Examining the Edit Methods and Edit View

By Rick Anderson | last updated May 22, 2015
732 of 787 people found this helpful

In this section, you'll examine the generated `Edit` action methods and views for the movie controller. But first will take a short diversion to make the release date look better. Open the *Models\Movie.cs* file and add the highlighted lines shown below:

```
using System;
using System.ComponentModel.DataAnnotations;
using System.Data.Entity;

namespace MvcMovie.Models
{
    public class Movie
    {
        public int ID { get; set; }
        public string Title { get; set; }

        [Display(Name = "Release Date")]
        [DataType(DataType.Date)]
        [DisplayFormat(DataFormatString = "{0:yyyy-MM-dd}", ApplyFormatInEditMode =
true)]
        public DateTime ReleaseDate { get; set; }
        public string Genre { get; set; }
        public decimal Price { get; set; }
    }

    public class MovieDBContext : DbContext
    {
        public DbSet<Movie> Movies { get; set; }
    }
}
```

You can also make the date culture specific like this:

```
[Display(Name = "Release Date")]
[DataType(DataType.Date)]
[DisplayFormat(DataFormatString = "{0:d}", ApplyFormatInEditMode = true)]
public DateTime ReleaseDate { get; set; }
```

We'll cover **DataAnnotations (http://msdn.microsoft.com/en-us/library/system.componentmodel.dataannotations.aspx)** in the next tutorial. The **Display (http://msdn.microsoft.com/en-us/library/system.componentmodel.dataannotations.displayattribute.aspx)** attribute specifies what to display for the name of a field (in this case "Release Date" instead of "ReleaseDate"). The

DataType (http://msdn.microsoft.com/en-us/library/system.componentmodel.dataannotations.datatypeattribute.aspx) attribute specifies the type of the data, in this case it's a date, so the time information stored in the field is not displayed. The **DisplayFormat (http://msdn.microsoft.com/en-us/library/system.componentmodel.dataannotations.displayformatattribute.aspx)** attribute is needed for a bug in the Chrome browser that renders date formats incorrectly.

Run the application and browse to the `Movies` controller. Hold the mouse pointer over an **Edit** link to see the URL that it links to.

The **Edit** link was generated by the **`Html.ActionLink (http://msdn.microsoft.com/en-us/library/system.web.mvc.html.linkextensions.actionlink(v=vs.108).aspx)`** method in the *Views\Movies\Index.cshtml* view:

```
@Html.ActionLink("Edit", "Edit", new { id=item.ID })
```

The `Html` object is a helper that's exposed using a property on the **System.Web.Mvc.WebViewPage (http://msdn.microsoft.com/en-us/library/gg402107(VS.98).aspx)** base class. The **`ActionLink (http://msdn.microsoft.com/en-us/library/system.web.mvc.html.linkextensions.actionlink.aspx)`** method of the helper makes it easy to dynamically generate HTML hyperlinks that link to action methods on controllers. The first argument to the `ActionLink` method is the link text to render (for example, `<a>Edit Me</a>`). The second argument is the name of the action method to invoke (In this case, the `Edit` action). The final argument is an **anonymous object (//weblogs.asp.net/scottgu/archive/2007/05/15/new-orcas-language-feature-anonymous-types.aspx)** that generates the route data (in this case, the ID of 4).

The generated link shown in the previous image is *http://localhost:1234/Movies/Edit/4*. The default route (established in *App_Start\RouteConfig.cs*) takes the URL pattern `{controller}/{action}/{id}`. Therefore, ASP.NET translates *http://localhost:1234/Movies/Edit/4* into a request to the `Edit` action method of the `Movies` controller with the parameter `ID` equal to 4. Examine the following code from the *App_Start\RouteConfig.cs* file. The **MapRoute (/mvc/tutorials/older-versions/controllers-and-routing/asp-net-mvc-routing-overview-cs)** method is used to route HTTP requests to the correct controller and action method and supply the optional ID parameter. The **MapRoute (/mvc/tutorials/older-versions/controllers-and-routing/asp-net-mvc-routing-overview-cs)** method is also used by the **HtmlHelpers (http://msdn.microsoft.com/en-us/library/system.web.mvc.htmlhelper(v=vs.108).aspx)** such as **`ActionLink (http://msdn.microsoft.com/en-us/library/system.web.mvc.html.linkextensions.actionlink.aspx)`** to generate URLs given the controller, action method and any route data.

```
public static void RegisterRoutes(RouteCollection routes)
{
    routes.IgnoreRoute("{resource}.axd/{*pathInfo}");

    routes.MapRoute(
        name: "Default",
        url: "{controller}/{action}/{id}",
        defaults: new { controller = "Home", action = "Index",
            id = UrlParameter.Optional }
    );
}
```

