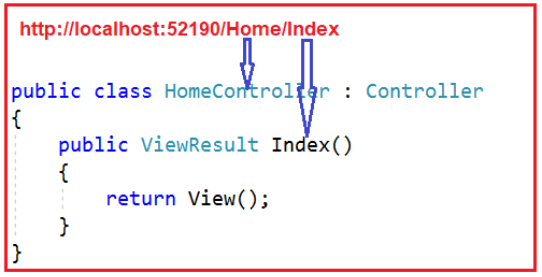
# Routing in ASP.NET MVC

## ****Routing in ASP.NET MVC Application****

1. **What is Routing in MVC?**
2. **Understanding Route and Route Table in ASP.NET MVC.**
3. **What are the different types of Routing supported by ASP.NET MVC?**
4. **How does the Routing work in ASP.NET Web Form?**
5. **What is a Route in MVC Application?**
6. **How Does the Routing work in ASP.NET MVC Application?**
7. **How to Configure a Route in ASP.NET MVC?**
8. **Understanding the URL Pattern in ASP.NET MVC Routing**
9. **How to Register Routes in ASP.NET MVC?**
10. **Examples to understand Routing.**

##### ****What is Routing in ASP.NET MVC?****

The ASP.NET MVC Routing module is responsible for mapping the incoming browser requests (i.e. the incoming URL or incoming HTTP Requests) to a particular controller action method. This mapping is done by the routing rules defined for your application. For example, if we issue a request to the **“/Home/Index**” URL, then it is the Index action method of the Home Controller class that is going to handle the request as shown in the below image.



Routing is not new or specific to the ASP.NET MVC framework. It can also be used with our traditional ASP.NET WebForms application.

##### ****What are the different types of Routing supported by ASP.NET MVC?****

In the ASP.NET MVC application, we can define routes in two ways. They are as follows:

1. **Convention Based Routing**
2. **Attribute-Based Routing.**

In this article, I am going to discuss the Convention Based Routing in ASP.NET MVC Application. In our upcoming article, we will discuss the [**Attribute Routing in MVC**](https://dotnettutorials.net/lesson/attribute-routing-imvc/). Before understanding how routing works in ASP.NET MVC, let us first understand how the routing works in our traditional Web Forms Application.

##### ****How does the Routing work in ASP.NET Web Form?****

As we know in ASP.NET Web Forms application, each and every URL should be matched with a specific .aspx file. For example, a URL **http://xxxxxxxxxxxxx/employeeinfo.aspx** must match with a physical file i.e. employeeinfo.aspx that contains necessary code and HTML for rendering the response to the browser. So in the case of ASP.NET Web Forms, the URL pointing to the file must have its physical existence.

Then ASP.NET Framework introduced the concept of Routing to eliminate the need of mapping each and every URL to a physical file. The Routing Concept enables us to define the URL pattern that maps to the request handler. That request handler can be a class (class methods) or file.

In the case of the ASP.NET Webform application, the request handler is a file (i.e. aspx file) and in the case of ASP.NET MVC Framework, the request handler is the Controller Methods i.e. action methods. For example, **http://xxxxxxxxxxxxxxx/employees** can be mapped to http://dotnettutorials/employeeinfo.aspx in ASP.NET Webforms application and the same URL can be mapped to Employee Controller and Index action method in ASP.NET MVC application.

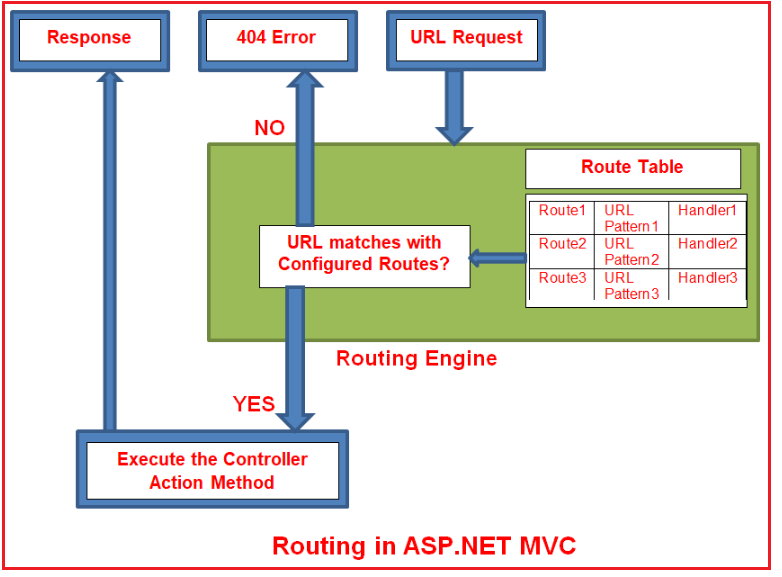
##### ****What is a Route in ASP.NET MVC Application?****

The Route defines the URL pattern and the handler information. The handler can be a physical file, such as an ASPX file in the case of the WebForms application. A handler can also be a class that processes the request, such as a controller in the case of the ASP.NET MVC application.

In the ASP.NET MVC application, all the Routes (URL pattern and Handler Information) are stored in the RouteTable and then the Routing engine uses this RouteTable to determine the appropriate handler class for an incoming HTTP request. Now, I hope you got some basic idea about Route. Let us proceed further and understand how does the routing works in the ASP.NET MVC Application in detail.

##### ****How Does the Routing work in ASP.NET MVC Application?****

Please have a look at the following diagram which illustrates the Routing Process in the ASP.NET MVC Application.



In simple words, we can say that Routing in ASP.NET MVC is a pattern matching mechanism that handles the incoming HTTP request (i.e. incoming URL) and figures out what to do with that incoming HTTP request.

When the client makes a request i.e. makes an HTTP Request, then that request is first received by the Routing Engine. Once the Routing engine receives an HTTP Request, then it figures out the URL Pattern of the incoming request and checks if that URL pattern is present in the Route table. If it found a matching URL pattern for the incoming request in the Route Table, then it fetches the corresponding handler information and forwards the request to the appropriate controller and action method. If there is no match found in the routing table for the incoming HTTP request URL Pattern, then it simply returns a 404 HTTP status code. The routing functionality is implemented in the **System.Web.Routing**.

##### ****How to Configure a Route in ASP.NET MVC?****

Every ASP.NET MVC application must configure (register) at least one route in the RouteConfig class and by default, the ASP.NET MVC Framework provides one default route. But you can configure as many routes as you want. You can register a route within the **RegisterRoutes** method of **RouteConfig** class, which you can find with the **RouteConfig.cs** class file under the **App\_Start** folder. You will find the following code in the RouteConfig class.

**namespace** *FirstMVCDemo*

**{**

**public** **class** RouteConfig

**{**

**public** **static** **void** RegisterRoutes**(**RouteCollection routes**)**

**{**

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

routes.MapRoute**(**

name: "Default", //Route Name

url: "{controller}/{action}/{id}", //Route Pattern

defaults: new

**{**

controller = "Home", //Controller Name

action = "Index", //Action method Name

id = UrlParameter.Optional //Defaut value for above defined parameter

**}**

**)**;

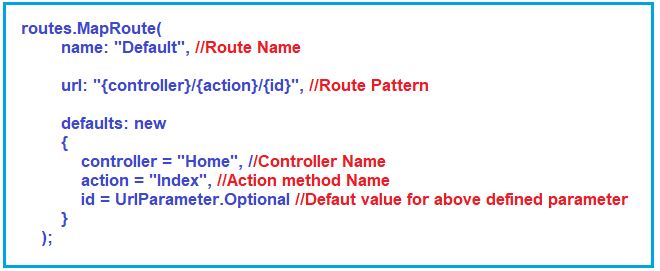
**}**

**}**

**}**

##### Default Route in ASP.NET MVC?

The following image shows the default route of the ASP.NET MVC application which is created by default when we create a new ASP.NET MVC 5 Application.



