The Learning House

427 S. 4th Street #300

Louisville KY 40202

HTML Helpers

.NET Cohort

Coding Bootcamp

SOFTWARE GUILD

HTML Attributes

Elements in HTML have attributes: additional values to configure them or adjust their behavior.

Attributes are defined as part of the opening portion of the HTML tag.

Ex:

Common HTML Attributes

Attribute	Usage
id	Identifies a unique element. Must be unique on the page.
name	Names an element. Multiple elements may share a name (and often do in the case of checkbox or radio button lists).
class	Used to reference CSS classes. Elements may have multiple CSS classes separated by a space. Class is a reserved word in C# so we have to say @class in Razor.
title	Most browsers display the title when the element is hovered over.
style	Applies a style directly to an element. (Best practice is to use CSS.)
placeholder	Puts placeholder text that is shown on an empty text input.
readonly	Makes an input tag read-only.

@HTML

The built-in HTML helper has a lot of methods to generate HTML elements that are “model-binder” ready.

@HTML.ActionLink

Syntax

ActionLink(text, action, controller, routeValues, attributes)

Usage

@HTML.ActionLink(“Link Text”, “Index”, “Home”, new { id=5}, new { @class=”CSSClass” })

Result

Link Text

@AntiForgeryToken

Syntax

AntiForgeryToken()

Usage

@HTML.AntiForgeryToken

Result

Hidden field with a token value that ensures the form is not submitted by any user or process that did not receive the original form render (Ex: bots).

@BeginForm

Syntax

BeginForm(action, controller, routeValues, FormMethod, attributes)

Usage

@HTML.BeginForm(“index”,”home”, new {id=1}, FormMethod.Post, new { role=“form”}

Result

<form action=“home/index/1” method=“post” role=“form”>

@CheckBox

Syntax

CheckBox(id, checked, attributes)

Usage

@HTML.CheckBox(“myCheck”, true)

Result

<input type=”checkbox” name=”myCheck” id=”myCheck” checked=”checked” />

@CheckBoxFor

Syntax

CheckBoxFor(lambda expression, attributes)

Usage

```
@HTML.CheckBoxFor(m => m.BoolProperty)
```

Result

```
<input type=”checkbox” name=”BoolProperty” id=”BoolProperty” />
```

@DropDownList

Syntax

DropDownList(name, IEnumerable<SelectListItem>, default text, attributes)

Usage

```
@HTML.DropDownList("myList",
   new List<SelectListItem>{
        new SelectListItem { Text="Item 1", Value="1"}},
   "- select -",    new { @class="ddl"})
```

Result

```
<select class="ddl" id="myList" name="myList">
  <option value="">- select -</option>
  <option value="1">Item 1</option>
</select>
```

@DropDownListFor

Syntax

DropDownListFor(lambda, IEnumerable<SelectListItem>, default text, attributes)

Usage

```
@HTML.DropDownList(m=>m.SelectedValue,
   new List<SelectListItem>{
        new SelectListItem { Text="Item 1", Value="1"}},
   "- select -",    new { @class="ddl"})
```

Result

```
<select class="ddl" id="myList" name="myList">
  <option value="">- select -</option>
  <option value="1">Item 1</option>
</select>
```

@Hidden

Syntax

Hidden(name, value)

Usage

@HTML.Hidden(“myHidden”, “banana”)

Result

<input type=“hidden” value=“banana” name=“myHidden” id=“myHidden” />

@HiddenFor

Syntax

HiddenFor(lambda)

Usage

```
@HTML.HiddenFor(m => m.Property)
```

Result

<input type="hidden" value="value of property" name="Property" id="Property" />

@ListBox

Syntax

ListBox(name, IEnumerable<SelectListItem>, default text, attributes)

Usage

```
@HTML.ListBox("myList",
   new List<SelectListItem>{
        new SelectListItem { Text="Item 1", Value="1"}},
   "- select -",    new { @class="ddl"})
```

