Articles » Web Development » ASP.NET » General

Route-friendly localization of ASP.NET MVC 5

DmitriyArh88

27 Apr 2016 CPOL

Rate this: ★★★★★ 4.78 (12 votes)

Creating of user-friendly configurable internationalization mechanism in ASP.NET MVC application

Download RoutedLocalizationExample.zip - 480.5 KB

Table of contents

Introduction

Localization is the common task for all the services created for multilingual usage.

There are a lot of well-known approaches to localize .NET applications, including ASP.NET MVC web applications.

But in this article I want to show how to create localization system, that will have next key features:

- routing system with support of locale;
- correct caching work for localised views;
- change current site locale from interface;
- if locale is not defined in route - then automatically get locale from user request context.

Background

Base of localisation in ASP.NET MVC is using resources file.

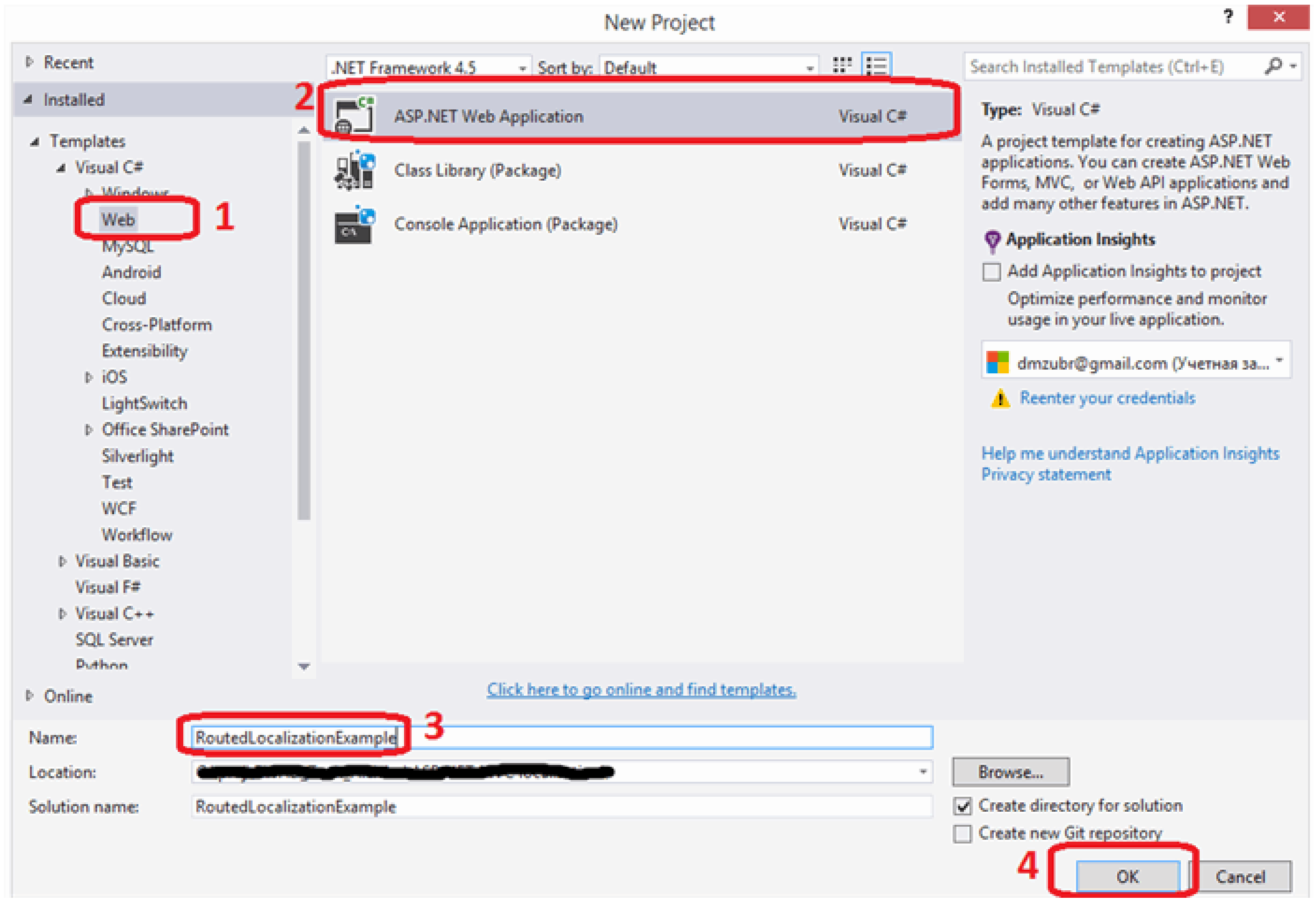
What is resources file and how to define the content of these files is described well in this Codeproject article.

Mechaninsm used to pass user localization preferences in primary source can be found on W3C site section.

Using the code

Create solution

First, let's create from scratch ASP.NET MVC Web 5 aplication +in Visual Studio (2015 Community Edition - in my case).



ASP.NET Web Application Visual C#
Class Library (Package) Visual C#
Console Application (Package) Visual C#

Type: Visual C#

A project template for creating ASP.NET applications. You can create ASP.NET Web Forms, MVC, or Web API applications and add many other features in ASP.NET.

Application Insights

☐ Add Application Insights to project

Optimize performance and monitor usage in your live application.

dmzubr@gmail.com (Учетная за...

Reenter your credentials

Help me understand Application Insights

Privacy statement

Click here to go online and find templates.

