// Copyright (c) Microsoft Open Technologies, Inc. All rights reserved. See License.txt in the project root for license information.

using System.Collections.Generic;

using System.Data.Linq;

using System.Diagnostics.CodeAnalysis;

using System.Globalization;

using System.Linq.Expressions;

using System.Text;

using System.Web.Mvc.Properties;

using System.Web.Routing;

namespace System.Web.Mvc.Html

{

public static class InputExtensions

{

// CheckBox

public static MvcHtmlString CheckBox(this HtmlHelper htmlHelper, string name)

{

return CheckBox(htmlHelper, name, htmlAttributes: (object)null);

}

public static MvcHtmlString CheckBox(this HtmlHelper htmlHelper, string name, bool isChecked)

{

return CheckBox(htmlHelper, name, isChecked, htmlAttributes: (object)null);

}

public static MvcHtmlString CheckBox(this HtmlHelper htmlHelper, string name, bool isChecked, object htmlAttributes)

{

return CheckBox(htmlHelper, name, isChecked, HtmlHelper.AnonymousObjectToHtmlAttributes(htmlAttributes));

}

public static MvcHtmlString CheckBox(this HtmlHelper htmlHelper, string name, object htmlAttributes)

{

return CheckBox(htmlHelper, name, HtmlHelper.AnonymousObjectToHtmlAttributes(htmlAttributes));

}

public static MvcHtmlString CheckBox(this HtmlHelper htmlHelper, string name, IDictionary<string, object> htmlAttributes)

{

return CheckBoxHelper(htmlHelper, metadata: null, name: name, isChecked: null, htmlAttributes: htmlAttributes);

}

public static MvcHtmlString CheckBox(this HtmlHelper htmlHelper, string name, bool isChecked, IDictionary<string, object> htmlAttributes)

{

return CheckBoxHelper(htmlHelper, metadata: null, name: name, isChecked: isChecked, htmlAttributes: htmlAttributes);

}

[SuppressMessage("Microsoft.Design", "CA1006:DoNotNestGenericTypesInMemberSignatures", Justification = "This is an appropriate nesting of generic types")]

public static MvcHtmlString CheckBoxFor<TModel>(this HtmlHelper<TModel> htmlHelper, Expression<Func<TModel, bool>> expression)

{

return CheckBoxFor(htmlHelper, expression, htmlAttributes: null);

}

[SuppressMessage("Microsoft.Design", "CA1006:DoNotNestGenericTypesInMemberSignatures", Justification = "This is an appropriate nesting of generic types")]

public static MvcHtmlString CheckBoxFor<TModel>(this HtmlHelper<TModel> htmlHelper, Expression<Func<TModel, bool>> expression, object htmlAttributes)

{

return CheckBoxFor(htmlHelper, expression, HtmlHelper.AnonymousObjectToHtmlAttributes(htmlAttributes));

}

[SuppressMessage("Microsoft.Design", "CA1006:DoNotNestGenericTypesInMemberSignatures", Justification = "This is an appropriate nesting of generic types")]

public static MvcHtmlString CheckBoxFor<TModel>(this HtmlHelper<TModel> htmlHelper, Expression<Func<TModel, bool>> expression, IDictionary<string, object> htmlAttributes)

{

if (expression == null)

{

throw new ArgumentNullException("expression");

}

ModelMetadata metadata = ModelMetadata.FromLambdaExpression(expression, htmlHelper.ViewData);

bool? isChecked = null;

if (metadata.Model != null)

{

bool modelChecked;

if (Boolean.TryParse(metadata.Model.ToString(), out modelChecked))

{

isChecked = modelChecked;

}

}

return CheckBoxHelper(htmlHelper, metadata, ExpressionHelper.GetExpressionText(expression), isChecked, htmlAttributes);

}

private static MvcHtmlString CheckBoxHelper(HtmlHelper htmlHelper, ModelMetadata metadata, string name, bool? isChecked, IDictionary<string, object> htmlAttributes)

{

RouteValueDictionary attributes = ToRouteValueDictionary(htmlAttributes);

bool explicitValue = isChecked.HasValue;

if (explicitValue)