As you can see in the above image, the Routing is configured using the **MapRoute()** extension method of the **RouteCollection** class, where the Route name is “**Default**” and the URL pattern is “**{controller}/{action}/{id}**“. The Defaults value for the **controller** is **Home**, and the default **action** method is **Index** and the **id** parameter is **optional**. This Route information i.e. the Route Name, URL Pattern, and handler information i.e. the controller name, action method name are stored in the Route table at the application start-up i.e. when the application runs for the first time.

Here Defaults specifies which controller, action method, or value of id parameter should be used if they do not exist in the incoming request URL. In the same way, you can configure other routes using the **MapRoute** method of **RouteCollection**. That we will discuss in our [**next article**](https://dotnettutorials.net/lesson/custom-routing-asp-dot-net-mvc/).

**Note:** The route name should be unique across the entire application. Route name can’t be duplicated.

##### ****Understanding the URL Pattern in ASP.NET MVC Routing:****

The URL pattern is considered only after the domain name in the URL. For example, Suppose your web application is running on www.dotnettutorials.net then the URL pattern **“{controller}/{action}/{id}”**  for your application would be look like **www.xxxxxxxxxxxxxx.net/{controller}/{action}/{id}**.

Anything after the “**www.xxxxxxxxxxxxxxx.net/**” would be considered as the controller name. In the same way, anything after the controller name would be considered as the action name and the value of the id parameter.

Hence you need to provide the controller name followed by the action name and id if it is required. If you will not provide any of the values then default values of these parameters will be provided by the routing engine that means the default controller and action method will handle the request.

##### ****How to Register Routes in ASP.NET MVC?****

Once you create the Route, next we need to register the Route at the Application Level. To register something at the application level, ASP.NET MVC Framework provides the **Global.asax** file. This file contains one class called MvcApplication which is inherited from the System.Web.HttpApplication class. This class provides many methods such as Application\_BeginRequest, Application\_Error, Application\_Start, Session\_Start, Session\_End, etc. So, the question is in which method, we need to register the Routes? We want the Route Table to load all the Routes at the Application Startup i.e. when we run the application for the first time. So, we need to register the routes within the Application\_Start method by calling **RouteConfig.RegisterRoutes(RouteTable.Routes)** method. If you open the **Global.asax** file, you will find the following code in it.

###### **Global.asax**

**namespace** *FirstMVCDemo*

**{**

**public** **class** MvcApplication : System.Web.HttpApplication

**{**

**protected** **void** Application\_Start**()**

**{**

AreaRegistration.RegisterAllAreas**()**;

FilterConfig.RegisterGlobalFilters**(**GlobalFilters.Filters**)**;

RouteConfig.RegisterRoutes**(**RouteTable.Routes**)**;

BundleConfig.RegisterBundles**(**BundleTable.Bundles**)**;

**}**

**}**

**}**

Once you call **RouteConfig.RegisterRoutes(RouteTable.Routes)** method, then notice here, we are passing the RouteCollection (i.e. **RouteTable.Routes**) as an input parameter and if you further notice, within the **RegisterRoutes**method, we add the routes to this RouteCollection property.

##### ****Understanding ASP.NET MVC Routing with an Example:****

To understand ASP.NET MVC Routing lets create a controller called HomeController as shown below

**namespace** *FirstMVCDemo.Controllers*

**{**

**public** **class** HomeController : Controller

**{**

**public** ActionResult Index**()**

**{**

**return** View**()**;

**}**

**}**

**}**

##### ****Example to understand Routing:****

1. **http://localhost:53605/** => controller = **Home**, action = **Index**, id = **none**, since default value of controller and action are Home and Index respectively.
2. **http://localhost:53605/Home** => controller = **Home**, action = **Index**, id = **none**, since default value of action is Index
3. **http://localhost:53605/Home/Index** => controller = **Home**, action = **Index**, id=**none**
4. **http://localhost:53605/Home/Index/5** => controller = **Home**, action = **Index**, id = **5**

The default route that ASP.NET MVC Framework creates for you when you create a new ASP.NET MVC 5 application assumes that you will follow this convention approach. But if you want to follow your own convention then you need to modify the routes or you need to create your own routes that we will discuss in our next article.

##### ****How is the routing table created in ASP.NET MVC?****

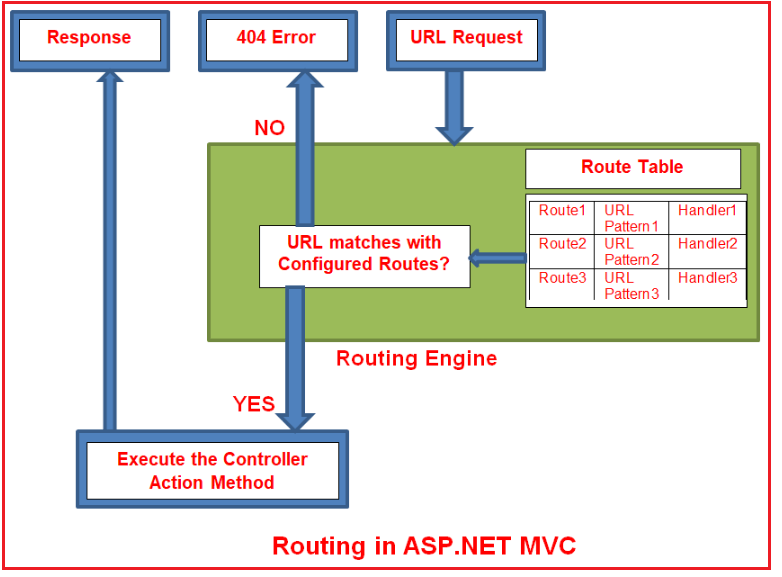
When an MVC application first starts, the Application\_Start() method in global.asax is called. This method calls the**RegisterRoutes()** method. The **RegisterRoutes()** method creates the routing table for the ASP.NET MVC application.

**Creating Custom Routes in ASP.NET MVC**

**How to Create Custom Routes in ASP.NET MVC Application**

1. **What is Custom Routing in MVC?**
2. **Why do we need Custom Routing in ASP.NET MVC?**
3. **How to Create Custom Route in ASP.NET MVC?**

As we already discussed the ASP.NET MVC Routing is a pattern matching mechanism that handles the incoming HTTP request (i.e. incoming URL) and figures out what to do with that incoming HTTP request. The following diagram shows how the routing work in ASP.NET MVC Framework and in our previous article, we already discussed this.



In our previous article, we discussed the default route that is created by ASP.NET MVC Framework and how it handles the incoming request. But if you don’t want to follow the default URL Pattern rather you want to create your own URL Pattern then you need to either modify the default route or you need to create your own route. Let’s discuss how to create our own route in ASP.NET MVC application.

**Creating Custom Routes in ASP.NET MVC Application:**

As we already discussed, if we want to configure any routes then we need to configure the routes within the RegisterRoute method of RouteConfig class using the MapRoute extension method. While configuring the Routes, at least two parameters we need to provide to the MapRoute method i.e. Route name and URL pattern. The Default parameter is optional.

The point that you need to remember is, the Route Names must be unique. You can register multiple custom routes with different names. Consider the following example where we register the “Employee” route.