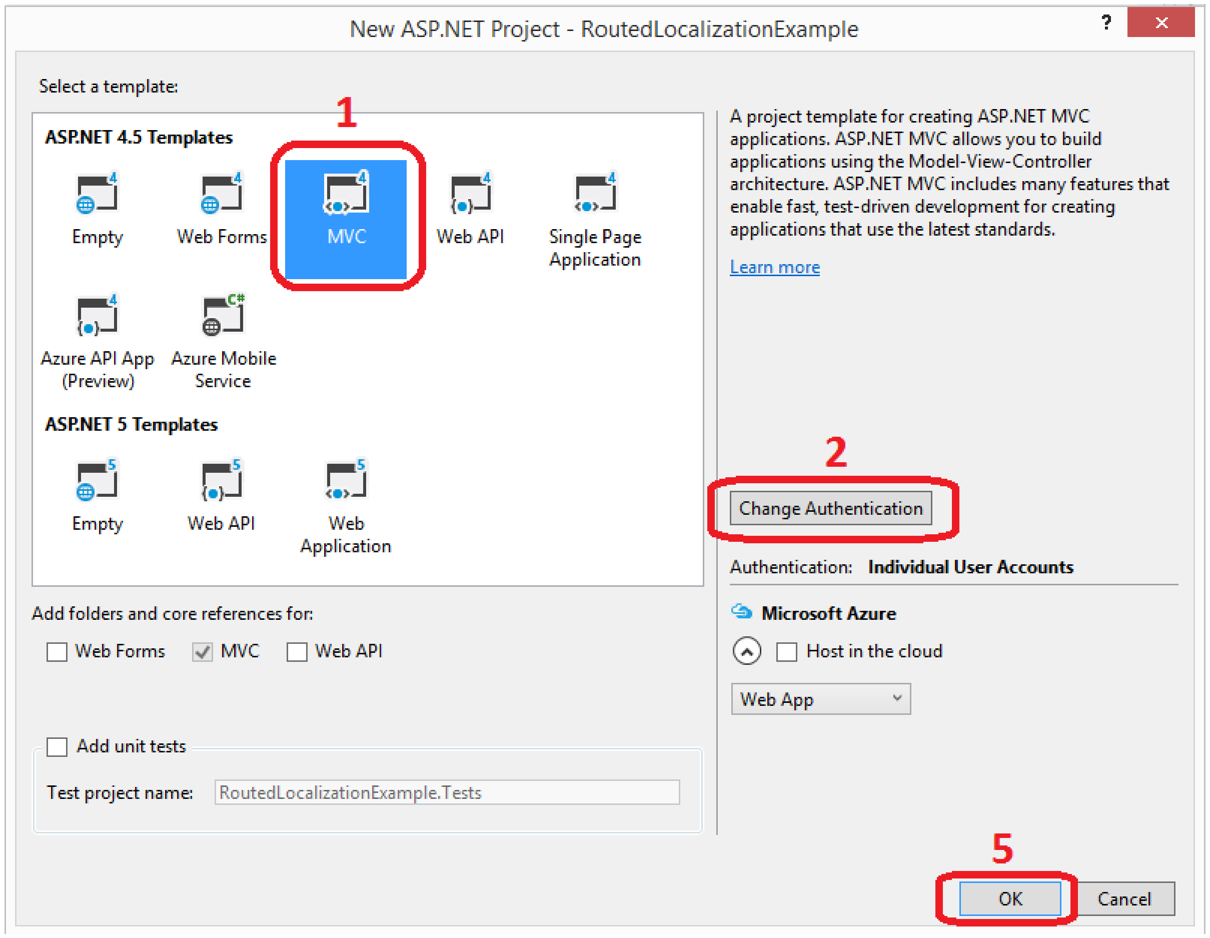
Name: RoutedLocalizationExample

Location: [redacted]

Solution name: RoutedLocalizationExample

☑ Create directory for solution

☐ Create new Git repository


New ASP.NET Project - RoutedLocalizationExample

Select a template:

ASP.NET 4.5 Templates: Empty, Web Forms, [✓ MVC], Web API, Single Page Application, Azure API App (Preview), Azure Mobile Service

ASP.NET 5 Templates: Empty, Web API, Web Application

A project template for creating ASP.NET MVC applications. ASP.NET MVC allows you to build applications using the Model-View-Controller architecture. ASP.NET MVC includes many features that enable fast, test-driven development for creating applications that use the latest standards.

Learn more

Change Authentication

Authentication: **Individual User Accounts**

Microsoft Azure

☐ Host in the cloud

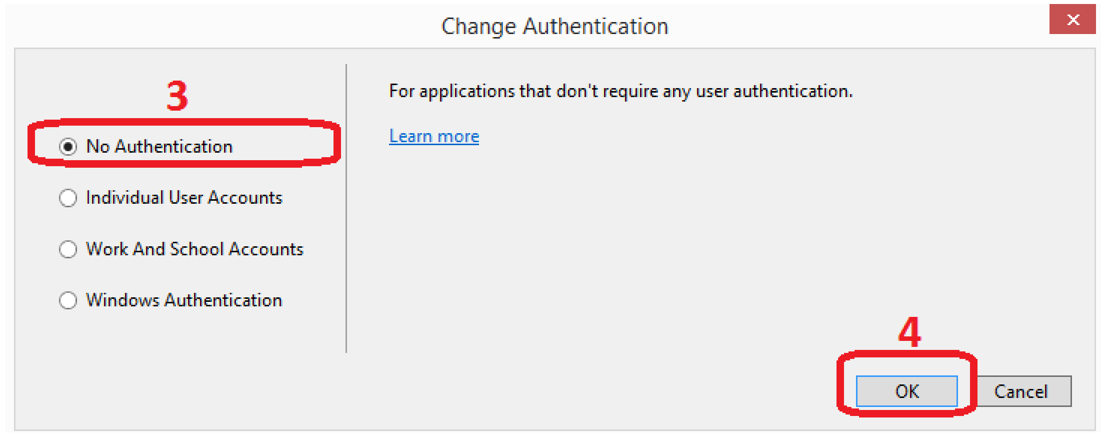
Web App

Add folders and core references for:

