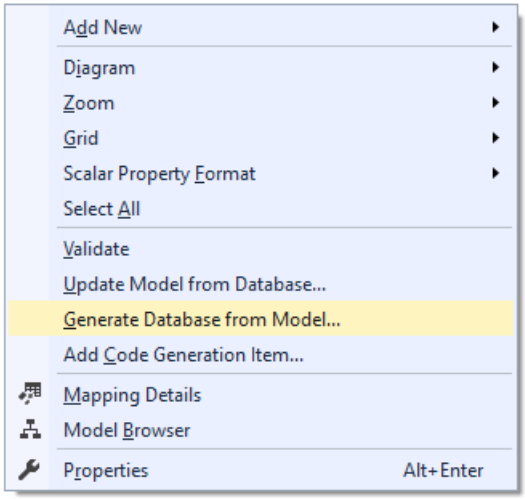
Copyright © 2019 MAX Technical Training

6. The entity should now look like this:

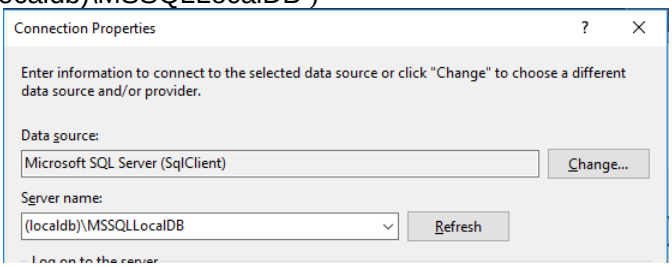



7. Right-click the Designer surface and click Generate Database from Model.

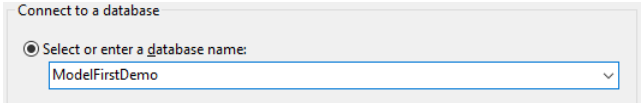


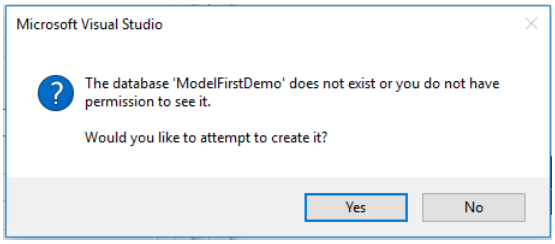

8. Enter the SQL Server Name. (If using a local SQL server then probably: (localdb)\MSSQLLocalDB )




9. Enter a database name such as "ModelFirstDemo".



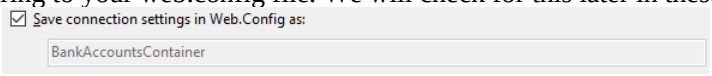

10. Click OK.




11. Click Yes to create the new database. (It won't actually be created until step 18.)

Copyright © 2019 MAX Technical Training

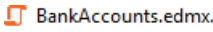
12. At the bottom of the Generate Database Wizard dialog, note the option to save the connection string to your web.config file. We will check for this later in these steps.



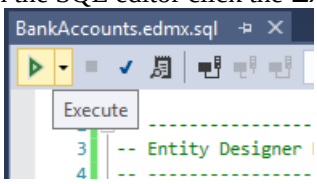

13. Click Next.
14. Note the generated SQL code that will be used to create the database. It will be saved to the Models folder (BankAccounts.edmx.sql).Note the "NOT NULL" text which makes these required fields. This is the default. You could have set these properties while adding/editing the fields.

```
-- Creating table 'SavingAccounts'
CREATE TABLE [dbo].[SavingAccounts] (
    [SavingAccountId] int IDENTITY(1,1) NOT NULL,
    [CustomerID] int  NOT NULL,
    [AccountNickname] nvarchar(max)  NOT NULL,
    [Balance] float  NOT NULL,
    [Created] datetime  NOT NULL,
    [Updated] datetime  NOT NULL,
    [Closed] datetime  NOT NULL,
    [InterestRate] float  NOT NULL
);
```

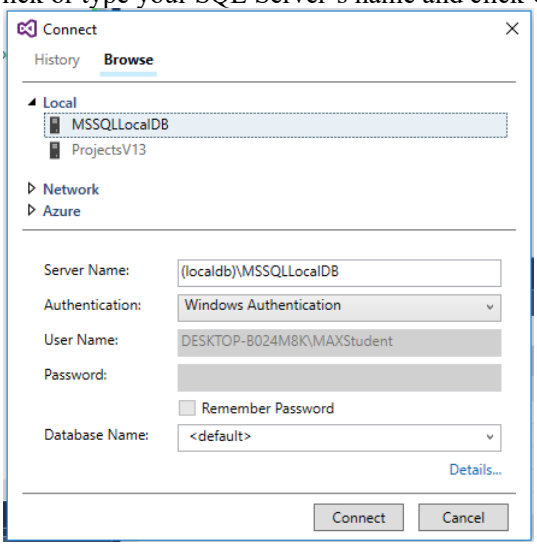
15. Click Finish and OK through any security warnings.
16. Expand the Models folder and click the SQL file.




17. In the SQL editor click the Execute button.




18. Click or type your SQL Server's name and click Connect.



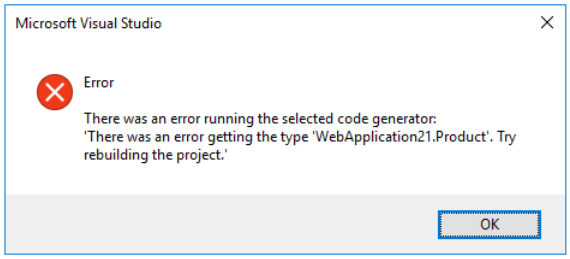

***WARNING! Running this SQL script will drop the existing tables and create new empty tables!***

# Make the Entities Available

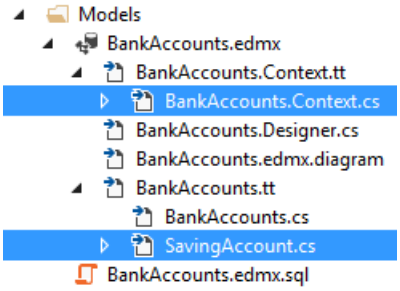
Note: The steps that follow are nearly identical to the steps in Database First example.

Before writing any code that accesses the entities, you will need to build the project, otherwise you get may this error:

Copyright © 2019 MAX Technical Training




1. Build the project. (Ctrl+Shift+B or from the Build menu.) Click OK in any security warning popups.
2. Expand the "Models" branch of your project. The edmx file is an XML that defines the entities which is displayed as the model designer surface.




3. Click the *.Context.cs file. (* = the name provided when you ran the wizard, and is by default the name of the database.
4. Note the comment at the beginning of this file, and all of the other files in this folder. After each edit and rebuild of the model, these files will be deleted and rebuilt! Don't edit them!

```
//------------------------------------------------------------------------------
// <auto-generated>
//     This code was generated from a template.
//
//     Manual changes to this file may cause unexpected behavior in your application.
//     Manual changes to this file will be overwritten if the code is regenerated.
// </auto-generated>
//------------------------------------------------------------------------------
```

5. Note the "partial" keyword in the class definition. We can create our own partial classes to extend these auto generated classes. A rebuild will not delete or modify our partial classes. We will take advantage of this C# feature in the Models module of this course. For more about partial classes see: https://msdn.microsoft.com/en-us/library/wa80x488.aspx

```
public partial class AdventureWorks2012Entities : DbContext
{
```

6. The *.Context.cs file defines the DbContext class for the Entity Model. Note that the class inherits from System.Data.Entity.DbContext.

**Web.config**

```
    <connectionStrings>
      <add name="BankAccountsContainer" connectionString="metadata=res://*/Models.BankAccounts.cs
    </connectionStrings>
</configuration>
```

***.Context.cs**

```
public partial class BankAccountsContainer : DbContext
{
    public BankAccountsContainer()
        : base("name=BankAccountsContainer")
    {
    }

    protected override void OnModelCreating(DbModelBuilder modelBuilder)
    {
        throw new UnintentionalCodeFirstException();
    }

    public virtual DbSet<SavingAccount> SavingAccounts { get; set; }
}
```

This is our connection string's name in web.config

These are the entities.

Singular Plural

Copyright © 2019 MAX Technical Training

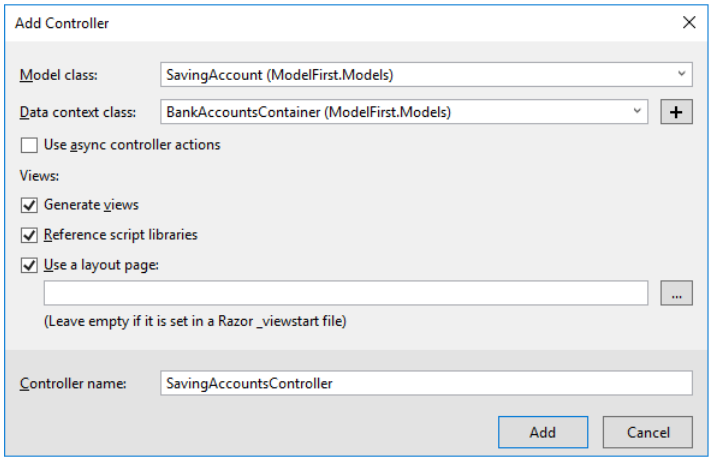
7. Note that the class's constructor receives the name of the connection string stored in web.config.
8. Note that each entity is defined as a property of the DbContext of a type of DbSet<T>.

We will explore the DbContext and DbSet types in the Code First section later in this module.

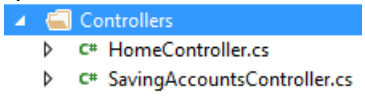
# Create an MVC Controller and Views

The following steps are almost identical to those found in the Database First demo.

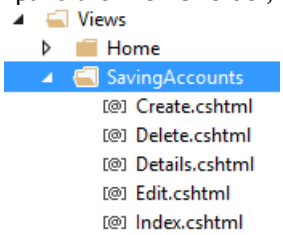
1. Right-click the Controllers folder in the Solution Explorer, click Add and Controller.
2. Click MVC 5 Controller with views, using Entity Framework and click Add.
3. Click the Model class dropdown and click SavingAccount. (Some of the classes listed here are not models!)
4. Click the Data context class dropdown and click BankAccountsContainer.
5. Make sure Generate views is checked.



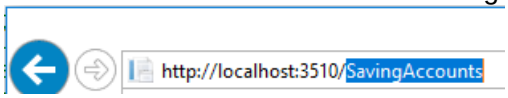

6. Note the Controller name. The "SavingAccounts" in a URL will by default map to a controller named SavingAccountsController.
7. Click Add. (If you get an error, build the project and try again.)
8. Expand the Controllers folder and note the new SavingAccountsController.




9. Expand the Views folder, expand the SavingAccounts folder and note the new views.

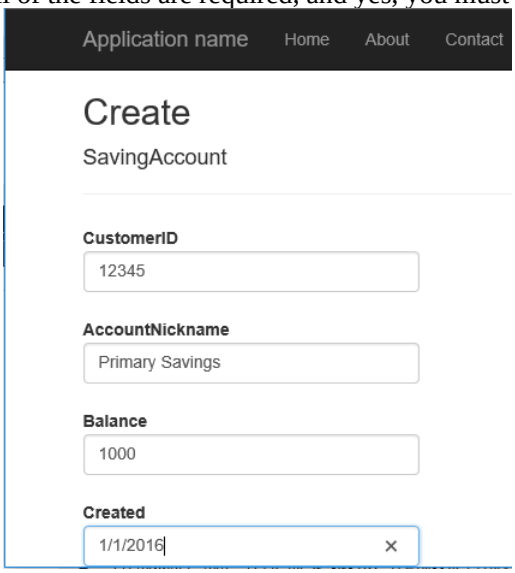



10. Click the SavingAccountsController.cs file and review the code.
    a. Note the Actions that have been created, the attributes above them and the code inside of them.
11. Click the Index view in the Views/SavingAccounts folder and note the HTML and the Razor "@" prefixed code.
12. Start your project. (Press F5 or click the Debug menu and click Start Debugging.)
13. Edit the URL of the browser and add /SavingAccounts.




Copyright © 2019 MAX Technical Training

14. Click Create New.
15. Enter some sample data to create a new savings account. As we did not configure them otherwise, all of the fields are required, and yes, you must enter a Closed date!




16. Click Create and return to the index page.
17. Click the Details button for the new account.
18. Scroll to the bottom of the Details page and click Edit.
19. Modify the account's nickname click Save.

While the views may not be very pretty, they were "free"!

You did not create:

- any SQL statements
- any ASP.NET SQL objects (SqlConnection, SqlCommand, SqlAdaptor, etc.)
- any C# code in the Controller (yet!)
- any pages, HTML or Razor code (yet!)

Next steps… make it pretty!

- Create a few View Models to only display and edit what the user needs.
- Add sortable columns and paging.
- Add your company's design, logos and art.

Also see:

- Entity Framework Model First
  https://msdn.microsoft.com/en-us/library/jj205424(v=vs.113).aspx

# Code First - .Net Framework MVC 5

In Code First Entity Framework projects you create simple classes. There's no graphical designer. After creating your entity classes, Customer, Product, etc., you create a context class and build the project. The Entity Framework can then create your database and tables. You can configure the project to not automatically build the database and tables, or even configure it to automatically populate the database with testing data.

Copyright © 2019 MAX Technical Training

# Create an MVC project

Remember MVC is not required for the Entity Framework and the Entity Framework is not required for MVC. They just work well together!

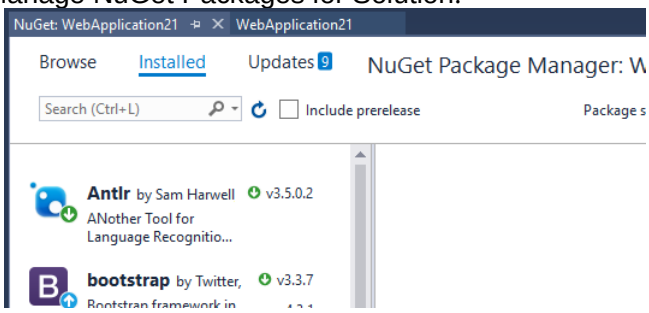
Note: The steps through the install of the Entity Framework are identical to the Database First example above.

1. Launch Visual Studio 2017.
2. Click New, Project.
3. Expand the templates to Visual C#, Web.
4. Click ASP.NET Web Application (.NET Framework) or ASP.NET Core Web Application.
5. Name the project and click OK.
6. In the New ASP.NET Project dialog box click Change Authentication, select No Authentication and OK.
7. Click OK.

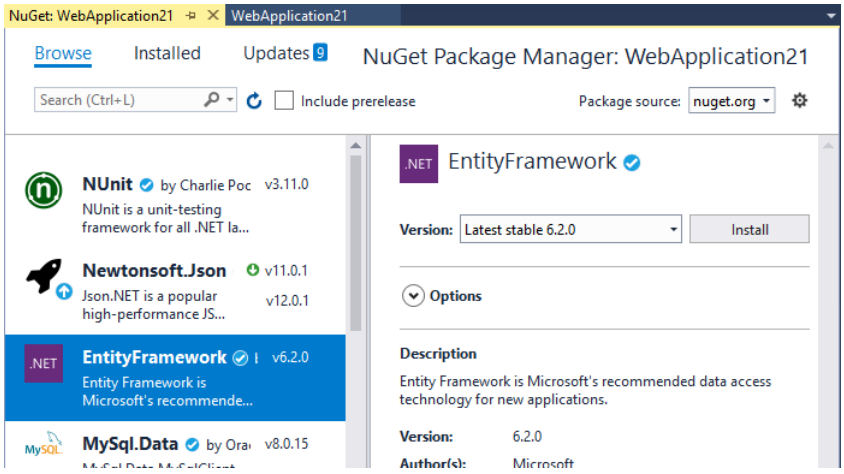
# Use NuGet to download and install the Entity Framework (.Net Framework only)

The Entity Frame is included in the Visual Studio .Net Core templates. The following is only need for .Net Framework.

1. Right-click References in the Solution Explorer pane, or click Tools, NuGet Package Manager, Manage NuGet Packages for Solution.

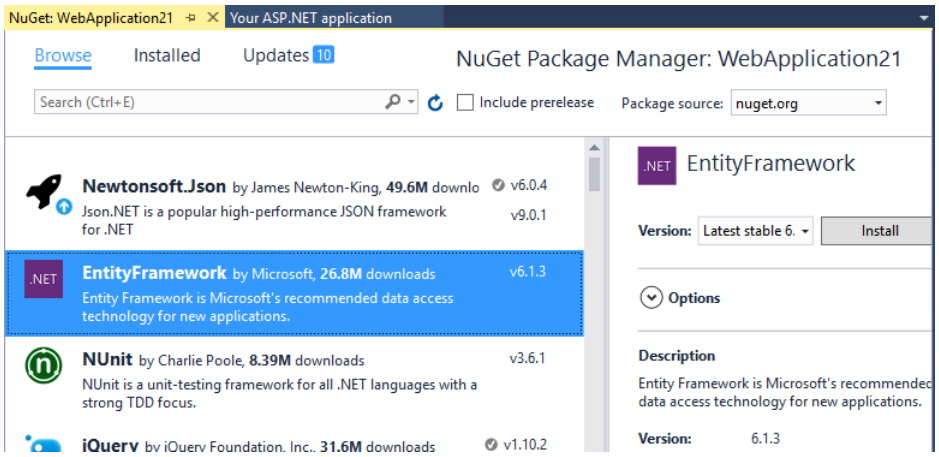



2. Click the Browse tab.




Copyright © 2019 MAX Technical Training

3. If displayed, click the EntityFramework package. Otherwise, search for the Entity Framework. (At the time of this writing, Entity Framework Core was not available in NuGet. If you see both Core and EF6, choose EF6.)
4. Click the Install button.




5. Click OK and accept the license agreement to complete the install.
6. Review the list of references and note that the EntityFramework assembly has been installed.

# Create the Model Classes

The entity and context classes are plain C# classes. You will be manually creating two or more classes. One class represents the context, or the database, and will inherit from System.Data.Entity.DbContext. One or more classes will represent your entities. These are either plain classes or ones that inherit from one of your own base classes.

When you create your classes, you can:

- Create one file for the context class and a separate file for each entity class.
- Create one file for the context class and one file with all of the entity classes.
- Create one file with the context class and all of the entity classes.

I.e. these are just classes and are managed like ordinary classes.

1. Right-click the Models folder, click Add, Class, enter a name for your class file and click Add. The name typically represents a single item, so is singular. For this demo, name the class file "Customer.cs". The new file will define a namespace and the class. Both can be renamed if desired.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;

namespace CodeFirstEFDemo.Models
{
    public class Customer
    {
    }
}
```

2. As you will typically be using attributes to define field properties like primary key, required, etc. you can add the using statement for annotations now.

```
using System.ComponentModel.DataAnnotations;
```

You may also need one more using statement to support additional attributes like

Copyright © 2019 MAX Technical Training

"[Column("name")]".

```
using System.ComponentModel.DataAnnotations.Schema;
```

3. Add the primary key field(s). You can hand type these or use the "prop" snippet.
   a. Inside of the Customer class type "prop" and press tab twice.

```
public class Customer
{
    public int MyProperty { get; set; }
}
```

   b. Press tab to jump to the property name highlight and type CustomerID.

   Note: Using Auto-magic / convention, any field named ID or Id, or any field named with the class name followed ID or Id will be used as the primary key. (I.e. ID, Id, CustomerID or CustomerId) You can use any name for the key if you add the [Key] attribute to the property.

```
[Key]
public int CustNum { get; set; }
```

4. Add the following properties to complete the Customer class:

```
public string FirstName { get; set; }
public string LastName { get; set; }
public string Manager { get; set; }
```

5. Repeat the above steps to add a CheckingAccount class:

```
public class CheckingAccount
{
    public int CheckingAccountID { get; set; }
    public int CustomerID { get; set; }
    public string AccountNickname { get; set; }
    public int Balance { get; set; }
    public DateTime Created { get; set; }
    public DateTime Updated { get; set; }
    public DateTime? Closed { get; set; }
    public int LastCheckNumberOrdered { get; set; }
}
```

   Note the "?" after the DateTime data type for the Closed property to make it nullable.

6. Define the relationship between the entities. This example will create a one (Customer) to many (CheckingAccounts) relationship and will create Foreign Keys in the SQL database..
   a. At the end of the Customer class add this line:

```
public virtual List<CheckingAccount> CheckingAccounts { get; set; }
```

   b. At the end of the CheckingAccounts class add this line:

```
public virtual Customer Customer { get; set; }
```

7. Create the context class. While the name will often include the word "context", this is not a requirement.
   a. Right-click the Models folder, click Add, Class, enter a name for your class file and click Add. (example: BankDbContext.cs)

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;

namespace CodeFirstEFDemo.Models
{
    public class BankDbContext
    {
    }
}
```

Copyright © 2019 MAX Technical Training

8. As this class will inherit from System.Data.Entity.DbContext, add a using statement for System.Data.Entity, and after the class name add ": DbContext".

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Data.Entity;

namespace CodeFirstEFDemo.Models
{
    public class BankDbContext : DbContext
    {
    }
}
```

9. Inside of the class we need to create two properties of type DbSet<T>, one for each entity. Notice the convention of using singular class names and plural DbContext property names.
10. In the context class type "prop" and press tab twice. Replace the int datatype with DbSet<Customer> and name the property Customers. Repeat for CheckingAccounts.

```
public class BankDbContext : DbContext
{
    public DbSet<Customer> Customers { get; set; }
    public DbSet<CheckingAccount> CheckingAccounts { get; set; }
}
```

11. For .Net Core…
    a. Add constructors to the context class.

```
public class BankDbContext : DbContext
{
    public DbSet<Customer> Customers { get; set; }
    public DbSet<CheckingAccount> CheckingAccounts { get; set; }

    public BankDbContext()
    {  }

    public BankDbContext(DbContextOptions<BankDbContext> options)
        : base(options)
    {  }
}
```

    b. Add a connection string to appsettings.json.

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Warning"
    }
  },
  "AllowedHosts": "*",
  "ConnectionStrings": {
    "BankDb": "Data Source=(localdb)\\MSSQLLocalDB;Initial Catalog=BankDatabase;Trusted_Connection=True;"
  }
}
```

    c. Load the connection string from Startup.cs. Just after the services.AddMvc line add a "services.AddDbContext" line.

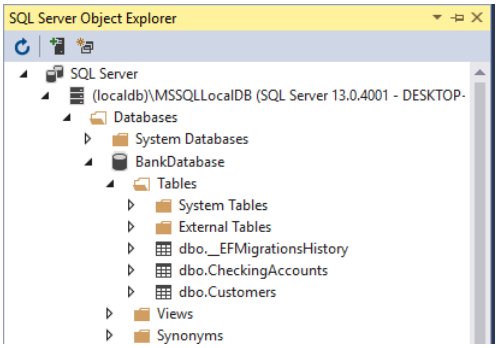
```
services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version_2_2);

services.AddDbContext<BankDbContext>(options =>
        options.UseSqlServer(Configuration.GetConnectionString("BankDb")));
```

    The "BankDb" name needs to match the name of the connection string in the appsettings.json file.

    d. Build your project and check for code errors.
    e. Configure migrations, if you want support for database migrations or if you want EF to build your database for you.
        i. Navigate to: Tools > NuGet Package Manager > Package Manager Console
        ii. Type Add-Migration InitialCreate and press Enter.
        iii. Type Update-Database and press Enter

Copyright © 2019 MAX Technical Training

f. Open the SQL Server Object Explorer view (or your favoriate SQL Server tool) and verify the tables have been created.

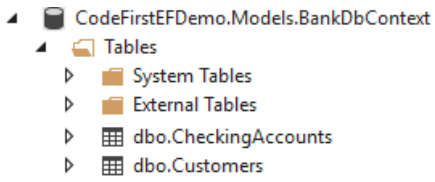



All done! At least with the model design. The rest is up to the Entity Framework. On the first run of the project, and on the first creation of the context, the Entity Framework will create the database and tables. (See the Auto-magic notes at the end of this demo for details.)

In order to use this data model you will use the following pattern:

- Build the project.
- In any class where you will be using the data:
  - Add a using statement for the Models: using CodeFirstEFDemo.Models;
  - Create an instance of the context class: BankDbContext db = new BankDbContext();
  - Perform some action against the context: int count = db.Customers.Count();
- Run the project.

When working with EF Core, the database tables are created after running the PowerShell migration code. For EF 6, the tables will be created after the first time code accesses the EF objects (typically in a Controller).




Tip! You can add calculated columns to the model by creating a property with the calculation and marking it as "NotMapped". This will prevent the property from being added to the database table. The wizards for adding views will not automatically add this property to your views. You will need to add it manually.

```
public string FirstName { get; set; }
public string LastName { get; set; }

[NotMapped]
public string FullName
{
    get { return this.FirstName + " " + this.LastName; }
}
```

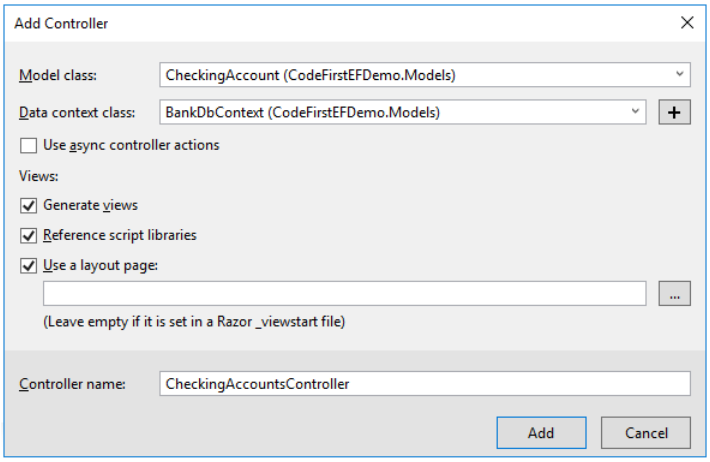
# Create an MVC Controller and Views

The following steps are almost identical to those found in the Database First demo.

1. Right-click the Controllers folder in the Solution Explorer, click Add and Controller.
2. Click MVC (or MVC 5) Controller with views, using Entity Framework and click Add.

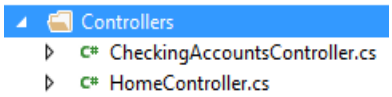
Copyright © 2019 MAX Technical Training

3. Click the Model class dropdown and click CheckingAccount. (Some of the classes listed here are not models!)
4. Click the Data context class dropdown and click BankDbContext.
5. Make sure Generate views is checked.

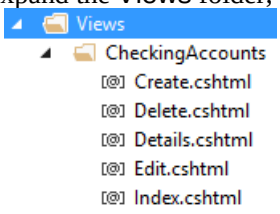



Note: EF Core will not have the "Use async" option as it's templates default to using async code.

6. Note the Controller's name. The "CheckingAccounts" in a URL will by default map to a controller named CheckingAccountsController.
7. Click Add. (If you get an error, build the project and try again.)
8. Expand the Controllers folder and note the new CheckingAccountsController.



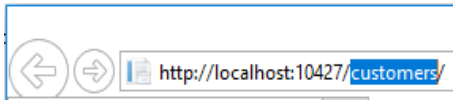

9. Expand the Views folder, expand the CheckingAccounts folder and note the new views.




10. Click the CheckingAccountsController.cs file and review the code.
    a. Note the Actions that have been created, the attributes above them and the code inside of them.
11. Click the Index view in the Views/CheckingAccounts folder and note the HTML and the Razor "@" prefixed code.

12. Repeat the steps above and create the Customer controller and views.

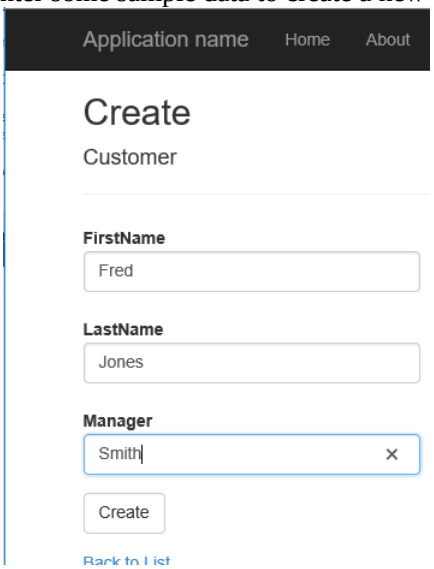
13. Start your project. (Press F5 or click the Debug menu and click Start Debugging.)
14. Edit the URL of the browser and add /Customers/CheckingAccounts.