**namespace** *FirstMVCDemo*

**{**

**public** **class** RouteConfig

**{**

**public** **static** **void** RegisterRoutes**(**RouteCollection routes**)**

**{**

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

routes.MapRoute**(**

name: "Employee",

url: " Employee/{id}",

defaults: new **{** controller = "Employee", action = "Index" **}**

**)**;

routes.MapRoute**(**

name: "Default", //Route Name

url: "{controller}/{action}/{id}", //Route Pattern

defaults: new

**{**

controller = "Home", //Controller Name

action = "Index", //Action method Name

id = UrlParameter.Optional //Defaut value for above defined parameter

**}**

**)**;

**}**

**}**

**}**

So, in this way you can configure as many as routes you want with your own URL pattern in ASP.NET MVC Application. Let’s add Employee Controller to our application

**namespace** *FirstMVCDemo.Controllers*

**{**

**public** **class** EmployeeController : Controller

**{**

**public** ActionResult Index**()**

**{**

**return** View**()**;

**}**

**}**

**}**

**Code Explanation:**

The URL pattern for the Employee route is **Employee/{id}** which specifies that any URL that starts with **domainName/Employee**, must be handled by EmployeeController. Notice that we haven’t specified **{action}** in the URL pattern because we want every URL that starts with Employee should always use Index action of EmployeeController. We have specified the default controller and action to handle any URL request which starts from the domain name/Employee.

The ASP.NET MVC framework evaluates each route in sequence. It starts with the first configured route and if the incoming URL doesn’t satisfy the First URL pattern of the route then it will evaluate the second route and so on. In the above example, the routing engine will evaluate the Employee route first and if the incoming URL doesn’t start with domainName/Employee then only it will consider the second route which is the default route.

The following URLs will be mapped to the Employee route.

1. **http://localhost:53605/Employee**
2. **http://localhost:53605/Employee/Index**
3. [**http://localhost:53605/Employee/Index/3**](http://localhost:53605/Employee/Index/3)

**Note:** Always put the more specific route on the top order while defining multiple routes, since the routing system checks the incoming URL pattern from the top and as it gets the matched route it will consider that. It will not check further routes after the matching pattern.

**Route Constraints in ASP.NET MVC**

**Route Constraints in ASP.NET MVC Application**

1. **What are Route Constraints in ASP.NET MVC Application?**
2. **Creating Route Constraint to a Set of Specific Values in MVC Application.**
3. **What is the difference between Routing and URL Rewriting?**
4. **How is the routing table created in ASP.NET MVC?**

**What are Route Constraints in ASP.NET MVC Application?**

The Route Constraint in ASP.NET MVC Routing allows us to apply a regular expression to a URL segment to restrict whether the route will match the request. In simple words, we can say that the Route constraint is a way to put some validation around the defined route. Suppose you have defined the following route in your application.

**namespace** *FirstMVCDemo*

**{**

**public** **class** RouteConfig

**{**

**public** **static** **void** RegisterRoutes**(**RouteCollection routes**)**

**{**

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

routes.MapRoute**(**

name: "Default", //Route Name

url: "{controller}/{action}/{id}", //Route Pattern

defaults: new

**{**

controller = "Home", //Controller Name

action = "Index", //Action method Name

id = UrlParameter.Optional //Defaut value for above defined parameter

**}**

**)**;

**}**

**}**

**}**

Now you want to restrict the incoming request URL with numeric id only. Now let’s see how we can do this with the help of regular expression in the ASP.NET MVC Application.

**Restrict to numeric Id only**

There is another overloaded version of the MapRoute extension method which takes constraints as a parameter. Using this parameter we can set a regular expression that will validate the incoming URL route parameters. In the below code, you can observe, we have passed **constraints :new { id = @”\d+” }** as the fourth parameter to the MapRoute extension method, and this regular expression will restrict the id parameter to be numeric only,

**namespace** *FirstMVCDemo*

**{**

**public** **class** RouteConfig

**{**

**public** **static** **void** RegisterRoutes**(**RouteCollection routes**)**

**{**

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

routes.MapRoute**(**

name: "Default", //Route Name

url: "{controller}/{action}/{id}", //Route Pattern

defaults: new

**{**

controller = "Home", //Controller Name

action = "Index", //Action method Name

id = UrlParameter.Optional //Defaut value for above defined parameter

**}**,

constraints :new **{** id = @"\d+" **}** //Restriction for id

**)**;

**}**

**}**

**}**

So now if you give a non-numeric value for the id parameter then that request will be handled by another route or if there are no matching routes then the “**The resource could not be found**” error will be thrown. So now for the above route, the routing engine will only consider the URLs which have only numeric id like **http://xxxxxxxxxxxx.com/Home/Index/10**

**Creating Route Constraint for Restricting Controller and Actions**

Suppose you want to restrict the user for those URLs that have controller name with **H prefix** and action name should be only **Details or About**. Now let’s see how we can achieve this with the help of regular expression.

**public** **class** RouteConfig

**{**

**public** **static** **void** RegisterRoutes**(**RouteCollection routes**)**

**{**

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

routes.MapRoute**(**

"Default", // Route name

"{controller}/{action}/{id}", // Route Pattern

new **{** controller = "Home", action = "Index", id = UrlParameter.Optional **}**, // Default values for parameters

new **{** controller = "^H.\*", action = "^Details$|^About$" **}** //Restriction for controller and action

**)**;

**}**

**}**

Now for this route, the routing engine will consider only those URLs which have controller name with **H prefix,** and action names should be only **Details** or **Index.** such as **http://xxxxxxxxxxxxxxxxx.net/Home/Index**, **http://xxxxxxxxxxxxxxx.net/Home/Details**, and **http://xxxxxxxxxxxxx.net/**, **http:// xxxxxxxxxxxx.net/Home** else it will consider that URL is not matched with this route.

Now you may be a little bit confused about why it will consider the **http://xxxxxxxxxxxx.net/**, **http:// xxxxxxxxxxxx.net/Home** URLs?

It will also consider both these since route constraint is checked after the provided default values for controller and action. In the above route default values for controller and action are Home and Index so these two URLs will also be matched. Like this, you can restrict the user according to your needs.

**What is the difference between Routing and URL Rewriting?**

Many developers compare Routing to URL rewriting since both look similar and can be used to make SEO-friendly URLs. The main differences between routing and URL rewriting are given below:

1. URL rewriting is focused on mapping one URL (new URL) to another URL (old URL) while routing is focused on mapping a URL to a resource i.e. controller action method.
2. URL rewriting rewrites your old URL to a new one while routing never rewrites your old URL to a new one but it maps to the original route.

# Attribute Routing in ASP.NET MVC

## ****Attribute Routing in ASP.NET MVC Application****

1. **What is Attribute Routing in MVC?**
2. **Why do we need Attribute Routing in ASP.NET MVC Application?**
3. **How to Enabling Attribute Routing in ASP.NET MVC Application?**
4. **Examples of using Attribute Routing.**
5. **What are the advantages of using Attribute Routing in MVC?**
6. **Can we use both Attribute Routing and Convention-based routing in a single MVC project?**

##### ****What is Attribute Routing in ASP.NET MVC?****

If we are defining Routes by using the [Route] attribute is called Attribute Routing. It provides you more control over the URIs by defining routes directly on actions and controllers in your ASP.NET MVC application.

**Note:** The earlier style of routing, called [convention-based routing](https://dotnettutorials.net/lesson/asp-dot-net-mvc-routing/), is still fully supported by ASP.NET MVC 5. In fact, you can combine both these routing approaches in the same project.

##### ****Why do we need Attribute Routing in ASP.NET MVC Application?****

Let us understand the need for attribute routing in the ASP.NET MVC Application with an example. Create a new Empty ASP.NET MVC application with the name **AttributeRoutingDemoInMVC.**Then Right-click on the “**Models**” folder and add a class file with the name **Student.cs**and then copy and paste the following code in it.

**namespace** *AttributeRoutingDemoInMVC.Models*

**{**

**public** **class** Student

**{**

**public** **int** Id **{** **get**; **set**; **}**

**public** string Name **{** **get**; **set**; **}**

**}**

**}**

##### ****Creating Students Controller:****

Right-click on the Controllers folder and then add a new ASP.NET MVC 5 Empty controller with the name **StudentsController.cs** and then copy and paste the following code in it.