You can also pass action method parameters using a query string. For example, the URL *http://localhost:1234/Movies/Edit?ID=3* also passes the parameter `ID` of 3 to the `Edit` action method of the `Movies` controller.

Open the `Movies` controller. The two `Edit` action methods are shown below.

```
// GET: /Movies/Edit/5
public ActionResult Edit(int? id)
{
    if (id == null)
    {
        return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
    }
    Movie movie = db.Movies.Find(id);
    if (movie == null)
    {
        return HttpNotFound();
    }
    return View(movie);
}

// POST: /Movies/Edit/5
// To protect from overposting attacks, please enable the specific properties you want to
bind to, for
// more details see http://go.microsoft.com/fwlink/?LinkId=317598.
[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult Edit([Bind(Include="ID,Title,ReleaseDate,Genre,Price")] Movie movie)
{
    if (ModelState.IsValid)
    {
        db.Entry(movie).State = EntityState.Modified;
        db.SaveChanges();
        return RedirectToAction("Index");
    }
    return View(movie);
}
```

Notice the second `Edit` action method is preceded by the **`HttpPost` (http://msdn.microsoft.com/en-us/library/system.web.mvc.httppostattribute.aspx)** attribute. This attribute specifies that that overload of the `Edit` method can be invoked only for POST requests. You could apply the **`HttpGet` (http://msdn.microsoft.com/en-us/library/system.web.mvc.httpgetattribute.aspx)** attribute to the first edit method, but that's not necessary because it's the default. (We'll refer to action methods that are implicitly assigned the `HttpGet` attribute as `HttpGet` methods.) The **Bind (http://msdn.microsoft.com/en-us/library/system.web.mvc.bindattribute(v=vs.108).aspx)** attribute is another important security mechanism that keeps hackers from over-posting data to your model. You should only include properties in the bind attribute that you want to change. You can read about overposting and the bind attribute in my **overposting security note (http://go.microsoft.com/fwlink/?LinkId=317598)** . In the simple model used in this tutorial, we will be binding all the data in the model. The **ValidateAntiForgeryToken (http://msdn.microsoft.com/en-us/library/system.web.mvc.validateantiforgerytokenattribute(v=vs.108).aspx)** attribute is used to prevent forgery of a request and is paired up with `@Html.AntiForgeryToken` (http://msdn.microsoft.com/en-us/library/system.web.mvc.htmlhelper.antiforgerytoken(v=vs.108).aspx) `()` in the edit view file (*Views\Movies\Edit.cshtml*), a portion is shown below:

```
@model MvcMovie.Models.Movie

@{
    ViewBag.Title = "Edit";
}
<h2>Edit</h2>
@using (Html.BeginForm())
{
    @Html.AntiForgeryToken()
    <div class="form-horizontal">
        <h4>Movie</h4>
        <hr />
        @Html.ValidationSummary(true)
        @Html.HiddenFor(model => model.ID)

        <div class="form-group">
            @Html.LabelFor(model => model.Title, new { @class = "control-label col-md-2" 
})
            <div class="col-md-10">
                @Html.EditorFor(model => model.Title)
                @Html.ValidationMessageFor(model => model.Title)
            </div>
        </div>
```

`@Html.AntiForgeryToken()` generates a hidden form anti-forgery token that must match in the `Edit` method of the `Movies` controller. You can read more about Cross-site request forgery (also known as XSRF or CSRF) in my tutorial **XSRF/CSRF Prevention in MVC (/mvc/overview/security/xsrfcsrf-prevention-in-aspnet-mvc-and-web-pages)** .