15. Click Create New.

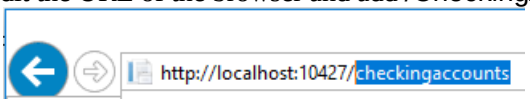
Copyright © 2019 MAX Technical Training

16. Enter some sample data to create a new customer.

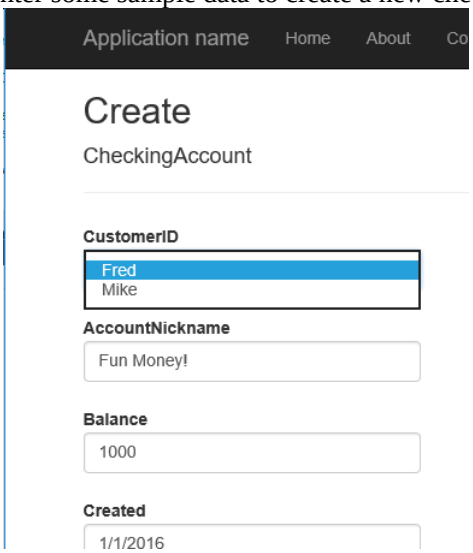



17. Click Create and return to the index page.
18. Add a second customer!

19. Edit the URL of the browser and add /CheckingAccounts.




20. Click Create New.
21. Enter some sample data to create a new checking account for one of your customers.




Copyright © 2019 MAX Technical Training

22. Notice the dropdown list for the customers! The "MVC 5 Controller with views" wizard was able to guess this from the relations we added earlier.

```
public class CheckingAccount
{
    public int CheckingAccountID { get; set; }
    public int CustomerID { get; set; }
    public string AccountNickname { get; set; }
    public int Balance { get; set; }
    public DateTime Created { get; set; }
    public DateTime Updated { get; set; }
    public DateTime Closed { get; set; }
    public int LastCheckNumberOrdered { get; set; }

    public virtual Customer Customer { get; set; }
}

public class Customer
{
    public int CustomerID { get; set; }
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public string Manager { get; set; }

    public virtual List<CheckingAccount> CheckingAccounts { get; set; }
}
```

23. Click Create and return to the index page.

While the views may not be very pretty, they were "free"!

You did not create:

- any SQL statements
- any ASP.NET SQL objects (SqlConnection, SqlCommand, SqlAdaptor, etc.)
- any C# code in the Controller (yet!)
- any pages, HTML or Razor code (yet!)

Next steps… make it pretty!

- Create a few View Models to only display and edit what the user needs.
- Add sortable columns and paging.
- Add your company's design, logos and art.
- Add links from the Customer View to a list of their checking accounts.

Also see:

- Entity Framework Code First to a New Database
  https://msdn.microsoft.com/en-us/library/jj193542(v=vs.113).aspx

# Auto-Magic at Work!

When you pressed F5 and started your project, magic happened!

- Where did your database get created? (For .Net Framework when no connection string supplied)
- What will your database be named?
- What will your tables be named?
- What SQL data types will be used?

Copyright © 2019 MAX Technical Training

**Well… it depends…**

- **Where did your database get created?** (Conventions!) (For .Net Framework only)
  - **If the DbContext constructor has a string that matches a <connectionStrings> entry** in your web.config, it will create the database as defined there. (SQL Server name and database name).
    Constructor:

```
public class ProductVendorDbContext : DbContext
{
    public ProductVendorDbContext() : base("ProductVendorsConnection")  {  }

    public DbSet<Product> Products { get; set; }
    public DbSet<Vendor> Vendors { get; set; }
}
```

    Connection string:

```
</appSettings>
<connectionStrings>
  <add name="ProductVendorsConnection"
       providerName="System.Data.SqlClient"
       connectionString="Server=(localdb)\MSSQLLocalDB;
       Database=ProductsVendors; Integrated Security=true"/>
</connectionStrings>
```

    If you want to set a location then add an "AttachDbFileName" parameter to the string. Enter the full path "C:\...\databasename" or use the "|DataDirectory|" placeholder to point to the "App_Data" folder of your project.

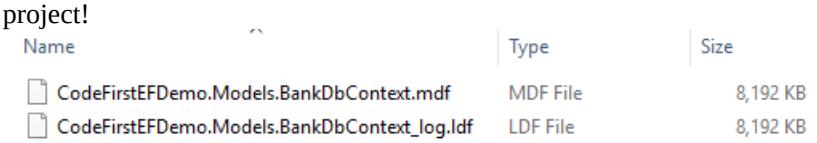
```
<connectionStrings>
  <add name="ProductVendorsConnection"
       providerName="System.Data.SqlClient"
       connectionString="Server=(localdb)\MSSQLLocalDB;
       Database=ProductsVendors; Integrated Security=true;
       AttachDbFileName=|DataDirectory|\mydata.mdf"/>
</connectionStrings>
```

  - **If the DbContext constructor has a string that DOES NOT match a <connectionStrings> entry** in your web.config, it will create a database using the string value in the default server. (See next…)

  - **If the DbContext constructor does not specify a string**, then
    - When the Entity Framework was installed using the NuGet package a check was made to find if there is a local database available. The first check is to see if SQL Express is running, and if so it is used, otherwise LocalDb is used. The web.config was then modified to set a default database.

```
<entityFramework>
  <defaultConnectionFactory type="System.Data.Entity.Infrastructure.LocalDbConnectior
    <parameters>
      <parameter value="mssqllocaldb" />
    </parameters>
  </defaultConnectionFactory>
```

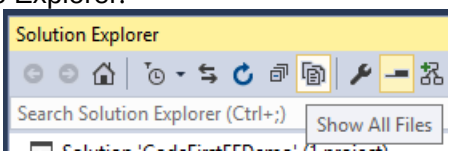
    - The default name will be based on the namespace and DbContext name. Example: CodeFirstEFDemo.Models.BankDbContext
    - The default location for the database files will be the App_Data folder of your project!




    To see the files, click Show All Files in the Solution Explorer and then expand the App_Data folder, or right-click the App_Data folder and click Open Folder in

Copyright © 2019 MAX Technical Training

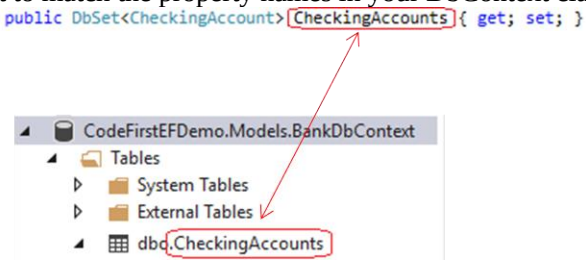
File Explorer.




or

  - The database will not be created until the DbContext is first accessed.

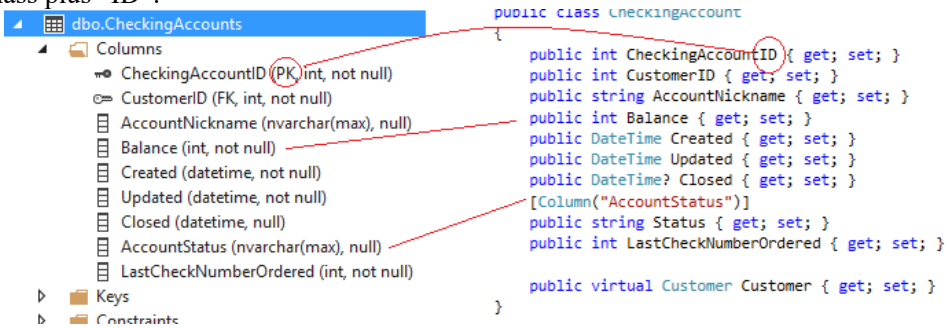
- **Table Names?**
  - Set to match the property names in your DbContext class, not the entity class names.




  - If you want to have different table and class names, just add an attribute to the entity's class. The following will create a table named Production.ProdTable.

```
[Table("ProdTable", Schema = "Production")]
public class Product
{
   public int ProductID { get; set; }
```

- **SQL column names?**
  - Set to match either the entity class property names or the name specified in an attribute. The primary key was also set by "convention" for the column with the same name as the class plus "ID".




# Notes for Code First

- To make non-nullable datatype nullable in C#, add a "?" to the type.
  public int Balance { get; set; }  // creates a SQL field that is not-nullable (i.e. "required").
  public int? Balance { get; set; } // creates a SQL field is it nullable
- To make a nullable datatype a required field add the Required attribute.
  [Required]
  public string FirstName { get; set; }
- If your database uses stored procedures for insert, update and delete then see:
  https://docs.microsoft.com/en-us/ef/ef6/modeling/code-first/fluent/cud-stored-procedures

Copyright © 2019 MAX Technical Training

# Code First Migrations

The Entity Framework can automate the migration of model changes from Code First Model classes to the SQL databases without the need to drop and recreate the databases.

See:

- Code First migrations:
  https://docs.microsoft.com/en-us/ef/ef6/modeling/code-first/migrations/index
- Code First Migrations and Deployment with the Entity Framework in an ASP.NET MVC Application
  https://docs.microsoft.com/en-us/aspnet/mvc/overview/getting-started/getting-started-with-ef-using-mvc/migrations-and-deployment-with-the-entity-framework-in-an-asp-net-mvc-application

# A More Complete Demo

You can download Microsoft's Contoso University sample Code First project and explore a complete MVC project that uses Entity Framework Code First.

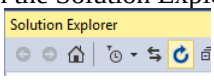
Notes:

- This project is missing controllers and views for Enrollment and Office Assignment. (You can add them!)
- This project includes:
  - A demo of the use of a Data Access Layer (the DAL folder).
  - An example Fluent API. (In the SchoolContext.cs file.)
  - Sort and paging in Views. (See StudentController.cs and /Views/Students/Index.cshtml.)
  - Numerous of Model class attributes.
  - Multiple examples of Entity relations. (See the model files and /Views/Students/Details.)

**Steps to download and setup:**

This project was designed for SQL Express and will need a tweak or two to work with MSSQLLocalDB.

1. Open a browser and navigate to https://code.msdn.microsoft.com/ASPNET-MVC-Application-b01a9fe8
2. Click the C# (541.9 KB) download button and save the file.
3. Expand the ZIP file to a folder. Perhaps to C:\Users*yourAccount*\Documents\Visual Studio 2015\Projects
4. Open the project in Visual Studio.
5. Ignore the message about "clicking the database file".
6. Expand the References folder and note the assemblies with yellow flags. These assemblies are missing.
7. Right-click the References folder, click Manage NuGet Packages and note the yellow bar across the top of the NuGet tab.
8. Click Restore.
9. In the Solution Explorer click the Refresh button and verify that the yellow flags are now gone.




10. Open the web.config file.
    a. Find the connection string and change "Data Source=(LocalDb)**v11.0**" to "Data Source=(localdb)**MSSQLLocalDB**".
    b. Find the "defaultConnectionFactory" section. Change "v11.0" to "mssqllocaldb".
11. Start the project. Add some departments, the courses and then some students.

Copyright © 2019 MAX Technical Training

# Testing with Standard Data (.Net Framework)

When working with Code First projects you can configure the Entity Framework to automatically drop the database, recreate the database and tables, and even populate a starting set of sample data! You simply create a class that inherits from one of three base classes and write code to create the sample data.

Three classes to inherit from:

- CreateDatabaseIfNotExists (This is the default)
- DropCreateDatabaseIfModelChanges<MyDataContext>
- DropCreateDatabaseAlways<MyDataContext>

Steps:

1. Create a class that inherits from either DropCreateDatabaseIfModelChanges or DropCreateDatabaseAlways and add your code to populate the database. While this class can be anywhere in your project, a common location is the App_Start folder. Remember to call SaveChanges!

```
public class CreateSampleData : DropCreateDatabaseAlways<BankDbContext>
{
    protected override void Seed(BankDbContext context)
    {
        context.Customers.Add(new Customer {
            FirstName = "Sam",
            LastName = "Jones",
            Manager = "Smith"
        });
        context.Customers.Add(new Customer
        {
            FirstName = "Susan",
            LastName = "Jones",
            Manager = "Smith"
        });

        context.Customers.Add(new Customer
        {
            FirstName = "Mike",
            LastName = "Smith",
            Manager = "Allen"
        });

        context.SaveChanges();

        context.CheckingAccounts.Add(new CheckingAccount
        {
```

2. Either one of the following:
    a. Add one line to the Application_Start method of your Global.asax:

```
System.Data.Entity.Database.SetInitializer( new App_Start.CreateSampleData() ) ;
```

    b. Add one entry to the <appSettings> section of the web.config:

```
<appSettings>
  <add key="DatabaseInitializerForType CodeFirstEFDemo.Models.BankDbContext, CodeFirstEFDemo"
       value="CodeFirstEFDemo.App_Start.CreateSampleData, CodeFirstEFDemo" />
```

    The web.config approach is preferred as there are no code changes needed between development and production.

# Testing with Standard Data (.Net Core)

While not done the same way, Core has built-in features to ensure that a database is dropped and recreated for testing purposes.

To always drop any existing database on each run add DataBase.EnsureDeleted() to the context's constructor(s):

Copyright © 2019 MAX Technical Training

```
public class BankDbContext : DbContext
{
    public DbSet<Customer> Customers { get; set; }
    public DbSet<CheckingAccount> CheckingAccounts { get; set; }

    public BankDbContext()
    {
        Database.EnsureDeleted();
        // code here to add sample data...
    }
```

If you want to just have the database automatically created if it does not exist, then use .EnsureCreated().

```
public class BankDbContext : DbContext
{
    public DbSet<Customer> Customers { get; set; }
    public DbSet<CheckingAccount> CheckingAccounts { get; set; }

    public BankDbContext()
    {
        Database.EnsureCreated();
        // code here to add sample data...
    }
```

# Calling SQL through the Entity Framework Context

You can make direct calls through the Entity Framework and take advantage of the existing configurations such as the connection string.

## Using SqlQuery

You can perform queries through the context using context.Database.SqlQuery<T> where "T" is the returned datatype.

```
var cnt = db.Database.SqlQuery<int>("Select count(*) from customers where manager is null");
ViewBag.ManagerCount = cnt.ToString() + " customers are not assigned to a manager!";
```

Keep in mind that your first choice for data access should usually be though the Entity Framework using LINQ. The following example is simpler to read, and less error prone due to IntelliSense and autocomplete.

```
var cnt2 = db.Customers.Where(c => c.Manager == null).Count();
ViewBag.ManagerCount = cnt.ToString() + " customers are not assigned to a manager!";
```

## Using ExecuteSqlCommand

When calling stored procedures or performing Insert, Update or Delete operations you can use context.Database.ExecuteSqlCommand with a SQL string and optional parameters. (Always use SQL parameters to reduce the risk of SQL injection attacks.)

```
var x = db.Database.ExecuteSqlCommand("delete Customers where customerid = @p0 and
manager = @p1", 3, "Smith");
```

When to use direct SQL?

- To access stored procedures, tables or columns that are not defined in your entities,
- To perform SQL operations not supported by EF such as creating or dropping SQL objects.

Copyright © 2019 MAX Technical Training

# Tips!

## Locally Cached Content

The Entity Framework DbContext caches changes until you call .SaveChanges(). If your Customers table had 10 records, you deleted 3 and added 2, the local copy of the data would have 12 items… 3 marked for delete and 2 not yet saved. Prior to calling .SaveChanges() you can access the .Local property of the entity. I.e. *yourContext*.Customers.Local.

Keep in mind that in a web project the life of the DbContext object is one page load, and to be more precise, the life of the controller object. Each Action that makes changes to the context should call SaveChanges() before it returns.

For more on .Local see: https://msdn.microsoft.com/en-us/library/jj592872(v=vs.113).aspx

## Monitoring the SQL generated by the Entity Framework

You have several options to monitor the SQL code that generated by the Entity Framework:

- Use the SQL Server Profiler tool.
- Add one line to the constructor of your DbContext class. (EF 6.1 and later, but not Core, yet.)
- Define an "interceptor" in your web.config file. (EF 6.1 and later, but not Core, yet.)

### Database.Log (EF 6.1 and later, but not Core, yet.)

You can set the Log property to a delegate that logs the SQL to some destination.

```
public class BankDbContext : DbContext
{
    public BankDbContext()
    {
        Database.Log = str => System.Diagnostics.Debug.WriteLine(str);
    }

    public DbSet<SavingSaccount> SavingAccounts { get; set; }
```

Notes:

- In a Console application you can use: Database.Log = Console.Write;
- You can write your own class and method to do the logging:

```
class MyEFSqlLogger
{
    public void Log(string str)
    {
        // do something with the SQL string
        System.Diagnostics.Debug.WriteLine(str);
    }
}

public class BankDbContext : DbContext
{
    public BankDbContext()
    {
        var logSql = new MyEFSqlLogger();
        Database.Log = str => logSql.Log(str);


        //Database.Log = str => System.Diagnostics.Debug.WriteLine(str);
        //Database.Log = Logger;
    }
```

Copyright © 2019 MAX Technical Training

- You can set Database.Log to any method in your class that accepts a single string parameter.

```
public class BankDbContext : DbContext
{
    public BankDbContext()
    {
        //Database.Log = str => System.Diagnostics.Debug.WriteLine(str);
        Database.Log = Logger;
    }

    private void Logger(string str)
    {
        System.Diagnostics.Debug.WriteLine(str);
    }
```

See here for more about Database.Log: https://msdn.microsoft.com/en-us/library/dn469464(v=vs.113).aspx

## Entity Framework Interceptors (EF 6.1 and later)

You can add an entry to your web.config file to automatically collect the SQL sent from the Entity Framework to SQL Server.

```
<entityFramework>
  <interceptors>
    <interceptor type="System.Data.Entity.Infrastructure.Interception.DatabaseLogger, EntityFramework">
      <parameters>
        <parameter value="C:\MyLogs\LogOutput.txt"/>
        <parameter value="true" type="System.Boolean"/>
      </parameters>
    </interceptor>
  </interceptors>
```

Notes:

- If you do not include the parameter lines, the output will be logged to the Visual Studio console.
- The second parameter is optional and will cause the log file to be appended to and will cause the log to potentially grow quite large. Without the parameter the log will be overwritten with each run.

See: https://msdn.microsoft.com/en-us/library/jj556606(v=vs.113).aspx#Anchor_5

# Entity Framework Fluent API

The Fluent API offers an alternate to using attributes for entity model configuration. There are two benefits to the Fluent API: changes to the model can be made at runtime rather than only at compile time, and more flexibility and options. The Fluent API code runs from within the OnModelCreating method of the DbContext class.

**Using attributes to mark a field as a primary key in the entity class:**

```
public class Customer
{
   [Key]
   public int CustomerID { get; set; }
   public string FirstName { get; set; }
…
```

**Using the Fluent API to set a field as a primary key using the OnModelCreating in the context class:**

```
public class BankingContext : DbContext
{
```

Copyright © 2019 MAX Technical Training

```
    public DbSet<Customer> Customers { get; set; }

    protected override void OnModelCreating(DbModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Customer>().HasKey(t => t.CustomerID);
    }
}
```

For more on the Fluent API see:
https://docs.microsoft.com/en-us/ef/ef6/modeling/code-first/fluent/types-and-properties

# In Summary

The Entity Framework is an object-relational mapper that bridges between objects (Customers, Products, etc.) and relational databases.

Benefits:

- You write little or no SQL!
- You write little or no database access code.
- You work only with classes and objects.
- You can use LINQ to write your queries.
- Very easy to use.

Copyright © 2019 MAX Technical Training

# Module 5 – Models

## Models

A Model represents the application's data. The model typically defines the business objects as classes, retrieves and stores data to the database and maintains the model state. While the most common model framework for MVC is the Entity Framework, you can use other frameworks or create your own model code.

## MVC without the Entity Framework?

The concept of a Model does not depend on the Entity Framework. You can create your own model classes and the methods to implement Select, Create, Read, Update and Delete operations.

```
public class Location
{
    // Location properties
    public int LocationID { get; set; }
    public string LocationName { get; set; }
    public string Address { get; set; }
    public string phone { get; set; }
    public int Level { get; set; }

    public List<Location> List()
    {
        var locations = new List<Location>();
        // SQL code to populate list goes here...
        return locations;
    }

    public Location Find(int id)
    {
        var location = new Location();
        // SQL code to populate the object goes here
        return location;
    }

    public void Insert(Location loc)
    {
        // SQL code to insert the object goes here
    }

    public void Update(Location loc)
    {
        // SQL code to update the object goes here
    }

    public void Delete(Location loc)
    {
        // SQL code to delete the object goes here
    }
}
```

Copyright © 2019 MAX Technical Training

You would then add code to your controller to use this object. You can start with the Creating a Controller with "read/write actions" controller template.

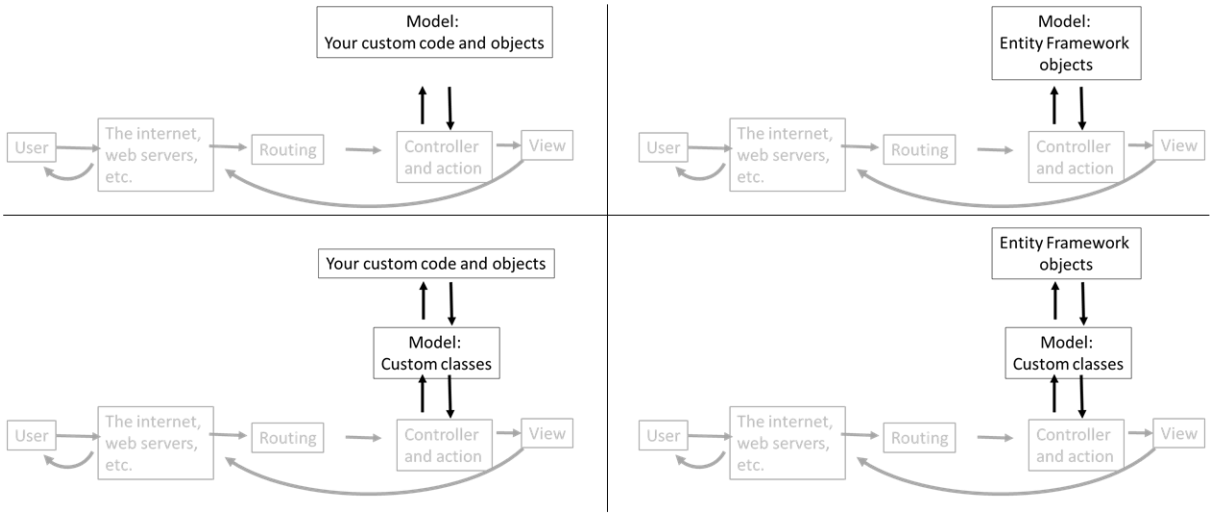
```
public class LocationController : Controller
{
    Models.Location db = new Models.Location();

    // GET: Location
    public ActionResult Index()
    {
        return View( db.GetList());
    }

    // GET: Location/Details/5
    public ActionResult Details(int id)
    {
        var location = db.Find(id);
        // if (location == null) ...
        return View(location);
    }

    // GET: Location/Create
    public ActionResult Create()
    {
        var location = new Models.Location();
        return View(location);
    }
```

And then create your views!

# Tiers

Models are often broken down into two or more tiers. In the top two examples below, there's only one model tier. This is the way that MVC is most often presented. In the real world you will rarely want to expose all of the properties of a table to a view for users to edit. You don't want people to approve their own expense reports or change the "created date" of objects. You also often need to enforce business rules that go beyond what can be built into SQL or Entity Framework features.

The tier closest to the database tends to work with "domain level" objects that include all of the properties of the objects. The tier closest to an MVC controller will often use a custom or "View Model" object that only exposes what needs to be displayed or edited.




Copyright © 2019 MAX Technical Training

# Model Attributes

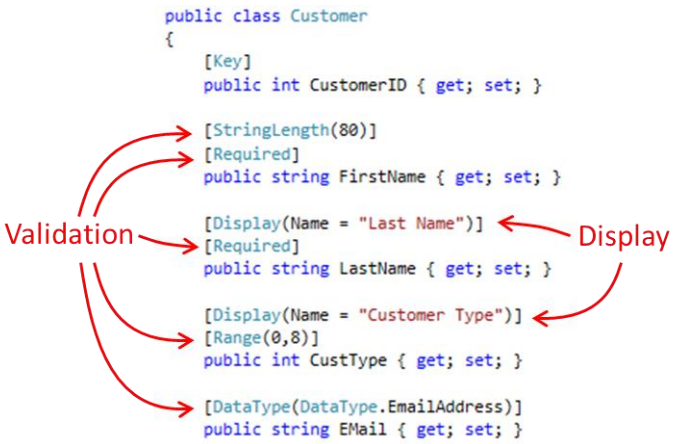
Although one of the goals of MVC is to completely separate the data (Models) from the presentation of the data (Views), there are many times we are going to skip across that boundary. One of the MVC features that does reach across this boundary is the use of model attributes to hint at how data is to be displayed and validated. While we can add these attributes, the person designing the view does not have to use them. Like the pirates say "the code is more what you'd call "guidelines" than actual rules"!

Model Attributes are used for:

- Entity Framework automatic database creation and migrations.
- Entity Framework data validation. ( if (ModelState.IsValid) … ).
- Razor View data display and validation.

Model attributes or annotations fall into two categories: display and validation. Some can be applied at the class level, while others are used at the property level.

Here's a sample Model class with a few attributes:




```
public class Customer
{
    [Key]
    public int CustomerID { get; set; }

    [StringLength(80)]
    [Required]
    public string FirstName { get; set; }

    [Display(Name = "Last Name")]
    [Required]
    public string LastName { get; set; }

    [Display(Name = "Customer Type")]
    [Range(0,8)]
    public int CustType { get; set; }

    [DataType(DataType.EmailAddress)]
    public string EMail { get; set; }
```

Note: The "CustType" property is a required field, even without the [Required] attribute, as it is a non-nullable integer.

Note: Changes to the Model, including the addition of attributes, made after you have created your Controllers and Views will not update the Controllers and Views. You will need to either manually make any needed changes or delete both the Controller and the matching View folder and recreate the Controller.

## Appling Attributes to Database First and Model First Models

If you create your Models using Code First then you can add attributes directly to the entity class properties. With EF 6's Database First and Model First you don't directly create the entity classes and should not edit the auto-generated class files. To apply model attributes to these entities we need to take advantage of the fact that these classes were created as "partial" classes and we need to create a metadata class with the attributes.

Steps:

1. Extend the entity class by creating a partial class with the same name. A good place to put this class file is the Models folder. Name the class to match the entity you want to add the attribute to and

Copyright © 2019 MAX Technical Training

make this a partial class.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;

namespace Northwind.Models
{
    public partial class Product
    {
    }
}
```

2. Add using statements for System.ComponentModel and System.ComponentModel.DataAnnotations.

```
using System.ComponentModel;
using System.ComponentModel.DataAnnotations;
```

3. In the same file, add a class for the metadata. While any name will work, you might want to use a pattern of "*entityname*Metadata".

```
public class ProductMetaData
{

}
```

4. Add each property and add your attributes.

```
public class ProductMetaData
{
    [StringLength(120)]
    public string Name { get; set; }

    [RegularExpression(@"^[A-Za-z]{2}[0-9]{3}$")]  // 2 letters and 3 digits
    public string ProductNumber { get; set; }

    [DataType(DataType.Currency)]
    public decimal ListPrice { get; set; }
}
```

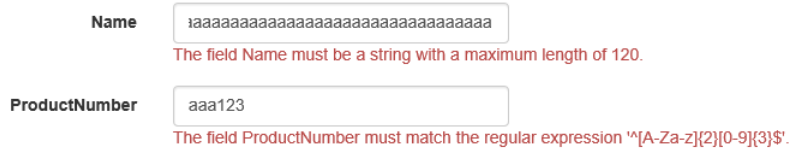
5. Add the metadata attribute to the partial class.

```
[MetadataType(typeof(ProductMetaData))]
public partial class Product
{
}
```

6. The result:

```
[MetadataType(typeof(ProductMetaData))]
public partial class Product
{
}

public class ProductMetaData
{
    [StringLength(120)]
    public string Name { get; set; }

    [RegularExpression(@"^[A-Za-z]{2}[0-9]{3}$")]  // 2 letters and 3 digits
    public string ProductNumber { get; set; }

    [DataType(DataType.Currency)]
    public decimal ListPrice { get; set; }
}
```

7. Test!




Copyright © 2019 MAX Technical Training

# Database Naming Attributes

By default, the Entity Framework will create new tables with the “dbo” schema. To override this, add the [Table] attribute. The following example will create a table named “Production.Product”. The internal class name will still be “Product”. Without this attribute, the SQL table would have been named “dbo.Products”.

```
[Table("ProdTable", Schema = "Production")]
public class Product
{
   public int ProductID { get; set; }
```

The Table attribute needs this using statement:

```
using System.ComponentModel.DataAnnotations.Schema;
```

# Display Attributes

Several attributes are used to supply “hints” and metadata to the view engines. In the Razor view engine, this additional data is picked up by the DisplayFor HTML helper.

Example Attributes:

- [DataType(DataType.DateTime)]
- [DisplayFormat(DataFormatString = "{0:MM/dd/yy}", ApplyFormatInEditMode = true)]
- [DisplayFormat(ConvertEmptyStringToNull = true, NullDisplayText = "[Null]")]

# Common options:

Many of the validation attributes have a few shared properties.

The validation attributes have default error messages that may be a bit confusing to the end user of your application. You can supply your own custom text for the error message.

```
ErrorMessage ="The password must be at least eight characters long."
```

If you are using ASP.NET Resource files to support multiple languages, you can add the following attributes to read language or region specific text from a resource file.

```
ErrorMessageResourceName ="InvalidEmail", ErrorMessageResourceType ="Resources.Resource"
```

## [DisplayName] and [Description]

Obsolete. Use [Display].

## [Display]

Display is used to define text to display as an alternate to the model’s property name. In Razor, this name can be displayed using the DisplayFor HTML helper.

Copyright © 2019 MAX Technical Training


Notes:

- If the ResourceType property is not supplied then value of the Name property is displayed using the Razor DisplayNameFor HTML helper.
- If the ResourceType property is supplied then value of the Name property is used to lookup the display text in an ASP.NET Resource file. (See "Globalization and Localization" later in this course.) In the example below, the key "AccountNickname" would be looked up in the appropriate resource file and the text for display would be returned (perhaps "Apodo de la cuenta".)

