

MVC Routes

MVC3 gives you great control over how URLs are mapped to your controllers. It gives you the ability to define your URLs in a human readable SEO (Search Engine Optimization) friendly fashion, to remap old URLs to new functionality and side-by-side utilize classic **ASP.NET** sites inside of MVC3. It also results in hiding what kind of page user is calling and what environment we are working in. Most of the new websites are following this and it is important to understand that routing is not URL rewriting as routing will have customizations and many attachments towards request/response.

When we create any type of MVC application by default GLOBAL.ASAX file is created becoz ASP.NET implements MVC using this global application class mainly. Routes defined in the Global.asax.cs file of our MVC3 web application/site. In this global.asax file most important element relative to our work is RegisterRoutes method. By default, there is only one route defined in the RegisterRoutes method that looks like the line below.

```
routes.MapRoute(
    "Default", // Route name
    "{controller}/{action}/{id}", // URL with parameters
    new { controller = "Home", action = "Index", id = UrlParameter.Optional } // Parameter defaults
);
```

This route defines the route name, which can be anything as long as it is unique, the URL template, and the parameter defaults. The default route that is pre-defined for you maps to Controller/Action/id. You can add additional routes by copying the same line and adjusting the URL parameters and the related default values. Remember when we add additional routes the order is important.

Custom MVC3 Routes

One of the many factors frequently considered by search engines to determine the relevance of a particular page to a particular search term is whether or not the URL link itself includes a particular term. In a classic **ASP.NET** site for a magazine, you might have a URL that looks like www.internet.com/ViewArticle.aspx?id=123. This URL passes the ID number of the article to view, but the URL itself doesn't describe the content in any human readable way. If the URL instead was www.internet.com/MVC3_Routing/123 a human-or a web crawler-could read that and know that the article is about **MVC3 Routing**.

Another frequent use of custom routing is to allow multiple sites to link to the same location while providing additional data about where they came from. For example, if you had a link to a product page and you wanted to provide custom co-branding based on which one of your partners linked in to a page, you could do so by including a "partner name" variable in the link. A link like this could be to www.internet.com/PartnerA/Article/123 or www.internet.com/PartnerB/Article/123 for example. This would allow two different sites to link to the same article while providing their own information in the process.

The code block below defines a new route called ArticleRoute that defines a new parameter called article in addition to the standard controller and action.

```
routes.MapRoute(  
    "ArticleRoute",  
    "{article}/{controller}/{action}",  
    new { article="Unknown", controller = "Home", action = "Index" } );
```

You can access this custom article part of the route in your controller by accessing the RouteData object.

```
RouteData.Values["article"]
```

To access the RouteData object in your **Razor** views, use the **@Url.RequestContext.RouteData** object.

When you are constructing URLs that use string data such as article titles or author's names, you will need to use some form of URL-friendly encoding. The easiest method is to use `HttpUtility.UrlEncode("Your String")` which will replace all of the URL unfriendly characters with the appropriate HTML escapes. This method is web spider friendly but not necessarily human readable URL friendly.

It is very important to remember that any data in a URL is easily user manipulated and shouldn't be trusted. It is not an appropriate place to pass application variables between pages unless they are of a nature that it would be acceptable if the user manipulated them.

Re-mapping Routes

From time to time it is necessary to re-route an old URL to a new location. Traditionally you would use a redirect page to let users know to update their bookmarks and include a link to the new location. Sometimes it is impractical or even impossible to change URLs as is the case when you have 3rd party software systems set up to access and scrape particular URLs.

If you wanted to route MyRazor.cshtml page to your **MVC3** Home/Index controller, you can do so by defining a route like the one below.

```
routes.MapRoute("RT", " MyRazor.cshtml ",  
    new { controller = "OtherController", action = "Packed" });
```

Including Classic ASP.NET as a Sub-directory in an MVC3 Web Application

If you have a large classic **ASP.NET** web site that you need to incrementally transition to **MVC3** you can put the entire classic **ASP.NET** web site in your **MVC3** web site as a sub-directory. You can then call the `IgnoreRoute` method seen in the code block below to tell MVC3 to not handle that particular URL path.

```
routes.IgnoreRoute("OldClassicASP/");
```

Constraints

You can define additional constraints to your defined routes to insure that the values passed for particular parts of your route are valid. This is useful not only for general security related needs but is also useful for scenarios where you might want to have additional routing logic in place for particular routes.

```
routes.MapRoute(  
    "ColorPath", // Route name  
    "{color}/{controller}/{action}/{id}", // URL with parameters  
    new { controller = "Home", action = "Index", id = UrlParameter.Optional } // Parameter defaults  
    , new { color="blue" }  
);
```

In the example route above, a new anonymous object is added to the MapRoute path call that adds a constraint to this path. The route will only be used if the color is blue.

Namespaces

In large ASP.NET MVC3 applications you can potentially have hundreds of controllers. This can become problematic because the **.NET Framework** looks in the Controllers folder and all sub-folders looking for controllers to match up with the defined routes.

To help you organize your code, you can add namespaces to your routes to constrain the controllers the route will match to particular namespaces. The route below will only use controllers defined in the Home.Guru namespace.

```
routes.MapRoute("NamespacedRoute", "Cool/{controller}/{action}",  
    new { controller = "Home", action = "Index", id = UrlParameter.Optional } , null,  
    , new string[] { "Home.Guru" });
```

Global Filters

If you want to define code that runs before or after your routing calls, you can define a global filter. Global filters are registered in the Global.asax.cs file by adding a line to the RegisterGlobalFilters method. This method, by default, registers the HandleErrorAttribute that is used to handle error conditions in ASP.NET MVC3 applications.

If you wanted to add a copyright notice at the bottom of all of your pages, you could add a global filter attribute that overrides the OnResultExecuted method which is run after your page has run.

```
public class CopyrightNoticeAttribute : ActionFilterAttribute  
{  
    public override void OnResultExecuted(ResultExecutedContext filterContext)
```

```
{
    filterContext.HttpContext.Response.Write(String.Format("<h1>Copyright
{0}</h1>",DateTime.Now.Year));
}
}
```

Once your custom attribute has been written you can add it to the global.asax.cs file.

```
public static void RegisterGlobalFilters(GlobalFilterCollection filters)
{
    filters.Add(new CopyrightNoticeAttribute());
    filters.Add(new HandleErrorAttribute());//Default in MVC3
}
```

-----*-----*-----*-----*-----