

# MVC :: An Introduction to URL Routing

In this tutorial, you are introduced to an important feature of every ASP.NET MVC application called *URL Routing*. The URL Routing module is responsible for mapping incoming browser requests to particular MVC controller actions.

In the first part of this tutorial, you learn how the standard route table maps requests to controller actions. In the second part of this tutorial, you learn how to modify the default route table with a custom route.

## Using the Default Route Table

When you create a new ASP.NET MVC application, the application is already configured to use URL Routing. URL Routing is setup in two places.

First, URL Routing is enabled in your application's Web configuration file (`Web.config` file). There are four sections in the configuration file that are relevant to routing: the `system.web.httpModules` section, the `system.web.httpHandlers` section, the `system.webserver.modules` section, and the `system.webserver.handlers` section. Be careful not to delete these sections because without these sections routing will no longer work.

Second, and more importantly, a route table is created in the application's `Global.asax` file. The `Global.asax` file is a special file that contains event handlers for ASP.NET application lifecycle events. The route table is created during the Application Start event.

The file in Listing 1 contains the default `Global.asax` file for an ASP.NET MVC application.

### Listing 1 – `Global.asax.vb`

```
Public Class GlobalApplication
    Inherits System.Web.HttpApplication

    Shared Sub RegisterRoutes(ByVal routes As RouteCollection)
        routes.IgnoreRoute("{resource}.axd/{*pathInfo}")

        ' MapRoute takes the following parameters, in order:
        ' (1) Route name
        ' (2) URL with parameters
        ' (3) Parameter defaults
        routes.MapRoute( _
            "Default", _
            "{controller}/{action}/{id}", _
            New With {.controller = "Home", .action = "Index", .id =
                ""} _
```

```

    )

End Sub

Sub Application_Start()
    RegisterRoutes(RouteTable.Routes)
End Sub

End Class

```

When an MVC application first starts, the `Application_Start()` subroutine is called. This subroutine, in turn, calls the `RegisterRoutes()` subroutine. The `RegisterRoutes()` subroutine creates the route table.

The default route table contains a single route (named `Default`). The `Default` route maps the first segment of a URL to a controller name, the second segment of a URL to a controller action, and the third segment to a parameter named **id**.

Imagine that you enter the following URL into your web browser's address bar:

```
/Home/Index/3
```

The `Default` route maps this URL to the following parameters:

```
controller = Home
```

```
action = Index
```

```
id = 3
```

When you request the URL `/Home/Index/3`, the following code is executed:

```
HomeController.Index(3)
```

The `Default` route includes defaults for all three parameters. If you don't supply a controller, then the controller parameter defaults to the value **Home**. If you don't supply an action, the action parameter defaults to the value **Index**. Finally, if you don't supply an id, the id parameter defaults to an empty string.

Let's look at a few examples of how the `Default` route maps URLs to controller actions. Imagine that you enter the following URL into your browser address bar:

```
/Home
```

Because of the `Default` route parameter defaults, entering this URL will cause the `Index()` method of the `HomeController` class in Listing 2 to be called.

### Listing 2 – `HomeController.vb`

```

<HandleError()> _
Public Class HomeController
    Inherits System.Web.Mvc.Controller

    Function Index(ByVal Id As String)

```

```
        Return View()  
    End Function  
  
End Class
```

In Listing 2, the `HomeController` class includes a method named `Index()` that accepts a single parameter named `Id`. The URL `/Home` causes the `Index()` method to be called with an empty string as the value of the `Id` parameter.

Because of the way that the MVC framework invokes controller actions, the URL `/Home` also matches the `Index()` method of the `HomeController` class in Listing 3.

### Listing 3 – `HomeController.vb` (Index action with no parameter)

```
<HandleError()> _  
Public Class HomeController  
    Inherits System.Web.Mvc.Controller  
  
    Function Index()  
        Return View()  
    End Function  
  
End Class
```

The `Index()` method in Listing 3 does not accept any parameters. The URL `/Home` will cause this `Index()` method to be called. The URL `/Home/Index/3` also invokes this method (the `Id` is ignored).

The URL `/Home` also matches the `Index()` method of the `HomeController` class in Listing 4.