☐ Web Forms ☑ MVC ☐ Web API

☐ Add unit tests

Test project name: RoutedLocalizationExample.Tests


Change Authentication

◉ No Authentication

○ Individual User Accounts

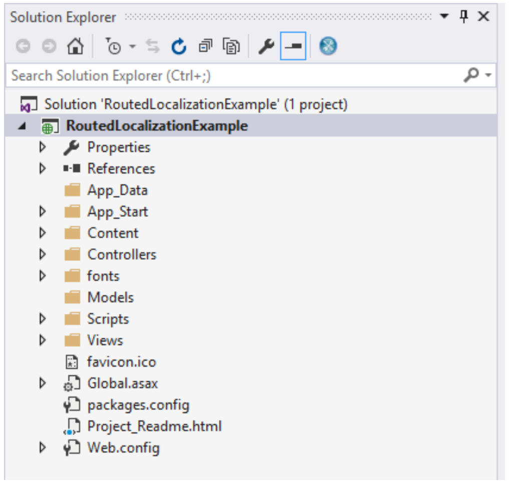
○ Work And School Accounts

○ Windows Authentication

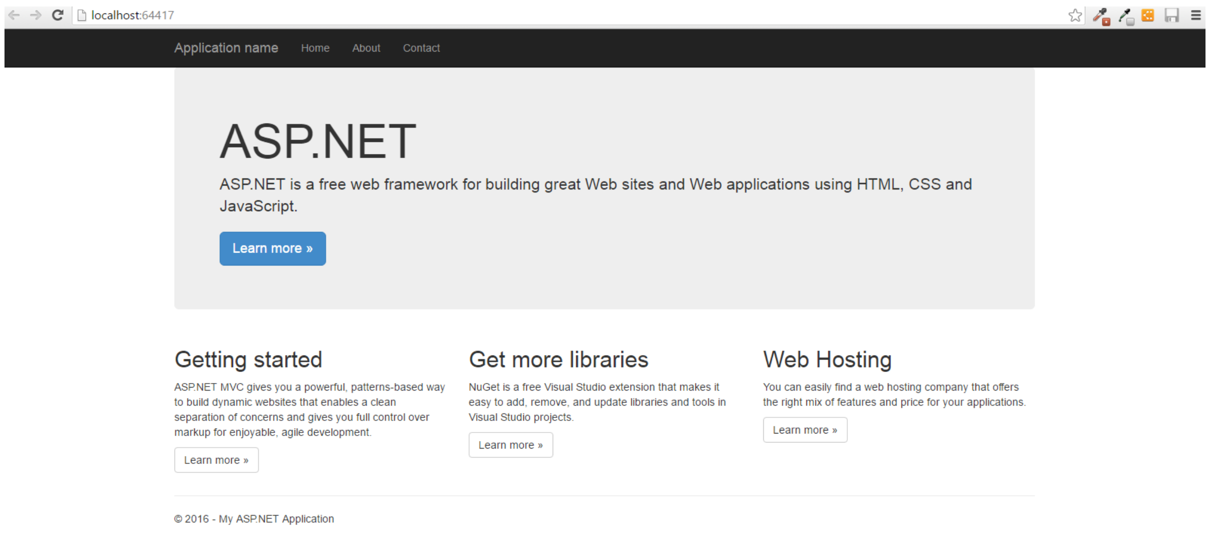
For applications that don't require any user authentication.

Learn more

So, after solution initialization we'll have structure like shown below.



Let's try to run project in Google Chrome. And we'll get standard ASP.NET MVC initial screen.



ASP.NET is a free web framework for building great Web sites and Web applications using HTML, CSS and JavaScript.

ASP.NET MVC gives you a powerful, patterns-based way to build dynamic websites that enables a clean separation of concerns and gives you full control over markup for enjoyable, agile development.

NuGet is a free Visual Studio extension that makes it easy to add, remove, and update libraries and tools in Visual Studio projects.

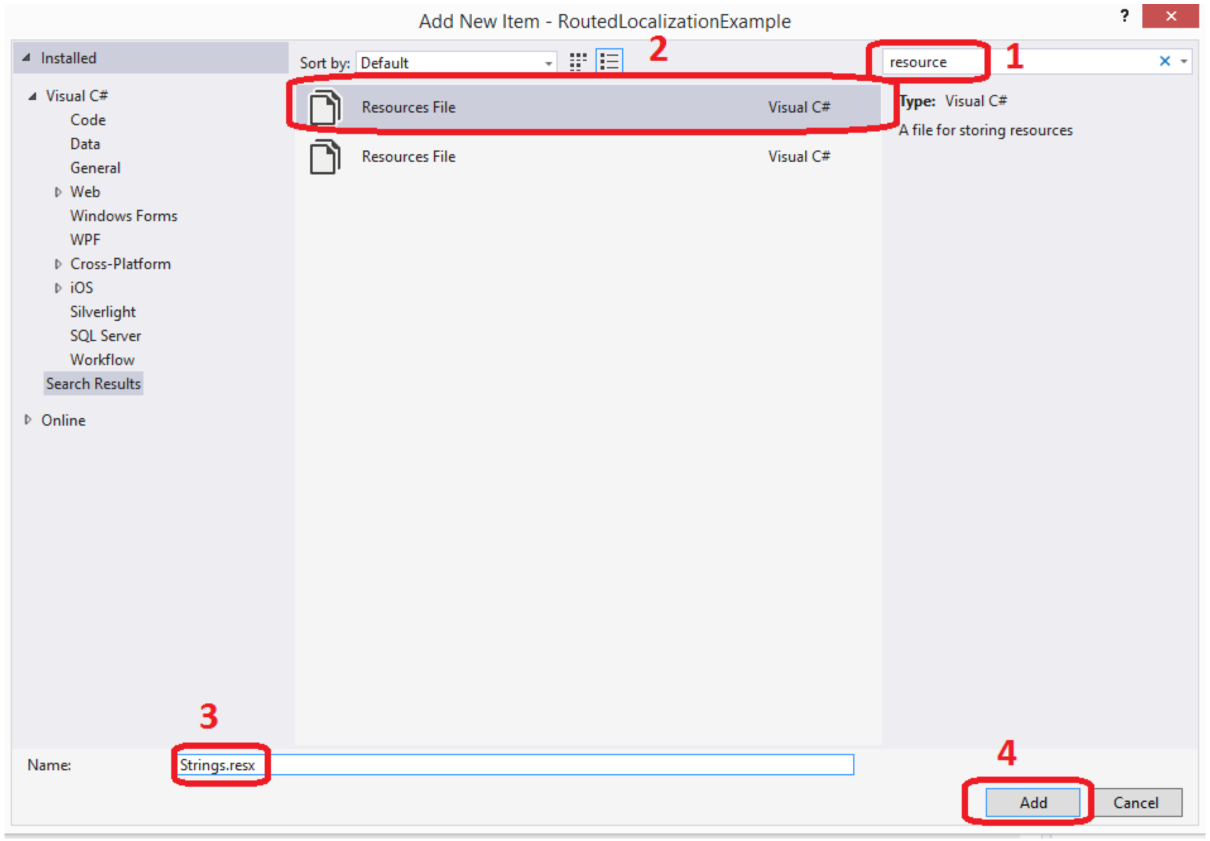
You can easily find a web hosting company that offers the right mix of features and price for your applications.