{

attributes.Remove("checked"); // Explicit value must override dictionary

}

return InputHelper(htmlHelper,

InputType.CheckBox,

metadata,

name,

value: "true",

useViewData: !explicitValue,

isChecked: isChecked ?? false,

setId: true,

isExplicitValue: false,

format: null,

htmlAttributes: attributes);

}

// Hidden

public static MvcHtmlString Hidden(this HtmlHelper htmlHelper, string name)

{

return Hidden(htmlHelper, name, value: null, htmlAttributes: null);

}

public static MvcHtmlString Hidden(this HtmlHelper htmlHelper, string name, object value)

{

return Hidden(htmlHelper, name, value, htmlAttributes: null);

}

public static MvcHtmlString Hidden(this HtmlHelper htmlHelper, string name, object value, object htmlAttributes)

{

return Hidden(htmlHelper, name, value, HtmlHelper.AnonymousObjectToHtmlAttributes(htmlAttributes));

}

public static MvcHtmlString Hidden(this HtmlHelper htmlHelper, string name, object value, IDictionary<string, object> htmlAttributes)

{

return HiddenHelper(htmlHelper,

metadata: null,

value: value,

useViewData: value == null,

expression: name,

htmlAttributes: htmlAttributes);

}

[SuppressMessage("Microsoft.Design", "CA1006:DoNotNestGenericTypesInMemberSignatures", Justification = "This is an appropriate nesting of generic types")]

public static MvcHtmlString HiddenFor<TModel, TProperty>(this HtmlHelper<TModel> htmlHelper, Expression<Func<TModel, TProperty>> expression)

{

return HiddenFor(htmlHelper, expression, (IDictionary<string, object>)null);

}

[SuppressMessage("Microsoft.Design", "CA1006:DoNotNestGenericTypesInMemberSignatures", Justification = "This is an appropriate nesting of generic types")]

public static MvcHtmlString HiddenFor<TModel, TProperty>(this HtmlHelper<TModel> htmlHelper, Expression<Func<TModel, TProperty>> expression, object htmlAttributes)

{

return HiddenFor(htmlHelper, expression, HtmlHelper.AnonymousObjectToHtmlAttributes(htmlAttributes));

}

[SuppressMessage("Microsoft.Design", "CA1006:DoNotNestGenericTypesInMemberSignatures", Justification = "This is an appropriate nesting of generic types")]

public static MvcHtmlString HiddenFor<TModel, TProperty>(this HtmlHelper<TModel> htmlHelper, Expression<Func<TModel, TProperty>> expression, IDictionary<string, object> htmlAttributes)

{

ModelMetadata metadata = ModelMetadata.FromLambdaExpression(expression, htmlHelper.ViewData);

return HiddenHelper(htmlHelper,

metadata,

metadata.Model,

false,

ExpressionHelper.GetExpressionText(expression),

htmlAttributes);

}

private static MvcHtmlString HiddenHelper(HtmlHelper htmlHelper, ModelMetadata metadata, object value, bool useViewData, string expression, IDictionary<string, object> htmlAttributes)

{

Binary binaryValue = value as Binary;

if (binaryValue != null)

{

value = binaryValue.ToArray();

}

byte[] byteArrayValue = value as byte[];

if (byteArrayValue != null)

{

value = Convert.ToBase64String(byteArrayValue);

}

return InputHelper(htmlHelper,

InputType.Hidden,

metadata,

expression,

value,

useViewData,

isChecked: false,

setId: true,

isExplicitValue: true,

format: null,

htmlAttributes: htmlAttributes);

}

// Password

public static MvcHtmlString Password(this HtmlHelper htmlHelper, string name)

{

return Password(htmlHelper, name, value: null);

}

public static MvcHtmlString Password(this HtmlHelper htmlHelper, string name, object value)

{

return Password(htmlHelper, name, value, htmlAttributes: null);

}

public static MvcHtmlString Password(this HtmlHelper htmlHelper, string name, object value, object htmlAttributes)

{

return Password(htmlHelper, name, value, HtmlHelper.AnonymousObjectToHtmlAttributes(htmlAttributes));

}

public static MvcHtmlString Password(this HtmlHelper htmlHelper, string name, object value, IDictionary<string, object> htmlAttributes)

{

return PasswordHelper(htmlHelper, metadata: null, name: name, value: value, htmlAttributes: htmlAttributes);

