Using Friendly URLs in Web Forms

Friendly URIs help you eliminate query string parameters and file extensions from the URL line. So, instead of…

```
www.url.com/Products.aspx
www.url.com/Products.aspx?ProductId=3
```

You use simple friendly URLs instead as shown below:

```
www.url.com/Products
www.url.com/Products/3
```

There are many benefits of using friendly URLs in your web applications.

- Cleaner query string
- User does not know the actual page name
- Easier for users to use

Friendly URLs are available in Web Forms and MVC. I see a lot of examples of using friendly URLs using MVC, but very few using Web Forms. So, I thought I would discuss how to use them in Web Forms. Actually, the process is almost identical.

First you need to download the **Microsoft.Asp.Net.FriendlyUrls.Core.dll** if you don't already have it in your project. If you have an older ASP.NET application you probably don't have it. If you are starting a new project in Visual Studio 2013, and choose the Web Forms template, this DLL is already present.

If you want to use friendly URLs in an older project, select **Tools** | **Nuget Package Manager** | **Manage NuGet Packages for Solution...** from the Visual Studio menu. Search online for Microsoft.AspNet.FriendlyUrls and install the Microsoft.AspNet.FriendlyUrls.Core. You don't need any of the other DLLs in the NuGet packages list, just the "Core" DLL.

If you have an App_Start folder, check and see if you have a class called RouteConfig.cs in there. If so, then you already have what you need. If you don't, then add a class called **RouteConfig** to your project.

Add the following ***using*** statements at the top of this new class file.

```
using System.Web.Routing;
using Microsoft.AspNet.FriendlyUrls;
```

Either add the following method, or modify it to look like the following. This assumes you have 3 pages in your project; Default.aspx, Customers.aspx, and Products.aspx. Feel free to substitute your page names as appropriate.

```
public static void RegisterRoutes(RouteCollection routes)
{
  routes.EnableFriendlyUrls();

  routes.MapPageRoute("", "Default", "~/Default.aspx");

  routes.MapPageRoute("", "Customers", "~/Customers.aspx");
  routes.MapPageRoute("GetCustomer",
                      "GetCustomer/{CustomerId}",
                      "~/Customers.aspx");

  routes.MapPageRoute("", "Products", "~/Products.aspx");
  routes.MapPageRoute("GetProduct", "GetProduct/{ProductId}",
                       "~/Products.aspx");
}
```

In the above page routes you have some parameter placeholders denoted by the curly braces {}. These are what you use to pass any parameters to and the names you use to retrieve those values.

The next step is to open your Global.asax and either add, or check to see, if you have the following ***using*** statement at the top of the file.

```
using System.Web.Routing;
```

In the Application_Start() event you now need to call the RegisterRoutes method you created in the last step. The **RouteTable.Routes** object, created by the ASP.NET engine, is what you add to in your RegisterRoutes method.

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

You can now run your ASP.NET application type in any of the following:

```
http://localhost:xxxx/Products
http://localhost:xxxx/GetProduct/22
http://localhost:xxxx/Customers
http://localhost:xxxx/GetCustomer/ABC
```

From any <a> tag on your web pages you can now use the following syntax:

```
<a href="Products">Get All Products</a>
<a href="GetProduct/22">Get Product #22</a>
```

Notice that you don't need the ".aspx" extension. If you are using the Response object to redirect from code behind, may also use the same shorthand for any route that does not have a parameter.

```
Response.Redirect("Products");
Response.Redirect("Customers");
```

Passing Parameters using the Response Object

If you are going to pass either a customer id or a product id to your pages, and you want to use the Response object, you need to setup things a little differently. Remember you created the following map page route in the RouteConfig class.

```
routes.MapPageRoute("GetProduct", "GetProduct/{ProductId}",
                    "~/Samples/ProductView.aspx");
```

To redirect to this page, you use the RedirectToRoute method of the Response object.

```
Response.RedirectToRoute("GetProduct",
 new {ProductId = 33});
```

The first parameter you pass to the RedirectToRoute method must match the first parameter in the MapPageRoute. The second parameter is an object with the name in the braces {ProductId} set to the value you wish to pass (in the above case 33).

Retrieve the Passed Parameters

To retrieve the value passed you use the Page.RouteData.Values property. Pass in the name of the parameter you are looking for, in this case "ProductId" and it will return either a null if not found, or the value. You typically retrieve these values from the Page_Load event procedure.

```
if (Page.RouteData.Values["ProductId"] != null)
{
  int ProductId =
     Convert.ToInt32(Page.RouteData.Values["ProductId"]);
}
```

Summary

Using friendly URLs is quite easy to accomplish in either Web Forms or MVC. You can download the friendly URLs “Core” DLL from NuGet to add to any project. Then with just a few lines of code you can start calling your pages in a very user-friendly manner.