The `HttpGet` `Edit` method takes the movie ID parameter, looks up the movie using the Entity Framework **`Find` (http://msdn.microsoft.com/en-us/library/gg696418(v=vs.103).aspx)** method, and returns the selected movie to the Edit view. If a movie cannot be found, **HttpNotFound (http://msdn.microsoft.com/en-us/library/gg453938(VS.98).aspx)** is returned. When the scaffolding system created the Edit view, it examined the `Movie` class and created code to render `<label>` and `<input>` elements for each property of the class. The following example shows the Edit view that was generated by the visual studio scaffolding system:

```
@model MvcMovie.Models.Movie

@{
    ViewBag.Title = "Edit";
}
<h2>Edit</h2>
@using (Html.BeginForm())
{
    @Html.AntiForgeryToken()
    <div class="form-horizontal">
        <h4>Movie</h4>
        <hr />
        @Html.ValidationSummary(true)
        @Html.HiddenFor(model => model.ID)

        <div class="form-group">
            @Html.LabelFor(model => model.Title, new { @class = "control-label col-md-2" 
})
            <div class="col-md-10">
                @Html.EditorFor(model => model.Title)
                @Html.ValidationMessageFor(model => model.Title)
            </div>
        </div>
        <div class="form-group">
            @Html.LabelFor(model => model.ReleaseDate, new { @class = "control-label col-
md-2" })
            <div class="col-md-10">
                @Html.EditorFor(model => model.ReleaseDate)
                @Html.ValidationMessageFor(model => model.ReleaseDate)
            </div>
        </div>
        @*Genre and Price removed for brevity.*@
        <div class="form-group">
            <div class="col-md-offset-2 col-md-10">
                <input type="submit" value="Save" class="btn btn-default" />
            </div>
        </div>
    </div>
}
<div>
    @Html.ActionLink("Back to List", "Index")
</div>
@section Scripts {
    @Scripts.Render("~/bundles/jqueryval")
}
```

Notice how the view template has a `@model MvcMovie.Models.Movie` statement at the top of the file — this specifies that the view expects the model for the view template to be of type `Movie`.

The scaffolded code uses several *helper methods* to streamline the HTML markup. The **`Html.LabelFor`** **(http://msdn.microsoft.com/en-us/library/gg401864(VS.98).aspx)** helper displays the name of the field ("Title", "ReleaseDate", "Genre", or "Price"). The **`Html.EditorFor`** **(http://msdn.microsoft.com/en-us/library/system.web.mvc.html.editorextensions.editorfor(VS.98).aspx)** helper renders an HTML `<input>` element. The **`Html.ValidationMessageFor`** **(http://msdn.microsoft.com/en-us/library/system.web.mvc.html.validationextensions.validationmessagefor(VS.98).aspx)** helper displays any validation messages associated with that property.