**namespace** *AttributeRoutingDemoInMVC.Controllers*

**{**

**public** **class** StudentsController : Controller

**{**

**static** List**<**Student**>** students = new List**<**Student**>()**

**{**

new Student**()** **{** Id = 1, Name = "Pranaya" **}**,

new Student**()** **{** Id = 2, Name = "Priyanka" **}**,

new Student**()** **{** Id = 3, Name = "Anurag" **}**,

new Student**()** **{** Id = 4, Name = "Sambit" **}**

**}**;

**[**HttpGet**]**

**public** ActionResult GetAllStudents**()**

**{**

**return** View**(**students**)**;

**}**

**[**HttpGet**]**

**public** ActionResult GetStudentByID**(int** studentID**)**

**{**

Student studentDetails = students.FirstOrDefault**(**s =**>** s.Id == studentID**)**;

**return** View**(**studentDetails**)**;

**}**

**[**HttpGet**]**

**public** ActionResult GetStudentCourses**(int** studentID**)**

**{**

List**<**string**>** CourseList = new List**<**string**>()**;

**if** **(**studentID == 1**)**

CourseList = new List**<**string**>()** **{** "ASP.NET", "C#.NET", "SQL Server" **}**;

**else** **if** **(**studentID == 2**)**

CourseList = new List**<**string**>()** **{** "ASP.NET MVC", "C#.NET", "ADO.NET" **}**;

**else** **if** **(**studentID == 3**)**

CourseList = new List**<**string**>()** **{** "ASP.NET WEB API", "C#.NET", "Entity Framework" **}**;

**else**

CourseList = new List**<**string**>()** **{** "Bootstrap", "jQuery", "AngularJs" **}**;

ViewBag.CourseList = CourseList;

**return** View**()**;

**}**

**}**

**}**

**Add the respective views for the above three action methods.**

##### ****GetAllStudents.cshtml****

The following view will render all the student data. The following is a strongly typed view and the model for this view is IEnumerable<AttributeRoutingDemoInMVC.Models.Student>.

@model IEnumerable**<AttributeRoutingDemoInMVC.Models.Student>**

@{

ViewBag.Title = "GetAllStudents";

}

**<h2>**GetAllStudents**</h2>**

**<table** class="table"**>**

**<tr>**

**<th>**

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

**</th>**

**<th>**

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

**</th>**

**</tr>**

@foreach (var item in Model)

{

**<tr>**

**<td>**

@Html.DisplayFor(modelItem => item.Id)

**</td>**

**<td>**

@Html.DisplayFor(modelItem => item.Name)

**</td>**

**</tr>**

}

**</table>**

##### ****GetStudentByID.cshtml****

The following view will render the data of a particular student. The following view is a strongly typed view and the model for this view is AttributeRoutingDemoInMVC.Models.Student.

@model AttributeRoutingDemoInMVC.Models.Student

@{

ViewBag.Title = "GetStudentByID";

}

**<h2>**GetStudentByID**</h2>**

**<div>**

**<h4>**Student**</h4>**

**<hr** **/>**

**<dl** class="dl-horizontal"**>**

**<dt>**

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

**</dt>**

**<dd>**

@Html.DisplayFor(model => model.Name)

**</dd>**

**<dt>**

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

**</dt>**

**<dd>**

@Html.DisplayFor(model => model.Id)

**</dd>**

**</dl>**

**</div>**

###### **GetStudentCourses.cshtml**

The following view will render the courses of a particular student. The following view is not a strongly-typed view and here we are using ViewBag to fetch the courses.

@{

ViewBag.Title = "GetStudentCourses";

}

**<h2>**GetStudentCourses**</h2>**

@foreach (var item in ViewBag.CourseList)

{

**<li>**@item**</li>**

}

In ASP.NET MVC, we have convention-based routing that defines routes using the route templates. When we create a new ASP.NET MVC 5 Application using Visual Studio, then by default a route (i.e. default route) is created in the **RouteConfig.cs** class file. The default route is shown below.

**namespace** *AttributeRoutingDemoInMVC*

**{**

**public** **class** RouteConfig

**{**

**public** **static** **void** RegisterRoutes**(**RouteCollection routes**)**

**{**

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

routes.MapRoute**(**

name: "Default",

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

defaults: new **{** controller = "Home", action = "Index", id = UrlParameter.Optional **}**

**)**;

**}**

**}**

**}**

Now we can access the above three methods using the following URL

1. **http://localhost:58316/Students/GetAllStudents**
2. **http://localhost:58316/Students/GetStudentByID?studentID=1**
3. <http://localhost:58316/Students/GetStudentCourses?studentID=1>

In some scenarios, convention-based routing is very difficult to support certain URL patterns. But those URL patterns can be easily achieved by using the Attribute Routing in ASP.NET MVC. For example, resources often contain child resources like Clients have Ordered; Movies have Actors, and Books have Authors, and so on. It’s natural to create URIs that reflects these relations like as **/clients/1/orders**

In our example, If we want the URL Pattern “**/students/1/courses**” and if that URL Pattern should be mapped to **GetStudentCourses(int studentID),** then this type of URL is very difficult to create using the convention-based routing in ASP.NET MVC Application. But by using Attribute Routing which is a new feature introduced in ASP.NET MVC 5, this type of URL pattern can be achieved very easily.

##### ****How to use Attribute Routing in ASP.NET MVC?****

The attribute routing can only be used with ASP.NET MVC 5, if you are using MVC 4, or lower version, you can not use Attribute Routing The first thing that you need to do is Enabling Attribute Routing in RouteConfig.cs class.

##### ****How to Enabling Attribute Routing in ASP.NET MVC Application?****

Enabling the Attribute Routing in ASP.NET MVC 5 Application is very simple. You just need to add a call to **routes.MapMvcAttributeRoutes()** method within the **RegisterRoutes()** method of **RouteConfig.cs** file. So, open the **RouteConfig.cs** file which is stored within the **App\_Start** folder, and then just adds **routes.MapMvcAttributeRoutes();** method just above the **routes.MapRoute** method as the show is below. The Attribute routing should configure before the convention-based routing.

**namespace** *AttributeRoutingDemoInMVC*

**{**

**public** **class** RouteConfig

**{**

**public** **static** **void** RegisterRoutes**(**RouteCollection routes**)**

**{**

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

//Enabling attribute routing

routes.MapMvcAttributeRoutes**()**;

routes.MapRoute**(**

name: "Default",

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

defaults: new **{** controller = "Home", action = "Index", id = UrlParameter.Optional **}**

**)**;

**}**

**}**

**}**

With the above changes in place, now your MVC application is going to support **Attribute Routing**. The next step is to decorate either the controller or action method with the Route attribute.

##### ****Using Route Attribute in ASP.NET MVC:****

Modify the GetStudentCourses Action Method as shown below. As you can see, here we decorate the **GetStudentCourses**Action Method with **[Route(“students/{studentID}/courses”)]** Attribute.

**[**HttpGet**]**

**[**Route**(**"students/{studentID}/courses"**)]**

**public** ActionResult GetStudentCourses**(int** studentID**)**

**{**

List**<**string**>** CourseList = new List**<**string**>()**;

**if** **(**studentID == 1**)**

CourseList = new List**<**string**>()** **{** "ASP.NET", "C#.NET", "SQL Server" **}**;

**else** **if** **(**studentID == 2**)**

CourseList = new List**<**string**>()** **{** "ASP.NET MVC", "C#.NET", "ADO.NET" **}**;

**else** **if** **(**studentID == 3**)**

CourseList = new List**<**string**>()** **{** "ASP.NET WEB API", "C#.NET", "Entity Framework" **}**;

**else**

CourseList = new List**<**string**>()** **{** "Bootstrap", "jQuery", "AngularJs" **}**;

ViewBag.CourseList = CourseList;

**return** View**()**;

**}**

With the above changes in place, you can access the above action by using **students/{studentID}/courses**URL**.**Here, **studentID**will be replaced with the values. Now navigate to the following URL and you should get the output as expected.