Add localization resources and get first localized result

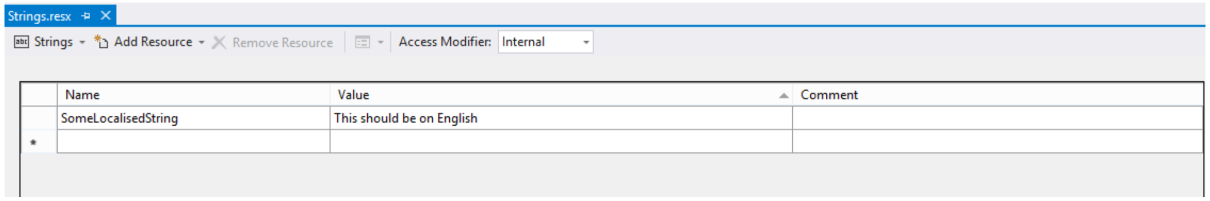
We'll use an approach to localize web applicaion using .NET resources files.

Details of creating the .NET resources files can be found in this MSDN article.

First, add the resources files.

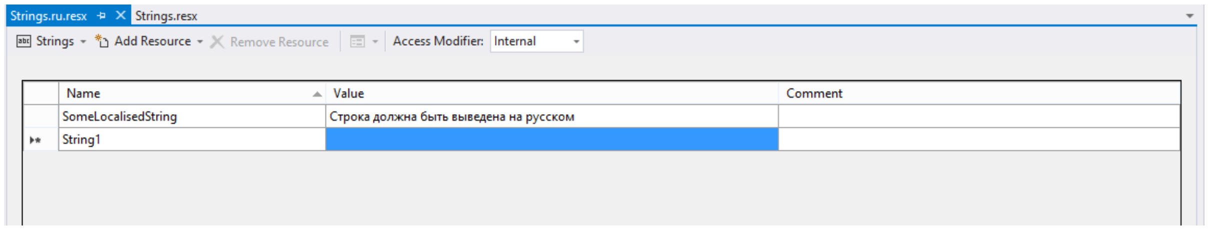


Then, add sample string that will be passed to the view layer.



Name	Value	Comment
SomeLocalisedString	This should be on English	

And add string to *Strings.ru.resx* file with identical key.



Name	Value	Comment
SomeLocalisedString	Строка должна быть выведена на русском	
String1		

Then, let's create method in *Home* controller to test localization behaviour.

Clean all existing methods in file *HomeController.cs*.

Then, add ViewModel class for our experiments.

Hide Copy Code

```
using System;

namespace RoutedLocalizationExample.ViewModels
{
    public class FullViewModel
    {
        public FullViewModel()
        {
            CreationDateTime = DateTime.Now;
        }

        /// <summary>
        /// This will contain localised string value
        /// </summary>
        public string LocalisedString { get; set; }

        /// <summary>
        /// For see difference of cretion time
        /// </summary>
        public DateTime CreationDateTime { get; set; }
    }
}
```

And our simple controller method.

Hide Copy Code

```
        // Localize string without any external impact
        public ActionResult Index()
        {
            // Get string from strongly typed localzation resources
            var vm = new FullViewModel { LocalisedString = Strings.SomeLocalisedString };
            return View(vm);
        }
```

Then, our *Index.cshtml* view should like this.

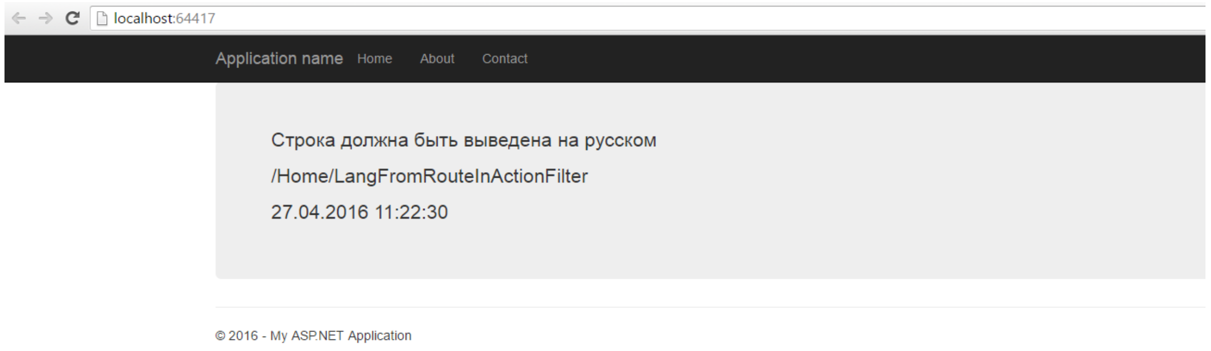
Hide Copy Code

```
@model RoutedLocalizationExample.ViewModels.FullViewModel

@{
    ViewBag.Title = "Home Page";
}

<div class="jumbotron">
    <p>@(Model.LocalisedString)</p>
    <p>@Url.Action("LangFromRouteInActionFilter", "Home")</p>
    <p>@Model.CreationDateTime.ToString()</p>
</div>
```

What do we have here?

