

# ASP.NET Core Convention based Routing

[ASP.NET Core](#) Convention based routing is typically used with the MVC controllers ( i.e. Controller with views). It configures set of Endpoints based on conventions of URL Patterns. We configure these conventions in the `UseEndpoints` method in the `Configure` method of [startup class](#).

## Setting up the Routes to Controller action

We create [controller](#) & [action methods](#) in a [MVC App](#) and in Web API App (REST API).

The API Controller's inherit from the [ControllerBase class](#), while MVC Controllers inherit from [Controller class](#). The [Controller class](#) itself inherits from [ControllerBase class](#) and adds functions that are needed to support views.

The API Controllers are expected to return data in serialized to format, While MVC Controllers are expected return views. Apart from that, there is not much difference between them.

There are two different ways by which we can set up routes to them

1. Convention based routing
2. [Attribute routing](#)

## Convention based routing

The Convention based Routing uses the a set of conventions to specify the URL Pattern. The URL Pattern is then used to map URL paths into controller action method.

## Attribute routing

[Attribute routing](#) creates routes based on attributes placed on controller actions.

The two routing systems can co-exist in the same system. Let's first look at Convention-based routing. We will look at the [Attribute Based Routing](#) in another tutorial

## What is a Route

The Route is similar to a roadmap. We use a roadmap to go to our destination. Similarly, the [ASP.NET Core](#) Apps uses the Route to go to the controller action.

The Each Route contains a Name, URL Pattern , Defaults and Constraints. The URL Pattern is compared to the incoming URLs for a match. An example of URL Pattern is `{controller=Home}/{action=Index}/{id?}`

## How to Set up Conventional Routes

The Convention based Routes are configured in the `Configure` method of the Startup class. We configure it using the `MapControllerRoute` method inside the call to `UseEndpoints`.

```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    //Middlewares

    app.UseRouting();

    //Middlewares

    app.UseEndpoints(endpoints =>
    {
        endpoints.MapControllerRoute(
            name: "default",
            pattern: "{controller=Home}/{action=Index}/{id?}");
    });
}
```

### **MapControllerRoute**

In the above example, the `MapControllerRoute` creates a single route, which is named as `default` and the URL Pattern of the route is `{controller=Home}/{action=Index}/{id?}`

## MapControllerRoute API

```
app.UseEndpoints(endpoints =>
{
    endpoints.MapControllerRoute(
        name: "default",
        pattern: "{controller=Home}/{action=Index}/{id?}");
});
```

Name of the Route

URL Pattern

MapControllerRoute is an [extension method](#) on the EndpointRouteBuilder and accepts the following argument.

name: The name of the route

pattern: The URL pattern of the route

defaults: An object that contains default values for route parameters. The object's properties represent the names and values of the default values

constraints: An object that contains constraints for the route. The object's properties represent the names and values of the constraints.

dataTokens: An object that contains data tokens for the route. The object's properties represent the names and values of the data tokens.

### URL Patterns

Each route must contain a URL Pattern. This Pattern is compared to an incoming URL. If the pattern matches the URL, then it is used by the routing system to process that URL.

Each URL Pattern consists of one or more **Segments**. Forward slash delimits one Segment from another segment.

Each segment can be either a Constant (literal) or Route Parameter.

