Writing custom HTML Helpers for ASP.NET MVC

Ed Charbeneau

Table of Contents

Chapter 1: Transitioning from webforms 4

Chapter 2: How to begin 4

Chapter 3: Writing a specification 4

Chapter 4: Unit testing 4

Chapter 5: Basic implementation 4

Chapter 6: Fluent configuration 4

Chapter 7: Strongly typed helpers 4

Chapter 8: Final results 4

Chapter 9: Considerations 4

Chapter 1: Transitioning from web forms

As a web forms developer I found the transition to MVC was a bit of a shock at first. Without fully understanding the nature of MVC I found the lack of a Toolbox filled with server controls confusing. However once it became clear that the goal of MVC was to expose HTML markup and give developers full control over what is rendered to the browser I quickly embraced the idea.

In MVC development, HTML helpers replace the server control, but the similarities aren’t exactly parallel. Web form’s and server controls were intended to bring the workflow of desktop forms to the web. In MVC HTML helpers provide a shortcut to writing out raw HTML elements that are frequently used.

The HTML helper, in most cases, is a method that returns a string. When called in a View using the razor syntax @Html, we are accessing the Html property of the View which is an instance of the HtmlHelper class.

Writing extensions for the HtmlHelper class will allow us to create our own custom helpers to encapsulate complex HTML markup. Custom helpers promote the use of reusable code and are unit testable. Custom helpers can be configured by passing values to the constructor, via fluent configuration, strongly typed or a combination of and ultimately return a string.

Chapter 2: How to begin

Reusable Html elements

Alert

HTML, CSS and Javascript

Chapter 3: Writing a specification

First step

Drafting the syntax

Chapter 4: Unit testing

HtmlHelperFactory

Checking the output

Chapter 5: Basic implementation

HtmlHelper extension method

Alert constructor

Advanced options

Attributes merge

Chapter 6: Fluent Configuration

Appending the spec

Method chaining with interfaces

More testing

Chapter 7: Strongly typed helpers

Appending the spec again

HtmlHelper extension methods again

The Element**For** convention

<TModel>

Expression<Func<T,T>>

Model Metadata

Chapter 8: Final results

Basic

Fluent

Strongly typed

Chapter 9: Considerations

When to create a helper

When to use basic, fluent, strongly typed or all.

What’s next, templates, and complex controls

Resources