- **Model.LocalisedString** is to show localized content;
- **Url.Action("LangFromRouteInActionFilter", "Home")** is to show how url are builded. We will need it to show some features of routing mechanism;
- **Model.CreationDateTime.ToString()** - we will need to know the time when a page was rendered.

Finally, let's try to get result in web browser (Google Chrome in my case).



Строка должна быть выведена на русском

/Home/LangFromRouteInActionFilter

27.04.2016 11:22:30

© 2016 - My ASP.NET Application

So, here we have localised string. How is target locale defined? Magic takes place in backgound. Prefered locale is catched from HTTP header "Accept-Language" and is applied to view rendering result.

So, in this case we have no explicit control over used locale, except of languages list in browser settings. But, like the User, I want to have option to define language with less efforts.

Get language from request query string

We can define used localization "Dictionary" (appropriate resources file) by using some .NET capabilities:

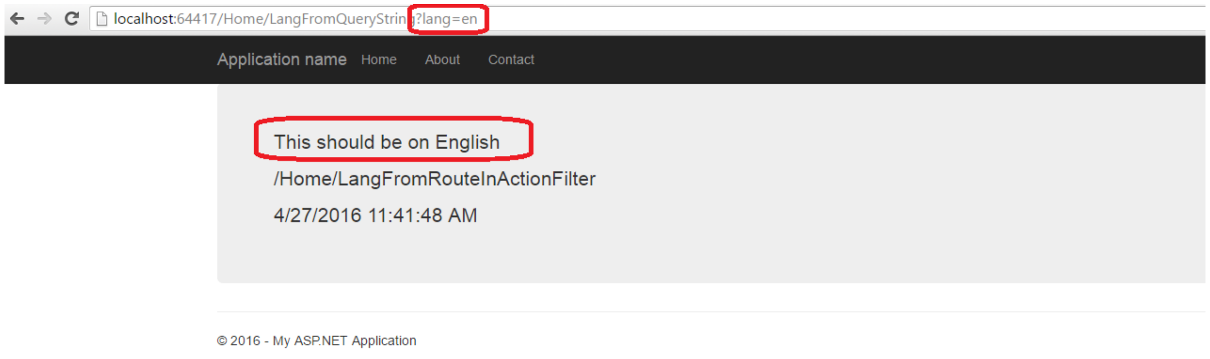
1. System.Threading.Thread.CurrentThread.CurrentCulture - defines culture used in current thread;
2. System.Threading.Thread.CurrentThread.CurrentUICulture - defines UI culture used in current thread.

So, let's try to take control over localization result with help of the mentioned above features. Add new method to *HomeController*.

Hide Copy Code

```
        // Get language from query string (by binder)
        public ActionResult LangFromQueryString(string lang)
        {
            Thread.CurrentThread.CurrentCulture = CultureInfo.GetCultureInfo(lang);
            Thread.CurrentThread.CurrentUICulture = CultureInfo.GetCultureInfo(lang);

            var vm = new FullViewModel { LocalisedString = Strings.SomeLocalisedString };
            return View("Index", vm);
        }
```

Then, try to use it. Please, pay attention to the query string of requested URL .

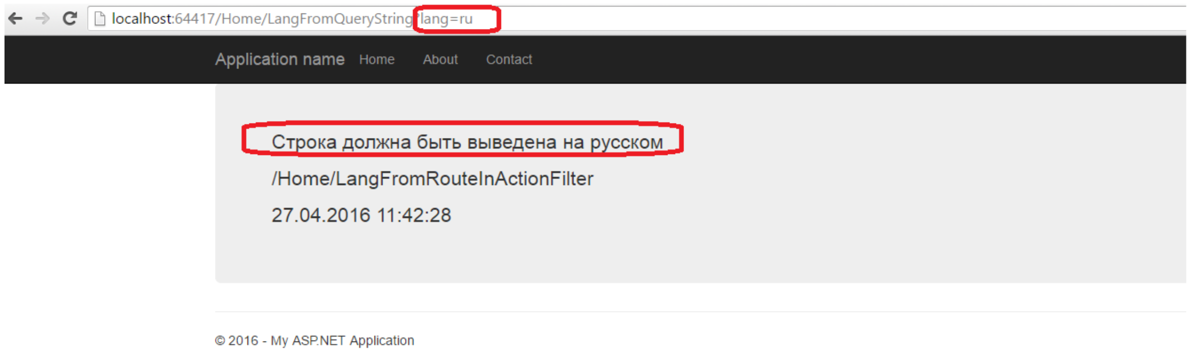

localhost:64417/Home/LangFromQueryString?lang=en

This should be on English

/Home/LangFromRouteInActionFilter

4/27/2016 11:41:48 AM

© 2016 - My ASP.NET Application


localhost:64417/Home/LangFromQueryString?lang=ru

Строка должна быть выведена на русском

/Home/LangFromRouteInActionFilter

27.04.2016 11:42:28

© 2016 - My ASP.NET Application

Excellent! Now we have control of locale by query parameter.

But... Is it user friendly? With it's qestion mark and parameter definition... I think that for the average user - ***it doesn't seems friendly enough.***

Define localization routing rule

ASP.NET MVC have the great out-of-box feature for building informative and nice looking URLs. It's the routing mechanism. So, we definetly need to use it for our localization purposes.

Let's add new routing rule in *App_Start/RouteConfig.cs* file (to void method *RegisterRoutes*).

Hide Shrink Copy Code

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

namespace RoutedLocalizationExample
{
    public class RouteConfig
    {
        public static void RegisterRoutes(RouteCollection routes)
        {
            routes.IgnoreRoute("{resource}.axd/{*pathInfo}");

            // Localization route - it will be used as a route of the first priority
            routes.MapRoute(
                name: "DefaultLocalized",
                url: "{lang}/{controller}/{action}/{id}",
                defaults: new
                {
                    controller = "Home",
                    action = "Index",
                    id = UrlParameter.Optional,
                    lang = "en"
                });

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

Then, add *Home* controller method that will catch *lang* route parameter.