### Listing 4 – `HomeController.vb` (Index action with nullable parameter)

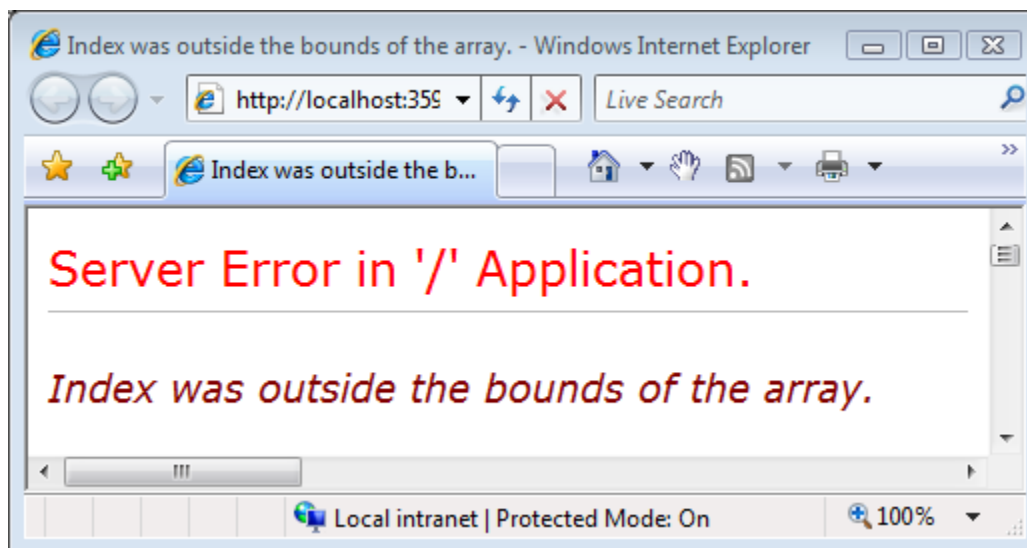
```
<HandleError()> _  
Public Class HomeController  
    Inherits System.Web.Mvc.Controller  
  
    Function Index(ByVal Id? As Integer)  
        Return View()  
    End Function  
  
End Class
```

In Listing 4, the `Index()` method has one Integer parameter. Because the parameter is a nullable parameter (can have the value `Nothing`), the `Index()` can be called without raising an error.

Finally, invoking the `Index()` method in Listing 5 with the URL `/Home` causes an exception since the `Id` parameter *is not* a nullable parameter. If you attempt to invoke the `Index()` method then you get the error page in Figure 1.

#### Listing 5 – HomeController.vb (Index action with Id parameter)

```
<HandleError()> _  
Public Class HomeController  
    Inherits System.Web.Mvc.Controller  
  
    Function Index(ByVal Id As Integer)  
        Return View()  
    End Function  
  
End Class
```



**Figure 1 – Invoking a controller action that expects a parameter value**

The URL `/Home/Index/3`, on the other hand, works just fine with the `Index` controller action in Listing 5. The request `/Home/Index/3` causes the `Index()` method to be called with an `Id` parameter that has the value 3.

## Creating a Custom Route

For many simple ASP.NET MVC applications, the default route table will work just fine. However, you might discover that you have specialized routing needs. In that case, you can create a custom route.

Imagine, for example, that you are building a blog application. You might want to handle incoming requests that look like this:

```
/Archive/12-25-2009
```

When a user enters this request, you want to return the blog entry that corresponds to the date 12/25/2009. In order to handle this type of request, you need to create a custom route.

The `Global.asax` file in Listing 6 contains a new custom route, named `Blog`, which handles requests that look like `/Archive/entry date`.

### Listing 6 – `Global.asax` (with custom route)

```
Public Class GlobalApplication
    Inherits System.Web.HttpApplication

    Shared Sub RegisterRoutes(ByVal routes As RouteCollection)
        routes.IgnoreRoute("{resource}.axd/{*pathInfo}")

        routes.MapRoute( _
            "Blog", _
            "Archive/{entryDate}", _
            New With {.controller = "Archive", .action = "Entry"} _
        )

        ' MapRoute takes the following parameters, in order:
        ' (1) Route name
        ' (2) URL with parameters
        ' (3) Parameter defaults
        routes.MapRoute( _
            "Default", _
            "{controller}/{action}/{id}", _
            New With {.controller = "Home", .action = "Index", .id =
""} _
        )
    End Sub
End Class
```

```

End Sub

Sub Application_Start()
    RegisterRoutes(RouteTable.Routes)
End Sub
End Class

```

The order of the routes that you add to the route table is important. Our new custom `Blog` route is added before the existing `Default` route. If you reversed the order, then the `Default` route always will get called instead of the custom route.

The custom `Blog` route matches any request that starts with `/Archive/`. So, it matches all of the following URLs:

```

/Archive/12-25-2009
/Archive/10-6-2004
/Archive/apple

```

The custom route maps the incoming request to a controller named `Archive` and invokes the `Entry()` action. When the `Entry()` method is called, the entry date is passed as a parameter named `entryDate`.

You can use the `Blog` custom route with the controller in Listing 7.

**Listing 7** – `ArchiveController.vb`

```

Public Class ArchiveController
    Inherits System.Web.Mvc.Controller

    Function Index()
        ' Add action logic here
        Throw New NotImplementedException()
    End Function

    Function Entry(ByVal entryDate As DateTime)
        Return "You requested the entry on " & entryDate.ToString()
    End Function
End Class

```

Notice that the `Entry()` method in Listing 7 accepts a parameter of type `DateTime`. The MVC framework is smart enough to convert the entry date from the URL into a `DateTime` value automatically. If the entry date from the URL cannot be converted to a `DateTime`, an error is raised.

## Summary

The goal of this tutorial was to provide you with a brief introduction to `URL Routing`. First, we examined the default route table that you get with a new ASP.NET MVC application. You learned how the default route maps URLs to controller actions.

Next, you learned how to create a custom route. You learned how to add a custom route to the route table in the `Global.asax` file that represents blog entries. We discussed how to map requests for blog entries to a controller named `ArchiveController` and a controller action named `Entry()`.