```
[Display(Name = "AccountNickname", ResourceType = typeof(Resources.Resource))]
public string AccountNickname { get; set; }
```

# Validation Attributes

The validation attributes provide server-side validation, and in many cases JavaScript or HTML5 form element validation via the view engine.

Example validation attributes:

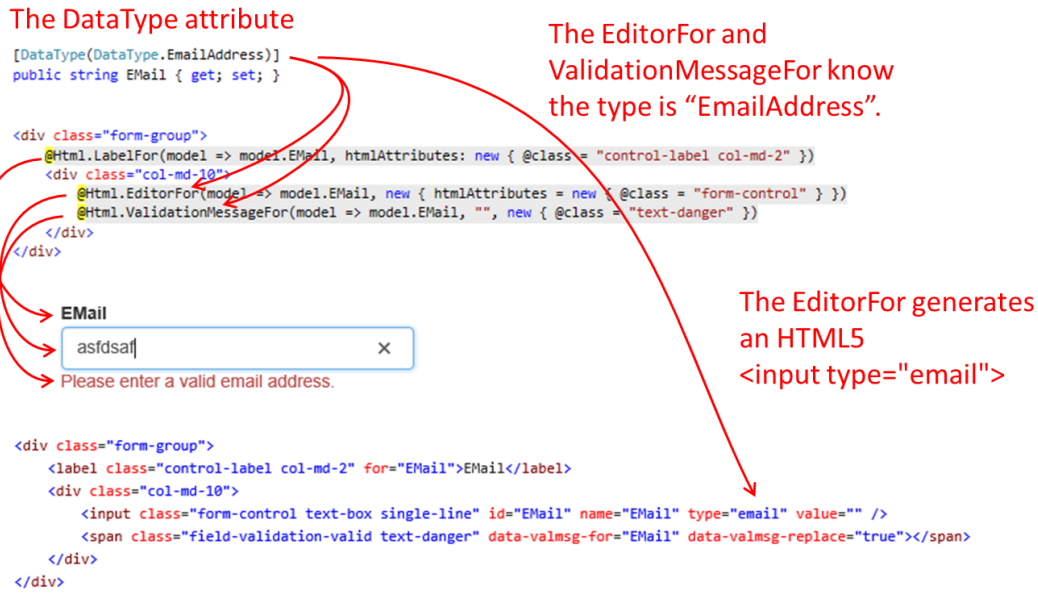
- [DataType(DataType.EmailAddress)]
- [Required]
- [Required(ErrorMessage="Please enter a name.")]
- [Range(0, 400)]
- [RegularExpression(".+\\@.+\\..+")]
- [StringLength]

## [DataType]

DataType can have both a display and a validation impact on the view depending on the datatype and the browser. DataType is used when the default SQL and .NET datatypes are not precise enough. As an example, HTML5 supports an "email" form input type that is stored in a simple string in SQL or .NET. Setting the DataType to "EmailAddress" will both set the input type to "email" and validate the entered text as an email address.

The New Controller and New View wizards add client side validation to match the DataType. DataType does not validate server-side. You will need to add Required and/or RegularExpression attributes for server-side validation.

Copyright © 2019 MAX Technical Training




Notes:

- The use of DataType may generate client-side validation when used with HTML.EditorFor. It will not generate server side validation.
- The actual display of the input controls on the page will vary by browser and browser version.
- The “Input type” column in the table below shows the type generated by HTML.EditorFor for the <input> tags’ These are often the new HTML5 types and are not equally supported across all browsers. For browser support see: http://caniuse.com and http://html5test.com.
- HTML5 browsers that do not support an input type fallback to “<input type="text">”.
- For dates and numbers, some HTML5 browsers fallback to type “text”, some do validation and some will display date and number pickers.
- Bottom line… use tools like jQueryUI and shims. And, test, test, test.

**Types:** (CSV = Client side validation)

| DataType value | @Html.DisplayFor() | Input type | CSV |
|---|---|---|---|
| CreditCard | | text | |
| Currency | | text | |
| Custom | | text | |
| Date | | date | Y |
| DateTime | | time | |
| Duration | | text | |
| EmailAddress | Clickable link (<a href="mailto:a@b">a@b.com</a>) | email | Y |
| Html | | text | |
| ImageUrl | | text | |
| MultilineText | | textarea | |
| Password | | password | |
| PhoneNumber | | tel | |
| PostalCode | | text | |
| Text | | text | |
| Time | | time | |
| Upload | | text | |

Copyright © 2019 MAX Technical Training

| | | | |
|---|---|---|---|
| Url | Clickable link (<a href="http://a.com">http://a.com</a>) | url | Y |

## [Required]

The required field marks a property as required. Even without the attribute, fields may be "required" when processed server-side. For example an "int" datatype does not support null values. (You can declare the int with a "?" (int?) to make it nullable.) SQL server may also have the property defined as "not null".

- Title (nvarchar(8), null)
- FirstName (Name(nvarchar(50)), not null)
- MiddleName (Name(nvarchar(50)), null)
- LastName (Name(nvarchar(50)), not null)

You can supply a custom error message.

```
[Required(ErrorMessage="Please enter a name.")]
```

```
[Required]
public string AccountNickname { get; set; }
```

The AccountNickname field is required.

```
[Required(ErrorMessage ="Please enter an account nickname for quicker phone support.")]
public string AccountNickname { get; set; }
```

Please enter an account nickname for quicker phone support.

## [Range]

Range restricts a field to a range of values. By default the range is numeric. While you can specify a datatype and the validation works correctly server-side, the client-side jQuery library appears to only support numbers. You may want to create a custom validator for non-numeric range validations. Also see [RegularExpression] below for an example of a range of letters.

```
[Range(0, 400)]
```

```
[Range(typeof(string), "A","E")]
```
(Does not appear to currently be supported for client-side validation.)

## [RegularExpression]

You can validate all kinds of text patterns using Regular Expressions. As these expressions frequently include C# string escape characters, they are usually written using C#'s literal string notation: @"abc".

```
[RegularExpression("[A-E]")]
```
Must be a letter from A-E.

```
[RegularExpression(@"^[A-Za-z]{2}[0-9]{3}$")]
```
Must have 2 letters and 3 digits.

```
[RegularExpression(@"^\d{3}-\d{3}-\d{4}$")]
```
Must look like a phone number. (123-123-1234)

```
[RegularExpression(@"^\d+[F,C] $")]
```
A temperature in F or C. (120F)

Search the web for "regular expression library", "regular expression tutorial" or "regular expression cheat sheet" to learn more.

## [Remote]

Remote is a validator attribute used to call an action in a controller to confirm the existence of a value.

(see [Remote()] in the "Additional MVC Attributes" section below.)

Copyright © 2019 MAX Technical Training

# [StringLength]

Use to set a maximum length for a string value and optionally set a minimum length. If there are multiple possible acceptable lengths then consider using a regular expression.

```
[StringLength(80)]

[StringLength(80,MinimumLength = 5)]

[StringLength(80,MinimumLength = 5, ErrorMessage = "Must be between 5 and 80 characters.")]
```

# [Custom]

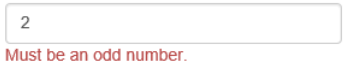
You can create your own validation attributes by creating a class that inherits from System.ComponentModel.DataAnnotations.ValidationAttribute and overriding the IsValid() method.

A few ideas:

- [RequiredIf(some Boolean expression)]
- [OddOrOven("odd")]
- [Category("toys")]

## Adding the Server Side Code

Here's an example that validates a field as either being odd, even or null.




Steps:

1. Create a new class that inherits from ValidationAttribute. (And add a using statement for: System.ComponentModel.DataAnnotations) Typically store this class in the Models folder.
2. Override the IsValid method and write code to test your validation.
3. Add the attribute to a Model property.

**The Model property attribute:**

```
[MustBeOddOrEven("odd")]
public int? somenumber { get; set; }
```

**The class for the custom validator:**

Copyright © 2019 MAX Technical Training

```
public class MustBeOddOrEven : ValidationAttribute
{
    // get the attribute's parameters
    private readonly string _OddOrEven;
    public MustBeOddOrEven(string OddOrEven)
    {
        _OddOrEven = OddOrEven.ToLower();
    }

    // do the validation
    protected override ValidationResult IsValid(object value, ValidationContext validationContext)
    {
        if (value == null)  // assuming Null is ok...
        {
            return ValidationResult.Success;
        }
        // test for odd or even...
        int x = (int)value;
        bool IsEven = (x % 2 == 0);
        if (IsEven & (_OddOrEven == "even") | !IsEven & (_OddOrEven == "odd"))
        {
            return ValidationResult.Success;
        }
        return new ValidationResult("Must be an " + _OddOrEven + " number.");

    }
}
```

## Adding Client Side Validation

You will need to create JavaScript code to do the client side validation. To make that easier you can implement the IClientValidatable interface on your custom validation attribute class to pass the validation details to the client. This is done by adding "data dash" (data-*name*) attributes to the <input> tag. In the example below the "data-" attributes for our custom validator include the text "oddoreven". (The field's name is "somenumber".)

<input class="input-validation-error form-control text-box single-line"
data-val="true" data-val-number="The field somenumber must be a number."
**data-val-oddoreven=**"Must be an odd number." **data-val-oddoreven-option=**"odd"
id="somenumber" name="somenumber" type="number" value="2" />

All you need to complete the client-side validation is some JavaScript to pick up on the "data dash" items for our "oddoreven" validator.

**Here's the changes needed to implement IClientValidatable:**

Copyright © 2019 MAX Technical Training

```
public class MustBeOddOrEven : ValidationAttribute, IClientValidatable
{
    // get the attribute's parameters
    private readonly string _OddOrEven;
    public MustBeOddOrEven(string OddOrEven)
    {
        _OddOrEven = OddOrEven.ToLower();
    }

    // do the validation
    protected override ValidationResult IsValid(object value, ValidationContext validationContext)
    {
        if (value == null)  // assuming Null is ok...
        {
            return ValidationResult.Success;
        }
        // test for odd or even...
        int x = (int)value;
        bool IsEven = (x % 2 == 0);
        if (IsEven & (_OddOrEven == "even") | !IsEven & (_OddOrEven == "odd"))
        {
            return ValidationResult.Success;
        }
        return new ValidationResult("Must be an " + _OddOrEven + " number.");
    }

    public IEnumerable<ModelClientValidationRule> GetClientValidationRules(ModelMetadata metadata, ControllerContext context)
    {
        var rule = new ModelClientValidationRule();

        rule.ErrorMessage = "Must be an " + _OddOrEven + " number.";
        rule.ValidationParameters.Add("option", _OddOrEven);
        rule.ValidationType = "oddoreven";
        yield return rule;
    }
}
```

Tip! If you have not seen "yield return" before then see: https://msdn.microsoft.com/en-us/library/9k7k7cf0.aspx

# Other Attributes

## [NotMapped]

NotMapped is used to add properties to an entity that are not duplicated in the SQL tables. As, an example, you can add calculated columns to the model by creating a property with the calculation and marking it as "NotMapped". The wizards for adding views will not automatically add this property to your views. You will need to add it manually.

```
public string FirstName { get; set; }
public string LastName { get; set; }

[NotMapped]
public string FullName
{
    get { return this.FirstName + " " + this.LastName; }
}
```

# Additional MVC Attributes

The following attributes require an additional using statement:

```
using System.Web.Mvc;
```

Copyright © 2019 MAX Technical Training

## [AdditionalMetadata]

AdditionalMetadata adds additional metadata about a model property that can be accessed from a view. There are no Razor HTML helpers to quickly access this data.

Code for the Model:

```
[AdditionalMetadata("DisplayClass", "h1.bold")]
public string BankName { get; set; }
```

Code for the View:

```
@ModelMetadata.FromLambdaExpression(x => x.BankName,
    ViewData).AdditionalValues["DisplayClass"]
```

or

```
@ModelMetadata.FromStringExpression("BankName",
  ViewData).AdditionalValues["DisplayClass"]
```

## [AllowHtml]

By default, MVC blocks HTML content. Using AllowHtml is a security risk. If you allow HTML then you should validate the safety of the content. For example, the following is blocked by default:

```
<b><a href=junklink>Click me!</a><b><script>alert('hi')</script>
```


## [Remote()]

The Remote attribute uses the jQuery validation plug-in remote validator to make an asynchronous call to a server side controller/action to check for the existence of a value.

Examples of use:

- Check to see the entered part number exists.
- Check to see if the conference room, airplane seat, etc. is available.
- Verify that an email address has not been added to the system yet.

Basic steps:

1. Create an Action in a Controller that returns a JSON result (return Json()) that returns either true or a string with an error message.
2. Add a Remote attribute to a property that follows this pattern:
   [Remote("ActionName","ControllerName", Error="some error message")]

The validator sends a request that looks like this:

```
/ControllerName/ActionName?propertyname=value
```

See:

- RemoteAttribute Class
  https://msdn.microsoft.com/en-us/library/system.web.mvc.remoteattribute(v=vs.118).aspx
- How to: Implement Remote Validation in ASP.NET MVC
  https://msdn.microsoft.com/en-us/library/gg508808(VS.98).aspx

Copyright © 2019 MAX Technical Training

- jQuery Validation Plugin
  https://jqueryvalidation.org/remote-method/

# View Models

The Domain Model describes our primary business objects and frequently represents data from a SQL table. The model includes all of the properties of the customer, bankaccount or product entities.

A View Model is subset or superset of the Domain Model. For example, the domain model for Customer includes a DateOfFirstOrder property. This value is never changed and we would not want accidental changes to be made or for a hacker to get access to the value. The View Model may also include non-domain model properties that are needed in the view but are not stored with the primary object. For example, our view may need the number of orders created by the customer and that value is not a property of the Domain Model.

Notes:

- View Models are typically stored in the Models folder of the project.
- Note! Do not add the new View Model to the DbContext class! It is not directly mapped to a table in the database.
- When a View needs additional objects, such as data for a dropdown list you have several options;
  - Do a direct to SQL or DbContent from the View. (Never do this!)
  - Pass the object in the ViewBag.
  - Create a View Model and add the additional object as a property. (This is the preferred technique as all data and relationships are defined in the models.)

## Example

In the example below we want to edit a subset of Customer data and pass through data to populate the CustType dropdown list. The CustomerEditView only contains the data needed for the edit page. There is no risk of a hacker getting to the “IsAdmin” property!

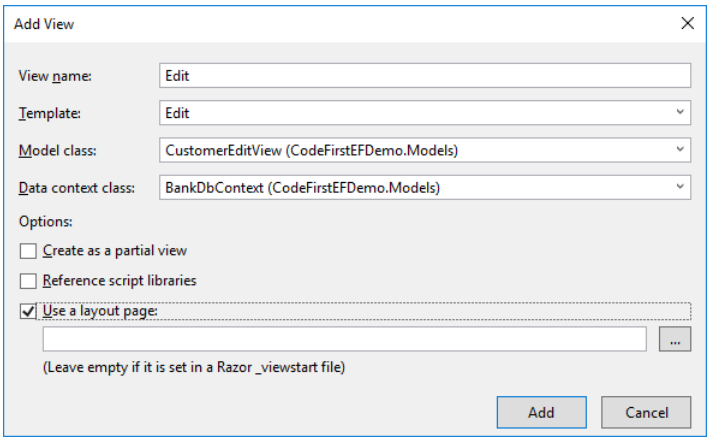
```
public class Customer
{
    [Key]
    public int CustomerID { get; set; }

    public string FirstName { get; set; }
    public string LastName { get; set; }
    public int? CustType { get; set; }
    [DataType(DataType.EmailAddress)]
    public string EMail { get; set; }
    public string Manager { get; set; }
    public DateTime? DateOfFirstOrder { get; set; }
    public bool IsAdmin { get; set; }
}
```

```
public class CustomerEditView
{
    [Key]
    public int CustomerID { get; set; }

    public string FirstName { get; set; }
    public string LastName { get; set; }
    public int? CustType { get; set; }
    [DataType(DataType.EmailAddress)]
    public string EMail { get; set; }
    public List<CustomerType>  custTypes { get; set; }
}
```

```
public class CustomerType
{
    public int CustomerTypeId { get; set; }
    public string Name { get; set; }
    public string Status { get; set; }
}
```

Create the View and set the Model class to your new View Model. Make sure you select the View Model class (CustomerEditView) and not the Domain Model class (Customer).

Copyright © 2019 MAX Technical Training

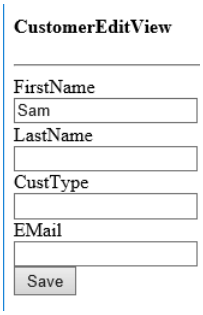



You will need to edit the Customer Controller to add the code to populate an instance of the View Model class. Customer properties are copied and the "custType" property is populated as a list of "CustomerTypes".

```
// GET: Customers/Edit/5
public ActionResult Edit(int? id)
{
    if (id == null)
    {
        return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
    }
    Customer customer = db.Customers.Find(id);
    if (customer == null)
    {
        return HttpNotFound();
    }

    CustomerEditView editview = new CustomerEditView();
    editview.CustomerID = customer.CustomerID;
    editview.FirstName = customer.FirstName;
    editview.CustType = customer.CustType;
    editview.EMail = customer.EMail;
    editview.custTypes = db.CustomerTypes.ToList();

    return View(editview);
}
```

Tip: Consider moving this code into a class in the business tier that returns a CustomerEditView object. This removes this business logic code from the Controller.

The View only exposes the properties we want exposed!




With a little work in the View you can add a dropdown list! Just replace:

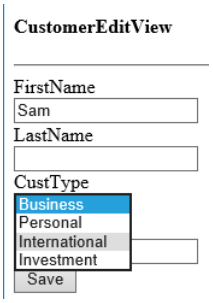
```
@Html.EditorFor(model => model.CustType, new { htmlAttributes = new { @class = "form-control" } })
```

Copyright © 2019 MAX Technical Training

with:

```
@Html.DropDownListFor(model => model.CustType,
    Model.custTypes.Select(s => new SelectListItem
    {
        Value = s.CustomerTypeId.ToString(),
        Text = s.Name
    }))
```

And you will get a dropdown with no HTML work!




Html.DropDownListFor is described in the Views module of this course.

# Best Practices

- Never create HTML in Model code, that's what Views are for!
- Never pass model properties to a view that are not needed there. Create a View Model with only what is needed.
- Add validation attributes to Models. Even if they are not used in Views, they often impact how the tables are created in SQL Server and also server as in-line documentation.

Copyright © 2019 MAX Technical Training

# Module 6 – Views

## Route to Controller to Action to View

After the server receives a request, the router parses the URL and selects a Controller and an Action. The Controller's Action code then collects data and retrieves a View. Unlike Web Forms, an MVC URL never points to a page. Instead, it causes the Controller to choose a view and return it. There is no direct path from a URL to a View, or from a View back to a browser. The Controller is always in between the two.

## View Engines

A View Engine is what takes a view file, processes it and then returns the HTML to return to the user's browser.

.Net Core only ships with support for Razor views. Custom view engines can be used.

The Visual Studio .Net Framework MVC templates include two view engines: ASP.NET Web Forms and Razor. Additional View Engines are available:

- ASP.NET Web Forms
- Razor
- NHAML
- Spark
- And more…
- And… you can write your own!

Links for comparisons of View Engines:

- ASP.NET MVC View Engine Comparison
  http://stackoverflow.com/questions/1451319/asp-net-mvc-view-engine-comparison
- Introducing "Razor" – a new view engine for ASP.NET
  https://weblogs.asp.net/scottgu/introducing-razor
- Developer Review - Four ASP.NET MVC View Engines
  https://channel9.msdn.com/coding4fun/articles/Developer-Review-Four-ASPNET-MVC-View-Engines

## Auto-magic and Views

Note: The formal term for "auto-magic" is "conventions".

**No View Name Supplied**

When a controller uses the View() helper method without specifying a view, MVC tries the registered view engines and takes a few guesses as to where to find a view file. (The following view search examples are for .Net Framework.)

```
public ActionResult SalesReport()
{
   return View();
}
```

Copyright © 2019 MAX Technical Training

Server Error in '/' Application.

*The view 'salesreport' or its master was not found or no view engine supports the searched locations. The following locations were searched:*
*~/Views/home/salesreport.aspx*
*~/Views/home/salesreport.ascx*
*~/Views/Shared/salesreport.aspx*
*~/Views/Shared/salesreport.ascx*
*~/Views/home/salesreport.cshtml*
*~/Views/home/salesreport.vbhtml*
*~/Views/Shared/salesreport.cshtml*
*~/Views/Shared/salesreport.vbhtml*

Note that MVC first looks for ASP.NET forms and web controls files (.aspx and .ascx), then for C# and VB.Net Razor views. The search first guesses a folder based on the Controller name and a file based on the Action name. If it can't find the file in the folder with the Controller's name, it also checks in the Shared folder.

While the above search order looks inefficient, the file list being searched is cached and does not require disk access for each search.

**View Name Supplied**

If you supply a View name, and it's not found, then the same search pattern is followed, but using the supplied View name.

```
public ActionResult SalesReport()
{
   return View("Report1");
}
```

Server Error in '/' Application.

*The view 'Report1' or its master was not found or no view engine supports the searched locations. The following locations were searched:*
*~/Views/home/Report1.aspx*
*~/Views/home/Report1.ascx*
*~/Views/Shared/Report1.aspx*
*~/Views/Shared/Report1.ascx*
*~/Views/home/Report1.cshtml*
*~/Views/home/Report1.vbhtml*
*~/Views/Shared/Report1.cshtml*
*~/Views/Shared/Report1.vbhtml*

**Why look for the .aspx and .ascx pages?**

If you will not be using ASP.NET web forms as views in your project you can remove the web forms view engine and just search using the Razor engine, but it may not be worth the trouble as the file list is cached and the search is very fast. If you want to experiment, you can add code to your Global.asax file's Application_Start method to limit the search to just Razor by customizing ViewEngines.Engines.

```
ViewEngines.Engines.Clear();
ViewEngines.Engines.Add(new RazorViewEngine());
```

Copyright © 2019 MAX Technical Training

Server Error in '/' Application.

*The view 'Report1' or its master was not found or no view engine supports the searched locations. The following locations were searched:*
*~/Views/Home/Report1.cshtml*
*~/Views/Home/Report1.vbhtml*
*~/Views/Shared/Report1.cshtml*
*~/Views/Shared/Report1.vbhtml*

# Controller Helpers for Views

## View()

Views are accessed from a controller using the View() helper method of the Controller base class.

```
// GET: Customers
public ActionResult Index()
{
    return View(db.Customers.ToList());
}
```

The View() method has several overrides to let you select the view and pass data to the view.

- View()
  - Searches for a view with the same name as the Action.
  - No model is passed to the View.
- View(string)
  - Searches for a view using the specified name.
    return View("SalesReport")
  - No model is passed to the View.
- View(object)
  - Searches for a view with the same name as the Action.
  - The object is the model being passed to the View.
    return View(customer);
    return View(db.Customers.ToList());
- View(string,object)
  - Searches for a view using the specified name.
  - The object is the model being passed to the View.
    return View("Details", customer);
    return View("Index", db.Customers.ToList());
- View(string,string)
  - Searches for a view using the specified name and layout (masterName).
    return View("SalesReport", "MobileLayout")
  - No model is passed to the View.
- View(string,string,object)
  - Searches for a view using the specified name and layout (masterName).
  - The object is the model being passed to the View.
    return View("SalesReport", "MobileLayout", db.SalesReport.ToList())

Copyright © 2019 MAX Technical Training

# Passing Data to Views

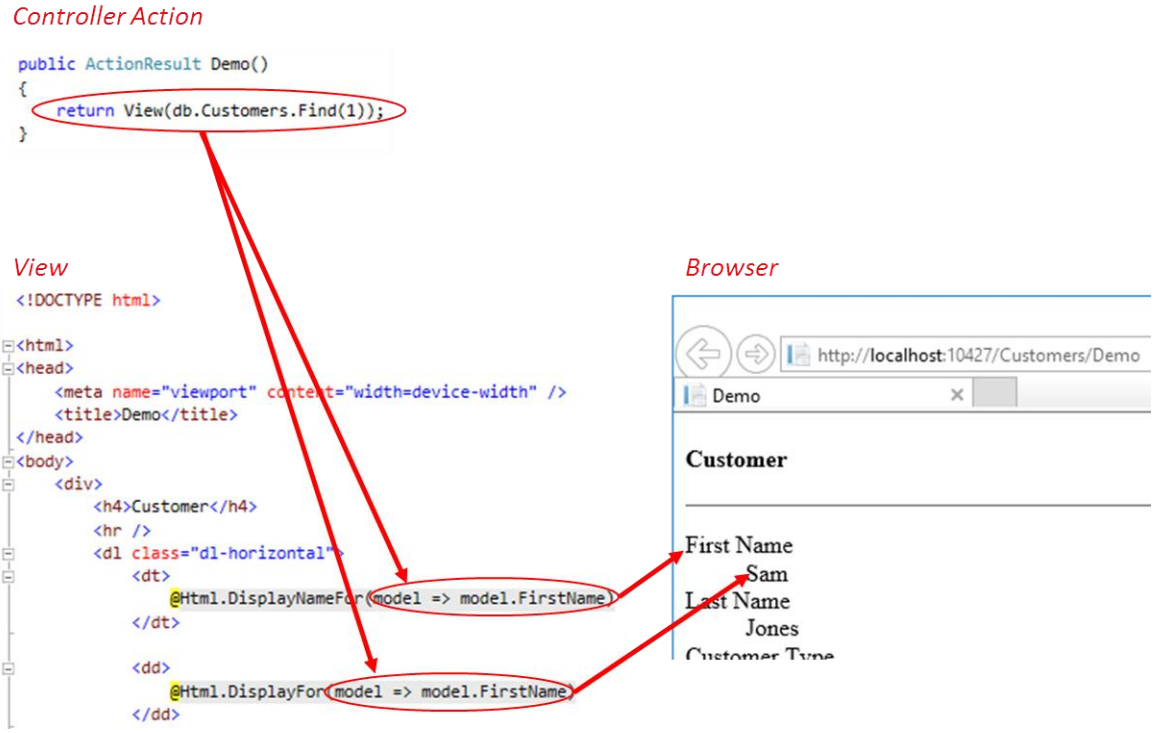
Data can only be passed from a Controller to a View using a Model object and/or the ViewBag object. You will frequently create a custom "View Model" has hides certain properties of the base object and adds additional properties needed by the View.

- Pass a Model (an object): View(domainobject);
- Pass a View Model: var object = CreateCustomerView(domainobject); View(object);
- Pass ViewData / ViewBag ViewBag.username="Fred"; View();
- Pass a model and view data: ViewBag.username="Fred"; View(someobject);

## ViewBag


Copyright © 2019 MAX Technical Training

# Model



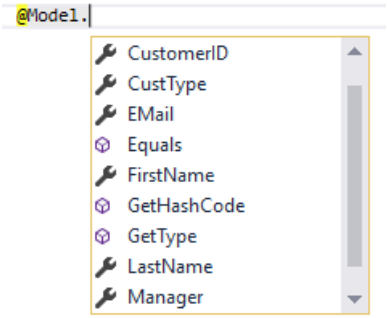

# "Model." vs. "model." vs. "@model"

In the sample View below notice the three uses of the word "model". All three have different meaning!


**Model?**

Copyright © 2019 MAX Technical Training

- **"@model"** is a directive, or instruction to the compiler, that says this View is tightly bound to the following datatype. In this case, "CodeFirstEFDemo.Models.Customer". I.e. this View only will work when passed a Customer object.
- **"Model."** (upper case) is a variable that contains the passed in model object. From it the view has access to all of the properties of the passed in object.




- **"model."** (lower case) is a lambda expression (anonymous function) parameter. The word "model" has no special meaning here and is scoped to just this lambda expression. You can use any placeholder name you like! All three of the following produce the exact same result.