}

[SuppressMessage("Microsoft.Design", "CA1006:DoNotNestGenericTypesInMemberSignatures", Justification = "This is an appropriate nesting of generic types")]

public static MvcHtmlString PasswordFor<TModel, TProperty>(this HtmlHelper<TModel> htmlHelper, Expression<Func<TModel, TProperty>> expression)

{

return PasswordFor(htmlHelper, expression, htmlAttributes: null);

}

[SuppressMessage("Microsoft.Design", "CA1006:DoNotNestGenericTypesInMemberSignatures", Justification = "This is an appropriate nesting of generic types")]

public static MvcHtmlString PasswordFor<TModel, TProperty>(this HtmlHelper<TModel> htmlHelper, Expression<Func<TModel, TProperty>> expression, object htmlAttributes)

{

return PasswordFor(htmlHelper, expression, HtmlHelper.AnonymousObjectToHtmlAttributes(htmlAttributes));

}

[SuppressMessage("Microsoft.Design", "CA1011:ConsiderPassingBaseTypesAsParameters", Justification = "Users cannot use anonymous methods with the LambdaExpression type")]

[SuppressMessage("Microsoft.Design", "CA1006:DoNotNestGenericTypesInMemberSignatures", Justification = "This is an appropriate nesting of generic types")]

public static MvcHtmlString PasswordFor<TModel, TProperty>(this HtmlHelper<TModel> htmlHelper, Expression<Func<TModel, TProperty>> expression, IDictionary<string, object> htmlAttributes)

{

if (expression == null)

{

throw new ArgumentNullException("expression");

}

return PasswordHelper(htmlHelper,

ModelMetadata.FromLambdaExpression(expression, htmlHelper.ViewData),

ExpressionHelper.GetExpressionText(expression),

value: null,

htmlAttributes: htmlAttributes);

}

private static MvcHtmlString PasswordHelper(HtmlHelper htmlHelper, ModelMetadata metadata, string name, object value, IDictionary<string, object> htmlAttributes)

{

return InputHelper(htmlHelper,

InputType.Password,

metadata,

name,

value,

useViewData: false,

isChecked: false,

setId: true,

isExplicitValue: true,

format: null,

htmlAttributes: htmlAttributes);

}

// RadioButton

public static MvcHtmlString RadioButton(this HtmlHelper htmlHelper, string name, object value)

{

return RadioButton(htmlHelper, name, value, htmlAttributes: (object)null);

}

public static MvcHtmlString RadioButton(this HtmlHelper htmlHelper, string name, object value, object htmlAttributes)

{

return RadioButton(htmlHelper, name, value, HtmlHelper.AnonymousObjectToHtmlAttributes(htmlAttributes));

}

public static MvcHtmlString RadioButton(this HtmlHelper htmlHelper, string name, object value, IDictionary<string, object> htmlAttributes)

{

// Determine whether or not to render the checked attribute based on the contents of ViewData.

string valueString = Convert.ToString(value, CultureInfo.CurrentCulture);

bool isChecked = (!String.IsNullOrEmpty(name)) && (String.Equals(htmlHelper.EvalString(name), valueString, StringComparison.OrdinalIgnoreCase));

// checked attributes is implicit, so we need to ensure that the dictionary takes precedence.

RouteValueDictionary attributes = ToRouteValueDictionary(htmlAttributes);

if (attributes.ContainsKey("checked"))

{

return InputHelper(htmlHelper,

InputType.Radio,

metadata: null,

name: name,

value: value,

useViewData: false,

isChecked: false,

setId: true,

isExplicitValue: true,

format: null,

htmlAttributes: attributes);

}

return RadioButton(htmlHelper, name, value, isChecked, htmlAttributes);

}

public static MvcHtmlString RadioButton(this HtmlHelper htmlHelper, string name, object value, bool isChecked)

{

return RadioButton(htmlHelper, name, value, isChecked, htmlAttributes: (object)null);

}

public static MvcHtmlString RadioButton(this HtmlHelper htmlHelper, string name, object value, bool isChecked, object htmlAttributes)

{

return RadioButton(htmlHelper, name, value, isChecked, HtmlHelper.AnonymousObjectToHtmlAttributes(htmlAttributes));

}

public static MvcHtmlString RadioButton(this HtmlHelper htmlHelper, string name, object value, bool isChecked, IDictionary<string, object> htmlAttributes)

{

if (value == null)

{

throw new ArgumentNullException("value");

}

// checked attribute is an explicit parameter so it takes precedence.