**http://localhost:58316/students/2/courses** (You need to change the port number)

##### ****What are the advantages of using Attribute Routing in ASP.NET MVC5?****

1. It gives us more control over the URIs than convention-based routing. Creating URI patterns like hierarchies of resources (For example, students have courses, Departments have employees) is very difficult with convention-based routing.
2. Reduces the chances for errors, if a route is modified incorrectly in RouteConfig.cs then it may affect the entire application’s routing.
3. May decouple controller and action names from route entirely.
4. Easy to map two routes pointing to the same action.

##### ****Can we use both Attribute Routing and Convention-based routing in a single MVC project?****

Yes, both the routing mechanisms can be combined in a single ASP.NET MVC project. The controller action methods that have the [Route] attribute uses Attribute Routing, and the action methods without the [Route] attribute uses Convention-based routing.

# ASP.NET MVC Attribute Routing with Optional Parameter

##### ****ASP.NET MVC Attribute Routing with Optional Parameter:****

You can also define a URI parameter as optional by adding a **question mark (“?”)** to the route parameter. You can also specify the default value by using **parameter = value**. If this is not clear at the moment, then don’t worry, we will see both these approaches with examples.

To understand this concept, let’s add a controller within the controller folder with the name as HomeController and then copy and paste the below codes. As you can see, in the MVC action method, we place a question mark after the parameter studentName within the Route attribute as **[Route(“MVCTest/{studentName ?}”)]** which makes the studentName parameter optional. If you have not specified any value for the studentName parameter in the URL, then it will store a null value in it and if you specified any value in the URL for the studentName parameter, then that value will be mapped to the studentName Parameter. In the WEBAPI action method, we have specified a default value for the studentName Parameter directly in the Route attribute as **Route(“WEBAPITest/{studentName = Pranaya}”)]**.

**namespace** *AttributeRoutingDemoInMVC.Controllers*

**{**

**public** **class** HomeController : Controller

**{**

// Optional URI Parameter

// URL: /MVCTest/

// URL: /MVCTest/Pranaya

**[**Route**(**"MVCTest/{studentName ?}"**)]**

**public** ActionResult MVC**(**string studentName**)**

**{**

ViewBag.Message = "Welcome to ASP.NET MVC!";

**return** View**()**;

**}**

// Optional URI Parameter with default value

// URL: /WEBAPITest/

// URL: /WEBAPITest/Pranaya

**[**Route**(**"WEBAPITest/{studentName = Pranaya}"**)]**

**public** ActionResult WEBAPI**(**string studentName**)**

**{**

ViewBag.Message = "Welcome to ASP.NET WEB API!";

**return** View**()**;

**}**

**}**

**}**

In the above example, both **/MVCTest** and **/MVCTest/Pranaya** will route to the “**MVC**” action method, similarly, **/WEBAPITest** and **/WEBAPITest/Pranaya** will route to the “**WEBAPI**” action method. Then add the below two views:

###### **MVC.cshtml**

@{

ViewBag.Title = "MVC";

}

@ViewBag.Message

###### **WEBAPI.cshtml**

@{

ViewBag.Title = "WEBAPI";

}

@ViewBag.Message

Now run the application and navigate to the below four URLs and see everything is working as expected.

1. **URL1: http://localhost:58316/MVCTest**
2. **URL2: http://localhost:58316/MVCTest/Pranaya**
3. **URL3: http://localhost:58316//WEBAPITest**
4. **URL4:** <http://localhost:58316//WEBAPITest/Pranaya>

**Route Prefix in ASP.NET MVC Attribute Routing**

**Route Prefix in ASP.NET MVC Attribute Routing**

**Why do we need Route Prefix in ASP.NET MVC Attribute Routing?**

Let’s understand the need and use of Route Prefix with one example. Let’s modify the StudentController class as shown below.

**namespace** *AttributeRoutingDemoInMVC.Controllers*

**{**

**public** **class** StudentsController : Controller

**{**

**static** List**<**Student**>** students = new List**<**Student**>()**

**{**

new Student**()** **{** Id = 1, Name = "Pranaya" **}**,

new Student**()** **{** Id = 2, Name = "Priyanka" **}**,

new Student**()** **{** Id = 3, Name = "Anurag" **}**,

new Student**()** **{** Id = 4, Name = "Sambit" **}**

**}**;

**[**HttpGet**]**

**[**Route**(**"students"**)]**

**public** ActionResult GetAllStudents**()**

**{**

**return** View**(**students**)**;

**}**

**[**HttpGet**]**

**[**Route**(**"students/{studentID}"**)]**

**public** ActionResult GetStudentByID**(int** studentID**)**

**{**

Student studentDetails = students.FirstOrDefault**(**s =**>** s.Id == studentID**)**;

**return** View**(**studentDetails**)**;

**}**

**[**HttpGet**]**

**[**Route**(**"students/{studentID}/courses"**)]**

**public** ActionResult GetStudentCourses**(int** studentID**)**

**{**

List**<**string**>** CourseList = new List**<**string**>()**;

**if** **(**studentID == 1**)**

CourseList = new List**<**string**>()** **{** "ASP.NET", "C#.NET", "SQL Server" **}**;

**else** **if** **(**studentID == 2**)**

CourseList = new List**<**string**>()** **{** "ASP.NET MVC", "C#.NET", "ADO.NET" **}**;

**else** **if** **(**studentID == 3**)**

CourseList = new List**<**string**>()** **{** "ASP.NET WEB API", "C#.NET", "Entity Framework" **}**;

**else**

CourseList = new List**<**string**>()** **{** "Bootstrap", "jQuery", "AngularJs" **}**;

ViewBag.CourseList = CourseList;

**return** View**()**;

**}**

**}**

**}**

As you can see in the above code, we are using route attributes at the action level to define routes, and further if you notice, all the routes in the **StudentsController** start with the same prefix i.e. **students.** That means “**students**” is the common prefix for all the routes in Student Controller.

Here, instead of writing the common prefix “**students**” at each action method, we can specify the common prefix **students**for the entire Student Controller (for all the action methods of student controller) using the **[RoutePrefix]** attribute at the controller level as shown below.

**namespace** *AttributeRoutingDemoInMVC.Controllers*

**{**

**[**RoutePrefix**(**"students"**)]**

**public** **class** StudentsController : Controller

**{**

**static** List**<**Student**>** students = new List**<**Student**>()**

**{**

new Student**()** **{** Id = 1, Name = "Pranaya" **}**,

new Student**()** **{** Id = 2, Name = "Priyanka" **}**,

new Student**()** **{** Id = 3, Name = "Anurag" **}**,

new Student**()** **{** Id = 4, Name = "Sambit" **}**

**}**;

**[**HttpGet**]**

**[**Route**]**

//This will be translated to /students

**public** ActionResult GetAllStudents**()**

**{**

**return** View**(**students**)**;

**}**

**[**HttpGet**]**

**[**Route**(**"{studentID}"**)]**

//This will be translated to /students/2

**public** ActionResult GetStudentByID**(int** studentID**)**

**{**

Student studentDetails = students.FirstOrDefault**(**s =**>** s.Id == studentID**)**;

**return** View**(**studentDetails**)**;

**}**

**[**HttpGet**]**

**[**Route**(**"{studentID}/courses"**)]**

//This will be translated to /students/2/course

**public** ActionResult GetStudentCourses**(int** studentID**)**

**{**

List**<**string**>** CourseList = new List**<**string**>()**;

**if** **(**studentID == 1**)**

CourseList = new List**<**string**>()** **{** "ASP.NET", "C#.NET", "SQL Server" **}**;

**else** **if** **(**studentID == 2**)**

CourseList = new List**<**string**>()** **{** "ASP.NET MVC", "C#.NET", "ADO.NET" **}**;

**else** **if** **(**studentID == 3**)**

CourseList = new List**<**string**>()** **{** "ASP.NET WEB API", "C#.NET", "Entity Framework" **}**;

**else**

CourseList = new List**<**string**>()** **{** "Bootstrap", "jQuery", "AngularJs" **}**;

ViewBag.CourseList = CourseList;

**return** View**()**;

**}**

**}**

**}**

As you can see in the above code, we applied the **[RoutePrefix(“students”)]** attribute at the Controller level. This Route Prefix attribute eliminates the need to repeat the common prefix “**students**” on each and every controller action method. However, sometimes we may need to override the route prefix attribute.