```
@Html.DisplayNameFor(model => model.FirstName)
@Html.DisplayNameFor(customer => customer.FirstName)
@Html.DisplayNameFor(x => x.FirstName)
```

  - Tips!
    - Lambda expressions are commonly used in .NET development and are a very useful skill to add to your toolbox. See: https://msdn.microsoft.com/en-us/library/bb397687.aspx
    - If you are not using Display Templates, then instead of using "@Html.DisplayFor(model => model.FirstName)" you could just write "@Model.FirstName".

# Razor

This course will use Razor for our Views.

- Easy to learn.
- No new language to learn. Uses C# or VB.
- Works with Unit Testing.
- The Add View wizards generally assume you want to use Razor!

Notes:

- Embedded Razor runs on the server while JavaScript runs on the client. JavaScript cannot call Razor functions or helpers.
- Razor code is not compiled and errors are not reported from a Build.
- In Visual Studio, if you first display a View file (.cshtml or .vbhtml) and then start a debug, Visual Studio will launch a browser using the route to that View. This is not always useful and often will display a 400 error. It builds a URL from the view's folder name and the view's file name. (The Sales.cshtml file in the /Views/Reports folder will attempted as /Reports/Sales.)
- Views should not contain any application / business logic.
- Views should not directly access databases.

Copyright © 2019 MAX Technical Training

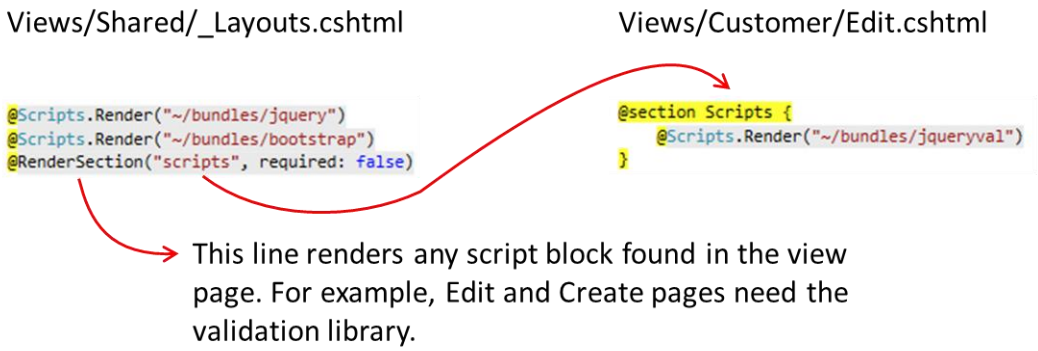
# JavaScript Libraries Included with .Net Framework MVC Razor Templates

The following JavaScript libraries are included with a Razor template. They are the versions shipped with Visual Studio 2015. You may want to update these either manually or by using NuGet.

Many of the files are bundled using App_Start/BundleConfig.cs.

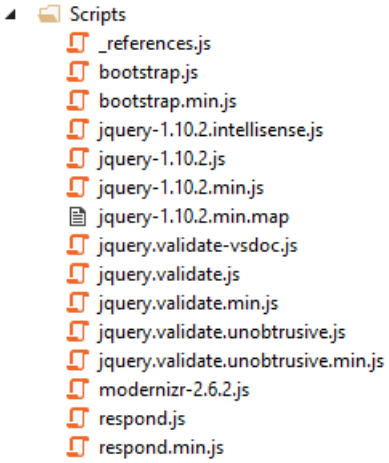
```
bundles.Add(new ScriptBundle("~/bundles/jquery").Include("~/Scripts/jquery-{version}.js"));
bundles.Add(new ScriptBundle("~/bundles/jqueryval").Include("~/Scripts/jquery.validate*"));
bundles.Add(new ScriptBundle("~/bundles/modernizr").Include("~/Scripts/modernizr-*"));
bundles.Add(new ScriptBundle("~/bundles/bootstrap").Include("~/Scripts/bootstrap.js","~/Scripts/respond.js"));
bundles.Add(new StyleBundle("~/Content/css").Include("~/Content/bootstrap.css","~/Content/site.css"));
```

These bundles are added to your Razor page and the Layouts page using Razor @Scripts.Render methods.




# JavaScript Libraries (/Scripts folder)

These are the JavaScript libraries loaded into your scripts folders by default.




When you create a new project, these files are copied from the Visual Studio installation directory:
C:\Program Files (x86)\Microsoft Visual Studio 14.0\Common7\IDE
\WebTemplates\DNX\CSharp\1033\StarterWeb\wwwroot\lib

You may want to update some of these to more recent versions. Be aware that some updates could break some of the MVC generated JavaScript code.

| | |
|---|---|
| bootstrap.js | A framework for developing responsive, mobile first projects. |

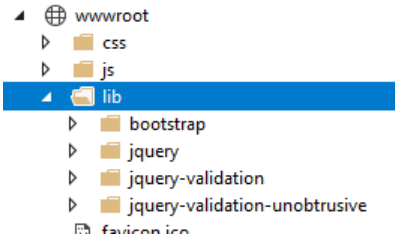
Copyright © 2019 MAX Technical Training

| | |
|---|---|
| | Includes Sass and Less CSS preprocessors.<br><br>Version 3.0.0. Current 3.3.7.<br><br>http://getbootstrap.com/ |
| bootstrap.min.js | Minimized version. |
| jquery-1.10.2.intellisense.js | Used to add jQuery IntelliSense features to Visual Studio. |
| jquery-1.10.2.js | The full, and readable, version of jQuery for development purposes. At the time of writing, the most recent versions are: 1.12.4, 2.2.4 and 3.1.1. (Version 2.x does not include support for IE before IE9, but is otherwise the same as 1.x. For more about 3.x see http://blog.jquery.com/2016/06/09/jquery-3-0-final-released/)<br><br>http://jquery.com/ |
| jquery-1.10.2.min.js | The minimized version of jQuery. |
| jquery-1.10.2.min.map | Used for debugging with the minimized version of jQuery. |
| jquery.validate-vsdoc.js | |
| jquery.validate.js | Version v1.11.1. Current version is 1.15.0<br>https://jqueryvalidation.org/ and<br>http://plugins.jquery.com/validation/ |
| jquery.validate.min.js | The minimized version of jqueryvalidate. |
| jquery.validate.unobtrusive.js | Microsoft created library to separate JavaScript code from the HTML markup. This version 3.2.2. Current version is 3.2.6<br><br>About Unobtrusive Validation:<br>http://bradwilson.typepad.com/blog/2010/10/mvc3-unobtrusive-validation.html<br>https://github.com/aspnet/jquery-validation-unobtrusive |
| jquery.validate.unobtrusive.min.js | Minimized version. |
| modernizr-2.6.2.js | A tool to test browser capabilities.<br><br>Version 2.6.2. Current version is 3.3.1.<br>https://modernizr.com/ |
| respond.js | Included with Bootstrap. A shim to support CSS3 Media Queries in older browsers. |
| respond.min.js | Minimized version. |
| _references.js | Contains a list of JavaScript files to automatically include for Visual Studio IntelliSense. The save you from having to add a JavaScript reference to all of your JavaScript files. Example:<br><br>/// <reference path="jquery-1.10.2.js" /><br><br>You can edit this file for your preferences.<br><br>See http://madskristensen.net/post/the-story-behind-_referencesjs |

Copyright © 2019 MAX Technical Training

## JavaScript Libraries Included with .Net Core MVC Razor Templates

The list of included libraries is similar to the .Net Framework templates.




The Visual Studio 2017 templates include these versions:

- Bootstrap 4.1.3
- jQuery 3.3.1
- jQuery Validation Plugin v1.17.0
- jquery.validate.unobtrusive.js 3.2.11

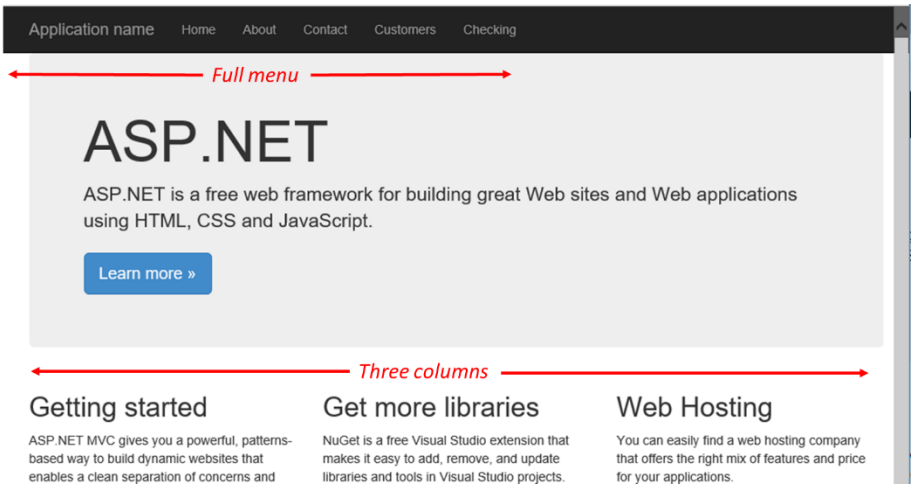
# HTML, CSS and Razor Views

The content of a Razor view is nothing more than traditional HTML, CSS, JavaScript and Razor embedded code. Plus… anything else would like to add for web development like jQuery, Angular, Knockout and Bootstrap.

The out of the box Razor views generated by MVC use Bootstrap to supply the default CSS to control the layout and display of the page.

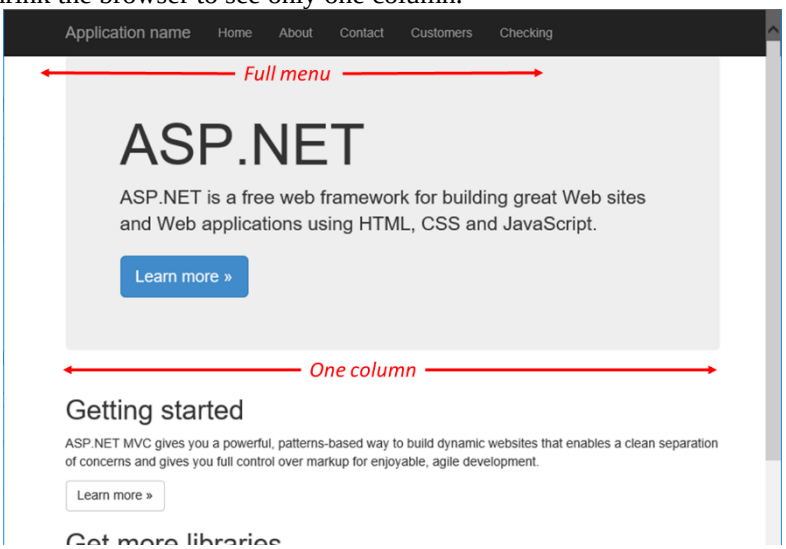
**Try this! A quick demo of bootstrap and CSS:**

1. Create a new MVC (.Net Framework) project.
2. Run the project.
3. Note the layout of the home page (Expand the width of the browser until you see the three columns across the bottom.

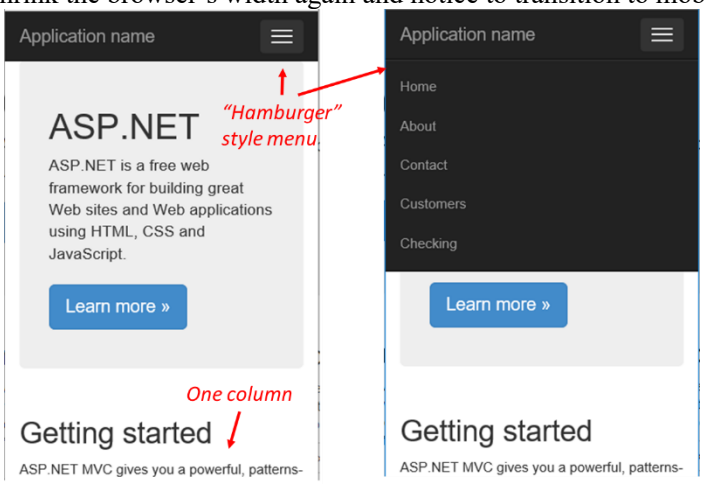



Copyright © 2019 MAX Technical Training

4. Shrink the browser to see only one column.



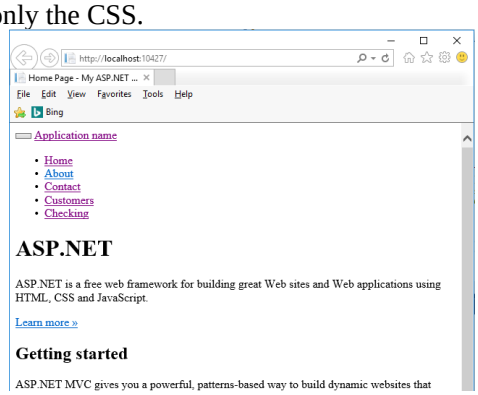

5. Shrink the browser's width again and notice to transition to mobile style display.




6. Stop the project, edit the App_Start/BundleConfig.cs file and comment out the line that bundles the Bootstrap files.

```
bundles.Add(new ScriptBundle("~/bundles/jquery").Include("~/Scripts/jquery-{version}.js"));
bundles.Add(new ScriptBundle("~/bundles/jqueryval").Include("~/Scripts/jquery.validate*"));
bundles.Add(new ScriptBundle("~/bundles/modernizr").Include("~/Scripts/modernizr-*"));
//bundles.Add(new ScriptBundle("~/bundles/bootstrap").Include("~/Scripts/bootstrap.js","~/Scripts/respond.js"));
//bundles.Add(new StyleBundle("~/Content/css").Include("~/Content/bootstrap.css","~/Content/site.css"));
```

7. Start the project note the appearance without the CSS from Bootstrap. The HTML has not changed, only the CSS.




8. To learn more about Bootstrap visit: https://getbootstrap.com/getting-started/. Play around with the size, resize the browser, etc., it's using Bootstrap of course!

Copyright © 2019 MAX Technical Training

# Creating Razor Views

You have several options for creating new Views.

- Manually.
- By using scaffolding while creating a new Controller.
- By right-clicking a View folder.
- By right-clicking an Action.

However you create the Razor View, remember:

- A Razor view is fundamentally an HTML document.
- Everything you know about HTML, JavaScript and CSS still apply.
- You can add any content to a Razor page that you can add to a generic HTML page.
- Razor code starts with "@" and is either C# or VB.NET.
- Most HTML helpers are optional but can be very valuable.
- The end result delivered from the View creation process is an HTML, CSS and JavaScript page.

## Manually

A View is just a text file with a file extension, depending on your choice of language, of .chthml or .vbhtml. The file can contain simple HTML or HTML plus Razor code. Note in the example below that the Razor code is prefixed with "@". The sample on the left is just a plain HTML file while the one on the right includes Razor code to turn it into a form with most of the data related HTML being created by "HTML helpers".

```
@{
    Layout = null;
}

<!DOCTYPE html>

<html>
<head>
    <meta name="viewport" content="width=de
    <title>View</title>
</head>
<body>
    <div>
    </div>
</body>
</html>
```

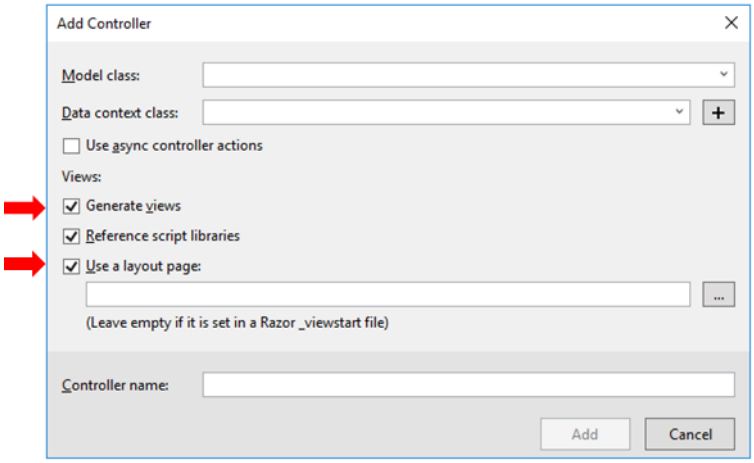
```
@model CodeFirstEFDemo.Models.Customer

@{
    Layout = null;
}

<!DOCTYPE html>

<html>
<head>
    <meta name="viewport" content="width=device-wic
    <title>View1</title>
</head>
<body>
    @using (Html.BeginForm())
    {
        @Html.AntiForgeryToken()

        <div class="form-horizontal">
            <h4>Customer</h4>
            <hr />
            @Html.ValidationSummary(true, "", new {
            <div class="form-group">
                @Html.LabelFor(model => model.somer
                <div class="col-md-10">
                    @Html.EditorFor(model => model.
                    @Html.ValidationMessageFor(mode
                </div>
            </div>
```

Note: If @{ Layout = null; } is not included in your file, the default Layouts file (default: Views\Shared_Layout.cshtml) will be loaded and your content merged where the "@RenderBody()" code is located.

Copyright © 2019 MAX Technical Training

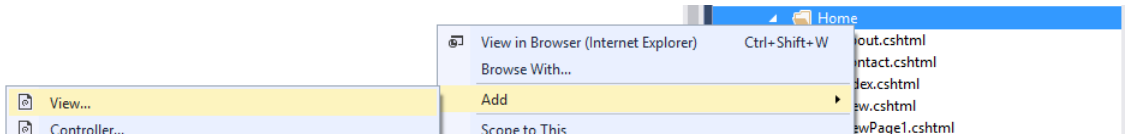
# While creating a new Controller

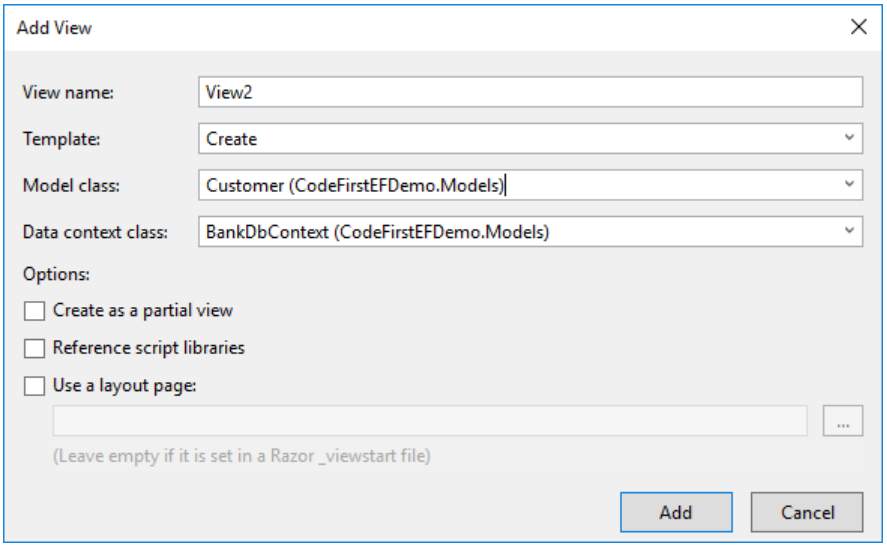
When you add a Controller to a project using the New Controller wizard you can request the creation of Views. Checking Generate views will create five Views: Create, Delete, Details, Edit and Index. Checking Use a layout page will create the Views with minimal HTML and merge the page with a layouts page (master page in Web Forms).




# Right-clicking a View folder

Right-click a View folder, click Add and View.

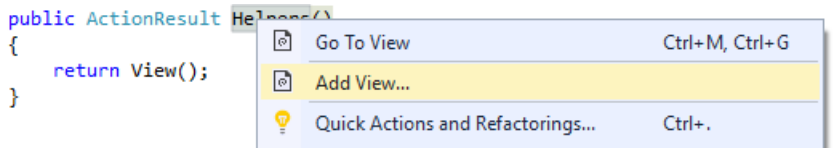



Copyright © 2019 MAX Technical Training

## Right-clicking an Action

From within a Controller, right-click an Action and click Add View. The dialog box will be prepopulated by setting the View name to the Action's name.




# Strongly Typed vs. Dynamically Typed Views

A View that starts with an @model declaration is a "strongly typed view". When you have the declaration "@model CodeFirstEFDemo.Models.Customer", Model.FirstName only works with a Customer object and will not work with an Employee object. I.e. this view only works with that one model. If your models for Customers, Employees and VendorContacts all have a core set of properties like FirstName and LastName then you might want to create a single "dynamic" View. A dynamic view is exactly the same as a strongly type view, except it is missing the @model declaration.

To create a dynamic view:

1. Write it by hand… or create a strongly typed View on one of the Models, making sure you design the view to only display or edit the common properties.
2. Delete the @model declaration!
3. Do not use the HTML helper methods! (They don't work with dynamic views.)

The View can now be used with any model that has at least the core set of properties: FirstName, LastName, etc. When you again edit the View, you may want to put the declaration back so IntelliSense will work during the editing session. Remove the declaration when you are finished with the edit, and the View is once more "dynamic".

# View Model Objects

Views can be bound, or strongly typed, on a Model type or on a collection.

For a single customer:

In the Action: `return View(db.Customers.Find(1));`

In the View: `@model CodeFirstEFDemo.Models.Customer`

For a collection of customers:

In the Action: `return View(db.Customers.ToList());`

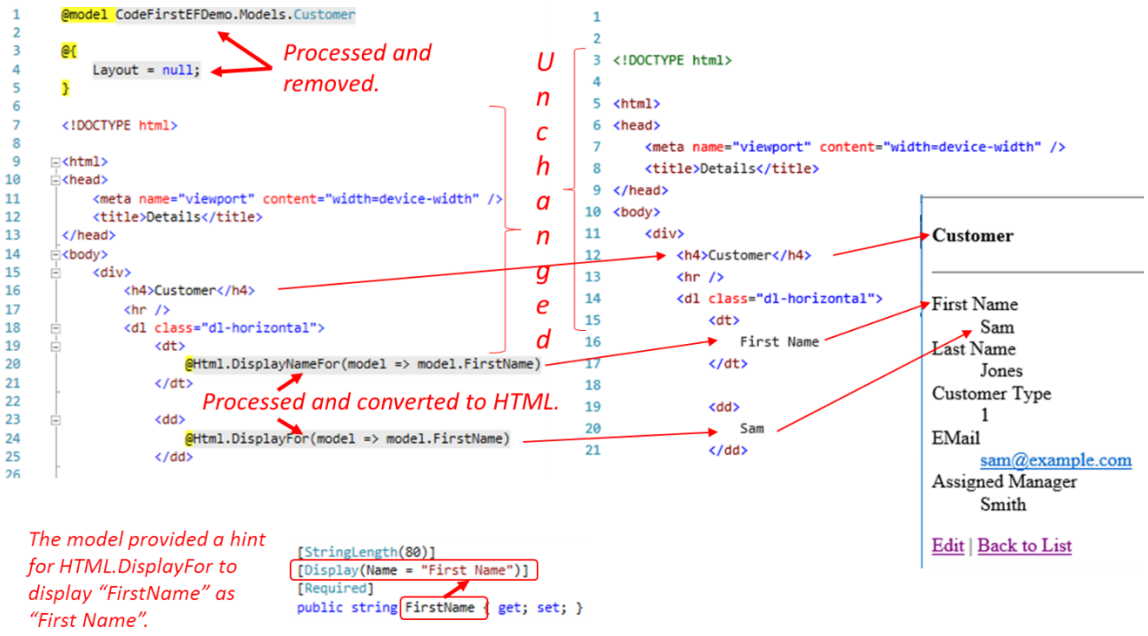
In the View: `@model IEnumerable<CodeFirstEFDemo.Models.Customer>`
This view would typically include a @foreach block to display each item in the collection.

# Embedded Razor Code

Razor code is C# or VB.NET code added to an otherwise plain HTML file. This code is parsed by the Razor engine and is removed, or replaced with HTML, before the being sent back to the user's browser.

Copyright © 2019 MAX Technical Training

Razor code is prefixed with the "@" symbol.




# Rules for embedding code

Intermixing code and content can be tricky! When something does not work as expected, be explicit and clearly mark the text as code or content.

- **Server side code starts with the "@" character**.
  - @Model.FirstName.ToUpper();

- **A comment.** Unlike an HTML or JavaScript comment, the Razor comment text will not be sent to the browser.
  - @* this is a comment *@

- **An explicit expression.** The normal Razor parsing will stop at the first white space character or unexpected character. For example, if the property "Cost" equals 1234.56 then:
  - @Model.Cost will display the cost value.
    1234.56
  - @Model.Cost * .95 will display the cost followed by "* .95"
    1234.56 * .95
  - @(Model.Cost * .95) will display the correct value.
    1172.832

- **A code block.** All of the code between the curly brackets will be processed as code with no concern for white space or unexpected switching back to HTML.
  - @{ string s = "abc"; string s2 = "Hello " + s; }
  - @{
    string s = "abc";
    string s2 = "Hello " + s;
    }

- **A switch back to HTML will occur when**:

Copyright © 2019 MAX Technical Training

- **A "<" is seen.** The following works when the <b> is in front of the "Account" text. If fails without the tag.

```
@foreach (var item in Model.CheckingAccounts)
{
    <b>Account ID: </b> @item.CheckingAccountID;
```

- **When "<text" is seen.** If a tag is not needed, you can use the pseudo tag <text> to trigger a switch back to HTML. These tags are removed before the page is sent to the browser.

```
@foreach (var item in Model.CheckingAccounts)
{
    <text>Account ID: </text> @item.CheckingAccountID;
```

- **When a line starts with "@:".** The rest of the line up to the end of the line, or another "@", will be treated and content and not code.

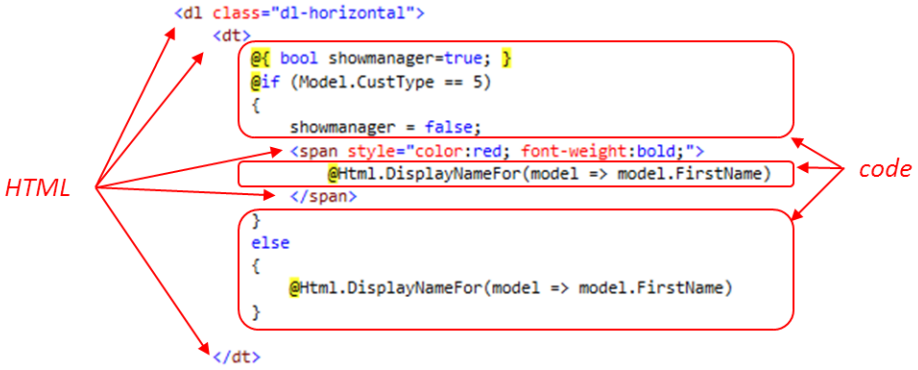
```
@foreach (var item in Model.CheckingAccounts)
{
    @:Account ID:
    @item.CheckingAccountID;
```

# Programming structures

Note that the following are actually C# style code structures. The only difference is the intermixing of code and HTML, and of course the leading "@".

Note: While it's tempting to write code in a View, business logic should be reserved for the Model, and possibly the Controller levels.

Below is an example of a C# IF block intermixed with HTML.




## Comments

You can use Razor comments anywhere in a page.

```
@* This is a comment *@
```

You can use C# comments inside of any code block.

```
@{
    // do some counting…
   for(var i = 10; i < 100; i++)  { … }
}
```

## if else

This is a standard C# IF block that allows intermixed HTML and code.

Copyright © 2019 MAX Technical Training

```
@if(condition)
{
   // code if true
}
else
{
   // code if false
}
```

## for

This is a standard C# FOR block that allows intermixed HTML and code.

```
@for(var i = 10; i < 100; i++)
{
  <b>Hello world!</b>
}
```

## foreach

This is a standard C# FOREACH block that allows intermixed HTML and code.

```
@foreach (var item in collection)
{
   // code
   <b>@item;</b>
}
```

## switch

This is a standard C# SELECT block that allows intermixed HTML and code.

```
@{ string txt = ""; }
@switch (Model.CustType)
{
   case 1:
      txt = "Business";
      break;
   case 2:
      txt = "Personal";
      break;
   default:
      txt = "Other";
      break;
}
Account type: @txt;
```

## while and do until

Guess the pattern! ☺

Copyright © 2019 MAX Technical Training

## @functions { }

You can create reusable functions and properties at the View level with the "functions" code block. You can kind of think of this as a way of extending the View as if were a partial class.

```
@functions {
    string eColor = "red";
    public string ErrorColor {
        get { return eColor; }
        set { eColor = value; }
    }

    string SayHello(string name)
    {
        return "Hello " + name + "!";
    }
}

@SayHello("Susan") <br />

@{ ErrorColor = "Green"; }
<span style="color:@ErrorColor">Error!</span>
```

# Razor HTML Helpers

Strictly speaking, the Razor HTML helpers are not necessary for the creation of Razor pages. They can add additional functionality.

Examples:

- You could use Model.property to return the value of a property, but by using Html.DisplayFor Razor can return a better display for the type of data. You can even create custom templates to further enhance the display of certain types of properties.
    - Example "IsManager" is a Boolean.
      Model.IsManager will display "true" or "false".
      Html.DisplayFor(model => model.IsManager) will display a checked or unchecked checkbox.
    - Example "UserEmail" is a string that was annotated as [DataType("EmailAddress")].
      Model.UserEmail will display as the text of the email address.
      Html.DisplayFor(model => model.UserEmail) will display as a "mailto" hyperlink:
      <a href='mailto:mike@example.com'>mike@example.com</a>

- You could type a property name like "First Name" or use Html.DisplayNameFor to check the model and return the [Display(Name="First Name")] value.
    - Example:
        - Model:
          [Display(Name = "First Name")]
          public string FirstName { get; set; }
        - View:
          Html.DisplayNameFor(model => model.FirstName)
        - Displayed in browser:
          First Name

## HTML Helpers vs. Tag Helpers

Razor supplies an HTML class with a collection of methods that create HTML tags.

Copyright © 2019 MAX Technical Training

```
@Html.LabelFor( model => model.CreateDate )
creates:  <label for="CreateDate">Create Date</label>
```

.NET Core Razor also adds Tag helpers for similar tasks. (Not every Html helper currently has a Tag helper.)

```
<label asp-for="Create Date"></label>
creates:  <label for="CreateDate">Create Date</label>
```

Out of the box Tag Helpers:

| Tag Helper | Equivalent HTML helper |
|---|---|
| Anchor | Html.ActionLink |
| Cache | n/a |
| Distributed Cache | n/a |
| Environment | n/a |
| Form | Html.BeginForm and Html.BeginRouteForm |
| Image | An image tag plus a URL helper |
| Input | Html.TextBoxFor, Html.EditorFor and others. |
| Label | LabelFor |
| Partial | Html.Partial and Html.RenderPartial |
| Select | Html.DropDownListFor and Html.ListBoxFor |
| Textarea | Html.TextAreaFor |
| Validation Message Validation Summary | Html.ValidationSummary |

For more information see:

- Tag Helpers in forms in ASP.NET Core
  https://docs.microsoft.com/en-us/aspnet/core/mvc/views/working-with-forms?view=aspnetcore-2.2
- And in the left navigation of the above page, expand “Built-in Tag Helpers”.

# Html.DisplayFor()

DisplayFor displays the value of the specified property using a default formatting or using a custom template.