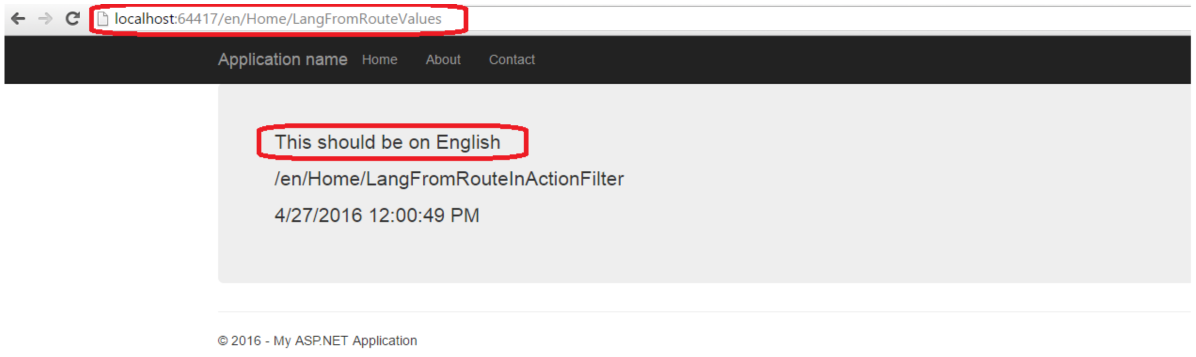
Hide Copy Code

```
        // Get language as a parameter from route data
        public ActionResult LangFromRouteValues(string lang)
        {
            Thread.CurrentThread.CurrentCulture = CultureInfo.GetCultureInfo(lang);
            Thread.CurrentThread.CurrentUICulture = CultureInfo.GetCultureInfo(lang);

            var vm = new FullViewModel { LocalisedString = Strings.SomeLocalisedString };
            return View("Index", vm);
        }
```

It does not differ from method "*LangFromQueryString*" declared previosly. Why is it so?

Because of result of binders working. It's the out-of-box magic. We don't need to care how to catch parameter value - is it should to be catched from query string or from RouteTable values?

Finally, let's see new controller method in action.

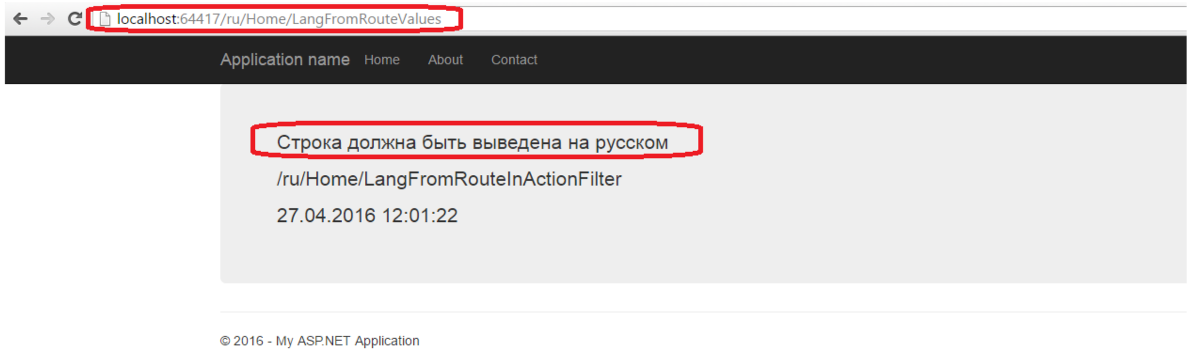


This should be on English

/en/Home/LangFromRouteInActionFilter

4/27/2016 12:00:49 PM

© 2016 - My ASP.NET Application



Строка должна быть выведена на русском

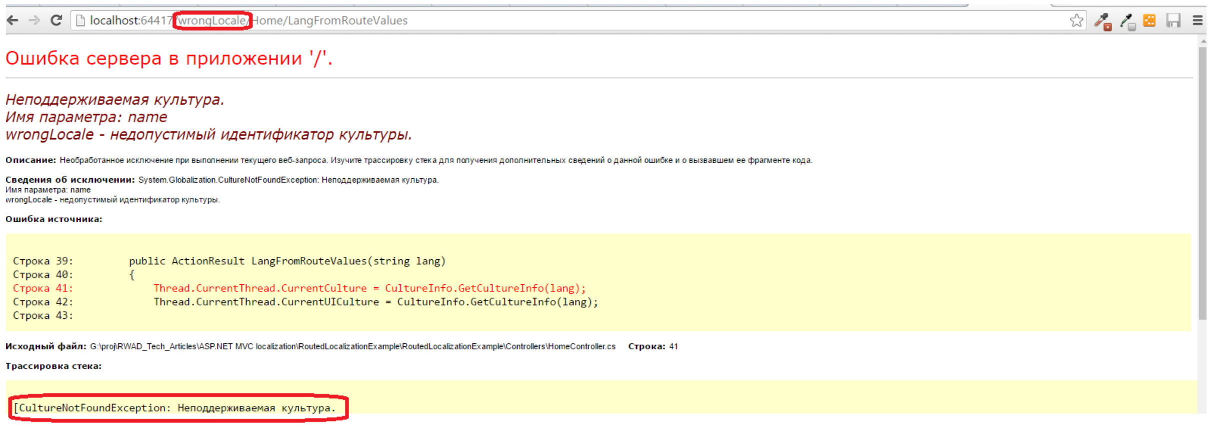
/ru/Home/LangFromRouteInActionFilter

27.04.2016 12:01:22

© 2016 - My ASP.NET Application

I think that it's obviously better than "*Controller/Method?lang=ru*". But...