**How to Override Route Prefix Attribute in ASP.NET MVC Attribute Routing?**

Let us understand this with an example. First, add a class file with the name “**Teacher.cs**” within the Models Folder. To do so right-click on the **Models** folder, and then add a new class file with the name **“Teacher.cs”**, and then copy and paste the following code in it.

**namespace** *AttributeRoutingDemoInMVC.Models*

**{**

**public** **class** Teacher

**{**

**public** **int** Id **{** **get**; **set**; **}**

**public** string Name **{** **get**; **set**; **}**

**}**

**}**

Now, add the following **GetTeachers()** action method within the **“StudentsController”**.

**public** ActionResult GetTeachers**()**

**{**

List**<**Teacher**>** teachers = new List**<**Teacher**>()**

**{**

new Teacher**()** **{** Id = 1, Name = "James" **}**,

new Teacher**()** **{** Id = 2, Name = "Patrik" **}**,

new Teacher**()** **{** Id = 3, Name = "Smith" **}**

**}**;

**return** View**(**teachers**)**;

**}**

After adding the **GetTeachers()** action method the **“StudentsController”** class should look as shown below

**namespace** *AttributeRoutingDemoInMVC.Controllers*

**{**

**[**RoutePrefix**(**"students"**)]**

**public** **class** StudentsController : Controller

**{**

**static** List**<**Student**>** students = new List**<**Student**>()**

**{**

new Student**()** **{** Id = 1, Name = "Pranaya" **}**,

new Student**()** **{** Id = 2, Name = "Priyanka" **}**,

new Student**()** **{** Id = 3, Name = "Anurag" **}**,

new Student**()** **{** Id = 4, Name = "Sambit" **}**

**}**;

**[**HttpGet**]**

**[**Route**]**

//This will be translated to /students

**public** ActionResult GetAllStudents**()**

**{**

**return** View**(**students**)**;

**}**

**[**HttpGet**]**

**[**Route**(**"{studentID}"**)]**

//This will be translated to /students/2

**public** ActionResult GetStudentByID**(int** studentID**)**

**{**

Student studentDetails = students.FirstOrDefault**(**s =**>** s.Id == studentID**)**;

**return** View**(**studentDetails**)**;

**}**

**[**HttpGet**]**

**[**Route**(**"{studentID}/courses"**)]**

//This will be translated to /students/2/course

**public** ActionResult GetStudentCourses**(int** studentID**)**

**{**

List**<**string**>** CourseList = new List**<**string**>()**;

**if** **(**studentID == 1**)**

CourseList = new List**<**string**>()** **{** "ASP.NET", "C#.NET", "SQL Server" **}**;

**else** **if** **(**studentID == 2**)**

CourseList = new List**<**string**>()** **{** "ASP.NET MVC", "C#.NET", "ADO.NET" **}**;

**else** **if** **(**studentID == 3**)**

CourseList = new List**<**string**>()** **{** "ASP.NET WEB API", "C#.NET", "Entity Framework" **}**;

**else**

CourseList = new List**<**string**>()** **{** "Bootstrap", "jQuery", "AngularJs" **}**;

ViewBag.CourseList = CourseList;

**return** View**()**;

**}**

**public** ActionResult GetTeachers**()**

**{**

List**<**Teacher**>** teachers = new List**<**Teacher**>()**

**{**

new Teacher**()** **{** Id = 1, Name = "James" **}**,

new Teacher**()** **{** Id = 2, Name = "Patrik" **}**,

new Teacher**()** **{** Id = 3, Name = "Smith" **}**

**}**;

**return** View**(**teachers**)**;

**}**

**}**

**}**

**GetTeachers.cshtml**

Next, add **GetTeachers.cshtml** view and then copy and paste the below code.

@model IEnumerable**<**AttributeRoutingDemoInMVC.Models.Teacher**>**

@**{**

ViewBag.Title = "GetTeachers";

**}**

**<**h2**>**GetTeachers**<**/h2**>**

**<**table **class**="table"**>**

**<**tr**>**

**<**th**>**

@Html.DisplayNameFor**(**model =**>** model.Id**)**

**<**/th**>**

**<**th**>**

@Html.DisplayNameFor**(**model =**>** model.Name**)**

**<**/th**>**

**<**/tr**>**

@**foreach** **(**var item in Model**)**

**{**

**<**tr**>**

**<**td**>**

@Html.DisplayFor**(**modelItem =**>** item.Id**)**

**<**/td**>**

**<**td**>**

@Html.DisplayFor**(**modelItem =**>** item.Name**)**

**<**/td**>**

**<**/tr**>**

**}**

**<**/table**>**

We want the **GetTeachers()** method to be mapped to URI **“/tech/teachers”.**

**[**Route**(**"tech/teachers"**)]**

**public** ActionResult GetTeachers**()**

**{**

List**<**Teacher**>** teachers = new List**<**Teacher**>()**

**{**

new Teacher**()** **{** Id = 1, Name = "James" **}**,

new Teacher**()** **{** Id = 2, Name = "Patrik" **}**,

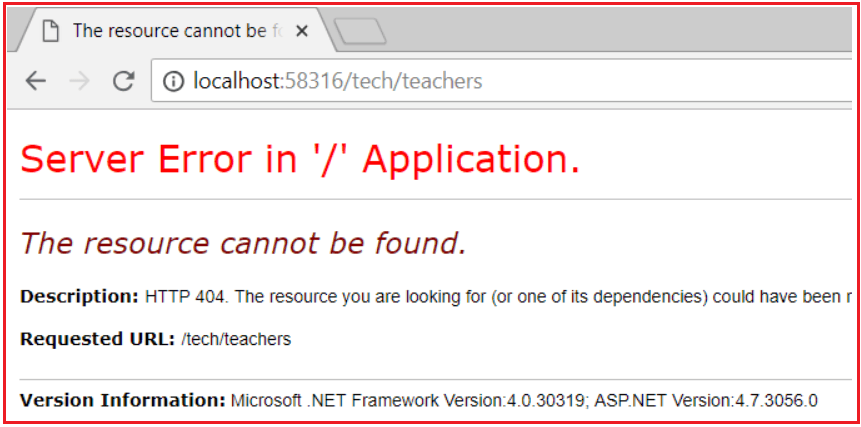
new Teacher**()** **{** Id = 3, Name = "Smith" **}**

**}**;

**return** View**(**teachers**)**;

**}**

If we use the **[Route]** attribute on the **GetTeachers()** method as shown in the above code and when we navigate to **tech/teachers**URL, we will get the following error.



But if we navigate to **/students/tech/teachers** then we will get the output as expected that is the list of teachers. This is because the **[RoutePrefix(“students”)]**attribute on StudentsController. So there is definitely a need to override the **RoutePrefix** attribute used on the StudentsController. To override the RoutePrefix attribute, we need to use the **~** (tilde) symbol while defining the route as shown in the below code. Here, we are using **[Route(“~/tech/teachers”)]** attribute and the tilde (~) symbol in the Route attribute will override or you can ignore the Route Prefix attribute defined at the StudentsController level.