```
Html.DisplayFor(model => model.IsManager)

Html.DisplayFor(model => model.UserEmail)
```

You can also specify the name of a custom template to use to format the data. The CustType field might be an integer, but you want to create a custom template the display to readable text values.

```
Html.DisplayFor(model => model.CustType,"CustTypeTemplate")
```

If the display template needs more data than just the current field then you can pass in an existing or custom object to the template.

```
Html.DisplayFor(model => model.CustType,"CustTypeTemplate", new { custid =
Model.CustomerID })
```

These additional properties are added to the ViewBag and can be accessed from within the Display Template.

Notes:

Copyright © 2019 MAX Technical Training

- If you specify a template, then MVC will look for a template by that name first in the controller's DisplayTemplates folder and then in the Views\Shared\DisplayTemplates folder. The default template is used if a custom template is not found.

# Custom Display Template

A custom Display Template is very similar to a View. You pass in data as a model and render HTML as needed.

Examples of use:

- Replace numbers or codes from the database with readable text.
- Display images or icons based on the value of a field.
- Inject CSS into the page based on a ProductCategory field.

Steps:

1. Create a folder named “DisplayTemplates” under “Views/Shared”.
2. Add a new empty view. (Right-click the new folder and click Add, New and View.)
3. Name the view something like “CustTypeTemplate” and click Add.
4. Delete the default content.
5. Add a model directive for the datatype. For this example the CustType field is an integer. As we want to allow nulls, the type will be “int?”.
6. Add the HTML and code to render the field.

```
@model int?

@{ string txt; }
@switch (Model)
{
    case 1:
        txt = "Business";
        break;
    case 2:
        txt = "Personal";
        break;
    default:
        txt = "Unknown";
        break;
}

@txt
```

7. Save the file.
8. Edit the View where you would like to use the custom template and update the Html.DisplayFor to include the template name.

```
@Html.DisplayFor(model => model.CustType, "CustTypeTemplate")
```

9. Run the project and verify the results!
   You should now see something like Customer Type: Business instead of Customer Type: 1.

Add additional properties:

10. Modify the Html.DisplayFor to include the object with the additional properties. This can be an inline custom object.

```
@Html.DisplayFor(model => model.CustType,"CustTypeTemplate",
            new { custid = Model.CustomerID, color = "red" })
```

11. Modify the Display Template to access the data from the ViewBag.

```
<span style="color:@ViewBag.color;">
    Customer ID @ViewBag.custid has a @txt account.
</span>
```

Copyright © 2019 MAX Technical Training

# Html.DisplayNameFor()

Html.DisplayNameFor retrieves the property name from the model, “FirstName” for example. If the model’s property has a [Display()] attribute that specifies either a name or an entry in a resource file, then that name will be used.


# Html.Display()

Html.Display is not bound to any property and simply displays whatever string is passed to it. You can add a second parameter to specify a Display Template to use. Html.Display becomes quite useful with the addition of a Display Template as you can format simple data in a more user friendly fashion.

# HTML Encoding and Html.Raw()

For security purposes Razor encodes text to prevent in the injection of malicious HTML or JavaScript. If a user enters HTML tag brackets they will be encoded. For example, “<” will be converted to “<”. If you otherwise validate the safeness of the text, you can then write out raw HTML using Html.Raw().

```
Html.Raw(model => model.BlogPostText)
```

# Html.ActionLink

Html.ActionLink creates <a> tags using properties for the Action and Controller.

**To create this <a> tag:**
<a href="/Products/Index">Product List</a>

type:
@Html.ActionLink("Product List", "Index", "Products") ( *displayText, action, controller* )
or:
@Html.ActionLink(linkText:="Product List", controllerName:="Products", actionName:="Index")

**Why use ActionLink when it looks like a lot more typing?**

- This would fail if the application was moved to production in a folder:
  <a href="/Products/Index">Product List</a>
- This would fail if the application was moved from a folder to the root of a site:
  <a href="Products/Index">Product List</a>

Copyright © 2019 MAX Technical Training

- The ActionLink will always create the correct URL.

**ActionLink works with variables and be changed at run time.**

```
@Html.ActionLink(item.ProductName, "Details", "Products", new { id=item.ProductID }, null )
```
( *displayText, action, controller, objectWithParameters* )

Yes, you could have also written (with the potential issue of the leading "/"):

```
<a href="/Products/Details/@item.ProductID">@item.ProductName</a>
```

**Notes:**

- When including parameters, they are passed as an object that defines the parameters found in the routes. Note that the "id=" in the ActionLink above matches to the route /{controller}/{action}/{id}, not to Model property named "id".
- The "null" parameter is needed otherwise another overload will be selected.

## How ActionLink works…

ActionLink does not just take the controller, action and the parameters and build a string separated by slashes. It take these values and searches through all of the routes and find the first one that matches. This may cause it to create a URL that's something other than what you expected, but should still work.

For example, you have the following routes defined:

```
routes.MapRoute(
    name: "AcctShortCut",
    url: "acct/{id}",
    defaults: new { controller = "customers", action = "details" },
    constraints: new { id = @"\d+" }
);

routes.MapRoute(
    name: "Default",
    url: "{controller}/{action}/{id}",
    defaults: new { controller = "Home", action = "Index", id = UrlParameter.Optional }
);
```

You create an ActionLink like this:

```
@Html.ActionLink("Details", "Details", new { id=item.CustomerID })
```

You would expect it to create a URL like:
http://yourserver/Customers/Details/1

Instead it creates this URL:
http://yourserver/acct/1

Both the custom route and the default route have defaults for the controller and the action, and both can be considered a match to the values in the ActionLink. The first rule that matches wins!

This link goes to the same destination for our example, but might not work as expected in another example. The fix? To insure that the ActionLink builds the URL you want, create another route that is more specific than the exiting default and add it before the custom route.

With the routes below, our ActionLink will now create the URL we expected.
http://yourserver/Customers/Details/

Copyright © 2019 MAX Technical Training

```
routes.MapRoute(
    name: "AcctShortCutNot",
    url: "Customer/Details/{id}",
    defaults: new { controller = "customers", action = "details" },
    constraints: new { id = @"\d+" }
);

routes.MapRoute(
    name: "AcctShortCut",
    url: "acct/{id}",
    defaults: new { controller = "customers", action = "details" },
    constraints: new { id = @"\d+" }
);

routes.MapRoute(
    name: "Default",
    url: "{controller}/{action}/{id}",
    defaults: new { controller = "Home", action = "Index", id = UrlParameter.Optional }
);
```

## Url.Action

The Url.Action helper simply builds a reliable MVC route. It is often used with <image> tags.

```
<img src="@Url.Action("GetProductPicture","Products", new { id = Model.ProductID })" />
```

This generates:

```
<img src="/Products/GetProductPicture/1">
```

Url.Action can also be used with <a> tags and anywhere else you need a URL to a Controller and Action.

```
<a href="@Url.Action("Details", "Products", new { id = Model.ProductID })" >@item.ProductName</a>
```

# HTML Helpers for Forms

## Html.BeginForm

Html.BeginForm creates a <form> tag using the same features as Url.Action and Html.ActionLink.

Html.BeginForm can be used with Html.EndForm.

```
@{ Html.BeginForm(); }
   My Form
   <!-- Normal HTML form code and/or HTML helpers here. -->
@{ Html.EndForm(); }
```

This will generate:

```
<form action="/Customers/EditView/1" method="post">
  <b>My Form</b>
  <!--Normal HTML form code and/or HTML helpers here. -->
</form>
```

Copyright © 2019 MAX Technical Training

A more common version uses a using block and does not require the EndForm method.

```
@using(Html.BeginForm())
{
   <b>My Form</b>
   <!-- Normal HTML form code and/or HTML helpers here. -->
}
```

This generates the same result:

```
<form action="/Customers/EditView/1" method="post">
  <b>My Form</b>
  <!--Normal HTML form code and/or HTML helpers here. -->
</form>
```

# Html.LabelFor

Html.LabelFor is similar to Html.DisplayNameFor, but is used in forms (<form>) to create a <label> tag and populate the "for" attribute. If the model's property has a [Display()] attribute that specifies either a name or an entry in a resource file, then that name will be used.

```
@Html.LabelFor(model => model.FirstName)
```

Will render this HTML: <label for="FirstName">FirstName</label>

Or if the model has a [Display(Name="First Name")] attribute then:
<label for="FirstName">First Name</label>

**.Net Core Tag Helper Equivalent:**

```
<label asp-for="Create Date"></label>
```

# Html.EditorFor

Html.EditorFor is similar to Html.DisplayFor, but is used in forms (<form>) to create datatype appropriate form controls. For example, a Boolean property would be displayed as an <input type="check">. You can also create Custom Editor Templates using steps that are very similar to Custom Display Templates.

- Simple text types display as <input type="text" name="*propertyname*">
- Boolean properties display as a checkbox.
- Model properties marked as [DataType(DataType.MultilineText)] will be rendered as a <textarea>.

```
@Html.EditorFor(model => model.HireDate)

@Html.EditorFor(model => model.HireDate,"MyCustomDatePickerTemplate")
```

# Custom Editor Templates

You can create, or find on the web, custom editors to improve the user experience and validate data as it is typed. A date picker is a very good example. You can create full custom code, or wrap existing tools like jQueryUI in a Razor template.

These templates are stored in Views/Shared/EditorTemplates.

Copyright © 2019 MAX Technical Training

Here's a few places to start:

- Using Display Templates and Editor Templates in ASP.NET MVC
  http://www.codeguru.com/csharp/.net/net_asp/mvc/using-display-templates-and-editor-templates-in-asp.net-mvc.htm
- Extending Editor Templates for ASP.NET MVC (includes a jQueryUi date picker example)
  https://www.simple-talk.com/dotnet/asp-net/extending-editor-templates-for-asp-net-mvc/

# Html.DropDownListFor

One of the really useful helpers can generate a dropdown list by just supplying a collection objects that have a Value property and a Text property. This collection can be passed to the view from the controller using either the ViewBag or as a property of a View Model.

Without the dropdown:

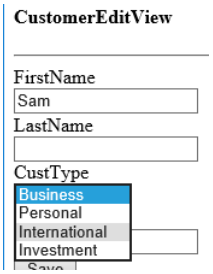

With a little work in the View file you can add a dropdown list! Just replace:

```
@Html.EditorFor(model => model.CustType, new { htmlAttributes = new { @class = "form-control" } })
```

with:

```
@Html.DropDownListFor(model => model.CustType,
    Model.custTypes.Select(s => new SelectListItem
    {
        Value = s.CustomerTypeId.ToString(),
        Text = s.Name
    }))
```

And you will get a dropdown with no additional HTML work!




# Addition HTML Helpers for Forms

Here's a list of the HTML helpers for form controls. We leave it up to you to do a web search for examples.

- Html.TextBoxFor
- Html.TextAreaFor
- Html.PasswordFor
- Html.HiddenFor

Copyright © 2019 MAX Technical Training

- Html.CheckBoxFor
- Html.RadioButtonFor
- Html.DropDownListFor
- Html.ListBoxFor

## Custom HTML Helpers

You can create your own Razor HTML helpers by either creating a class or an extension method.

See: https://docs.microsoft.com/en-us/aspnet/mvc/overview/older-versions-1/views/creating-custom-html-helpers-cs

## The Anti-Forgery Token

When working with forms that update data you want to make sure that a hacker does not "replay" a form post or try a direct attack. The ValidateAntiForgeryToken attribute verifies that the anti-forgery token value has not expired. This token is used to prevent cross site request forgery attacks and is stored both in an HTTP-only cookie and an <input type="hidden"> tag. To use, add the attribute to your Action and add an HTML helper, @Html.AntiForgeryToken(), to your View's <form> section.

Two edits are necessary:

- Add the ValidateAntiForgeryToken attribute to the Actions that change data.

```
[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult Delete(int id, FormCollection collection)
{
```

- Add an HTML helper to your form.

```
@using(Html.BeginForm())
{
    @Html.AntiForgeryToken()

    <b>My Form</b>
    <!--Normal HTML form code and/or HTML helpers here. -->
}
```

This will send a cookie to the browser and add a hidden <input> tag to your page.

```
<form action="/Customers/EditView/1" method="post">
  <input name="__RequestVerificationToken" type="hidden" value="c1hx9_TKbmA2S3CjI62Pcn1cA1o
  ugjy6Lt5Qwy8Pgr50c8LgkfwHmxiu4NR4CQbQXTFY2BqIIcfatRCt-_kn4v7AVwew0cWTREfx17U0r4g1" />
  <b>My Form</b>
  <!--Normal HTML form code and/or HTML helpers here. -->
</form>
```

Also see:

- XSRF/CSRF Prevention in ASP.NET MVC and Web Pages
  https://docs.microsoft.com/en-us/aspnet/mvc/overview/security/xsrfcsrf-prevention-in-aspnet-mvc-and-web-pages

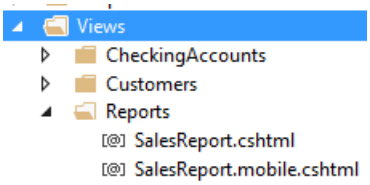
# Mobile-Specific Views

There are two strategies for building mobile views. One is to build a view that will work well in most devices by using an adaptive design using tools like Bootstrap. The second is to build two versions of each

Copyright © 2019 MAX Technical Training

view, one for mobile and one for non-mobile. The single view, "mobile first", is generally considered the best practice. Some pages though, reports for example, may work best by creating both a mobile and non-mobile view.

Mobile views are often created by copying an existing view and then renaming it.




**The non-mobile View**

- This is the default View!
  /Views/Reports/SalesReport.cshtml

**The mobile View**

- This is a View file with the addition of "mobile" to its name.
  /Views/Reports/SalesReport.**mobile**.cshtml

MVC detects the type of device the user has by looking at the User Agent String sent by the browser. Out of the box this capability is very limited and not 100% reliable.

ASP.NET's default method of "guessing" what is a mobile device can be improved by adding your own detection. See this article for more on this topic:
https://docs.microsoft.com/en-us/azure/app-service-web/web-sites-dotnet-deploy-aspnet-mvc-mobile-app#a-namebkmkbrowserviewsa-create-browser-specific-views

## Testing Mobile Views

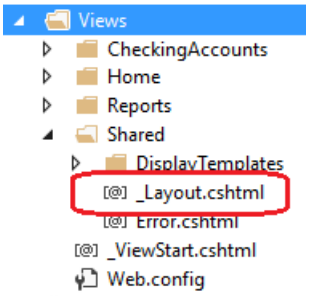
Options:

- Publish your project to an internet facing server and use a mobile device to access the site. Microsoft's Azure is an easy way to do this.
- Publish your project to an internet facing server and use an internet emulation service. Examples:
  - Turbo.Net Browser Sandbox https://turbo.net/browsers
  - BrowserStack (includes Mobile Browser Emulators) https://www.browserstack.com/
  - http://www.mobilephoneemulator.com/
- Use the User Agent String "emulation" options in Internet Explorer's F12 tools or the Chrome "Google Chrome DevTools".
- Use a downloadable emulator for a specific device.
  - iPad and iPhone: http://www.electricplum.com/studio.aspx
  - A list of emulators: https://docs.microsoft.com/en-us/aspnet/mobile/device-simulators
  - Note: Downloadable emulators often run as virtual machines and will not run inside of another virtual machine like Hyper-V or VirtualBox.

# Layouts

ASP.NET Web Form projects have the concept of a "master page" that contains the HTML, JavaScript and CSS that is common to all, or most, of the pages in our project. In Razor this same concept is called "layouts". When you create a new MVC project it adds a sample layout page.

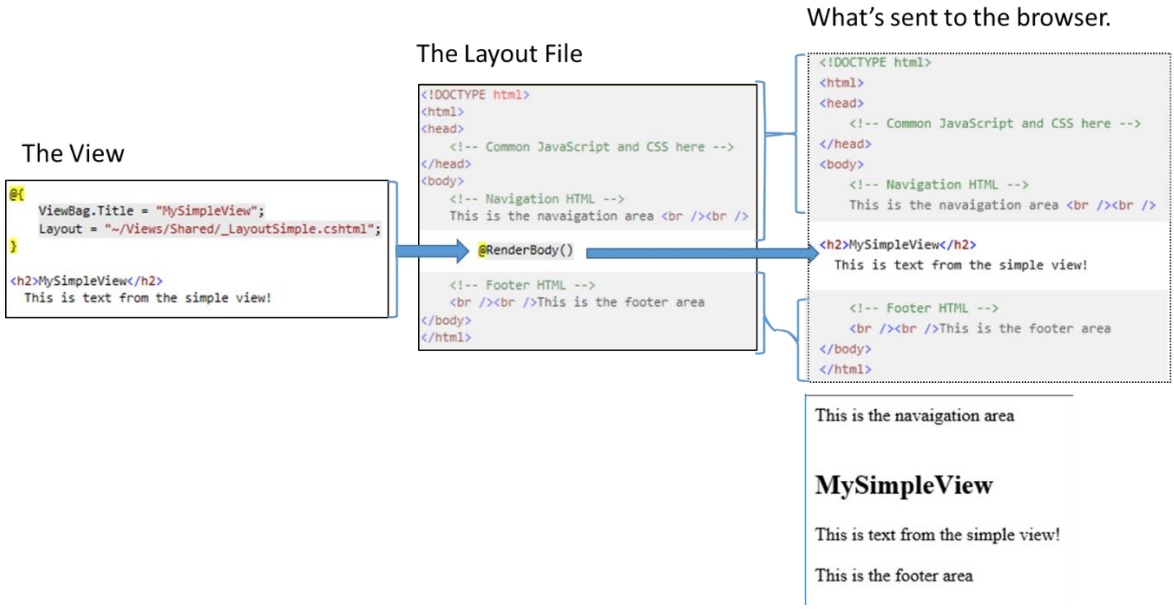
Copyright © 2019 MAX Technical Training




By convention, the default layouts page is stored in /Views/Shared and is named "_Layout.cshtml". The simplest layout page might look something like this:

```
<!DOCTYPE html>
<html>
<head>
    <!-- Common JavaScript and CSS here -->
</head>
<body>
    <!-- Navigation HTML -->
    This is the navaigation area <br /><br />

        @RenderBody()

    <!-- Footer HTML -->
    <br /><br />This is the footer area
</body>
</html>
```

The content of the View that uses this layout file will be inserted using the **@RenderBody()** method.




# Options for specifying a layout file

- Add an entry to the _ViewStart.cshtml file in the /Views folder. This will be the default to be used for any page that does not specify its own layout file.

```
@{
    Layout = "~/Views/Shared/_Layout.cshtml";
}
```

Copyright © 2019 MAX Technical Training

- Add an entry to the top of each View.

```
@{
    Layout = "~/Views/Shared/_LayoutSimple.cshtml";
}
```

- Add an entry to the top a View to not use any layout file. If the View does not specify its own layout file, or add "Layout = null", then the default listed in the _ViewStart file will be used.

```
@{
    Layout = null;
}
```

If your View is using a layout then:

- It should not have <html>, <head> or <body> tags as these are in the layout file.
- The Model object passed to the View is available to the layout.
- Data can be exchanged between the Controller, the View and the layout file through the ViewBag. Example: The View set the title text to be displayed in the layout file.

```
@{
    ViewBag.Title = "Contact";
}
```

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>@ViewBag.Title - My ASP.NET Application</title>
    @Styles.Render("~/Content/css")
    @Scripts.Render("~/bundles/modernizr")
</head>
<body>
```

Notes:

- Prior to processing a View file, the _ViewStart file is loaded and processed.
- The _ViewStart file is not loaded for Partial Views.

# Partial Views

Partial Views are created primarily to have reusable blocks of HTML and code that can be used in other Views. They can be as simple as one line of text or HTML, or a complete shopping cart. Examples might include a stock ticker, the special of the day or a legal disclaimer that is used on every page that asks for an email address.

By convention, Partial Views are named starting with an underscore. _StockPrice.cshtml

Notes:

- _viewstart.cshtml is not executed for Partial Views.
- Partial Views do not use layout pages, but the page that uses the Partial View can.
- Data can be passed between the Controller, the View, the layout file and the Partial View using the ViewBag.
- A common location for Partial Views that will be used throughout the application is the /Views/Shared folder.

## A simple Partial View

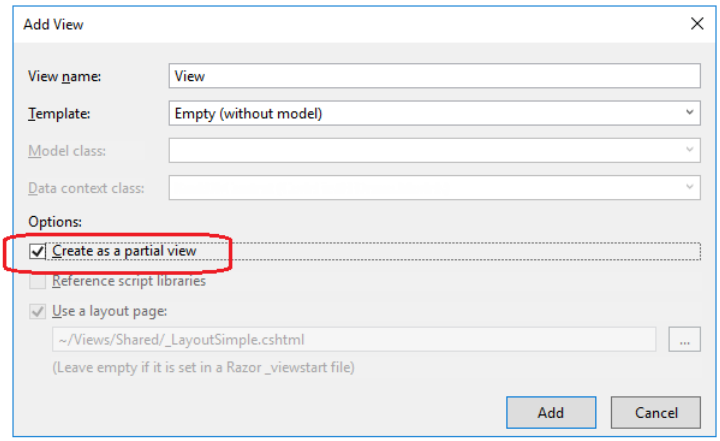
Notice that there nothing more here than needed. The  may include a class or id attribute so the parent View can style it.

Copyright © 2019 MAX Technical Training

```
@{
    // code to call a web service to get the stock price
    string price = "$123.45";
}

<span class="stockPrice">The current stock price is @price</span>
```

# Creating a Partial View

Just like normal Views, a Partial View is just a text file with HTML, and optionally code and Razor HTML helpers. You can also create a Partial View using the wizards. Complete the form as usual and checkmark Create as a partial view.




# Using a Partial View

For a Partial View that uses no model, or the same model as the parent, use Html.Partial to load it.

A simple Partial Page can be loaded with the Html.Partial helper.

```
@Html.Partial("_StockPrice")
```

If the Partial View needs data from the parent view, you can pass an object or a simple data type.

```
@Html.Partial("_StockPrice",StockData)
```

If the Partial View needs its own model, or its own caching, then:

1. Create an Action in a Controller.
2. Optionally add an attribute to set caching options.
3. Optionally add the [ChildActionOnly] attribute to prevent the action from being called directly from a route. (i.e. from a URL in a browser.)
4. Create the object.
5. Return using the PartialView method.

```
[ChildActionOnly]
public ActionResult StockInfo()
{
    stock stk = new stock();
    // set properties
    return PartialView(stk);
}
```

6. Access the Partial View using: @{ Html.RenderAction("action","controller") }

Copyright © 2019 MAX Technical Training

# Best Practices

- Don’t directly access a database server from a View, that’s what Models are for!
- Avoid adding any business logic to your views. (“if” statements in a view are often hints of business logic.)
- Use View Models to pass only the data needed by the View.

Copyright © 2019 MAX Technical Training

# Module 7 – Adding Charts to MVC Projects

## Charts

While MVC does not have features for charts and graphs, the ASP.NET Chart class and third party charting tools are easy additions to an MVC project. Here we will take a look at the server-side .NET Chart class and a client-side open source JavaScript project.

### Server-side Charts

- Requires a server request for chart refreshes.
- Can be saved and cached server-side.
- Do not require the download of data.
- No browser decencies other than the ability to display an image.
- No additional JavaScript or CSS files to load.

### Client-side Charts

- No server-side processing and overhead.
- Data can be dynamic and updated from JavaScript.
- Chart can be interactive, depending on the JavaScript library.
- Possible browser dependencies. (The example we use depends on the HTML5 canvas tag.)
- Possible licensing of the charting tools. (Not everything is free!)

## System.Web.Helpers.Chart

The System.Web.Helpers namespace contains a Chart class that is based on the ASP.NET Chart control. The Chart class supports 35 chart types and can be styled using themes.

Most of the Microsoft documentation for System.Web.Helpers.Charts is for use in standalone cshtml pages and not for MVC Views. The examples typically use the .Write() method while in an MVC controller you will want to return an image file using the .GetBytes() method. While you will need to modify the sample code a bit, the links below do show the features of the Charts object.

- Chart Class
  https://docs.microsoft.com/en-us/previous-versions/aspnet/web-frameworks/gg538296(v=vs.111)
  https://msdn.microsoft.com/library/system.web.helpers.chart(v=vs.99)
- Displaying Data in a Chart with ASP.NET Web Pages (Razor)
  https://docs.microsoft.com/en-us/aspnet/web-pages/overview/data/7-displaying-data-in-a-chart

Copyright © 2019 MAX Technical Training

The Chart object has two required and two optional parameters in its constructor:

- int width,
- int height,
- string theme, (optional) Blue, Green, Vanilla, Yellow and Vanilla3D
- string themePath (optional)

If both the theme and themePath are provided, only the theme parameter will be used.

```
var myChart = new Chart(width: 600, height: 400, theme: ChartTheme.Blue)
```

The Chart object has three optional properties.

- Height (pixels)
- Width (pixels)
- ChartImage (name of background image)

The Chart object has numerous methods to define the chart. Only a few are listed here.

- AddLegend(title, uniqueName) *The name is optional.*
  .AddLegend("Categories")
- AddTitle(title, uniqueName) *The name is optional.*
  .AddTitle("Sales by Category")
- AddSeries(
  string name, *optional*
  string chartType, *(see table below)*
  string chartArea,
  string axisLabel,
  string legend,
  int markerStep, *how often to display data point markers. (default value = 1)*
  IEnumerable xValue, *the data to plot (often an array, but can be bound to an object collection)*
  string xField,
  IEnumerable yValues, *the data to plot (often an array, but can be bound to an object collection)*
  string yFields
  )
  - The above parameters are documented here: https://docs.microsoft.com/en-us/previous-versions/aspnet/web-frameworks/gg547988%28v%3dvs.111%29
  - Also review how some of these are used in the examples that follow.
  - If multiple Series are supplied, the "xValue" labels will be the same for each series, and only need to be supplied for the first Series.
  - When using more than one Series, the "chartType" will usually be the same. While you can mix line and bar chart types, you cannot mix bar and column chart types in the same chart

# Chart Types

The Chart class includes 35 kinds of charts, and most can also be displayed as a 3D chart.

Descriptions can be found here: https://msdn.microsoft.com/en-us/library/dd489233.aspx

| | | | | |
|---|---|---|---|---|
| Area | Bar | BoxPlot | Bubble | Candlestick |
| Column | Doughnut | ErrorBar | FastLine | FastPoint |
| Funnel | Kagi | Line | Pie | Point |

Copyright © 2019 MAX Technical Training

| | | | | |
|---|---|---|---|---|
| PointAndFigure | Polar | Pyramid | Radar | Range |
| RangeBar | RangeColumn | Renko | Spline | SplineArea |
| SplineRange | StackedArea | StackedArea100 | StackedBar | StackedBar100 |
| StackedColumn | StackedColumn100 | StepLine | Stock | ThreeLineBreak |

# Creating a Chart from an Action

You can generate a chart and then return it from an Action as a JPEG or other web format. The chart can then be used in an <IMG> tag.

Basic steps for creating a chart:

- Create an Action in a Controller.
- Retrieve data (unless hardcoded)
- Create the chart object.
- Return a File ActionResult.
- Consume the data from an IMG tag.

## An Action to Create a Chart

The sample Action below creates a Chart object and then returns it as a JPEG file type.

Notes:

- The data is hardcoded as inline arrays (new[] { "Toys", "Hobbies", … }).
- This chart has two data series.
- The "xValue" in the second series could be removed as is will be ignored.
- If the multiple series do not have the same number of elements, the first series must have the largest number of elements.
- Numeric values can be passed as numbers or strings. Both of the following examples are valid.
  yValues: new[] { "11250", "8900", "32505", "10520", "46230" }
  yValues: new[] { 11250, 8900, 32505, 10520, 46230, 10000 }

```
using System.Web.Helpers;
…
public ActionResult Chart1()
{
    var myChart = new Chart(width: 800, height: 400, theme: ChartTheme.Blue)
        .AddTitle("Sales by Category")
        .AddLegend("This is the legend")
        .AddSeries(
            chartType: "column",
            xValue: new[] { "Toys", "Hobbies", "TVs", "Computers", "Appliances" },
            yValues: new[] { "11250", "8900", "32505", "10520", "46230" },
            name: "2017")
        .AddSeries(
            chartType: "column",
            xValue: new[] { "Toys", "Hobbies", "TVs", "Computers", "Appliances" },
            yValues: new[] { "12125", "8593", "37250", "7752", "48123" },
            name: "2018");

    return File(myChart.GetBytes("jpeg"), "image/jpeg");
}
```

Copyright © 2019 MAX Technical Training

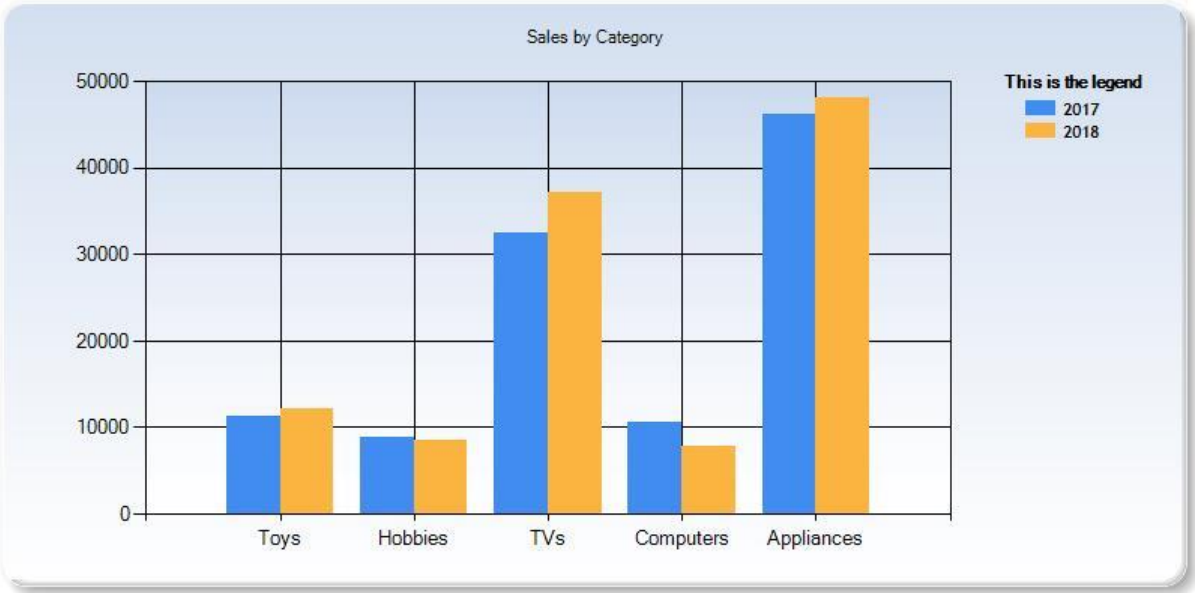
## Consuming the Action's Result

In a Razor View use either of the following IMG tags.

```
<img src="/Home/Chart1" />
```

or

```
<img src="@Url.Action("Chart1","Home")" />
```




## Using Data from a Model

The Chart class supports data binding by supplying multiple arrays for the X and Y values or a single collection of objects along with a column/property name.