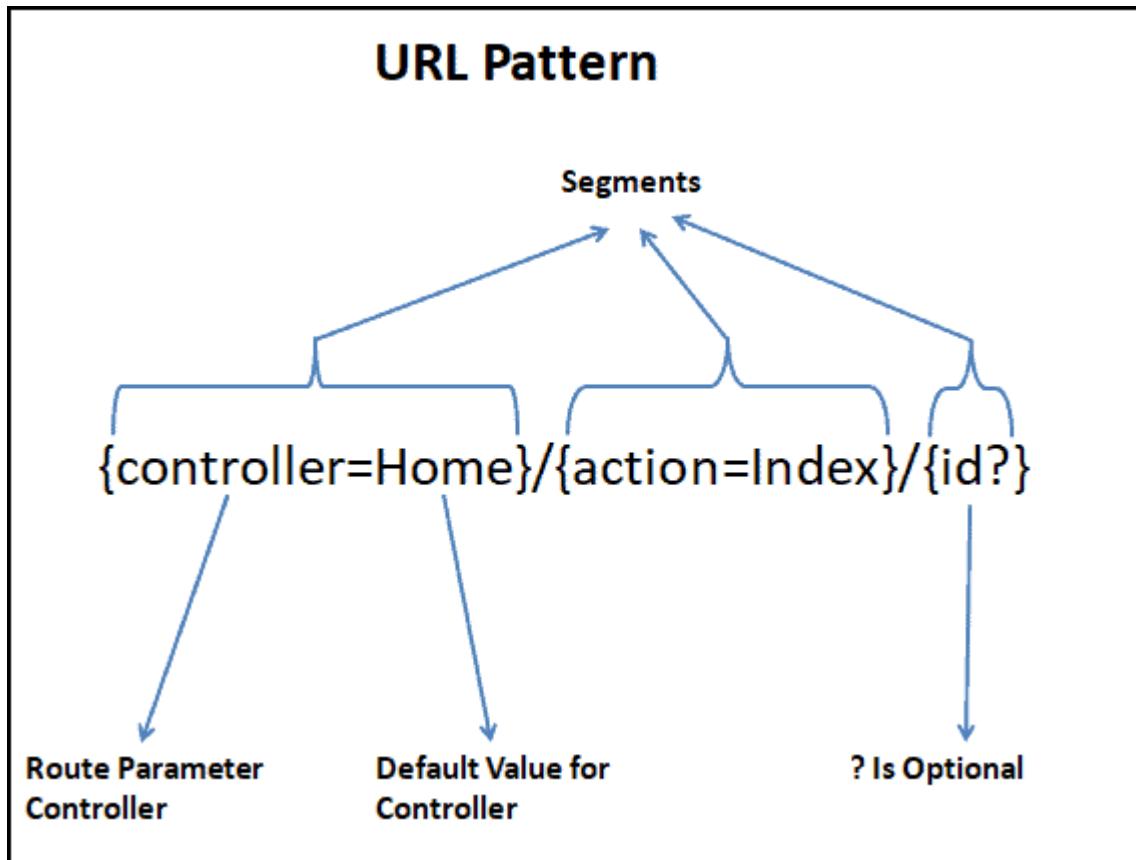
The Route Parameters are wrapped in curly braces for example {controller}, {action}.

The Route Parameters can have default value like {controller=Home}, where **Home** is the default value for the controller. An equals = sign followed by a value after the Route Parameter name defines a default value for the parameter.

You can also have the Constant segments as shown in this route. admin/{controller=Home}/{action=Index}/{id?}. Here admin is a Constant and must present in the requested URL.

The ? in {id ?} indicates that it is optional. A question mark ? after the route parameter name defines the parameter as optional

The URL Pattern {controller=Home}/{action=Index}/{id?} Registers route where the first part of the URL is Controller, the second part is the Action method to invoke on the controller. The third parameter is an additional data in the name of id.



#### URL Matching

Each segment in the incoming URL is matched to the corresponding segment in the URL Pattern. The Route {controller=Home}/{action=Index}/{id?} has three segments. The last one is optional.

Consider the Example URL www.example.com/product/list. This URL has two segments. This URL Matches with the above pattern as the third segment in the URL Pattern is Optional

The Routing engine will parse as {controller}= Product & {action}= List

## URL Matching

{controller=Home}/{action=Index}/{id?}

www.example.com/product/list

First Segment is mapped to first segment  
of the pattern

The URL www.example.com/product also matches with the above URL Pattern, although it has only one segment. This is because the action placeholder has a default value of the Index. if there is no corresponding segments are in the URL and segment has a default value in the Pattern, then the default value is chosen by the Routing Engine.