Result

```
<select class="ddl" id="myList" name="myList“ multiple=“multiple”>
   <option value="">- select -</option>
   <option value="1">Item 1</option>
</select>
```

@ListBoxFor

Syntax

DropDownListFor(lambda, IEnumerable<SelectListItem>, default text, attributes)

Usage

```
@HTML.ListBoxFor(m=>m.SelectedItems, Model.Items, new
{@class="lbx"})
```

Result

```
<select class="lbx" id="SelectedItems" multiple="multiple" name="SelectedItems">
        <option value="1">option 1</option>
        <option value="2">option 2</option>
        <option value="3">option 3</option>
        <option value="4">option 4</option>
</select>
```

@Password

Syntax

Password(name, value)

Usage

@HTML.Password(“pwd”, “banana”)

Result

<input type=“password” value=“banana” name=“pwd” id=“pwd” />

@PasswordFor

Syntax

PasswordFor(lambda, attributes)

Usage

```
@HTML.PasswordFor(m => m.Property)
```

Result

<input type="password" value="value of property" name="pwd" id="pwd" />

@RadioButton

Syntax

RadioButton(name, value, checked, attributes)

Usage

@HTML.RadioButton(“Gender”, “male”, true)
@HTML.RadioButton(“Gender”, “female”, false)

Result

<input type=”radio” id=”Gender” name=”Gender” value=”male” checked=”checked” />

<input type=”radio” id=”Gender” name=”Gender” value=”female” />

@RadioButtonFor

Syntax

RadioButtonFor(lambda expression, value, attributes)

Usage

```
@HTML.RadioButtonFor(m =>m.Gender, “male”, new
{@checked=”checked”)
@HTML.RadioButtonFor(m =>m.Gender, “female”)
```

Result

```
<input type=”radio” id=”Gender” name=”Gender” value=”male”
checked=”checked” />
<input type=”radio” id=”Gender” name=”Gender” value=”female” />
```

@TextArea

Syntax

TextArea(name, value)

Usage

@HTML.TextArea(“myText”, “some text”)

Result

<textarea name=”myText” id=”myText”>some text</textarea>

@TextAreaFor

Syntax

TextAreaFor(lambda, attributes)

Usage

```
@HTML.TextAreaFor(m => m.Property)
```

Result

```
<textarea name=”Property” id=”Property”>value of Property</textarea>
```

@TextBox

Syntax

TextBox(name, value)

Usage

@HTML.TextBox(“myText”, “some text”)

Result

<input type=“text” name=”myText” id=”myText”>some text</input>

@TextBoxFor

Syntax

TextBoxFor(lambda, attributes)

Usage

```
@HTML.TextBoxFor(m => m.Property)
```

Result

<input type="text" value="value of property" name="Property" id="Property" />

Custom Helpers

The HTML helper methods are extension methods, so we can easily add our own helpers to the stack. Helpers must return an MvcHtmlString and can use the TagBuilder class to format HTML.

```
public static class HtmlHelperExtensions
{
    public static MvcHtmlString PlaceHolderTextBox(this HtmlHelper htmlHelper,
                                                   string name, string placeHolderText)
    {
        var builder = new TagBuilder("input");
        builder.MergeAttribute("type", "text");
        builder.MergeAttribute("name", name);
        builder.MergeAttribute("id", name);
        builder.MergeAttribute("placeholder", placeHolderText);

        return MvcHtmlString.Create(builder.ToString(TagRenderMode.SelfClosing));
    }
}
```

Using it in a view

Then, you can just put a using statement in your view to the namespace of the extensions and call it! Ex: @HTML.PlaceHolderTextBox

```
@using MovieTracker.UI.Extensions

@{
    ViewBag.Title = "Index";
    Layout = null;
}

<h2>Index</h2>

@Html.PlaceHolderTextBox("myText", "Some placeholder text")
```

Gut Check

- What is the difference between @HTML.TextBox() and @HTML.TextBoxFor()
  - Why do we care?
  - Remember, it's the same for all the <Control>For()
- How do you add random attributes to the generated HTML?