### Arrays

Building individual arrays for the X and Y axis data can be useful when the data for both cannot be derived from the same data source or query. Here's an example where the text on the X axis is hardcoded text and the data for the Y axis is from a data source converted to an array.

```
public ActionResult Chart5()
{
   var vendprod = from v in db.Vendors
                  where v.Products.Count() > 0
                  select new { Vendor = v.Name, Count = v.Products.Count() };
   var xArray = new[] { "FMA", "SBS", "CFM" };   // HARDCODED TEXT!
   var yArray = vendprod.Select(v => v.Count).ToArray();

   var myChart = new Chart(width: 600, height: 400, theme: ChartTheme.Blue)
                  .AddTitle("Product Count by Vendor")
                  .AddSeries(
                     xValue: xArray,
                     yValues: yArray);

   return File(myChart.GetBytes("jpeg"), "image/jpeg");
```

Copyright © 2019 MAX Technical Training

```
}
```

## Data Binding

When data is available as a collection, such as the result of a LINQ query, you can bind the collection to the xValue and yValue series properties.

If the collection is named "vendprod" and the two properties for the chart are named "Vendor" and "Count" you can bind the data like this:

```
.AddSeries(
   xValue: vendprod, xField: "Vendor",
   yValues: vendprod, yFields: "Count");
```

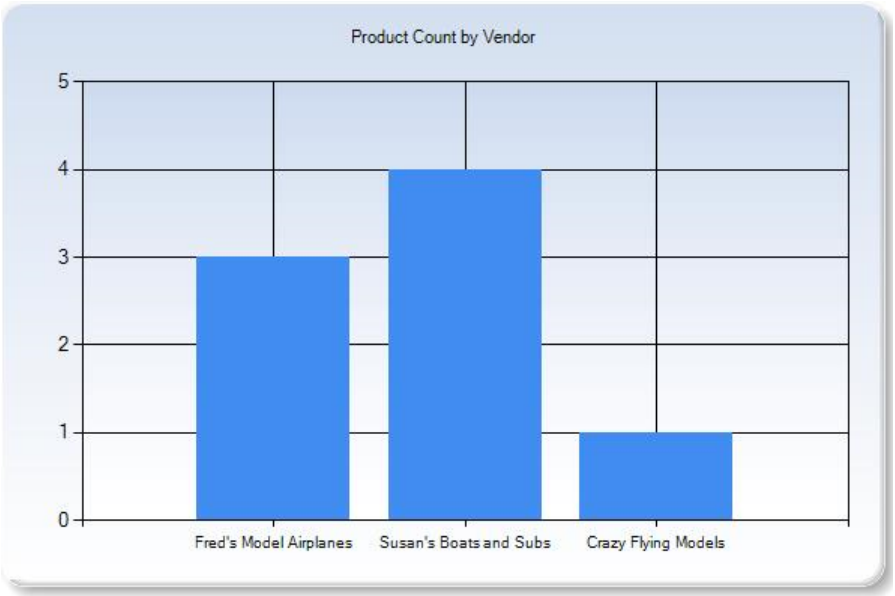
Here are three examples of LINQ queries that return a list of vendors and a count of products purchased from each vendor. The first two work with two entities that do not have a defined relationship. (using a JOIN) The third example works when the two entities have a defined relationship, and uses Vendors.Products to drilldown to the Products data.

```
var vendprod = from p in db.Products
               join v in db.Vendors on p.VendorID equals v.VendorID into pv
               from data in pv
               group data by data.Name into g
               select new { Vendor = g.Key, Count = g.Count() };

var vendprod = from v in db.Vendors
               join p in db.Products on v.VendorID equals p.VendorID into vp
               select new { Vendor = v.Name, Count = vp.Count() };

var vendprod = from v in db.Vendors
               where v.Products.Count()>0
               select new { Vendor = v.Name, Count = v.Products.Count() };
```

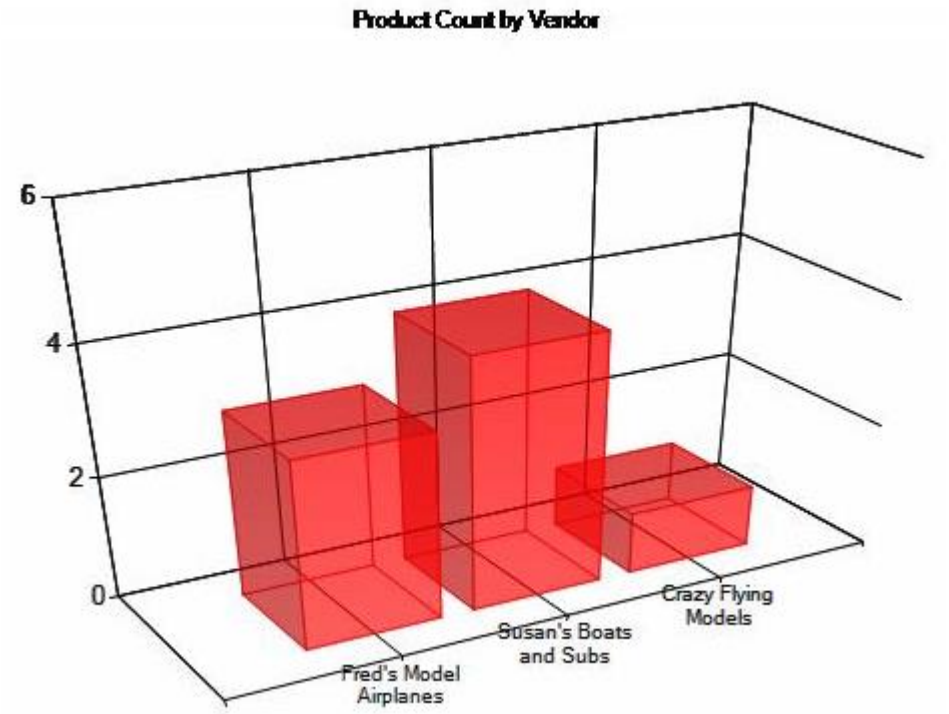
Here's the complete Action using one of the above LINQ queries to get the Vendors and the Product counts. As no chart type has been specified, it defaults to a bar chart.

```
using System.Web.Helpers;
…
public ActionResult VendorProductCountChart()
{
   var vendprod = from v in db.Vendors
                  where v.Products.Count()>0
                  select new { Vendor = v.Name, Count = v.Products.Count() };

   var myChart = new Chart(width: 600, height: 400, theme: ChartTheme.Blue)
                  .AddTitle("Product Count by Vendor")
                  .AddSeries(
                     xValue: vendprod, xField:"Vendor",
                     yValues: vendprod, yFields: "Count");

   return File(myChart.GetBytes("jpeg"), "image/jpeg");
}
```

Copyright © 2019 MAX Technical Training




Here's the 3D version you get by setting the theme to 3D (theme: ChartTheme.Vanilla3D).




# Client-side Charts

If you do a quick web search for "JavaScript charts" you will find a large number of free and commercial charting libraries. Most of these libraries use the HTML5 canvas and SVG features to create the charts. A few have fallbacks to support older browsers.

For an example of using JavaScript charts in an MVC project we will use the ChartJS library. This library uses the HTML5 canvas feature.

Copyright © 2019 MAX Technical Training

- https://www.chartjs.org/

Basic steps:

- Download ChartJS and add it to your project, or link to a CDN hosting ChartJS such as:
  https://cdnjs.cloudflare.com/ajax/libs/Chart.js/2.7.2/Chart.js
  (other options here: https://cdnjs.com/libraries/Chart.js)
- Create an Action in a Controller that loads a View with the chart code.
- If using live data, add code to the Action to retrieve the data.
- Create a View or PartialView for the chart code.
- Create an HTML5 canvas tag:
  `<canvas id="myChart" width="600" height="400"></canvas>`
- Create a context object for the canvas:
  `var ctx = document.getElementById("myChart").getContext('2d');`
- Create the chart object. (see example below)

# An Example with Hardcoded Data

This first example is a very simple chart with the minimal configuration needed.

## The Action

```
public class ChartsController : Controller
{
    // Hardcoded data
    public ActionResult ChartJS1()
    {
        return View();
    }
}
```

## The View

Add the following HTML and JavaScript to your View or PartialView. This example is named "ChartJS1.cshtml" to match the Action's name.

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/Chart.js/2.7.2/Chart.min.js"></script>

<canvas id="myChart" width="600" height="400"></canvas>

<script>
    var ctx = document.getElementById("myChart").getContext('2d');

    var myChart = new Chart(ctx, {
        type: 'bar',
        data: {
            labels: ["Toys", "Hobbies", "TVs", "Computers", "Appliances"],
            datasets: [{
                label: 'Sales by Category',
                data: ["12125", "8593", "37250", "7752", "48123"],
                borderWidth: 1
```

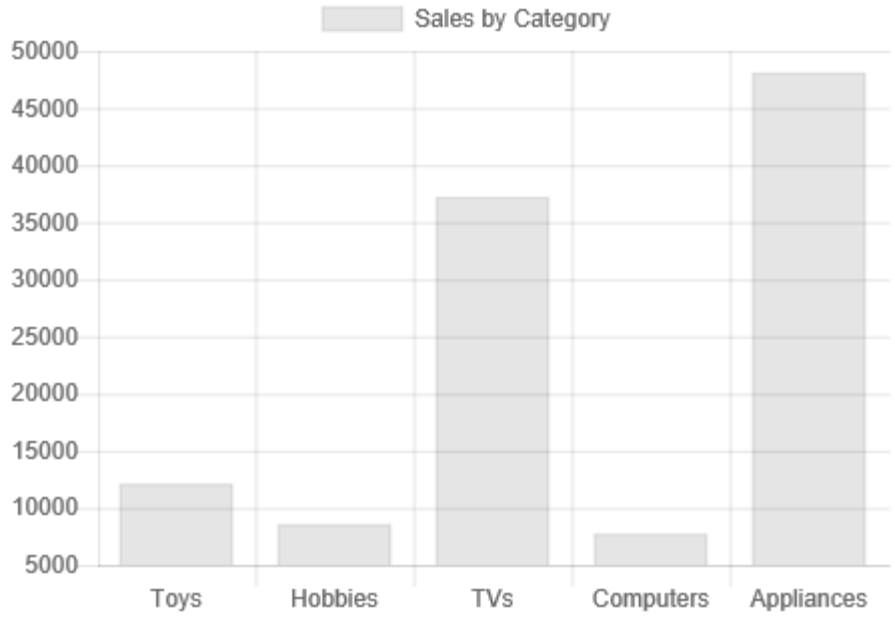
Copyright © 2019 MAX Technical Training

```
            }]
        }
    });
```

```
</script>
```

Notes:

- The Chart.js file can be loaded from a local copy in your project or a CDN.
- This example has only one data range (datasets).




## Chart Types

The core types include: line, bar, radar, polar area, doughnut, pie, bubble, scatter, area and mixed. For a list see: http://www.chartjs.org/samples/latest/

## Colors

You can supply colors using any of the common HTML color conventions:

- "red"
- "#ff0000"
- "rgba(255, 0, 0)"

To assign colors to the individual bars in the previous chart example, add the following in the “datasets” section:

```
backgroundColor: [ "red", "blue", "green", "yellow", "cyan"  ],
```

To assign a color to the entire dataset:

```
backgroundColor: "red" ,
```

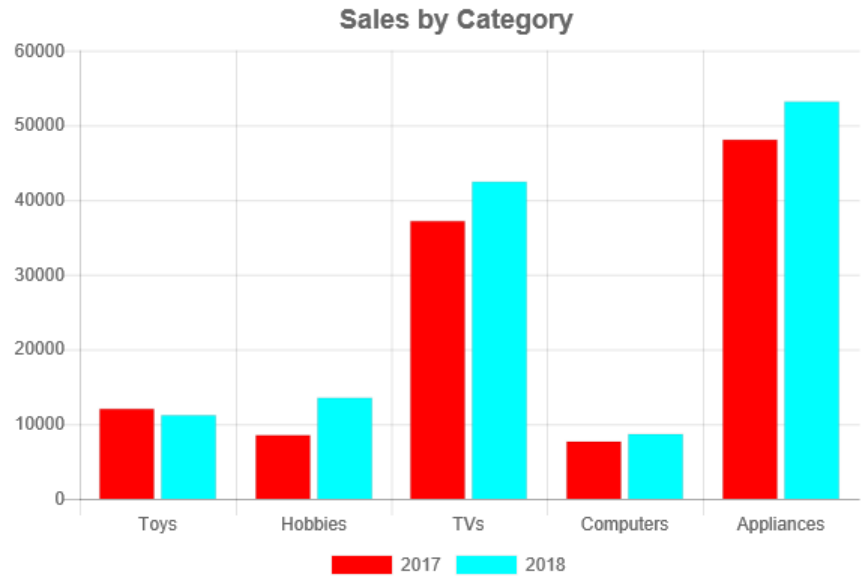
Copyright © 2019 MAX Technical Training

## Options

There are a large number of options for formatting the chart, axis and data (bars, etc.) In the example below, we have added options for “responsive”, “legend”, “title” and “scales”. (The default for “responsive” is true.)

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/Chart.js/2.7.2/Chart.min.js"></script>

<canvas id="myChart" width="600" height="400"></canvas>

<script>
   var ctx = document.getElementById("myChart").getContext('2d');

   var myChart = new Chart(ctx, {
      type: 'bar',
      data: {
         labels: ["Toys", "Hobbies", "TVs", "Computers", "Appliances"],
         datasets: [
            {
               label: '2017',
               data: ["12125", "8593", "37250", "7752", "48123"],
               backgroundColor: "red",
               borderWidth: 1
            },
            {
               label: '2018',
               data: ["11250", "13593", "42501", "8752", "53230"],
               backgroundColor: "cyan",
               borderWidth: 1
            }
         ]
      },
      options: {
         responsive: true,
         legend: {
            position: 'bottom',
         },
         title: {
            display: true,
            text: 'Sales by Category',
            fontSize: 18
         },
         scales: {
            yAxes: [{
               ticks: {
                  beginAtZero: true
               }
            }]
         }
      }
   });

</script>
```

Copyright © 2019 MAX Technical Training




# An Example with Live Data

To pass data to a JavaScript chart:

- Create an array of strings for the labels. (x axis)
- Create an array for each dataset.
- Convert the arrays to JSON.
- Pass the JSON content to the View using either a ViewBag or a custom object.
- Insert the JSON content using an HTML helper.
  labels: @Html.Raw(ViewBag.xArray),

You could also access the data for the chart from the View using an AJAX call to a WebAPI web service. The Action in the Controller would then only need to return the View.

## The Action

Here's a sample Action that collects these data using a LINQ query and then converts the data to JSON. (There are multiple ways of doing both of these steps.)

```
// Live data
public ActionResult ChartJS2wData()
{
    var vendprod = from v in db.Vendors
                   where v.Products.Count() > 0
                   select new { Vendor = v.Name, Count = v.Products.Count() };
    var xArray = vendprod.Select(v => v.Vendor).ToArray();
    var yArray = vendprod.Select(v => v.Count).ToArray();

    var js = new JavaScriptSerializer();  // using System.Web.Script.Serialization;
    ViewBag.xArray = js.Serialize(xArray);  // labels
    ViewBag.yArray = js.Serialize(yArray);

    return View();
```

Copyright © 2019 MAX Technical Training

```
}
```

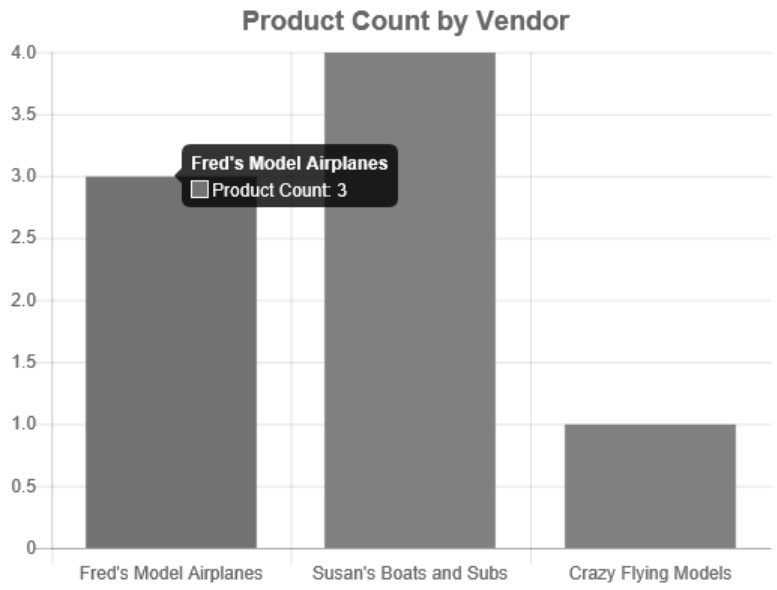
## The View

Here's the chart code:

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/Chart.js/2.7.2/Chart.min.js"></script>

<canvas id="myChart" width="600" height="400"></canvas>

<script>
    var ctx = document.getElementById("myChart").getContext('2d');

    var myChart = new Chart(ctx, {
        type: 'bar',
        data: {
            labels: @Html.Raw(ViewBag.xArray),
            datasets: [
                {
                    label: 'Product Count',
                    data: @Html.Raw(ViewBag.yArray),
                    backgroundColor: "gray",
                    borderWidth: 1
                }
            ]
        },
        options: {
            responsive: false,
            legend: {
                display: false,
            },
            title: {
                display: true,
                text: 'Product Count by Vendor',
                fontSize: 18
            },
            scales: {
                yAxes: [{
                    ticks: {
                        beginAtZero: true
                    }
                }]
            }
        }

    });
</script>
```

And here's the result.

Copyright © 2019 MAX Technical Training

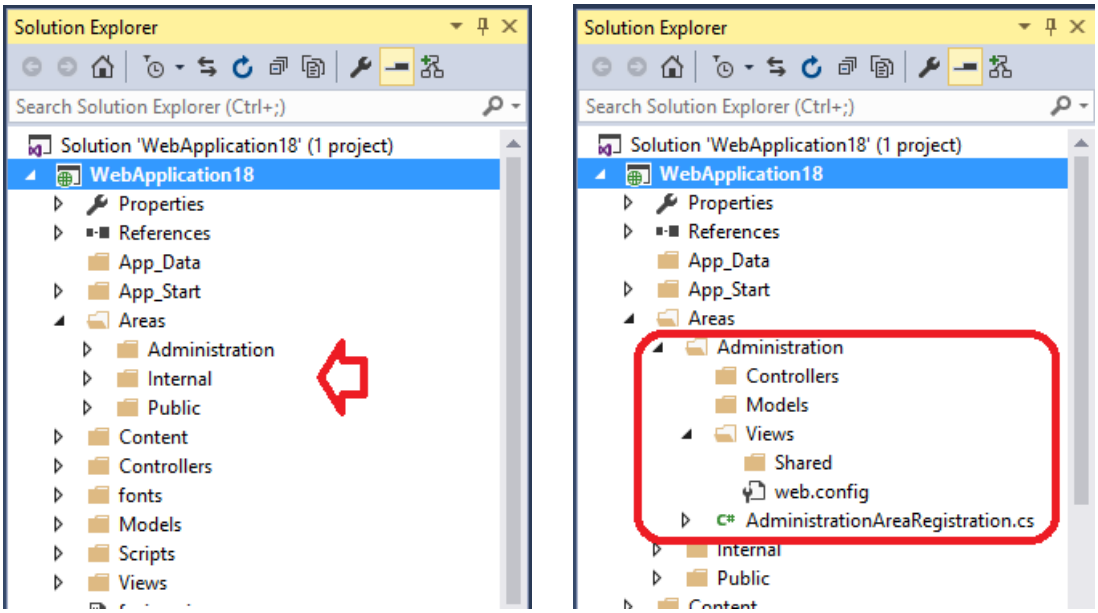



Copyright © 2019 MAX Technical Training

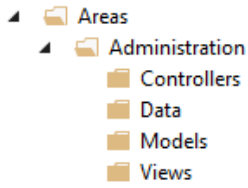
# Module 8 – MVC Areas

## Areas

Larger projects tend to have separate areas of responsibility, which MVC supports as Models, Views and Controllers. As applications grow, your Models, Views and Controllers folders will start to collect a large number of items and will become more difficult to manage. To better support larger and more complex projects, MVC supports the concept of Areas to break a project into sections.




.NET Core areas are similar:




.Net Core notes:

- Controllers in Areas need an Area attribute:

```
[Area("Products")]
public class DemoController : Controller
{
```

- Route templates need an Area section:

```
routes.MapRoute(
      name: "MyArea",
      template: "{area:exists}/{controller=Home}/{action=Index}/{id?}");
```

Copyright © 2019 MAX Technical Training

a named area example:

```
routes.MapRoute(
      name: "AdminRoute",
      template: "Administration/{controller=Home}/{action=Index}/{id?}");
```

Each Area has its own collections of Models, Views and Controllers.

For .Net Framework, each area also has its own routing configuration in a file named *areaname*AreaRegistration.cs. This code is called from Global.asax from "AreaRegistration.RegisterAllAreas()".

```
public override void RegisterArea(AreaRegistrationContext context)
{
    context.MapRoute(
        "Administration_default",
        "Administration/{controller}/{action}/{id}",
        new { action = "Index", id = UrlParameter.Optional }
    );
}
```

If you were to look at a sample routing for the above example project with RouteDebugger (see Module 3) you would see that the Area routes are being checked first:

| Matches Current Request | Url | Defaults | C |
|---|---|---|---|
| False | Public/{controller}/{action}/{id} | action = Index, id = UrlParameter.Optional | ( |
| False | Internal/{controller}/{action}/{id} | action = Index, id = UrlParameter.Optional | ( |
| False | Administration/{controller}/{action}/{id} | action = Index, id = UrlParameter.Optional | ( |
| False | {resource}.axd/{*pathInfo} | (null) | ( |
| True | {controller}/{action}/{id} | controller = Home, action = Index, id = UrlParameter.Optional | ( |
| True | {*catchall} | (null) | ( |

This is because the project's Global.asax App_Start first calls RegisterAllAreas, and then later calls Register Routes.

```
protected void Application_Start()
{
    AreaRegistration.RegisterAllAreas();
    FilterConfig.RegisterGlobalFilters(GlobalFilters.Filters);
    RouteConfig.RegisterRoutes(RouteTable.Routes);
    BundleConfig.RegisterBundles(BundleTable.Bundles);
}
```

Notes:

- You can have as many areas as needed.
- Each Area has its own Models, Views and Controllers.
- You can have multiple controllers with the same name, as long as each is in its own Area. Each lives in its own namespace.

Copyright © 2019 MAX Technical Training

- A top-level controller:

```
namespace WebApplication18.Controllers
{
    0 references
    public class HomeController : Controller
    {
```

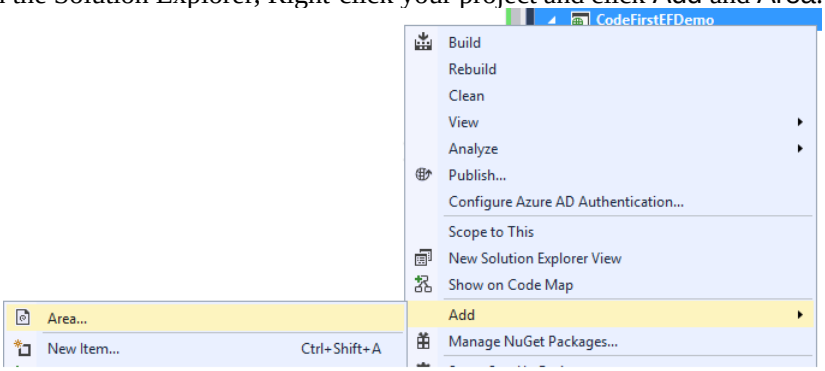
- An area controller:

```
namespace WebApplication1.Areas.Administration.Controllers
{
    0 references
    public class HomeController : Controller
    {
```

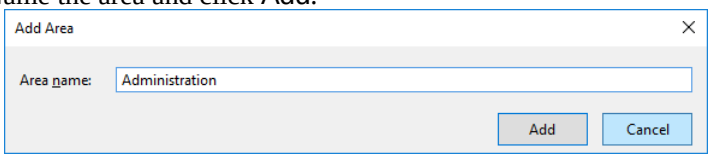
# Creating Areas

To create an Area:

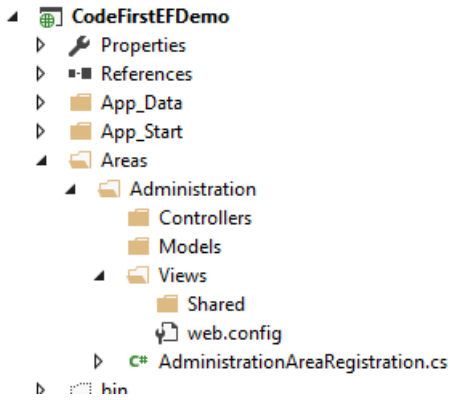
1. In the Solution Explorer, Right-click your project and click Add and Area.




2. Name the area and click Add.




3. A new collection of folders will be added to the project to support the new Area.




# Linking between Areas

Areas typically include one more level in the path to a controller:

Non-area URL: http://www.example.com/products/index

Copyright © 2019 MAX Technical Training

Area URL: http://www.example.com/**administration**/products/index

## HTML Helpers

The ActionLink HTML helper uses simple strings to define a link in the same area:

```
@Html.ActionLink("Update Products ", "Products", "Update")
```

When using ActionLink to link to another Area you also need to pass an object, not a simple string, to select the area:

```
@Html.ActionLink("Update Products", "Products", "Update", new { area = "Administration" })
```

Copyright © 2019 MAX Technical Training

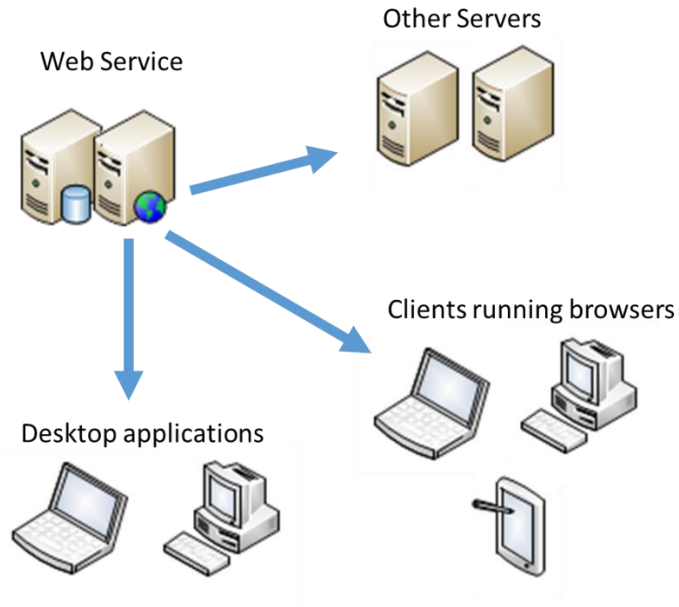
# Module 9 – Web API

In this module we will take what you already know about MVC to quickly create RESTful web services using ASP.NET Web API. Web API could even be thought of as MVC without the "V". You still create routes, models and controllers, but instead of returning views you return data.

## Web Services

Web Services Definitions:

- If you think of a web page as a machine to human (with a web browser) communication, then a web service is a machine to machine communication. The machines could be anything network connected and includes web browsers.
- While a web site typically returns web pages, a web service returns data. The data could be in text, JSON, XML or other format.
- A way to expose a "method" to the network.




Characteristics of Web Service

- Server – Client based.
  - Server receives requests using HTTP with the query defined by the URL, an HTTP header or a payload (XML for example).
  - Returns data to client in text format. This could be simple text, encoded binary data, XML or JSON.
- Can be hosted on local servers, cloud servers or local PCs.
- Stateless – No context is stored between requests.

Copyright © 2019 MAX Technical Training

- Can be private and secured, or public and anonymous.
- Often used as the "API" into applications such as SharePoint.
- Often cacheable.

**Microsoft Tools to Create Web Services**

- ASP.NET 2.0 web services (2002)
  - SOAP XML based.
  - ASMX file extension.
  - Requires a web server.
- Windows Communications Foundation (WCF).
  - Multiple configurable protocols. Could be HTTP over the internet or binary over the local LAN.
  - SOAP, and other formats.
- Web API 2.0
  - GET, POST, PUT, DELETE based web services
  - Representational State Transfer (REST)
- SQL Server
  - Stored procedures can be exposed as HTTP end points.

## A Brief History of Web Services

- Remote Procedure Calls (RPC) - Exchanges of files (FTP, etc.) (way be when!)
- Electronic Data Interchange (EDI) - Uses proprietary data formats (mid 1970's) and formalized using XML (mid 1990's).
- HTTP and XML (SOAP) based web services (2000)
- Microsoft Windows Communications Foundation (WCF) (2005 as part of .NET Framework 3.0)
- Representational State Transfer (REST) (First defined in 2000 and later by W3C in 2002)

# Adding Web Services to an MVC Project

Web services can be added to MVC projects in multiple ways.

- Return data from an MVC Controller using the Content or JSON methods instead of the View method.
- Add Web API classes to the project.
- Add a WCF project to your solution.
- Add ASP.NET 2.0 Web Service files (.asmx) to your project or solution.

# MVC Actions vs. Web API

You can add Actions to your MVC Controllers to provide much of the functionality of web services. The MVC templates also let you include a more full featured web services option in the form of Web API. Web API uses Routing and Controllers and uses much of what you already know about MVC development, but with a more complete feature set for web services.

**MVC Actions Pros**

- Easy to add to existing projects.

Copyright © 2019 MAX Technical Training

- Can use the same models and view models.
- Automatic serialization to JSON.
- Routes based on Action names. (/Vendors/GetVendors)

**MVC Actions Cons**

- No support for REST.
- No automatic serialization to XML.

**Web API Pros**

- Built-in and automatic detection for XML and JSON serialization.
- Separation of concerns. (Separate controllers.)
- A focus on HTTP methods (GET, POST, etc.) instead of Action names. (/Vendors + Get, or /Vendors + Post, etc.)
- REST!

**Web API Cons**

- Additional Controllers.
- Additional routing work.

# Data Formats

While a web service can return almost any kind of HTTP compatible data, the most common formats used to represent objects are XML and JSON. XML and JSON are intended to be both human readable and machine readable.

.NET object cannot be sent using HTTP as is. The object needs to be serialized into a text format for transmissions and then deserialized from text by the consumer of the web service.

## XML

eXtensible Markup Language

XML was first released in 1996 and the XML 1.0 standard was released in 1997. “1.0” is the latest version in common use today! (There is a 1.1, but it is not widely used.)

- XML is a tag based markup language with no pre-defined tags. <mytoys><toy id="1" name="boat"/></mytoys> is perfectly valid.
- XML dialects have been created for standardized objects. For examples see: https://en.wikipedia.org/wiki/List_of_XML_markup_languages
- XML is case sensitive.
- XML is used to define many .NET Framework files such as web.config.

Tools to reformat or validate XML:

- Visual Studio has a built-in editor for files with an .XML extension (and many other files with XML content) and IntelliSense support for XML objects.
- XML is supported from built-in .NET Framework classes within the System.Xml namespace.

Copyright © 2019 MAX Technical Training

# JSON

JavaScript Object Notation

While defined in the early 2000's JSON became an ECMA standard in 2013.

The tools needed to parse JSON are built into JavaScript and are available for other languages with tools like the Newtonsoft.Json library for the .NET Framework. The Newtonsoft.Json library is included with MVC projects and can be added to other projects using NuGet.

- Properties are defined as name/value pairs.
  "PartNumber" : "1234"
- JSON defines objects using curly brackets. Multiple properties are separated by commas.
  { "PartNumber" : "1234", "Qty" : 12 }
- Collections of objects are defined with square brackets.
  [{ "PartNumber" : "1234", "Qty" : 12 }, { "PartNumber" : "4567", "Qty" : 2 } ]
- Objects can have a hierarchal collection of properties. For example, Widget has three properties named Name, ID and Parts. Parts is a collection of part objects.