Run the application and navigate to the */Movies* URL. Click an **Edit** link. In the browser, view the source for the page. The HTML for the form element is shown below.

```
<form action="/movies/Edit/4" method="post">
   <input name="__RequestVerificationToken" type="hidden" value="UxY6bkQyJCXO3Kn5AXg-
6TXxOj6yVBi9tghHaQ5Lq_qwKvcojNXEEfcbn-FGh_0vuw4tS_BRk7QQQHlJp8AP4_X4orVNoQnp2cd8kXhykS01"
/>  <fieldset class="form-horizontal">
```

```
      <legend>Movie</legend>

      <input data-val="true" data-val-number="The field ID must be a number." data-val-
required="The ID field is required." id="ID" name="ID" type="hidden" value="4" />

      <div class="control-group">
         <label class="control-label" for="Title">Title</label>
         <div class="controls">
            <input class="text-box single-line" id="Title" name="Title" type="text"
value="GhostBusters" />
            <span class="field-validation-valid help-inline" data-valmsg-for="Title"
data-valmsg-replace="true"></span>
         </div>
      </div>

      <div class="control-group">
         <label class="control-label" for="ReleaseDate">Release Date</label>
         <div class="controls">
            <input class="text-box single-line" data-val="true" data-val-date="The field
Release Date must be a date." data-val-required="The Release Date field is required."
id="ReleaseDate" name="ReleaseDate" type="date" value="1/1/1984" />
            <span class="field-validation-valid help-inline" data-valmsg-
for="ReleaseDate" data-valmsg-replace="true"></span>
         </div>
      </div>

      <div class="control-group">
         <label class="control-label" for="Genre">Genre</label>
         <div class="controls">
            <input class="text-box single-line" id="Genre" name="Genre" type="text"
value="Comedy" />
            <span class="field-validation-valid help-inline" data-valmsg-for="Genre"
data-valmsg-replace="true"></span>
         </div>
      </div>

      <div class="control-group">
         <label class="control-label" for="Price">Price</label>
         <div class="controls">
            <input class="text-box single-line" data-val="true" data-val-number="The
field Price must be a number." data-val-required="The Price field is required."
id="Price" name="Price" type="text" value="7.99" />
            <span class="field-validation-valid help-inline" data-valmsg-for="Price"
data-valmsg-replace="true"></span>
         </div>
      </div>

      <div class="form-actions no-color">
         <input type="submit" value="Save" class="btn" />
      </div>
   </fieldset>
</form>
```

The `<input>` elements are in an HTML `<form>` element whose `action` attribute is set to post to the */Movies/Edit* URL. The form data will be posted to the server when the **Save** button is clicked. The second line shows the hidden **XSRF (/mvc/overview/security/xsrfcsrf-prevention-in-aspnet-mvc-and-web-pages)** token generated by the `@Html.AntiForgeryToken()` call.

Processing the POST Request

The following listing shows the `HttpPost` version of the `Edit` action method.

```
[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult Edit([Bind(Include="ID,Title,ReleaseDate,Genre,Price")] Movie movie)
{
    if (ModelState.IsValid)
    {
        db.Entry(movie).State = EntityState.Modified;
        db.SaveChanges();
        return RedirectToAction("Index");
    }
    return View(movie);
}
```

The **ValidateAntiForgeryToken (http://msdn.microsoft.com/en-us/library/system.web.mvc.validateantiforgerytokenattribute(v=vs.108).aspx)** attribute validates the **XSRF (/mvc/overview/security/xsrfcsrf-prevention-in-aspnet-mvc-and-web-pages)** token generated by the `@Html.AntiForgeryToken()` call in the view.

The **ASP.NET MVC model binder (http://msdn.microsoft.com/en-us/library/dd410405.aspx)** takes the posted form values and creates a `Movie` object that's passed as the `movie` parameter. The `ModelState.IsValid` method verifies that the data submitted in the form can be used to modify (edit or update) a `Movie` object. If the data is valid, the movie data is saved to the `Movies` collection of the `db(MovieDBContext` instance). The new movie data is saved to the database by calling the `SaveChanges` method of `MovieDBContext`. After saving the data, the code redirects the user to the `Index` action method of the `MoviesController` class, which displays the movie collection, including the changes just made.

As soon as the client side validation determines the values of a field are not valid, an error message is displayed. If you disable JavaScript, you won't have client side validation but the server will detect the posted values are not valid, and the form values will be redisplayed with error messages. Later in the tutorial we examine validation in more detail.

The **`Html.ValidationMessageFor (http://msdn.microsoft.com/en-us/library/system.web.mvc.html.validationextensions.validationmessagefor(v=vs.108).aspx)`** helpers in the *Edit.cshtml* view template take care of displaying appropriate error messages.