Hence this URL Is parsed as {controller}=Product and {action}=Index

The www.example.com also matches with the above URL Pattern as the first segment controller has the default value of Home. This URL is Parsed as {controller}=Home and {action}=Index

The URL www.example.com/product/list/10 is parsed as {controller}=Home,{action}=Index and {id}=10.

The URL www.example.com/product/list/10/detail fails. That is because this URL has four segments and the URL Pattern has three.

### Routing in Action

Lets us build an example ASP.NET Core app to check how the routing works.

[Create a new ASP.NET Core Web App](#) using framework version **NET 5.0 & VS Community 2019**

**Version 16.9.1.** & select the template **ASP.NET Core Web App (Model-View-Controller)**

Run the app and ensure that it runs without error.

Under the Configure method (in the [startup.cs](#) ), you will find the following code, which registers the default route.

```
app.UseEndpoints(endpoints =>
```

```
{  
    endpoints.MapControllerRoute(  
        name: "default",  
        pattern: "{controller=Home}/{action=Index}/{id?}");  
};
```

Now, remove the routes from the code.

```
app.UseEndpoints(endpoints =>  
{  
});
```

Run this app and try the following URLs. `/`, `/Home`, `/Home/Index` or any other URL

In all the above cases you will receive the “Fails to Find a Route” message. This is because we have not registered any Route.

### Route Parameters

The Route Parameters are enclosed in curly braces. Each segment in the incoming URL is matched to the corresponding segment in the URL Pattern.

Consider the following route where the Controller and action route parameter do not have any default values.

```
app.UseEndpoints(endpoints =>  
{  
    endpoints.MapControllerRoute(  
        name: "default",  
        pattern: "{controller}/{action}");  
};
```

The following table shows the response

URL	Match ?
/	No No defaults specified for Controller and Action
/Home	No No defaults specified for Action
/Home/Index	Yes
/home/index/10	No There are three segments in the URL, while pattern has only two

The **controller & action** are reserved route parameters. Whenever they get value, the Routing modules knows , which controller & action method to invoke.

The following is the list of reserved route parameter names.

- action
- area
- controller
- handler
- page

### Route Defaults

We can provide a default values for route parameters. Route Defaults can be specified in two ways.

The following route has a default value for **action** URL Parameter.

```
endpoints.MapControllerRoute(
    name: "default",
    pattern: "{controller}/{action=index}");
```

Another way is to use the **defaults** argument. Here, we create an instance of an anonymous type, which contains properties that represents the URL Parameter. Values of those properties become the default values of the URL Parameter.

```
endpoints.MapControllerRoute(  
    name: "default",  
    pattern: "{controller}/{action}",  
    defaults: new { action = "Index" });
```

URL	Match ?	Pattern
/	No	{controller}/{action}
/Home	Yes	Controller=Home
		Action parameter uses the default value index
/Home/Index	Yes	Controller=Home, Action=Index
/home/index/10	No	URL Pattern has only two segments

Default value for both Controller & action

```
endpoints.MapControllerRoute(  
    name: "default",  
    pattern: "{controller=home}/{action=index}");
```

Or

```
endpoints.MapControllerRoute(  
    name: "default",  
    pattern: "{controller}/{action}",  
    defaults: new { controller=home, action = "Index" });
```

URL	Match ?	Parsed As
/	Yes	Controller=Home Action=Index
/Home	Yes	Controller=Home Action=Index
/Home/Index	Yes	Controller=Home Action=Index

The following route has **admin** as Route Parameter.

```
endpoints.MapControllerRoute(
    name: "default",
    pattern: "{admin}/{controller=home}/{action=index}");
```

Now, test it with the following URLs.

URL	Match ?
/	No No defaults for admin. Hence first segment is mandatory
/Home	Yes The First segment Home matches to the Admin
/Abc	Yes
/Home/Index	No Admin=Home Controller=Index There is No IndexController, Hence it fails

URL	Match ?
/Xyz/Home	Yes
/Admin/Home	Yes

### Constants

We can also use a Constant (literal) in a segment. The Constant is without curly braces. The Constant must be part of the URL always.