```
{
  "Widget" : {
    "Name" : "thingamajig",
    "ID" : "A100",
    "Parts" :  [
                 { "PartNumber" : "1234", "Qty" : 12 },
                 { "PartNumber" : "4567", "Qty" : 2 }
               ]
  }
}
```

- JSON strings are typically code generated and not usually pretty formatted in order to not send unneeded whitespace characters.

```
{"Widget":{"Name":"thingamajig","ID":"A100","Parts":[{"PartNumber":"1234","Qty":12},{"Part
Number":"4567","Qty":2}]}}
```

Tools for reformat or validate JSON:

- Visual Studio has a built-in editor for files with a .JSON extension and IntelliSense support inside of JavaScript code.
- Built into Bing search
  https://www.bing.com/search?q=JSON%20formatter

## Using JSON in JavaScript

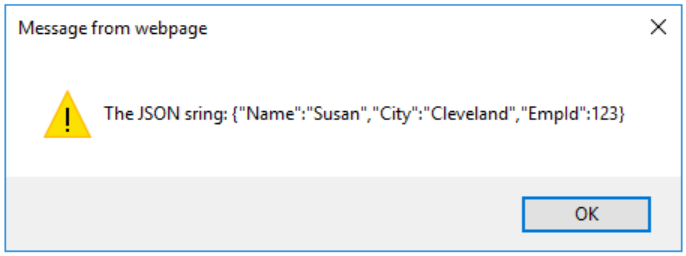
JavaScript includes a JSON object with methods to convert objects to and from JSON strings.

### JSON.stringify( object )

JSON.stringify will take a JavaScript object and convert it into a JSON string.

Copyright © 2019 MAX Technical Training

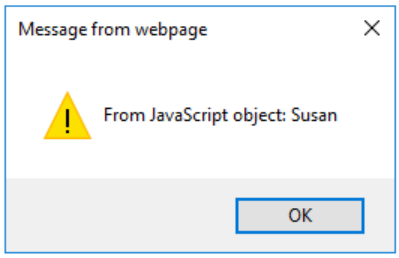
```
var person = {};
person.Name = "Susan";
person.City = "Cleveland";
person.EmpId = 123;

alert("The JSON sring: " + JSON.stringify(person));
```




### JSON.parse( string )

JSON.parse takes a JSON formatted string and returns a JavaScript object.

```
var jsonstring = '{"Name":"Susan","City":"Cleveland","EmpId":123}';
var person = JSON.parse(jsonstring)

alert("From JavaScript object: " + person.Name)
```




## Using JSON in C#

While C# does have native support for JSON conversions in the System.Web.Helpers.Json namespace, Microsoft supplies the Newtonsoft.Json library with Visual Studio and the MVC templates as a richer and faster JSON solution. For non-MVC projects, you can download the Newtonsoft.Json library using NuGet.

- The general recommendation is to use Newtonsoft.Json.
- The .Json method of the Controller class uses the JSON class that System.Web.Helper.Json uses.
- The examples below use Newtonsoft.Json. (Documentation: http://www.newtonsoft.com/json/help/html/Introduction.htm)

### JsonConvert.DeserializeObject

JsonConvert.DeserializeObject can convert a JSON string into a C# Dynamic object or any predefined type.

```
string jsonstring = "{'Name':'Susan','City':'Cleveland','EmpId':123}";

dynamic csharpobject = JsonConvert.DeserializeObject(jsonstring);
string name = csharpobject.Name;

// or if we have Person class
Person aperson = JsonConvert.DeserializeObject<Person>(jsonstring);
string city = aperson.City;
```

Copyright © 2019 MAX Technical Training

### JsonConvert.SerializeObject( object )

JsonConvert.SerializeObject converts an object into a JSON string.

```
Person testperson = new Person();
testperson.Name = "Susan";
testperson.City = "Cleveland";
testperson.EmpId = 123;

string jsonstring1 = JsonConvert.SerializeObject(testperson);

// or a multiline string...
string jsonstring2 = JsonConvert.SerializeObject(testperson, Formatting.Indented);
```

## JSON and Dates

JSON does not know what a date datatype is and each JSON serializer will return its own favorite date format. Here are some examples:

- "\/Date(1489627020170)\/" is a JavaScript serial number for number of milliseconds since January 1, 1970.
- "\/Date(1489627020170-0500)\/" is the same, but with a time zone adjustment.
- "2017-03-15T17:46:02.1153043" is the ISO8601 standard with a time zone adjustment.
- "2017-03-15T17:46:02.1153043-07:00" is the ISO8601 standard with a time zone adjustment.

### JavaScript Client

The first example, "\/Date(1489627020170)\/", is what is returned by the MVC Controller's Json() method. To do anything useful with this on the client side using JavaScript will need something like this:

- var myobject = JSON.parse(jsonstring);
  var mydate = new Date(parseInt(myobject.dateproperty.substr(6)))

The third example, "2017-03-15T17:46:02.1153043", is what is returned by Newtonsoft.Json. JavaScript's JSON.parse will still convert this to a string, but it's a little easier to convert to a date.

- var myobject = JSON.parse(jsonstring);
  var mydate = new Date(myobject.dateproperty)

### C# Client

The Newtonsoft.Json library will properly map from a JSON date-like property and a .NET object's DateTime property using either a cast to your type (VendorStatus in the example below) or to a JObject collection.

```
VendorStatus vs = JsonConvert.DeserializeObject<VendorStatus>(data);
DateTime d1 = vs.Created;

JObject obj = (JObject)JsonConvert.DeserializeObject(data);
DateTime d2 = (DateTime)obj["Created"];
```

What if you are using a web service that returns "\/Date(1489627020170)\/" and your C# client code does not understand that format? You will have to take those milliseconds and add them to 1/1/1970!

Copyright © 2019 MAX Technical Training

- Int64 numberFromJson = 1489661748371;
  DateTime date1970 = new DateTime(1970, 1, 1, 0, 0, 0, DateTimeKind.Utc);
  DateTime mydate = date1970.AddMilliseconds(numberFromJson);

If you need to create the JavaScript date number from C# you could do this:

- DateTime date1970 = new DateTime(1970, 1, 1, 0, 0, 0, DateTimeKind.Utc);
  Int64 javascriptdatenumber = (Int64)(yourDateVariable - date1970).TotalMilliseconds;

# Creating Web Services from an MVC Application

An MVC controller can return Views to deliver HTML web pages to a browser. By using the Content(), Json() and other methods a controller can return data. (Kind of like MVC without the V!)

## Returning JSON Data

MVC Controllers include a Json() method that can be used to return an object in JSON format.

```
var venddata = from v in db.Vendors
               select new { Vendor = v.Name, VendorID = v.VendorID,
                   Products = from p in v.Products select new { p.Description, p.Price } };

return Json(venddata, JsonRequestBehavior.AllowGet);
```

Or using the Newtonsoft.Json library:

```
var venddata = from v in db.Vendors
               select new { Vendor = v.Name, VendorID = v.VendorID,
                   Products = from p in v.Products select new { p.Description, p.Price } };

string jsonstring = JsonConvert.SerializeObject(venddata);
return Content(jsonstring, "application/json; charset=utf-8");
```

## Returning XML Data

While an MVC Controller does not have an XML() method, it does have a Content() method that can return a string of data and a content type. As long as we can built the string, Content() can return it to the browser as “text/xml”.

In its simplest form, returning XML from an Action might look like this:

```
public ActionResult EmployeesAsXML()
{
    return Content("<employees><employee id='1'><name>mike</name></empoyee></employees>", "text/xml");
}
```

There are several options within the .NET Framework for serializing objects into XML. The example below uses an XmlSerializer object and a StringWriter object to return a string of XML.

The using statements:

```
using System.Xml.Serialization;
using System.IO;
```

The Action:

Copyright © 2019 MAX Technical Training

```
public ActionResult GetVendorStatusCodes()
{
    // get the data
    var vstatus = new VendorStatus();
    vstatus.VendorStatusID = "1A";
    vstatus.VendorStatusDescription = "test";
    vstatus.Created = DateTime.Now;

    // serialize it
    XmlSerializer xmlSerializer = new XmlSerializer(vstatus.GetType());
    string xmlstring = "";
    // StringWriter needs to be disposed!
    using (StringWriter textWriter = new StringWriter())
    {
        xmlSerializer.Serialize(textWriter, vstatus);
        xmlstring = textWriter.ToString();
    }

    return Content(xmlstring, "text/xml");
}
```

# Consuming Web Services

Once you have a web service created that returns XML or JSON, you need to write the client-side code to request and consume the data.

## Calling a Web Service from Server Side Code

The .NET Framework has multiple classes that can make calls to web servers. Here are two common examples.

Option 1: System.Web.WebClient.

```
public class VendorStatus
{
    public string VendorStatusID { get; set; }
    public string VendorStatusDescription { get; set; }
    public DateTime Created { get; set; }
}


class Program
{
    static void Main(string[] args)
    {
        // using Newtonsoft.Json; using Newtonsoft.Json.Linq;
        string url = "http://localhost:24366/vendors/GetVendorStatusCodes";
        var wc = new System.Net.WebClient();
        wc.UseDefaultCredentials = true;
        wc.Headers.Add("Content-Type", "application/json");
        string data = wc.DownloadString(url);

        VendorStatus vs = JsonConvert.DeserializeObject<VendorStatus>(data);
        DateTime d1 = vs.Created;

        JObject obj = (JObject)JsonConvert.DeserializeObject(data);
        DateTime d2 = (DateTime)obj["Created"];
```

Option 2: System.Net.WebClient

Copyright © 2019 MAX Technical Training

If you are creating a project that otherwise is not using System.Web, then consider using System.Net.WebClient. Here is a very basic use of the class.

```
// using System.Net
using (WebClient proxy = new WebClient())
{
    var response = proxy.DownloadString("http://localhost:24366/vendors/GetVendorStatusCodes");

    VendorStatus vs2 = JsonConvert.DeserializeObject<VendorStatus>(data);
    DateTime vsdate = vs2.Created;
```

# Calling Web Services from Ajax and jQuery

Ajax used to be AJAX (Asynchronous JavaScript and XML). While it uses JavaScript, it does not have to be asynchronous or return XML. Today Ajax is a name a technology and is not usually in all upper case.

Many web services are accessed from client-side code using JavaScript. Most of these calls will be asynchronous to prevent the user's browser from freezing during updates. While there are numerous Ajax libraries, we will use the jQuery library for these examples. jQuery lets you create a detailed request with many options and simple request to just get JSON data.

## $.ajax()

The Ajax method gives you the most control and the most options. See: http://api.jquery.com/jQuery.ajax/

In the example below:

- The curly brackets define a JavaScript object that is passed to the $.ajax method.
- A custom HTTP header has been added to request JSON formatted data.
    - The service you are calling might be able to request multiple formats.
    - This will not guarantee that you will get JSON back. That's up to the service.
- Asynchronous functions for processing a success or an error. This example is using anonymous in-line functions. (An alert is not the best way to show data!)

```
$.ajax(
  {
      url: "http://localhost:24366/vendors/GetVendorStatusCodes",
      method: "GET",
      headers: {
          "accept": "application/json; odata=verbose",
      },
      success: function (data) {
          alert(JSON.stringify(data));
          // display data in the page
      },
      error: function (err) {
          alert(JSON.stringify(err));
      }
  }
);
```

Copyright © 2019 MAX Technical Training

## $.getJSON()

getJSON is just a shorthand for:

```
$.ajax({
    dataType: "json",
    url: url,
    data: data,
    success: success
});
```

The following is the equivalent to the $.ajax example above.

```
$.getJSON(
    "http://localhost:24366/vendors/GetVendorStatusCodes",
    function (data) { alert(JSON.stringify(data)) }).fail(alert("Error!")
)
```

$.getJSON has two shortcomings:

- If the JSON is malformed, it will “silently fail”.
- You do not get the cause of the error from .fail().

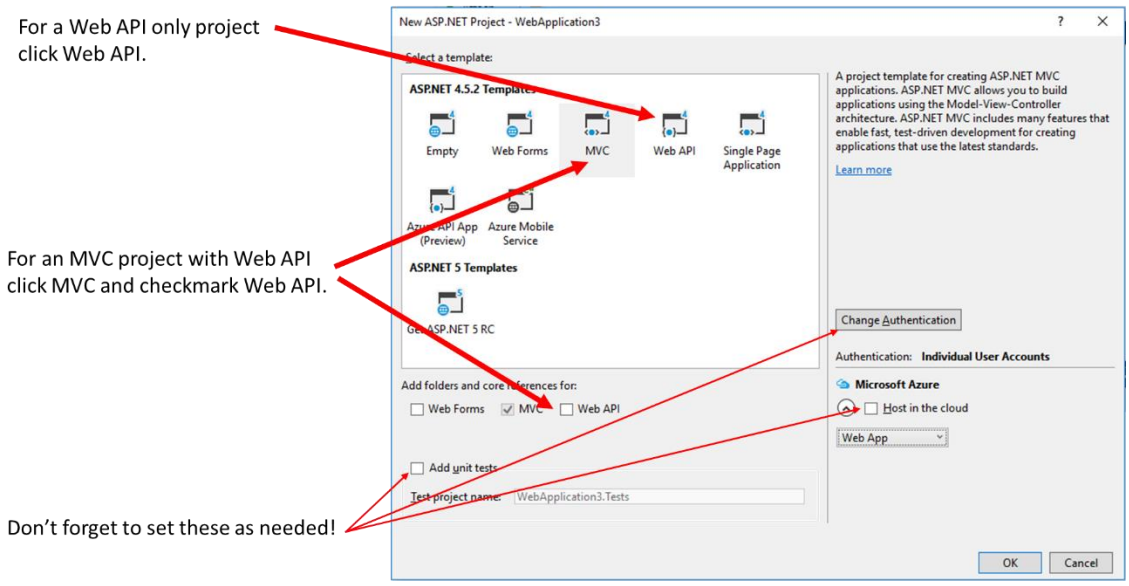
The solution to these is to use the full $.ajax() method.

# Web API Projects

Web API projects are similar to MVC projects in that they include Routes, Controllers and Actions. They also include the Web API 2.0 libraries (available as a NuGet package) and a Controller that inherits from System.Web.Http.ApiController.

## Starting a New Web API or MVC plus Web API Project

Starting a new Web API project is almost identical to start a new MVC project.




Copyright © 2019 MAX Technical Training

# Adding Web API to an Existing MVC Project

If you have already created your MVC project, you can add Web API to that project.

Steps:

- Install Microsoft.AspNet.WebAPI from the NuGet Package Manager.
- Add support for WebAPI routing.
  - Then add a public static class, typically in the App_Start folder:

```
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 }
          );
      }
  }
```

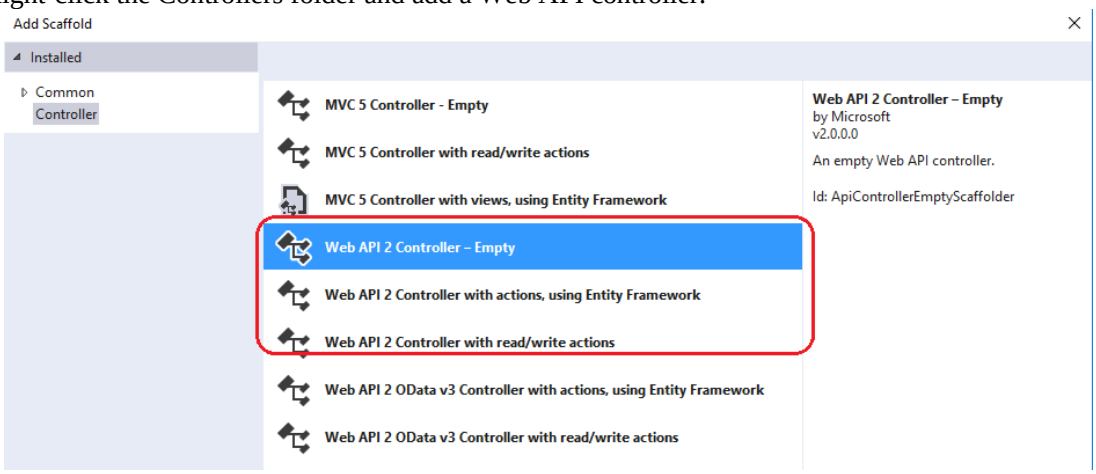
  - Edit Global.asax and add:
    using statement:

```
using System.Web.Http;
```

    and in the Application_Start method:

```
GlobalConfiguration.Configure(WebApiConfig.Register);
```

- Right-click the Controllers folder and add a Web API controller.




# Web API Routing

Like MVC, Web API uses routing and controllers. While Web API uses its own route collection, those routes are also added to the MVC route table. To make the Web API routes distinct from MVC routes they typically include a prefix like "/api/". You can use any name you like such as "/service/" or "/data/".

**Example Web API URL**: http://www.example.com/**api**/products/123

Visual Studio templates that include Web API add an entry to Global.asax and a file, WebApiConfig.cs, to the App_Start folder.

**Added to Global.asax:** GlobalConfiguration.Configure(WebApiConfig.Register);

Copyright © 2019 MAX Technical Training

The WebApiConfig.cs in a new project adds two entries to the route table by default.

- MapHttpAttributeRoutes – Adds support for routes defined using method attributes (new in MVC5).
- Routes.MapHttpRoute – Adds a route using convention-based routing based on a template.

**WebApiConfig.cs:**

Here is the default routing for Web API.

```
public static class WebApiConfig
{
    public static void Register(HttpConfiguration config)
    {
        // Web API configuration and services

        // Web API routes
        config.MapHttpAttributeRoutes();

        config.Routes.MapHttpRoute(
            name: "DefaultApi",
            routeTemplate: "api/{controller}/{id}",
            defaults: new { id = RouteParameter.Optional }
        );
    }
}
```

## config.Register.MapHttpAttributeRoutes();

Adds support for Web API 2.0 attribute routing. This uses attributes on methods in controllers instead of predefined maps to manage routes. If MapHttpAttributeRoutes is enabled, then the Route attribute in the following example says to call this Action when a request is received using the specified URL pattern.

```
[Route("customers/{customerId}/orders")]
public IEnumerable<Order> GetOrdersByCustomer(int customerId) { ... }
```

The example path above with the customerId in the middle cannot be handled using MapHttpRoute.

For details see: Attribute Routing in ASP.NET Web API 2
https://www.asp.net/web-api/overview/web-api-routing-and-actions/attribute-routing-in-web-api-2

## config.Register.Routes.MapHttpRoute

Routing can also be configured using convention-based routing where you supply a route template. The following is the example found in a new Web API project.

```
config.Routes.MapHttpRoute(
    name: "DefaultApi",
    routeTemplate: "api/{controller}/{id}",
    defaults: new { id = RouteParameter.Optional }
);
```

The “api” in the routeTemplate is a convention and not a requirement. It is recommended that you use something similar to avoid conflicts with MVC routes in the same project.

Note that the above example for MapHttpRoute is using named parameters. You could also have written the above using positional parameters:

Copyright © 2019 MAX Technical Training

```
config.Routes.MapHttpRoute(  "DefaultApi", "api/{controller}/{id}",
  new { id = RouteParameter.Optional }  );
```

Also see: https://www.asp.net/web-api/overview/web-api-routing-and-actions/routing-in-aspnet-web-api

# Web API Controllers

Web API controllers inherit from System.Web.Http.ApiController and typically have Actions named with HTTP verbs. In the example below note the Action names.

Action naming conventions:

- Actions are named to match the HTTP verbs:  Get, Post, Put, Delete.
- Actions are named with HTTP verbs as prefixes: GetVendor, PostVendor, PutVendor, DeleteVendor.
- Use any name you like and add an HttpGet, HttpPut, HttpPost, or HttpDelete attribute or the [AcceptVerbs("verbname", "verbname",…)] attribute.
- Create routes and route directly to any named Action.

```
public class ProductsController : ApiController
{
    // GET: api/Products
    public IEnumerable<string> Get()
    {
        return new string[] { "value1", "value2" };
    }

    // GET: api/Products/5
    public string Get(int id)
    {
        return "value";
    }

    // POST: api/Products
    public void Post([FromBody]string value)
    {
    }

    // PUT: api/Products/5
    public void Put(int id, [FromBody]string value)
    {
    }

    // DELETE: api/Products/5
    public void Delete(int id)
    {
    }
}
```

# HTTP Verbs

In most of your web work you have used the HTTP verbs “GET” and “POST” and only associated those with URL Query Strings and form <form> submissions. These verbs are part of a larger set that define the basic Create, Read, Update and Delete set of data activities (CRUD).

- POST – this is the **C**reate activity, or the equivalent to a SQL Insert statement.
- GET – This the **R**ead activity, or the equivalent to a SQL Select statement.
- PUT – this is the **U**pdate activity, or the equivalent to a SQL Update statement. In REST PUT is used to replace the entire entity while PATCH/MERGE is used to update part of an entity.

Copyright © 2019 MAX Technical Training

- DELETE – this is the **D**elete activity, or the equivalent to a SQL Delete statement.

Notes:

- A GET action is easy to manually test as it is just a URL typed into a browser. The other actions will require writing C# or JavaScript code or by using a tool like Fiddler.
- PUT vs PATCH:
  - PUT is intended to completely replace an existing item. The client should always send the entire object. In SQL terms this would be the same as an UPDATE that updates all of the fields.
  - PATCH is intended to update selected properties. I.e. change the Qty in Stock of a product.

# Web API Actions

MVC Actions typically return only an ActionResult object. Web API Actions can return various datatypes that can be accepted by a web browser.

- **void** – returned as "204 (No Content)"
- **string**
- **IEnumerable**<string> – A collections of strings
- **IEnumerable**<Vendor> or **IQueryable**<Vendor> – A collection of a strongly typed object that has been serialized to XML or JSON and returned as a string.
- **IHttpActionResult** – Lets you easily customize the results, for example the HTML Status code. (System.Web.Http.Results) Basically the returned data with response header options.
- **Any other type, including "object"** – These are serialized to XML or JSON and returned as a string.

# Returning Data – Best Practices

When you create a View, you should only pass the data needed by the View. The same applies to web services… only return the minimum data needed. You can create custom models, or reuse View Models created elsewhere in the project.

An anonymous typed collection:

```
//GET: api/VendorsData
public object GetVendors()
{
    //var venddata= db.Vendors;    // too much data!

    var venddata = from v in db.Vendors
                   select new
                   {
                       Name = v.Name,
                       VendorID = v.VendorID,
                       Phone = v.Phone,
                       Email = v.Email
                   };
    return venddata;
}
```

A strongly typed collection using a custom Model (VendorAPIModel):

Copyright © 2019 MAX Technical Training

```
// GET: api/VendorsData
public IQueryable<VendorAPIModel> GetVendors()
{
    //var venddata= db.Vendors;     // too much data!

    var venddata = from v in db.Vendors
                   select new VendorAPIModel
                   {
                       Name = v.Name,
                       VendorID = v.VendorID,
                       Phone = v.Phone,
                       Email = v.Email
                   };
    return venddata;
}
```

# Using IHttpActionResult

Similar to how when using MVC's Action Result you can return data using the ApiController's base class helper methods like View(), Json(), File(), etc., with IHttpActionResult you can use base class methods like OK(), Redirect, and Unauthorized, BadRequest, NotFound() and others.

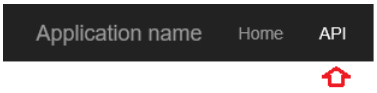
IHttpActionResult is typically used when performing Actions that might fail. Things like duplicate keys, bad data, etc.

See: https://msdn.microsoft.com/en-us/library/system.web.http.apicontroller(v=vs.118).aspx