What will occur if user try to use some locale for which we don't have appropriate resources file in our project?


localhost:64417/wrongLocale/Home/LangFromRouteValues

Ошибка сервера в приложении '/'.

Неподдерживаемая культура.
Имя параметра: name
wrongLocale - недопустимый идентификатор культуры.

Описание: Необработанное исключение при выполнении текущего веб-запроса. Изучите трассировку стека для получения дополнительных сведений о данной ошибке и о вызвавшем ее фрагменте кода.

Сведения об исключении: System.Globalization.CultureNotFoundException: Неподдерживаемая культура.
Имя параметра: name
wrongLocale - недопустимый идентификатор культуры.

Ошибка источника:

```
Строка 39:         public ActionResult LangFromRouteValues(string lang)
Строка 40:         {
Строка 41:             Thread.CurrentThread.CurrentCulture = CultureInfo.GetCultureInfo(lang);
Строка 42:             Thread.CurrentThread.CurrentUICulture = CultureInfo.GetCultureInfo(lang);
Строка 43:
```

Исходный файл: G:\proj\RWAD_Tech_Articles\ASP.NET MVC localization\RoutedLocalizationExample\RoutedLocalizationExample\Controllers\HomeController.cs **Строка:** 41

Трассировка стека:

```
[CultureNotFoundException: Неподдерживаемая культура.
```

We will see predictable ***CultureNotFoundException*** exception.

We'll try to resolve this problem a little later in this article.

Localisation logic in action filter

Using localization logic will be the common task for the majority of controllers methods, of course.

So, we definetly need to follow DRY principle when perform localization task. And, again, platform has a feature that will help us. It's ***ActionFilterAttribute*** class. You can get more information about action filters in this article.

Let's create catalogue "ActionFilters" and code our ***InternationalizationAttribute***.

Hide Shrink ▲ Copy Code

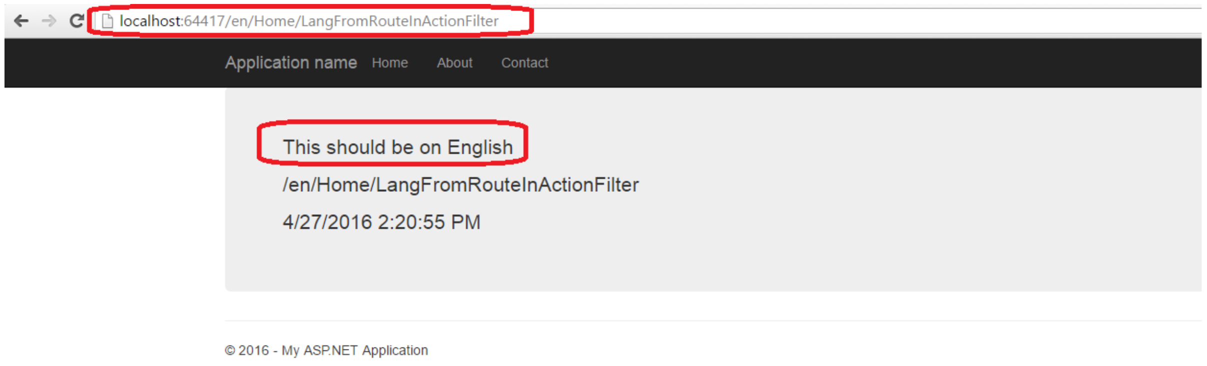
```
using System.Web.Mvc;
using System.Collections.Generic;
using System.Globalization;
using System.Threading;

namespace RoutedLocalizationExample.ActionFilters
{
    /// <summary>
    /// Set language that is defined in route parameter "lang"
    /// </summary>
    public class InternationalizationAttribute : ActionFilterAttribute
    {
        private readonly IList<string> _supportedLocales;
        private readonly string _defaultLang;

        public InternationalizationAttribute()
        {
            // Get supported locales list
            _supportedLocales = Utils.LocalizationHelper.GetSupportedLocales();

            // Set default locale
            _defaultLang = _supportedLocales[0];
        }

        /// <summary>
        /// Apply locale to current thread
        /// </summary>
        /// <param name="lang">locale name</param>
        private void SetLang(string lang)
        {
            Thread.CurrentThread.CurrentCulture = CultureInfo.GetCultureInfo(lang);
            Thread.CurrentThread.CurrentUICulture = CultureInfo.GetCultureInfo(lang);
        }

        public override void OnActionExecuting(ActionExecutingContext filterContext)
        {
            // Get locale from route values
            string lang = (string)filterContext.RouteData.Values["lang"] ?? _defaultLang;

            // If we haven't found appropriate culture - seet default locale then
            if (!_supportedLocales.Contains(lang))
                lang = _defaultLang;

            SetLang(lang);
        }
    }
}
```

Then, we need to add controller method decorated with created action filter.

Hide Copy Code

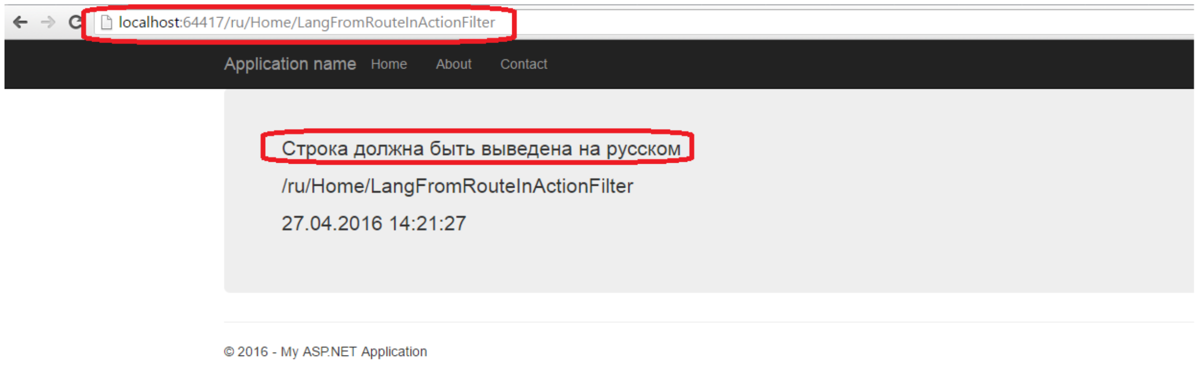
```
        // Get language in action filter (from route parameter)
        [Internationalization]
        public ActionResult LangFromRouteInActionFilter()
        {
            var vm = new FullViewModel { LocalisedString = Strings.SomeLocalisedString };
            return View("Index", vm);
        }
```