The admin is Constant in the following Route. This means the first segment of the URL must always contain the word admin.

```
endpoints.MapControllerRoute(
    name: "default",
    pattern: "admin/{controller=home}/{action=index}");
```

URL	Match ?	Pattern
/	No because First segment is mandatory	/
/Home	No The first segment must contain the word Admin	/Admin/{controller=home}/{action=index}
/Abc	No The first segment must contain the word Admin	/Admin/{controller=home}/{action=index}
/Admin	Yes	/Admin/{controller=home}/{action=index}
/Admin/Home	Yes	/Admin/{controller=home}/{action=index}

### Optional Route Parameters

Now, Consider the following Route. This is the default Route that we get, when we [create a new MVC Web app](#)

```
endpoints.MapControllerRoute(  
    name: "default",  
    pattern: "{controller}/{action}/{id?}");
```

It has the third segment named id. It is a Route parameter as it enclosed in curly braces. But it also has ?, which makes it optional.

The id is passed as the parameter to the Controller Action method.

Open the HomeController and change the Index action method

```
public string Index(string id)  
{  
    if (id != null)  
    {  
        return "Received " + id.ToString();  
    }  
    else  
    {  
        return "Received nothing";  
    }  
}
```

A request for /Home/Index/10 will match the above route and the value 10 is passed as id parameter to the Index action.

### Multiple Routes

In the examples above, we used only one route

You can configure the ASP.NET Core to handle any no of conventional routes as shown below

```
endpoints.MapControllerRoute(
```

```
name: "test",
pattern: "testme",
defaults: new { controller = "test", Action="index" });
```

```
endpoints.MapControllerRoute(
```

```
    name: "admin",
    pattern: "admin/{action=Index}/{id?}",
    defaults: new { controller = "admin" });
```

```
endpoints.MapControllerRoute(
```

```
    name: "default",
    pattern: "{controller=Home}/{action=Index}/{id?}");
```

The example above has three routes. Each route must have a Unique Name. We have named it as test, admin and default.

The test route is interesting. It has only one segment. We have setup defaults values for controller and action method on this route. The Controller & Action defaults to Index method of the TestController.

Add the an empty TestController and change the Index method as shown below

```
public class TestController : Controller
{
    public string Index()
    {
        return "Hello Testing done";
    }
}
```

The URL /testme matches this route.

Similarly, create an empty AdminController and copy the following code

```
public class AdminController : Controller
```

```
{
    public string Index()
    {
        return "Hi this is admin controller";
    }
}
```

All the URL that begin with admin, arrive at this Route. It will always use the Controller AdminController

#### **Order matters. First Route wins**

Order in which routes are registered is very important.

Conventional routes are matched in the order in which they are configured.

they appear URL Matching starts at the top and enumerates the collection of Routes searching for a match. It stops when it finds the first match.

Now consider this route

```
endpoints.MapControllerRoute(
    name: "home",
    pattern: "{home}  ",
    defaults: new { Controller = "Home", Action = "Index" });
```

```
endpoints.MapControllerRoute(
    name: "admin",
    pattern: "admin",
    defaults: new { Controller = "Admin", Action = "Index" });
```

The first route has URL Parameter {home} and matches everything. The second route has constant admin, which is a more specific route.

When you use the URL /admin, it does not invoke the AdminController, but instead the HomeController. This is because the first /admin also matches the first route.

To make this route work, move the secure route ahead of the Home Route

```
endpoints.MapControllerRoute(  
    name: "admin",  
    pattern: "admin",  
    defaults: new { Controller = "Admin", Action = "Index" });
```

```
endpoints.MapControllerRoute(  
    name: "home",  
    pattern: "{home} ",  
    defaults: new { Controller = "Home", Action = "Index" });
```