RouteValueDictionary attributes = ToRouteValueDictionary(htmlAttributes);

attributes.Remove("checked");

return InputHelper(htmlHelper,

InputType.Radio,

metadata: null,

name: name,

value: value,

useViewData: false,

isChecked: isChecked,

setId: true,

isExplicitValue: true,

format: null,

htmlAttributes: attributes);

}

[SuppressMessage("Microsoft.Design", "CA1006:DoNotNestGenericTypesInMemberSignatures", Justification = "This is an appropriate nesting of generic types")]

public static MvcHtmlString RadioButtonFor<TModel, TProperty>(this HtmlHelper<TModel> htmlHelper, Expression<Func<TModel, TProperty>> expression, object value)

{

return RadioButtonFor(htmlHelper, expression, value, htmlAttributes: null);

}

[SuppressMessage("Microsoft.Design", "CA1006:DoNotNestGenericTypesInMemberSignatures", Justification = "This is an appropriate nesting of generic types")]

public static MvcHtmlString RadioButtonFor<TModel, TProperty>(this HtmlHelper<TModel> htmlHelper, Expression<Func<TModel, TProperty>> expression, object value, object htmlAttributes)

{

return RadioButtonFor(htmlHelper, expression, value, HtmlHelper.AnonymousObjectToHtmlAttributes(htmlAttributes));

}

[SuppressMessage("Microsoft.Design", "CA1006:DoNotNestGenericTypesInMemberSignatures", Justification = "This is an appropriate nesting of generic types")]

public static MvcHtmlString RadioButtonFor<TModel, TProperty>(this HtmlHelper<TModel> htmlHelper, Expression<Func<TModel, TProperty>> expression, object value, IDictionary<string, object> htmlAttributes)

{

ModelMetadata metadata = ModelMetadata.FromLambdaExpression(expression, htmlHelper.ViewData);

return RadioButtonHelper(htmlHelper,

metadata,

metadata.Model,

ExpressionHelper.GetExpressionText(expression),

value,

null /\* isChecked \*/,

htmlAttributes);

}

private static MvcHtmlString RadioButtonHelper(HtmlHelper htmlHelper, ModelMetadata metadata, object model, string name, object value, bool? isChecked, IDictionary<string, object> htmlAttributes)

{

if (value == null)

{

throw new ArgumentNullException("value");

}

RouteValueDictionary attributes = ToRouteValueDictionary(htmlAttributes);

bool explicitValue = isChecked.HasValue;

if (explicitValue)

{

attributes.Remove("checked"); // Explicit value must override dictionary

}

else

{

string valueString = Convert.ToString(value, CultureInfo.CurrentCulture);

isChecked = model != null &&

!String.IsNullOrEmpty(name) &&

String.Equals(model.ToString(), valueString, StringComparison.OrdinalIgnoreCase);

}

return InputHelper(htmlHelper,

InputType.Radio,

metadata,

name,

value,

useViewData: false,

isChecked: isChecked ?? false,

setId: true,

isExplicitValue: true,

format: null,

htmlAttributes: attributes);

}

// TextBox

public static MvcHtmlString TextBox(this HtmlHelper htmlHelper, string name)

{

return TextBox(htmlHelper, name, value: null);

}

public static MvcHtmlString TextBox(this HtmlHelper htmlHelper, string name, object value)

{

return TextBox(htmlHelper, name, value, format: null);

}

public static MvcHtmlString TextBox(this HtmlHelper htmlHelper, string name, object value, string format)

{

return TextBox(htmlHelper, name, value, format, htmlAttributes: (object)null);

}

public static MvcHtmlString TextBox(this HtmlHelper htmlHelper, string name, object value, object htmlAttributes)

{

return TextBox(htmlHelper, name, value, format: null, htmlAttributes: htmlAttributes);

}

public static MvcHtmlString TextBox(this HtmlHelper htmlHelper, string name, object value, string format, object htmlAttributes)

{

return TextBox(htmlHelper, name, value, format, HtmlHelper.AnonymousObjectToHtmlAttributes(htmlAttributes));

