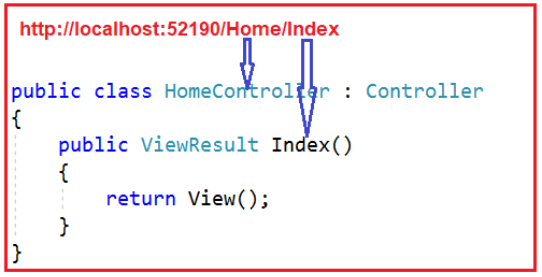
**What is Routing in ASP.NET MVC?**

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:

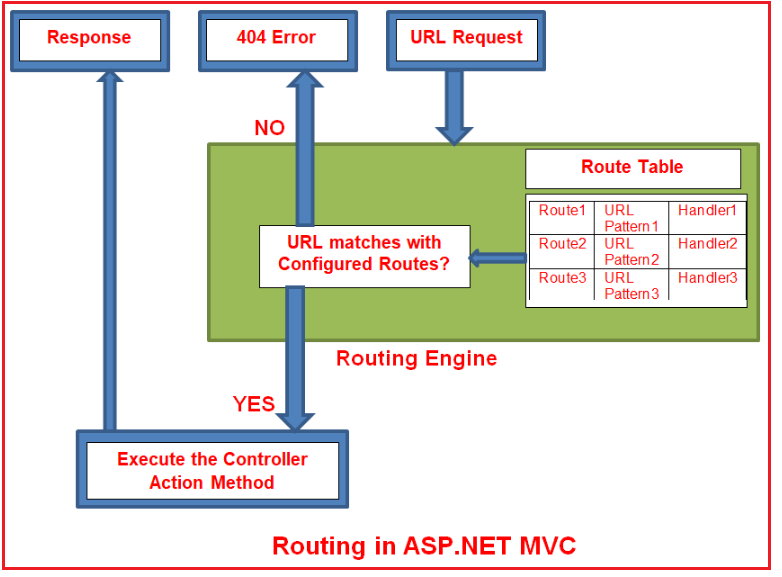
1. **Convention Based Routing**
2. **Attribute-Based Routing.**

##### ****What is a Route in 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.

##### ****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 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.

**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**

**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 Application**

**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://dotnettutorials.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://localhost:4221/Home/Index**, **http://localhost:4221/Home/Details**, and **http:// localhost:4221/**, **http localhost:4221/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:// localhost:4221/**, **http:// localhost:4221/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.