**[**Route**(**"~/tech/teachers"**)]**

**public** ActionResult GetTeachers**()**

**{**

List**<**Teacher**>** teachers = new List**<**Teacher**>()**

**{**

new Teacher**()** **{** Id = 1, Name = "James" **}**,

new Teacher**()** **{** Id = 2, Name = "Patrik" **}**,

new Teacher**()** **{** Id = 3, Name = "Smith" **}**

**}**;

**return** View**(**teachers**)**;

**}**

With this change **GetTeachers()** action method is now mapped to URI “**/tech/teachers**” as expected.

**What is the use of the RoutePrefix Attribute in ASP.NET MVC Attribute Routing?**

RoutePrefix attribute is used to specify the common route prefix at the controller level to eliminate the need to repeat that common route prefix on each and every controller action method

**How to override the route prefix?**

Use **~** character to override the route prefix

# Route Constraints in ASP.NET MVC Attribute Routing

##### ****Route Constraints in ASP.NET MVC Attribute Routing****

Route Constraints in ASP.NET MVC Attribute Routing are nothing but a set of rules that can be applied to the route parameters. By using the “:” symbol we can applied Route Constraints to the Route Parameters. Let us understand How to use the Route Constraints with one example. First, modify the Students Controller as shown below.

**namespace** *AttributeRoutingDemoInMVC.Controllers*

**{**

**[**RoutePrefix**(**"students"**)]**

**public** **class** StudentsController : Controller

**{**

**static** List**<**Student**>** students = new List**<**Student**>()**

**{**

new Student**()** **{** Id = 1, Name = "Pranaya" **}**,

new Student**()** **{** Id = 2, Name = "Priyanka" **}**,

new Student**()** **{** Id = 3, Name = "Anurag" **}**,

new Student**()** **{** Id = 4, Name = "Sambit" **}**

**}**;

**[**HttpGet**]**

**[**Route**(**"{studentID}"**)]**

**public** ActionResult GetStudentDetails**(int** studentID**)**

**{**

Student studentDetails = students.FirstOrDefault**(**s =**>** s.Id == studentID**)**;

**return** View**(**studentDetails**)**;

**}**

**}**

**}**

###### **GetStudentDetails.cshtml**

Now add the GetStudentDetails view and then copy and paste the below code into it.

@model AttributeRoutingDemoInMVC.Models.Student

@{

ViewBag.Title = "GetStudentDetails";

}

**<h2>**GetStudentDetails**</h2>**

**<div>**

**<h4>**Student**</h4>**

**<hr** **/>**

**<dl** class="dl-horizontal"**>**

**<dt>**

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

**</dt>**

**<dd>**

@Html.DisplayFor(model => model.Name)

**</dd>**

**<dt>**

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

**</dt>**

**<dd>**

@Html.DisplayFor(model => model.Id)

**</dd>**

**</dl>**

**</div>**

Now If you navigate to **/students/1** URL, then **GetStudentDetails(int studentID)** action method is executed and we get the details of the student whose id is 1 as expected. Let’s change our requirement, in addition to retrieving the student details by **“student Id”**, we also want to retrieve the student details by “**student Name**“. So let’s add another **GetStudentDetails()** method by taking a string parameter as shown below.

**namespace** *AttributeRoutingDemoInMVC.Controllers*

**{**

**[**RoutePrefix**(**"students"**)]**

**public** **class** StudentsController : Controller

**{**

**static** List**<**Student**>** students = new List**<**Student**>()**

**{**

new Student**()** **{** Id = 1, Name = "Pranaya" **}**,

new Student**()** **{** Id = 2, Name = "Priyanka" **}**,

new Student**()** **{** Id = 3, Name = "Anurag" **}**,

new Student**()** **{** Id = 4, Name = "Sambit" **}**

**}**;

**[**HttpGet**]**

**[**Route**(**"{studentID}"**)]**

**public** ActionResult GetStudentDetails**(int** studentID**)**

**{**

Student studentDetails = students.FirstOrDefault**(**s =**>** s.Id == studentID**)**;

**return** View**(**studentDetails**)**;

**}**

**[**HttpGet**]**

**[**Route**(**"{studentName}"**)]**

**public** ActionResult GetStudentDetails**(**string studentName**)**

**{**

Student studentDetails = students.FirstOrDefault**(**s =**>** s.Name == studentName**)**;

**return** View**(**studentDetails**)**;

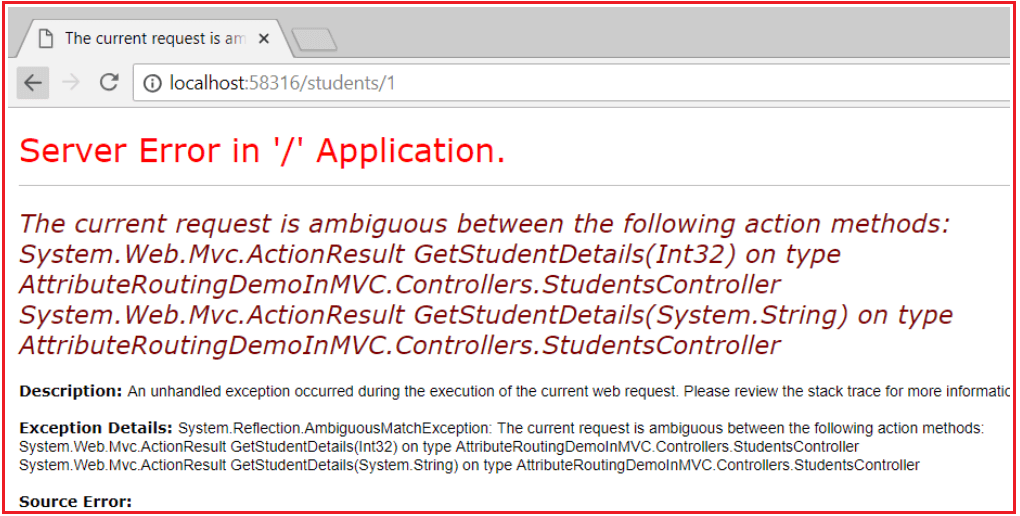
**}**

**}**

**}**

At this point build the solution, and navigate to the following URI’s  
**http://localhost:58316/students/1**  
**http://localhost:58316/students/Pranaya**

In both the case, you will get the following error.



This is because the ASP.NET MVC Framework does not know or does not identify which version of the **GetStudentDetails()** action method to use when the request comes. This is where constraints play a very important role. If an integer is specified in the URL (**/students/1**), then we want to execute the **GetStudentDetails(int studentId)**action method which has an integer parameter. If a string is specified in the URI (**/students/Pranaya**), then we want to execute the **GetStudentDetails(string studentName)**action method which has the string parameter

##### ****Applying Attribute Route Constraints in ASP.NET MVC****

This can be very easily achieved using Attribute Route Constraints in the ASP.NET MVC application. To specify attribute route constraint, the syntax is “**{parameter:constraint}**“. With these constraints in place, if the parameter segment in the URL is an integer, then **GetStudentDetails(int studentId)** action method with integer parameter is invoked, and if it is a string then **GetStudentDetails(string studentName)** action method with string parameter is invoked.

Let’s modify the Student Controller to use Attribute Route Constraints as shown below to achieve the above requirements.