}

public static MvcHtmlString TextBox(this HtmlHelper htmlHelper, string name, object value, IDictionary<string, object> htmlAttributes)

{

return TextBox(htmlHelper, name, value, format: null, htmlAttributes: htmlAttributes);

}

public static MvcHtmlString TextBox(this HtmlHelper htmlHelper, string name, object value, string format, IDictionary<string, object> htmlAttributes)

{

return InputHelper(htmlHelper,

InputType.Text,

metadata: null,

name: name,

value: value,

useViewData: (value == null),

isChecked: false,

setId: true,

isExplicitValue: true,

format: format,

htmlAttributes: htmlAttributes);

}

[SuppressMessage("Microsoft.Design", "CA1006:DoNotNestGenericTypesInMemberSignatures", Justification = "This is an appropriate nesting of generic types")]

public static MvcHtmlString TextBoxFor<TModel, TProperty>(this HtmlHelper<TModel> htmlHelper, Expression<Func<TModel, TProperty>> expression)

{

return htmlHelper.TextBoxFor(expression, format: null);

}

[SuppressMessage("Microsoft.Design", "CA1006:DoNotNestGenericTypesInMemberSignatures", Justification = "This is an appropriate nesting of generic types")]

public static MvcHtmlString TextBoxFor<TModel, TProperty>(this HtmlHelper<TModel> htmlHelper, Expression<Func<TModel, TProperty>> expression, string format)

{

return htmlHelper.TextBoxFor(expression, format, (IDictionary<string, object>)null);

}

[SuppressMessage("Microsoft.Design", "CA1006:DoNotNestGenericTypesInMemberSignatures", Justification = "This is an appropriate nesting of generic types")]

public static MvcHtmlString TextBoxFor<TModel, TProperty>(this HtmlHelper<TModel> htmlHelper, Expression<Func<TModel, TProperty>> expression, object htmlAttributes)

{

return htmlHelper.TextBoxFor(expression, format: null, htmlAttributes: htmlAttributes);

}

[SuppressMessage("Microsoft.Design", "CA1006:DoNotNestGenericTypesInMemberSignatures", Justification = "This is an appropriate nesting of generic types")]

public static MvcHtmlString TextBoxFor<TModel, TProperty>(this HtmlHelper<TModel> htmlHelper, Expression<Func<TModel, TProperty>> expression, string format, object htmlAttributes)

{

return htmlHelper.TextBoxFor(expression, format: format, htmlAttributes: HtmlHelper.AnonymousObjectToHtmlAttributes(htmlAttributes));

}

[SuppressMessage("Microsoft.Design", "CA1006:DoNotNestGenericTypesInMemberSignatures", Justification = "This is an appropriate nesting of generic types")]

public static MvcHtmlString TextBoxFor<TModel, TProperty>(this HtmlHelper<TModel> htmlHelper, Expression<Func<TModel, TProperty>> expression, IDictionary<string, object> htmlAttributes)

{

return htmlHelper.TextBoxFor(expression, format: null, htmlAttributes: htmlAttributes);

}

[SuppressMessage("Microsoft.Design", "CA1006:DoNotNestGenericTypesInMemberSignatures", Justification = "This is an appropriate nesting of generic types")]

public static MvcHtmlString TextBoxFor<TModel, TProperty>(this HtmlHelper<TModel> htmlHelper, Expression<Func<TModel, TProperty>> expression, string format, IDictionary<string, object> htmlAttributes)

{

ModelMetadata metadata = ModelMetadata.FromLambdaExpression(expression, htmlHelper.ViewData);

return TextBoxHelper(htmlHelper,

metadata,

metadata.Model,

ExpressionHelper.GetExpressionText(expression),

format,

htmlAttributes);

}

private static MvcHtmlString TextBoxHelper(this HtmlHelper htmlHelper, ModelMetadata metadata, object model, string expression, string format, IDictionary<string, object> htmlAttributes)

{

return InputHelper(htmlHelper,

InputType.Text,

metadata,

expression,

model,

useViewData: false,

isChecked: false,

setId: true,

isExplicitValue: true,

format: format,

htmlAttributes: htmlAttributes);

