To serve content in multiple languages on your MVC site, you first need to set up functionality that detects and sets the current culture for each request. You can then localize the content displayed on the site's pages.

Enabling localization in MVC projects

If you want to have a multilingual MVC site, you need to set up culture recognition in the MVC project. On the beginning of every request, you need to:

1. Retrieve or determine the correct culture to be used in the current request. The way you detect the culture depends on the implementation of your multilingual site. For example, possible options are culture prefixes in URLs, culture-specific domains, custom cookies, etc.
2. Set the **Thread.CurrentThread.CurrentUICulture** and **Thread.CurrentThread.CurrentCulture** properties of the current thread (available in the *System.Threading* namespace).

> **Note**: *Thread.CurrentThread.CurrentUICulture* and *Thread.CurrentThread.CurrentCulture* are properties of the .NET framework and use the *System.Globalization.CultureInfo* type. They are not directly comparable with the Kentico *CMS.Localization.CultureInfo* type, or the *CurrentUICulture* and *CurrentCulture* properties of the *CMS.Localization.LocalizationContext* class.

The Kentico localization API then automatically works with the given culture (for example when resolving resource strings).

Example

One common way to determine the culture of page requests is to use culture prefixes in your site's routes. For example:

- English – *www.example.com/**en-us**/path*
- Spanish – *www.example.com/**es-es**/path*

The following code examples showcase how to parse the culture from the route prefix and set the current culture for the MVC application. You can modify and adjust snippets from this example for use in your own projects.

Example - RouteConfig

```
using System.Web.Mvc;
using System.Web.Routing;

...

public static void RegisterRoutes(RouteCollection routes)
{
    ...

    // Parses a URL containing a culture route prefix
    var route = routes.MapRoute(
        name: "Default",
        url: "{culture}/{controller}/{action}",
        defaults: new { controller = "Home", action = "Index" },
        constraints: new { culture = new SiteCultureConstraint() }
    );

    // Assigns a custom route handler to the route
    route.RouteHandler = new MultiCultureMvcRouteHandler();
}
```

Example - MultiCultureMvcRouteHandler

```
using System.Globalization;
using System.Threading;
using System.Web;
using System.Web.Mvc;
using System.Web.Routing;

public class MultiCultureMvcRouteHandler : MvcRouteHandler
{
    protected override IHttpHandler GetHttpHandler(RequestContext requestContext)
    {
        // Retrieves the requested culture from the route
        var cultureName = requestContext.RouteData.Values["culture"].ToString();

        try
        {
            // Creates a CultureInfo object from the culture code
            var culture = new CultureInfo(cultureName);

            // Sets the current culture for the MVC application
            Thread.CurrentThread.CurrentUICulture = culture;
            Thread.CurrentThread.CurrentCulture = culture;
        }
        catch
        {
            // Handles cases where the culture parameter of the route is invalid
            // Returns a 404 status in this case, but you can also log an error, set a
default culture, etc.
            requestContext.HttpContext.Response.StatusCode = 404;
        }

        return base.GetHttpHandler(requestContext);
    }
}
```

Example - SiteCultureConstraint

```
using System.Web;
using System.Web.Routing;

using CMS.SiteProvider;

// Constraint that restricts culture parameter values
// Only allows the codes of cultures assigned to the current site in Kentico
public class SiteCultureConstraint : IRouteConstraint
{
    public bool Match(HttpContextBase httpContext,
                      Route route,
                      string parameterName,
                      RouteValueDictionary values,
                      RouteDirection routeDirection)
    {
        string cultureCodeName = values[parameterName]?.ToString();
        return CultureSiteInfoProvider.IsCultureOnSite(cultureCodeName, SiteContext.
CurrentSiteName);
    }
}
```

Localizing site content

When retrieving page content, load the correct culture version of pages based on the current culture of the request.

For individual text strings displayed on the site (which are not stored within page fields), we recommend using multilingual resource strings. You can create and edit the strings in the **Localization** application of the Kentico administration interface.

To work with the strings in your code, you can create an alias for the *CMS.Helpers.ResHelper* class and use the *GetString()* method:

```
@using Resources = CMS.Helpers.ResHelper;
...
<h2>@Resources.GetString("SiteName.OurProducts")</h2>
```

```
@using Resources = CMS.Helpers.ResHelper;
...
@{
    ViewBag.Title = Resources.GetString("SiteName.OurProducts");
}
```

Localizing validation results and model properties

The default approach to validating the data model of MVC applications is to decorate the model and its properties with attributes from the System.ComponentModel.DataAnnotations namespace. You use the attributes to define common validation patterns, such as range checking, string length and required fields.

The *Kentico.Web.Mvc* integration package provides a feature that allows you to use localized Kentico resource string keys as error messages in data annotation attributes. The strings are localized based on the current culture context of the current visitor. The *Kentico.Languages.English* integration package (installed with the *Kentico.Web.Mvc* integration package) contains a *.resx* file with the resource strings used in the English localization of Kentico. Aside from that, you can also use the standard resource strings stored in the database.

The feature supports localization of the following data annotation attributes:

- *Display*
- *DisplayName*
- *DataType*
- *MaxLength*
- *MinLength*
- *Range*
- *RegularExpression*
- *Required*
- *StringLength*

Enabling data annotation localization

To enable localization of data annotations, you only need to set up your MVC application according to the instructions in Starting with MVC development.

You can verify that the feature is enabled in the MVC application's **ApplicationConfig.cs** file:

```
using Kentico.Web.Mvc;

...

public static void RegisterFeatures(ApplicationBuilder builder)
{
        ...
    builder.UseDataAnnotationsLocalization();
        ...
}
```

If the feature is enabled, the validation results and display names of model properties are localized using Kentico localization services.

```
public class MessageModel
{
...
         [Required(ErrorMessage = "General.RequiresMessage")]
     [Display(Name = "General.Message")]
     [DataType(DataType.MultilineText)]
     [MaxLength(500, ErrorMessage = "General.MaxlengthExceeded")]
     public string MessageText
     {
         get;
         set;
     }
...
}
```

Note: The individual error messages are processed by the *System.String.Format* method and support composite formatting. That is, the strings themselves can contain format items that are specific to each validation attribute and represent their parameters. For example, the *minimum length* for the *MinLenghtAttribute* or the *minimum* and *maximum* for the *RangeAttribute*.