```
// DELETE: api/VendorsData/5
[ResponseType(typeof(Vendor))]
public IHttpActionResult DeleteVendor(int id)
{
    Vendor vendor = db.Vendors.Find(id);
    if (vendor == null)
    {
        return NotFound();
    }

    db.Vendors.Remove(vendor);
    db.SaveChanges();

    return Ok(vendor);
}
```

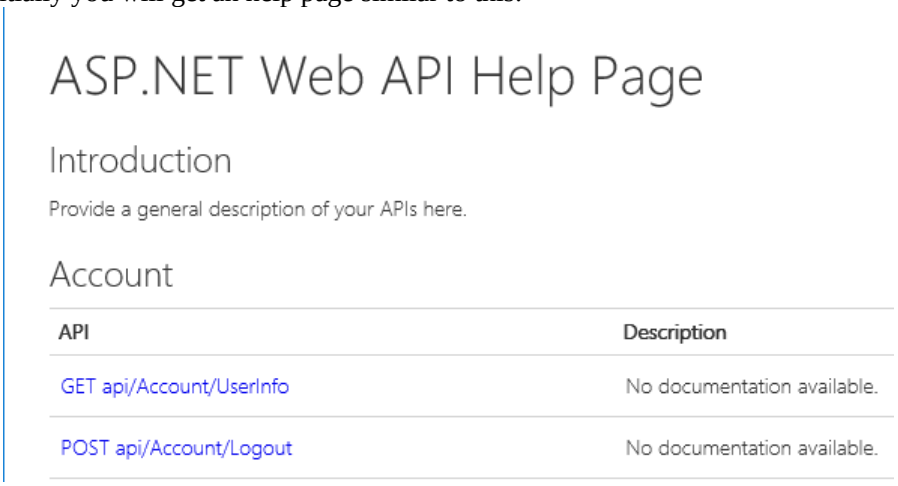
# Web API Help Pages

When you create a new Web API project, a help link name "API" is added to the navigation. This link displays a list of all of your ApiController Actions.




Copyright © 2019 MAX Technical Training

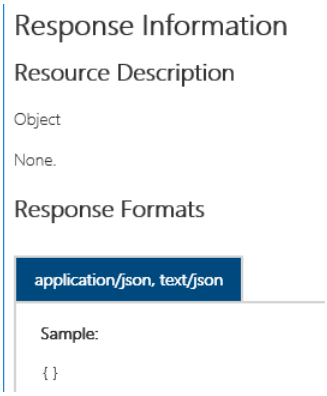
Initially you will get an help page similar to this:




Clicking one of the APIs will open a page with details about:

- Parameters needed.
- Properties returned.
- Sample JSON and/or XML results depending on your GlobalConfiguration.Configuration.Formatters configuration in Global.asax.

Initially the documentation generated will be a bit lacking, especially when using “object” and “IHttpActionResult” as the return types.




Additional documentation can be added using:

- Configuration in the Areas/HelpPage/App_Start/HelpPageConfig.cs. (Browse this file and read the comments.)
- The use of Visual Studio XML documentation.
- Web API controller and action attributes.
    - [ResponseType] – specifies the datatype being returned (for documentation purposes only)
    - Example: [ResponseType(typeof(Product))]
- Actions and entire Controllers can be excluded from documentation using: [ApiExplorerSettings(IgnoreApi=true)]

# XML Documentation

XML Documentation is:

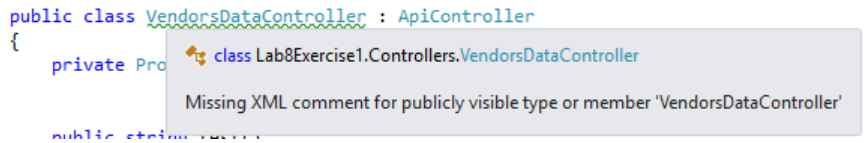
Copyright © 2019 MAX Technical Training

- A comment block using three slashes (“///”) with XML tagged sections.
- A Visual Studio option to extract these comments to an XML file.
- Automation to convert this data to a Windows Help file, page or other destination.

Example:

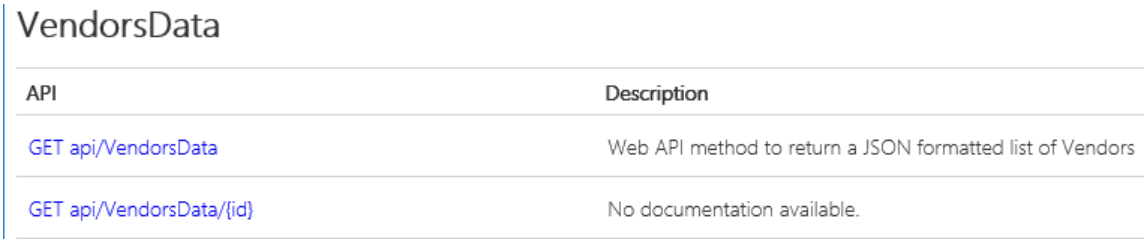
```
//GET: api/VendorsData
/// <summary>
/// Web API method to return a JSON formatted list of Vendors
/// </summary>
/// <returns>Name, VendorID, Phone and Email</returns>
/// <remarks>
///   Lasted updated by Sam 1/12/1206
/// </remarks>
public object GetVendors()
{
```

Once you have enabled XML Documentation, Visual Studio will add green “squiggles” under class and method decorations to remind to add documentation. These are not errors, just tips.




For more about the XML Documentation feature see: https://msdn.microsoft.com/en-us/library/b2s063f7.aspx

This example has an expanded description for the first API that was extracted from XML Documentation data.




# Customization

You can modify the API Help Controllers and Views as needed to create both custom documents and page design.

# Adding Web API Help Pages to Existing Projects

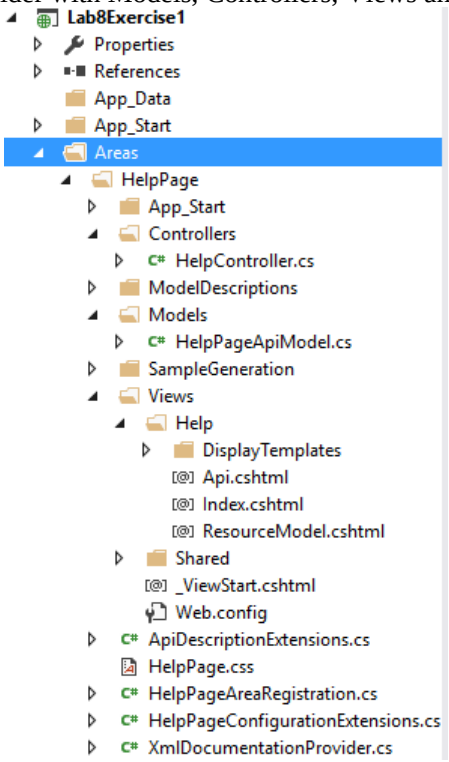
You do not automatically get Web API Help pages if you initially created an MVC project and then later added Web API controllers. Luckily there’s a NuGet package for that!

Steps:

1. Use the NuGet Package Manager to install the Microsoft.AspNet.WebApi.HelpPage package. (Or use the NuGet Package Manager Console and type Install-Package Microsoft.AspNet.WebApi.HelpPage.

Copyright © 2019 MAX Technical Training

2. You will find new assemblies in the References folder and the addition of an Areas/HelpPages folder with Models, Controllers, Views and other files.




3. Take a look at the Areas/HelpPage/App_Start/HelpPageConfig.cs file for ideas for some the help generation features that are available.
4. The help page is now available, but can be enhanced with a little more configuration. Build the project, run it and visit http://yourserver/Help to see what you get so far.
5. If the Visual Studio XML Documentation feature is enabled that data can be picked up by the help page.
    a. Edit the Areas/HelpPage/App_Start/HelpPageConfig.cs file and uncomment the line that starts with "config.SetDocumentationProvider".
    b. Add XML Document comments to an Action in your ApiController class. Type three slashes "///" to create a sample block.
    c. Rebuild the project, run it and redisplay the help page.

# RESTful Web Services

REpresentational State Transfer

Traditional web services do one thing and only one thing. They may accept one or more parameters to filter content, but they cannot be used in ways other than what they were written for. A GetVendors service might return all active vendors. If someone needs a list of all vendors from Ohio, or all vendors with for a particular product, then we need to write yet another web service. And as soon as that is delivered, they want one that returns vendors for a ZIP code or a phone number prefix.

A REST service URL points to a resource, not a page, file or service.

- ASP.NET ASPX SOAP web service:
  http://yourserver/SomeService**.asmx** a page, not a resource – returns a vendor or a product or a …?
- Windows Communications Foundation (WCF):
  http://yourserver/SomeService**.svc** a page, not a resource – returns a vendor or a product or a …?

Copyright © 2019 MAX Technical Training

- REST
  http://yourserver/Vendor no file extension! Clearly a request for a vendor!
  Note: Web API routes will typically include a path prefix:
  http://yourserver/**api/**Vendor

While REST services with OData can support full CRUD activities, its most useful feature is the ability to query data from a URL.

- http://yourserver/VendorsData?$select=name, phone&$filter=state eq 'OH'
- http://yourserver/VendorsData(1)/Products?$select=description, price

## REST:

- Hides the implementation
  - No .asmx or .svc files or pages in the URL (ideally)
  - http://site/resource/action?parameters
- A RESTful service models data entities.
  - Products, Customers, Announcements, Tasks, etc.
- Uses standard HTTP verbs for CRUD.
  - POST, GET, PUT, DELETE plus PATCH/MERGE
- Readable, predictable URLs.
- Typically returns JSON or XML.

# OData

OData is a standard that defines both a query syntax and a format for the returned data. OData can also generate its own documentation data.

For more about the OData standard see:

- **OData in ASP.NET Web API**
  https://docs.microsoft.com/en-us/aspnet/web-api/overview/odata-support-in-aspnet-web-api/
- **Understand OData in 6 steps**
  http://www.odata.org/getting-started/understand-odata-in-6-steps/
- **OData - the Best Way to REST**
  http://www.odata.org/
- **Open Data Protocol** (with some history)
  https://en.wikipedia.org/wiki/Open_Data_Protocol

Note: The Web API implementation of OData does not fully comply with what's described at the odata.org site.

## Service Metadata Document

OData is self-documenting and can generate an XML document that defines the list of entities supported.

http://yourserver/odata**/$metadata**

Copyright © 2019 MAX Technical Training

```
- <edmx:Edmx Version="1.0">
  - <edmx:DataServices m:DataServiceVersion="3.0" m:MaxDataServiceVersion="3.0">
    - <Schema Namespace="Lab8Exercise6.Models">
      - <EntityType Name="Vendor">
        - <Key>
            <PropertyRef Name="VendorID"/>
          </Key>
          <Property Name="VendorID" Type="Edm.Int32" Nullable="false"/>
          <Property Name="Name" Type="Edm.String" Nullable="false"/>
          <Property Name="Status" Type="Edm.String" Nullable="false"/>
          <Property Name="Phone" Type="Edm.String" Nullable="false"/>
          <Property Name="Email" Type="Edm.String" Nullable="false"/>
          <Property Name="CreditLimit" Type="Edm.Decimal"/>
          <Property Name="ContractReview" Type="Edm.DateTime"/>
          <Property Name="PaymentTerms" Type="Edm.String"/>
          <NavigationProperty Name="Products"
          Relationship="Lab8Exercise6.Models.Lab8Exercise6_Models_Vendor_Products_Lab8Exerci
          FromRole="ProductsPartner"/>
        </EntityType>
      - <EntityType Name="Product">
```

# Data Format

While Web API by default just returns an XML or JSON representation of an object, OData returns that data plus “metadata” about the object.

**A Web API request for a single vendor:**
http://yourserver/api/vendorsdata/1

```
{
    "Name": "Fred's Model Airplanes",
    "VendorID": 1,
    "Phone": "123-123-1234",
    "Email": "fred@example.com"
}
```

Web API by default returns just the data, and your object is the top level object. If you stored this into a JavaScript variable named yourVar you would access Name as yourVar.Name.

A REST request with OData formatted return for a single vendor can be returned in several formats
http://localhost:24366/odata/VendorsODATA(1)

**OData Atom Pub (XML):**

```
"accept": "application/atom+xml"
```

Copyright © 2019 MAX Technical Training

```
<?xml version="1.0" encoding="utf-8"?>
<entry xml:base="http://localhost:24366/odata"
       xmlns="http://www.w3.org/2005/Atom"
       xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices"
       xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata"
       xmlns:georss="http://www.georss.org/georss"
       xmlns:gml="http://www.opengis.net/gml">
  <id>http://localhost:24366/odata/VendorsODATA(1)</id>
  <category term="Lab8Exercise6.Models.Vendor"
            scheme="http://schemas.microsoft.com/ado/2007/08/dataservices/scheme" />
  <link rel="edit" href="http://localhost:24366/odata/VendorsODATA(1)" />
  <link rel="self" href="http://localhost:24366/odata/VendorsODATA(1)" />
  <title />
  <updated>2017-03-19T20:17:32Z</updated>
  <author>
    <name />
  </author>
  <content type="application/xml">
    <m:properties>
      <d:Name>Fred's Model Airplanes</d:Name>
      <d:VendorID m:type="Edm.Int32">1</d:VendorID>
      <d:Phone>123-123-1234</d:Phone>
      <d:Email>fred@example.com</d:Email>
    </m:properties>
  </content>
</entry>
```

**OData JSON Verbose:**

```
"accept": "application/json; odata=verbose"

{
    "d": {
        "__metadata": {
            "id": "http://yourserver/odata/VendorsODATA(1)",
            "uri": "http://yourserver/odata/VendorsODATA(1)",
            "type": "Lab8Exercise6.Models.Vendor"
        },
        "Name": "Fred's Model Airplanes",
        "VendorID": 1,
        "Phone": "123-123-1234",
        "Email": "fred@example.com"
    }
}
```

**OData JSON Light (the default starting with ODATA 3)**

```
"accept": "application/json"

{
    "d": {
        "__metadata": {
            "id": "http://localhost:24366/odata/VendorsODATA(1)",
            "uri": "http://localhost:24366/odata/VendorsODATA(1)",
            "type": "Lab8Exercise6.Models.Vendor"
        },
        "Name": "Fred's Model Airplanes",
        "VendorID": 1,
        "Phone": "123-123-1234",
        "Email": "fred@example.com"
    }
}
```

OData by default returns your data and metadata about the data. Your object is a property of a top level object named “d”. If you stored this into a JavaScript variable named yourVar you would access Name as yourVar**.d.**Name.

Copyright © 2019 MAX Technical Training

## OData is very case sensitive

All OData URLs and queries are case sensitive past the server name and any "/odata/" prefix.

- These are all valid:
  - http://yourserver/odata/VendorSerach?$select=Name
  - http://YourServer/oData/VendorSerach?$select=Name
  - http://YOURSERVER/ODATA/VendorSerach?$select=Name
  - http://YourServer/odata/VendorSerach?$select=Name
- Only one of these is valid:
  - http://yourserver/odata/VendorSerach?$select=Name
  - http://yourserver/odata/vendorserach?$select=Name
  - http://yourserver/odata/VendorSerach?$select=name
  - http://yourserver/odata/VendorSerach?$Select=Name
  - Which of the above is the correct one? The fourth is never correct. "select" and other keywords are always in lower case. For the first three it depends on how things were named in the controller.

# OData Queries

OData URLs are first RESTful, pointing a resource, and then queryable by using a query string.

URLs that point to a resource:

- http://yourserver/odata/VendorsData (returns all vendors – a collection of resources)
- http://yourserver/odata/VendorsData(1) (returns a single vendor)
- http://localhost:24366/odata/VendorsODATA(1)/Products (returns a single vendor's products)

URLs with a query:

- http://yourserver/odata/VendorsData?$select=Name (returns all vendors names)
- http://yourserver/odata/VendorsData(1)?$select=Name (returns a single vendor's name)

## Passing an ID

OData URLs pass parameters a little differently. While a Web API URL would select a vendor with ID=3 using a route, OData passes the parameter as you would for a method call.

- Web API: /api/Vendors/3
- OData: /odata/Vendors(3)

## Queries

OData queries are written as a URL query string and follow a pattern:

- Starts with a "?", if the first item in the query string, or with a "&". ?$select=…
- Keywords start with a "$". $select=…
- Keywords are followed by an equal sign.
- Boolean operators are names, not symbols. eq, ne, gt, lt, etc.

Copyright © 2019 MAX Technical Training

## $select

$select selects which properties to include in the response.

Notes:

- Property names can be separated by spaces. $select=Name,Status or $select=Name, Status
- Some ODATA providers may still return additional properties beyond those in your list.

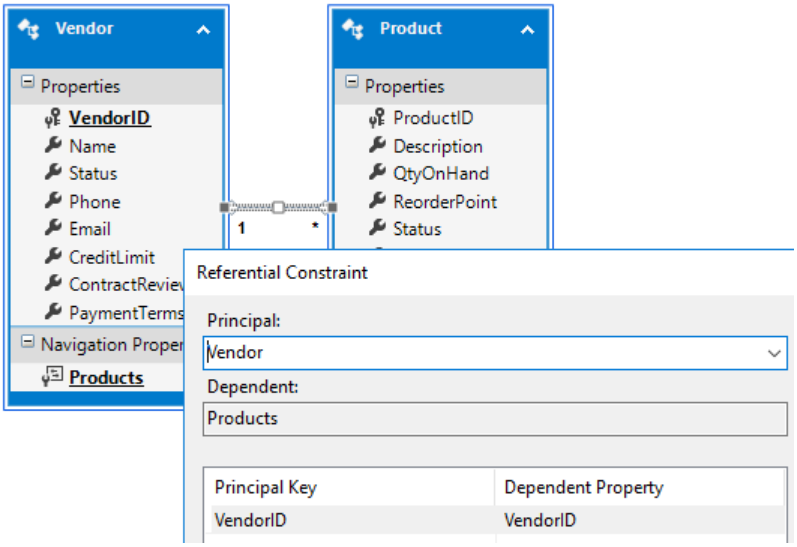
Examples:

- VendorsODATA?$select=Name, Status
- /VendorsODATA(1)? $select=Name, Status

## $select with drilldown

$select combined with $expand can be used to return parent/child sets of related data.

For this set of entities:




You could return a list of products for a vendor, or all vendors, using:

- http://yourserver/odata/VendorsODATA(1)?**$expand=Products**&$select=Name, Status, **Products/Description, Products/Price**
- http://yourserver/odata/VendorsODATA?**$expand=Products**&$select=Name, Status, **Products/Description, Products/Price**

## $filter

$filter Filters the results, based on a Boolean condition.

For details and examples see: http://www.odata.org/documentation/odata-version-3-0/odata-version-3-0-core-protocol/#queryingcollections

Examples:

- http://yourServer/odata/VendorsODATA?**$filter=Status eq 'Active'**
- http://yourServer/odata/VendorsODATA?**$filter=substringof('plane',Name)**
    - This is the equivalent of a “contains” function.

Copyright © 2019 MAX Technical Training

    - The searched text (“plane”) is not case sensitive.
- http://yourServer/odata/VendorsODATA?$**filter=year(ContractReview) eq 2018**

Notes:

- $filter can only be applied to collections, not single objects.
- All keywords are case sensitive!
- All literal text must be wrapped in single quotes.
- Boolean operators are names, not symbols: eq, ne, gt, ge, lt, le, and, or, not
- String functions: length, substringof, startswith, endswith, indexof, replace, substring, tolower, toupper, trim, concat
    - $filter=length(Name) gt 25
    - $filter=startswith(Name,'fred')
    - Note: Text used in these functions is not case sensitive.
- Arithmetic Operators: add, sub, mul, div, mod
    - Example: $filter=Price sub 10000 gt 0
    - Example: $filter=VendorID mod 2 eq 0  only returns even numbered vendor IDs
- Date functions:
    - year(*date*)
    - month(*date*)
    - day(*date*)
    - hour(*date*)
    - minute(*date*)
    - second(*date*)
- The current version does not support aggregates such as sum and average.

## $orderby

$orderby sorts the results. Uses a list of properties and can include “desc” or “asc”.

Example:

- $orderby=Name
- $orderby=CreditLimit desc,Name

Notes:

- $orderby can only be applied to collections, not single objects.

## $top

$top Returns only the first n the results.

Example:

- $top=20

Notes:

- $top can only be applied to collections, not single objects.

Copyright © 2019 MAX Technical Training

## $skip

$skip Skips the first n results. Typically used with $top and $inlinecount for paging of data.

Example:

- $skip=80&$top=20

Notes:

- $skip can only be applied to collections, not single objects.

## $inlinecount

$inlinecount Tells the server to include the total count of matching entities in the response. (Useful for paging.)

Example:

- $inlinecount=allpages


```
{
  "odata.metadata":"http://localhost:24366/odata/$metadata#VendorsODATA","odata.count":"104","value":[
    {
      "VendorID":104,"Name":"Test Vendor
0099","Status":"Inactive","Phone":"123-123-0099","Email":"V0099@example.com","CreditLimit":"10400000.0(
0:00:00","PaymentTerms":null
    },{
```


Notes:

- The returned property is “odata.count”.
- $inlinecount can only be applied to collections, not single objects.
- $inlinecount has only two options: none and allpages.

# OData Updates

OData can update data using the HTTP POST, PATCH and DELETE verbs.

| Verb | URL | HTTP body |
|---|---|---|
| DELETE | To resource:<br>http://yourServer/odata/resource(id) | None |
| POST – insert | To controller:<br>http://yourServer/odata/resource | Data needed to create a new item. Typically required fields.<br>JSON example:<br>{<br>"Name": "Tom’s Tanks",<br>"Phone": "123-555-5555",<br>"Email": "tom@example.com"<br>} |
| PATCH/MERGE – update | To resource:<br>http://yourServer/odata/resource(id) | Data to change.<br>JSON example:<br>{ |

Copyright © 2019 MAX Technical Training

| | | |
|---|---|---|
| | | "Status": "Inactive"<br>} |

# Examples

## DELETE

An example using jQuery's AJAX function.

```
$.ajax({
    url: "http://yourServer/odata/resource(123)",
    type: "DELETE",
    success: function (data) {
        //alert(data)
    },
    error: function (jqXHR, ex, msg) {
        alert("Error from server: \nStatus: " + jqXHR.status + "\nException: " + ex +
            "\nMessage: " + msg + "\nMessage from server: " + jqXHR.responseText)
    }
});
```

## POST - Insert

An example using jQuery's AJAX function.

```
$.ajax({
    url: "http://yourServer/odata/resource",
    contentType: 'application/json',
    type: "POST",
    dataType: 'json',
    data: '{ "Name": "Hobbies4you", "Phone": "123-555-5555", "Email": "tom@example.com" }',
    headers: { "Content-Length": 78 },
    success: function (data) {
        //alert(data)
    },
    error: function (jqXHR, ex, msg) {
        alert("Error from server: \nStatus: " + jqXHR.status + "\nException: " + ex +
            "\nMessage: " + msg + "\nMessage from server: " + jqXHR.responseText)
    }
});
```

## PATCH – Update

An example using jQuery's AJAX function.

```
$.ajax({
    url: "http://yourServer/odata/resource(123)",
    contentType: 'application/json',
    type: 'PATCH',
    dataType: 'json',
    data: '{ "Status": "Inactive"  }',
    headers: { "Content-Length": 25 },
    success: function (data) {
        //alert(data)
```

Copyright © 2019 MAX Technical Training

```
    },
    error: function (jqXHR, ex, msg) {
        alert("Error from server: \nStatus: " + jqXHR.status + "\nException: " + ex +
            "\nMessage: " + msg + "\nMessage from server: " + jqXHR.responseText)
    }
});
```

# Differences Between a Web API Project and an OData Project

- While they both have Get, Get(id), Post, Put and Delete, the OData version adds a Patch/Merge Action.
- Get Action
  - Identical!
- Get(id) Action
  - Code:
    - Web API: Creates a single object from EF .Find().
    - OData: Create a single object from a LINQ query.
  - Return type:
    - Web API: IHttpActionResult
    - OData: SingleResult<T> (a queryable type)
  - Return value:
    - Web API: OK(object) or NotFound()
    - OData: SingleResult.Create() (returns 0 or 1 items, i.e. not an error)
- Post Action
  - Identical except for return value.
  - Return value:
    - Web API: CreatedAtRoute() (returns route to the new item)
    - OData: Created(object) returns a location header that is the same as the edit link of the created entity
  - 
- Delete Action
  - Return type is the same. IHttpActionResult
  - Data in:
    - Web API: int id
    - OData: [FromODataUri] int key
  - Primary code is identical.
  - Return value:
    - Web API: OK(object)
    - OData: StatusCode(HttpStatusCode.NoContent)

# Adding REST and OData to a Project

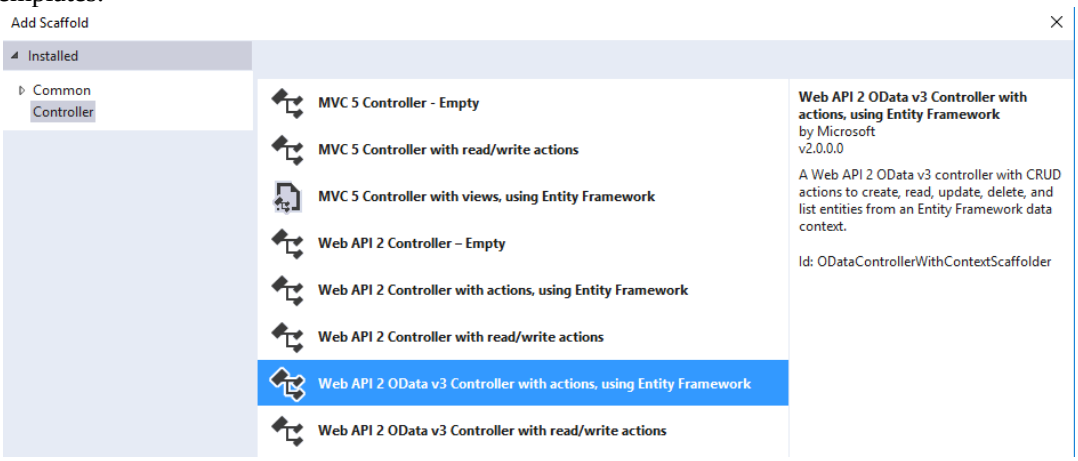
The project should already be configured as a Web API project. See earlier steps for creating Web API projects or adding Web API to existing MVC projects.

Adding an ODATA controller is identical to adding a Web API controller:

Copyright © 2019 MAX Technical Training

- Manually create a controller that inherits from ODataController. (Instead of ApiController for Web API services.)

  or
- Right-click the Controllers folder, click Add and Controller. Select one of the OData V3 Controller templates.




# Customizations

The customizations you make to OData controllers are the same as for Web API controllers.

- You rarely want to expose your full set of properties from your domain models or tables. Consider creating custom models to only expose the needed properties.
- Add try catch blocks around all SaveChanges() and any other code that could fail. Remember that web services have no user interfaces, and that the default error messages may either be useless or reveal too much information to a hacker.

Copyright © 2019 MAX Technical Training

# Module 10 – Where to Go from Here

Model View Controller (MVC) is an architectural pattern. In Visual Studio it is a collection of tools (classes and wizards) and pattern for software development. It is not law, or a religion.

The pattern in a nutshell:

- Encourages the separation of concerns.
- Controllers direct traffic. They don't read/write data, perform business logic or generate HTML.
- Models represent data as plain old C# objects. Models get and save the data to some kind of a database. Models don't generate HTML.
- Views accept objects and generate HTML. They don't directly access databases or perform business logic.

Like the pirates say "the code is more what you'd call "guidelines" than actual rules"!

## Questions to Ask and Decisions to Make

### Routes

- Should you follow the convention of mapping a URL to a Controller by a common name? (http://yourserver/Sales/Index maps to SalesController?) Or, should you write routing rules for each path and not worry about, or depend on, "auto-magic" routing.

- Should routes be preplanned and "carved in stone" so as to never break user bookmarks or search engine links, or should they changed seasonally or as new products are released as requested by marketing.
- Should routes be created with Search Engine Optimization in mind?

### Controllers

- Should there be one Controller per Model, one per section of a project or a Controller per worker role?
    - Should reports be distributed across Controllers, or in a single Controller?
      http://yourserver/Products/ProductListReport
      http://yourserver/Customers/CustomerReport
      or
      http://yourserver/Reports/ProductListReport
      http://yourserver/Reports/CustomerReport

    - Where should the administration functions live? Scattered across each Model Controller (and tagged with [Authorize(Roles…)]), on in a single Administration Controller?

- Should a Controller ever return anything other than Views?

Copyright © 2019 MAX Technical Training

    - The Controller has methods to return JSON, JavaScript, files or just about any kind of content.
    - Web API, and web services in general, are designed to return data.
    - Should non-View requests only be supported in Web API?

- Should you have a "No SQL code in a controller EVER" rule?

# Models

- Should your models be handcrafted with direct calls to SQL Server, or should you use an object to table mapper like the Entity Framework?
- Who owns your database? Can you even use Code First or Design First?
- How many "tiers" should you have in your data model?
    - Should you stick to the one tier that the Entity Framework models guide you to?
    - Should you prohibit ever passing a domain level, table mapped, object to a View? I.e. should you require that all objects being passed to a View be a View Model that only exposes the minimum amount of data that's needed by the View?
- Should a View Model be created for every view… as a matter of policy?
    - It's very rare that every column from a SQL table, or every property from a domain model, will be exposed in a View. Especially things like "datecreated" and all of those nasty GUIDs. ☺ To avoid the risk of an accidental release of data, or an opportunity for a hacker, you might require a View Model just to make the developer think about the entirety of the data being passed to the View.

# Views

- The tools used to build HTML user interfaces often feel like fads, or the JavaScript library of the month club.
    - Should you use the Razor HTML helpers? They are based on jQuery and jQuery Validate. Or should you use other validation libraries.
    - Should you use Bootstrap? After all it is included with MVC and automatically added to wizard generated Views.
    - Should you even use the wizard generated templates at all?

- Should your Views make direct queries to:
    - SQL Server?
        - Best practices say NEVER! But the C# code features can do it.
    - External web services?
        - Should they be allowed to call a Controller/Action or a WebAPI Controller that then makes the web service call? I.e. make the call from the client's browser or make the call from your web server? Going back through your server adds overhead and bandwidth, but give better error handling and maybe pre-filtering the results.
    - Models?
        - Or should models only be passed from a Controller to the View?

- Should you host your own JavaScript libraries, or should you use a Content Delivery Network (CDN).
    - If you host them, you know they will always be there and the version will not change.

Copyright © 2019 MAX Technical Training

    - If you use a CDN you get better performance, but the CDN could go away, be under a DNS attack or have been hacked. Will your site still run?

## Other

- Caching
    - At what tier should caching occur?
        - You can cache at the database server, model, controller or view levels.
    - Who on the team makes the caching decisions?
- Validation
    - Should validation be performed at the client level, the controller level or the model level? Or, everywhere?

Copyright © 2019 MAX Technical Training