}

// Helper methods

private static MvcHtmlString InputHelper(HtmlHelper htmlHelper, InputType inputType, ModelMetadata metadata, string name, object value, bool useViewData, bool isChecked, bool setId, bool isExplicitValue, string format, IDictionary<string, object> htmlAttributes)

{

string fullName = htmlHelper.ViewContext.ViewData.TemplateInfo.GetFullHtmlFieldName(name);

if (String.IsNullOrEmpty(fullName))

{

throw new ArgumentException(MvcResources.Common\_NullOrEmpty, "name");

}

TagBuilder tagBuilder = new TagBuilder("input");

tagBuilder.MergeAttributes(htmlAttributes);

tagBuilder.MergeAttribute("type", HtmlHelper.GetInputTypeString(inputType));

tagBuilder.MergeAttribute("name", fullName, true);

string valueParameter = htmlHelper.FormatValue(value, format);

bool usedModelState = false;

switch (inputType)

{

case InputType.CheckBox:

bool? modelStateWasChecked = htmlHelper.GetModelStateValue(fullName, typeof(bool)) as bool?;

if (modelStateWasChecked.HasValue)

{

isChecked = modelStateWasChecked.Value;

usedModelState = true;

}

goto case InputType.Radio;

case InputType.Radio:

if (!usedModelState)

{

string modelStateValue = htmlHelper.GetModelStateValue(fullName, typeof(string)) as string;

if (modelStateValue != null)

{

isChecked = String.Equals(modelStateValue, valueParameter, StringComparison.Ordinal);

usedModelState = true;

}

}

if (!usedModelState && useViewData)

{

isChecked = htmlHelper.EvalBoolean(fullName);

}

if (isChecked)

{

tagBuilder.MergeAttribute("checked", "checked");

}

tagBuilder.MergeAttribute("value", valueParameter, isExplicitValue);

break;

case InputType.Password:

if (value != null)

{

tagBuilder.MergeAttribute("value", valueParameter, isExplicitValue);

}

break;

default:

string attemptedValue = (string)htmlHelper.GetModelStateValue(fullName, typeof(string));

tagBuilder.MergeAttribute("value", attemptedValue ?? ((useViewData) ? htmlHelper.EvalString(fullName, format) : valueParameter), isExplicitValue);

break;

}

if (setId)

{

tagBuilder.GenerateId(fullName);

}

// If there are any errors for a named field, we add the css attribute.

ModelState modelState;

if (htmlHelper.ViewData.ModelState.TryGetValue(fullName, out modelState))

{

if (modelState.Errors.Count > 0)

{

tagBuilder.AddCssClass(HtmlHelper.ValidationInputCssClassName);

}

}

tagBuilder.MergeAttributes(htmlHelper.GetUnobtrusiveValidationAttributes(name, metadata));

if (inputType == InputType.CheckBox)

{

// Render an additional <input type="hidden".../> for checkboxes. This

// addresses scenarios where unchecked checkboxes are not sent in the request.

// Sending a hidden input makes it possible to know that the checkbox was present

// on the page when the request was submitted.

StringBuilder inputItemBuilder = new StringBuilder();

inputItemBuilder.Append(tagBuilder.ToString(TagRenderMode.SelfClosing));

TagBuilder hiddenInput = new TagBuilder("input");

hiddenInput.MergeAttribute("type", HtmlHelper.GetInputTypeString(InputType.Hidden));

hiddenInput.MergeAttribute("name", fullName);

hiddenInput.MergeAttribute("value", "false");

inputItemBuilder.Append(hiddenInput.ToString(TagRenderMode.SelfClosing));

return MvcHtmlString.Create(inputItemBuilder.ToString());

}

return tagBuilder.ToMvcHtmlString(TagRenderMode.SelfClosing);

}

private static RouteValueDictionary ToRouteValueDictionary(IDictionary<string, object> dictionary)

{

return dictionary == null ? new RouteValueDictionary() : new RouteValueDictionary(dictionary);

}

}

}

static object GetModelStateValue(HtmlHelper htmlHelper, string key, Type destinationType)

{

ModelState modelState;

if (htmlHelper.ViewData.ModelState.TryGetValue(key, out modelState))

{

if (modelState.Value != null)

{

return modelState.Value.ConvertTo(destinationType, null /\* culture \*/);

}

}

return null;

}