All the `HttpGet` methods follow a similar pattern. They get a movie object (or list of objects, in the case of `Index`), and pass the model to the view. The `Create` method passes an empty movie object to the Create view. All the methods that create, edit, delete, or otherwise modify data do so in the `HttpPost` overload of the method. Modifying data in an HTTP GET method is a security risk, as described in the blog post entry **ASP.NET MVC Tip**

#46 – Don’t use Delete Links because they create Security Holes (http://stephenwalther.com/blog/archive/2009/01/21/asp.net-mvc-tip-46-ndash-donrsquot-use-delete-links-because.aspx) . Modifying data in a GET method also violates HTTP best practices and the architectural **REST (http://en.wikipedia.org/wiki/Representational_State_Transfer)** pattern, which specifies that GET requests should not change the state of your application. In other words, performing a GET operation should be a safe operation that has no side effects and doesn't modify your persisted data.

If you are using a US-English computer, you can skip this section and go to the next tutorial. You can download the Globalize version of this tutorial **here (http://archive.msdn.microsoft.com/Project/Download/FileDownload.aspx?ProjectName=aspnetmvcsamples&DownloadId=16475)** . For an excellent two part tutorial on Internationalization, see **Nadeem's ASP.NET MVC 5 Internationalization (http://afana.me/post/aspnet-mvc-internationalization.aspx)** .

Note to support jQuery validation for non-English locales that use a comma (",") for a decimal point, and non US-English date formats, you must include *globalize.js* and your specific *cultures/globalize.cultures.js* file(from **https://github.com/jquery/globalize (https://github.com/jquery/globalize)**) and JavaScript to use `Globalize.parseFloat`. You can get the jQuery non-English validation from NuGet. (Don't install Globalize if you are using a English locale.)

1. From the **Tools** menu click **Library Package Manager**, and then click **Manage NuGet Packages for Solution**.

2. On the left pane, select **Online.** (See the image below.)
3. In the **Search Installed packages** input box, enter *Globalize.*

   Click **Install**. The *Scripts\jquery.globalize\globalize.js* file will be added to your project. The *Scripts\jquery.globalize\cultures\* folder will contain many culture JavaScript files. Note, it can take five minutes to install this package.

The following code shows the modifications to the Views\Movies\Edit.cshtml file:

```
@section Scripts {
    @Scripts.Render("~/bundles/jqueryval")

<script src="~/Scripts/globalize/globalize.js"></script>
<script
src="~/Scripts/globalize/cultures/globalize.culture.@(System.Threading.Thread.CurrentThrea
</script>
<script>
    $.validator.methods.number = function (value, element) {
        return this.optional(element) ||
            !isNaN(Globalize.parseFloat(value));
    }
    $(document).ready(function () {

Globalize.culture('@(System.Threading.Thread.CurrentThread.CurrentCulture.Name)');
    });
</script>
<script>
    jQuery.extend(jQuery.validator.methods, {
        range: function (value, element, param) {
            //Use the Globalization plugin to parse the value
            var val = Globalize.parseFloat(value);
            return this.optional(element) || (
                val >= param[0] && val <= param[1]);
        }
    });
    $.validator.methods.date = function (value, element) {
        return this.optional(element) ||
            Globalize.parseDate(value) ||
            Globalize.parseDate(value, "yyyy-MM-dd");
    }
</script>
}
```

To avoid repeating this code in every Edit view, you can move it to the layout file. To optimize the script download, see my tutorial **Bundling and Minification (/mvc/tutorials/mvc-4/bundling-and-minification)** .

For more information see **ASP.NET MVC 3 Internationalization (http://afana.me/post/aspnet-mvc-internationalization.aspx)** and **ASP.NET MVC 3 Internationalization - Part 2 (NerdDinner) (http://afana.me/post/aspnet-mvc-internationalization-part-2.aspx)** .

As a temporary fix, if you can't get validation working in your locale, you can force your computer to use US English or you can disable JavaScript in your browser. To force your computer to use US English, you can add the globalization element to the projects root *web.config* file. The following code shows the globalization element with the culture set to United States English.

```
<system.web>
  <globalization culture ="en-US" />
  <!--elements removed for clarity-->
</system.web>
```

In the next tutorial, we'll implement search functionality.

This article was originally created on October 17, 2013

Author Information

Rick Anderson – Rick Anderson works as a programmer writer for Microsoft, focusing on ASP.NET MVC, Windows Azure and Entity Framework. You can follow him on twitter via @RickAndMSFT.

Comments (64)