**namespace** *AttributeRoutingDemoInMVC.Controllers*

**{**

**[**RoutePrefix**(**"students"**)]**

**public** **class** StudentsController : Controller

**{**

**static** List**<**Student**>** students = new List**<**Student**>()**

**{**

new Student**()** **{** Id = 1, Name = "Pranaya" **}**,

new Student**()** **{** Id = 2, Name = "Priyanka" **}**,

new Student**()** **{** Id = 3, Name = "Anurag" **}**,

new Student**()** **{** Id = 4, Name = "Sambit" **}**

**}**;

**[**HttpGet**]**

**[**Route**(**"{studentID:int}"**)]**

**public** ActionResult GetStudentDetails**(int** studentID**)**

**{**

Student studentDetails = students.FirstOrDefault**(**s =**>** s.Id == studentID**)**;

**return** View**(**studentDetails**)**;

**}**

**[**HttpGet**]**

**[**Route**(**"{studentName:alpha}"**)]**

**public** ActionResult GetStudentDetails**(**string studentName**)**

**{**

Student studentDetails = students.FirstOrDefault**(**s =**>** s.Name == studentName**)**;

**return** View**(**studentDetails**)**;

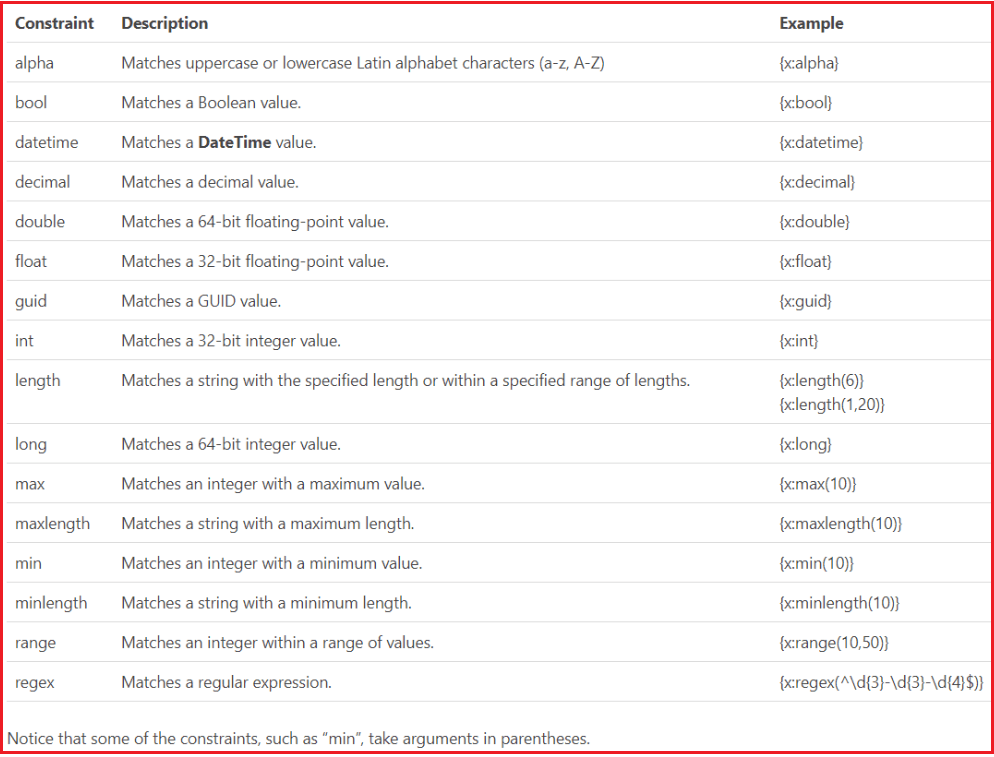
**}**

**}**

**}**

Now build the solution, and navigate to the following two URIs and see everything is working as expected.  
**http://localhost:58316/students/1**  
**http://localhost:58316/students/Pranaya**

Please note that “alpha” stands for uppercase or lowercase alphabet characters. Along with int and alpha, we also have constraints like decimal, double, float, long, bool, etc. as shown in the below image.



##### ****Example:****

If you want GetStudentDetails(int studentId) action method to be mapped to URI **/students/{studentId}**, only if studentId is a number greater than ZERO, then use the “min” constraint as shown below.

**[**Route**(**"{studentID:int:min(1)}"**)]**

**public** ActionResult GetStudentDetails**(int** studentID**)**

**{**

Student studentDetails = students.FirstOrDefault**(**s =**>** s.Id == studentID**)**;

**return** View**(**studentDetails**)**;

**}**

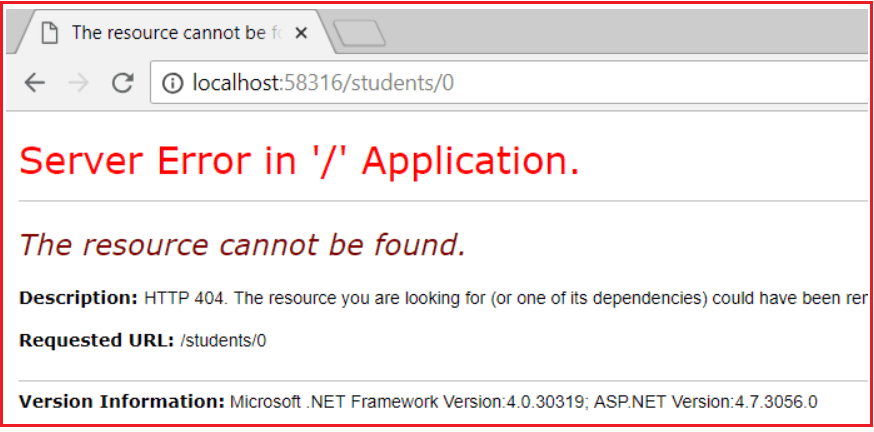
With the above change, if you specify a positive number like 1 in the URI, then it will be mapped to GetStudentDetails(int studentID) action method as expected

###### **/students/1**

However, if you specify 0 or a negative number, you will get an error. For example, if you specify 0 as the value for studentID in the URI,

**http://localhost:58316/students/0**

**You will get the below error**



Along with the “min” constraint, you can also specify the “max” constraint as shown below. For example, if you want the studentID value in the URI to be between 1 and 3 inclusive, then you can specify both “min” and “max” constraints as shown below.

**[**HttpGet**]**

**[**Route**(**"{studentID:int:min(1):max(3)}"**)]**

**public** ActionResult GetStudentDetails**(int** studentID**)**

**{**

Student studentDetails = students.FirstOrDefault**(**s =**>** s.Id == studentID**)**;

**return** View**(**studentDetails**)**;

**}**

The above example can also be achieved using just the “range” attribute as shown below

**[**HttpGet**]**

**[**Route**(**"{studentID:int:range(1,3)}"**)]**

**public** ActionResult GetStudentDetails**(int** studentID**)**

**{**

Student studentDetails = students.FirstOrDefault**(**s =**>** s.Id == studentID**)**;

**return** View**(**studentDetails**)**;

**}**

**Default Route and Route Name in Attribute Routing**

**Default Route in ASP.NET MVC Attribute Routing:**

You can also apply the Route attribute on top of the controller (i.e. at the controller level), to capture the default action method as the parameter. That route would then be applied to all actions in the controller; unless a specific Route has been defined on a specific action, overriding the default setting on the controller. Let’s understand this with an example.

**namespace** *AttributeRoutingDemoInMVC.Controllers*

**{**

**[**RoutePrefix**(**"MyPage"**)]**

**[**Route**(**"action = Index"**)]**

**public** **class** HomeController : Controller

**{**

// URL: /MyPage/

**public** ActionResult Index**()**

**{**

ViewBag.Message = "Index Page";

**return** View**()**;

**}**

// URL: /MyPage/Contact

**public** ActionResult Contact**()**

**{**

ViewBag.Message = "Contact page";

**return** View**()**;

**}**

// URL: /MyPage/About

**public** ActionResult About**()**

**{**

ViewBag.Message = "About";

**return** View**()**;

**}**

**}**

**}**

**Route Names in Attribute Routing**

You can specify a name for a route, in order to easily allow URI generation for it. For example, for the following route:

**namespace** *AttributeRoutingDemoInMVC.Controllers*

**{**

**[**Route**(**"menu", Name = "mymenu"**)]**

**public** **class** MenuController : Controller

**{**

**public** ActionResult MainMenu**()**

**{**

ViewBag.Message = "Menu Page";

**return** View**()**;

**}**

**}**

**}**

You could generate a link using **Url.RouteUrl**:

**<a href=”@Url.RouteUrl(“mymenu”)“>Main menu</a>**