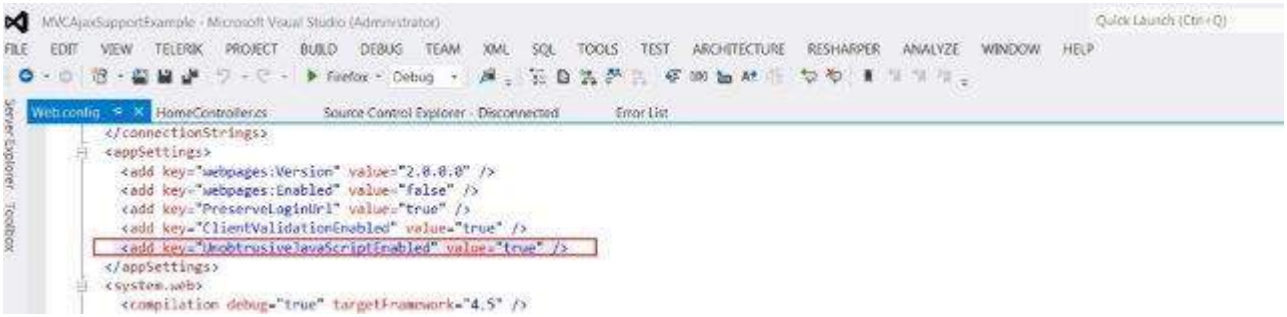
MVC FRAMEWORK - AJAX SUPPORT

Introduction

As you might be knowing, Ajax is a shorthand for Asynchronous JavaScript and XML. The MVC Framework contains built-in support for unobtrusive Ajax by which you can use the helper methods to define your Ajax features without adding code throughout all the views. This feature in MVC is based on the jQuery features.

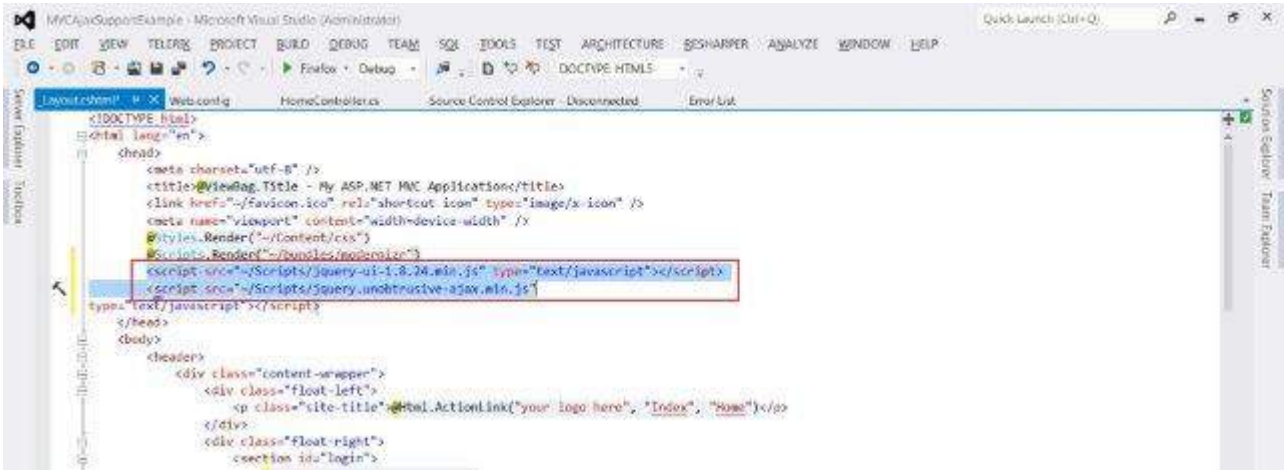
To enable the unobtrusive AJAX support in the MVC application, open the Web.Config file and set the UnobtrusiveJavaScriptEnabled property inside the appSettings section using the following code. If the key is already present in your application, you can ignore this step.

```
<add key="UnobtrusiveJavaScriptEnabled" value="true" />
```



After this, open the common layout file _Layout.cshtml file located under Views/Shared folder. We will add references to the jQuery libraries here using the following code:

```
<script src="~/Scripts/jquery-ui-1.8.24.min.js" type="text/javascript"></script>
<script src="~/Scripts/jquery.unobtrusive-ajax.min.js" type="text/javascript"></script>
```



Creating an Unobtrusive Ajax Application

In the example that follows, we will create a form which will display the list of users in the system. We will place a dropdown having three options: Admin, Normal and Guest. When you will select one of these values, it will display the list of users belonging to this category using unobtrusive AJAX setup.

Step 1:

Create a Model file Model.cs and copy the following code:

```
using System;

namespace MVCAjaxSupportExample.Models
{
```

```
    public class User
    {
        public int UserId { get; set; }
        public string FirstName { get; set; }
        public string LastName { get; set; }
        public DateTime BirthDate { get; set; }
        public Role Role { get; set; }
    }

    public enum Role
    {
        Admin,
        Normal,
        Guest
    }
}
```

Step 2:

Create a Controller file named UserController.cs and create two action methods inside that as shown below:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web.Mvc;
using MVCAjaxSupportExample.Models;

namespace MVCAjaxSupportExample.Controllers
{
    public class UserController : Controller
    {
        private readonly User[] userData =
        {
            new User {FirstName = "Edy", LastName = "Clooney", Role = Role.Admin},
            new User {FirstName = "David", LastName = "Sanderson", Role = Role.Admin},
            new User {FirstName = "Pandy", LastName = "Griffyth", Role = Role.Normal},
            new User {FirstName = "Joe", LastName = "Gubbins", Role = Role.Normal},
            new User {FirstName = "Mike", LastName = "Smith", Role = Role.Guest}
        };

        public ActionResult Index()
        {
            return View(userData);
        }

        public PartialViewResult GetUserData(string selectedRole = "All")
        {
            IEnumerable data = userData;
            if (selectedRole != "All")
            {
                var selected = (Role) Enum.Parse(typeof (Role), selectedRole);
                data = userData.Where(p => p.Role == selected);
            }
            return PartialView(data);
        }

        public ActionResult GetUser(string selectedRole = "All")
        {
            return View((object) selectedRole);
        }
    }
}
```

Step 3:

Now create a partial View named GetUserData with the following code. This view will be used to render list of users based on the selected role from the dropdown.

```
@model IEnumerable<MVCAjaxSupportExample.Models.User>

<table>
    <tr>
        <th>
            @Html.DisplayNameFor(model => model.FirstName)
        </th>
        <th>
            @Html.DisplayNameFor(model => model.LastName)
        </th>
        <th>
            @Html.DisplayNameFor(model => model.BirthDate)
        </th>
        <th></th>
    </tr>

@foreach (var item in Model) {
    <tr>
        <td>
            @Html.DisplayFor(modelItem => item.FirstName)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.LastName)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.BirthDate)
        </td>
        <td>

        </td>
    </tr>
}

</table>
```

Step 4:

Now create a View GetUser with the following code. This view will asynchronously get the data from previously created controller's GetUserData Action.

```
@using MVCAjaxSupportExample.Models
@model string
@{
ViewBag.Title = "GetUser";
AjaxOptions ajaxOpts = new AjaxOptions {
UpdateTargetId = "tableBody"
};
}
<h2>Get User</h2>
<table>
    <thead><tr><th>First</th><th>Last</th><th>Role</th></tr></thead>
        <tbody >
            @Html.Action("GetUserData", new {selectedRole = Model })
        </tbody>
</table>

@using (Ajax.BeginForm("GetUser", ajaxOpts)) {
    <div>
        @Html.DropDownList("selectedRole", new SelectList(
        new [] {"All"}.Concat(Enum.GetNames(typeof(Role)))))
        <button type="submit">Submit</button>
    </div>
}
```

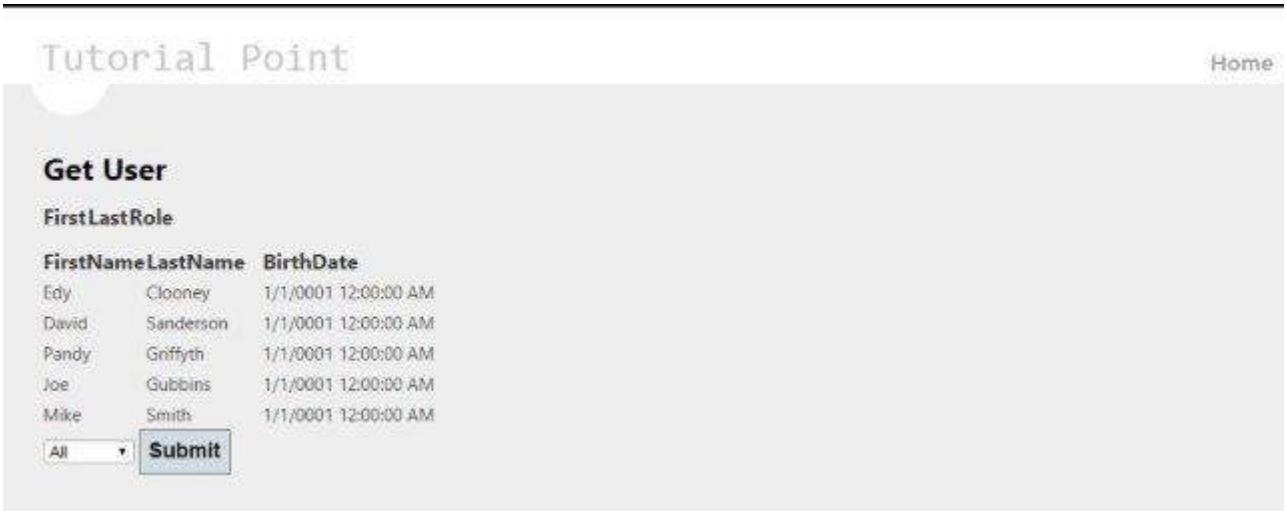
Step 5:

Finally just change the Route.config entries to launch the User Controller.

```
defaults: new { controller = "User", action = "GetUser", id = UrlParameter.Optional }
```

Step 6:

Finally, run the application which will look like the below screenshot:



If you select Admin from the dropdown, it will go and fetch all the users with Admin type. This is happening via AJAX and does not reload the entire page.