So, let's try to test it. Start in Chrome and see results.


localhost:64417/en/Home/LangFromRouteInActionFilter

This should be on English
/en/Home/LangFromRouteInActionFilter
4/27/2016 2:20:55 PM

© 2016 - My ASP.NET Application

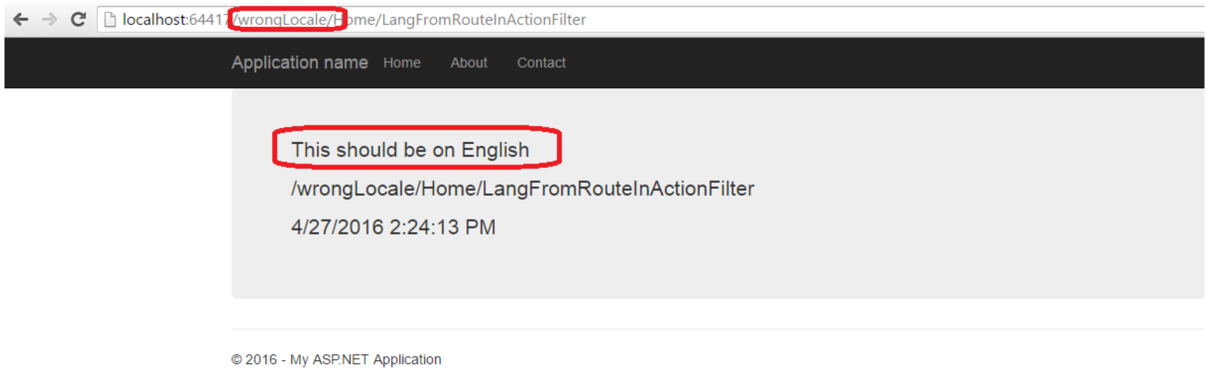

localhost:64417/ru/Home/LangFromRouteInActionFilter

Строка должна быть выведена на русском
/ru/Home/LangFromRouteInActionFilter
27.04.2016 14:21:27

© 2016 - My ASP.NET Application

Result is what we've expected.

What about handling situations with wrong locales?


localhost:64417/wrongLocale/Home/LangFromRouteInActionFilter

This should be on English
/wrongLocale/Home/LangFromRouteInActionFilter
4/27/2016 2:24:13 PM

© 2016 - My ASP.NET Application

Result is better, then screen with fatal error, of course. We see the effect of using default locale ("*en*") in action filter.

But it'll be better to return supported language in route.

Handle situations when locale is not defined in url

Let's assume, that we have some ASP.NET MVC site that initially hasn't support of localizaion. Site is in production environment already. It has many visitros and a lot hyperlinks in Internet follow to this site. This site has only english version.

Once, site's owner have decided to add localization feature.

So, there may be cases when user will request url of site without any *lang* parameter value. And we have to handle that requests correctly. Fatal error or 404 page shouldn't be the predictable case for system in production.

The problem is that we can't handle the situation described above in any action filter, because action filters methods are invoked in context of explicit controller, when target controller was defined according to requested url. But it does not in case when we have changed routing mechanism.

So - sutiation seems like a deadlock for the first time.

But... Actually we can control request handling logic before ASP.NET MVC core will decide whether appropriate controller exists ot not. We can do this in HTTPModules.

Let's try to create such a module by implementing interface *System.Web.IHttpModule*.

Hide Shrink Copy Code

```
using System;
using System.Reflection;
using System.Linq;
using System.Web;
using System.Web.Mvc;
using System.Collections.Generic;
using RoutedLocalizationExample.Utils;

namespace RoutedLocalizationExample.HttpModules
{
    /// <summary>
    /// Module to append lang parameter to the requested url if it's absent or unsupported
    /// </summary>
    public class LangQueryAppenderModule : IHttpModule
    {
        /// <summary>
        /// List of supported locales
        /// </summary>
        private readonly IList<string> _supportedLocales;

        /// <summary>
        /// We need to have controllers list to correctly handle situations
        /// when target method name is missed
        /// </summary>
        private readonly IList<string> _controllersNamesList;

        public LangQueryAppenderModule()
        {
            // Get list of supported locales
            _supportedLocales = Utils.LocalizationHelper.GetSupportedLocales();

            // Get controllers list of current project by reflection
            var asmPath = HttpContext.Current.Server.MapPath("~/bin/RoutedLocalizationExample.dll");
            Assembly asm = Assembly.LoadFile(asmPath);

            var controllerTypes = asm.GetTypes()
                .Where(type => typeof(Controller).IsAssignableFrom(type));
            _controllersNamesList = new List<string>();

            foreach (var controllerType in controllerTypes)
            {
                var fullName = controllerType.Name;

                // We need only name part of Controller class that is used in route
                _controllersNamesList.Add(fullName.Substring(0, fullName.Length - "Controller".Length));
            }
        }

        // In the Init function, register for HttpApplication
        // events by adding your handlers.
        public void Init(HttpApplication application)
        {
            application.BeginRequest += (new EventHandler(this.Application_BeginRequest));
        }

        private void Application_BeginRequest(Object source, EventArgs e)
        {
            try
            {
                HttpApplication app = (HttpApplication)source;
                HttpContext ctx = app.Context;

                // We will redirect to url with defined locale only in case for HTTP GET verb
                // cause we assume that all requests with other verbs will be called from site directly
                // where all the urls created with URLHelper, so it complies with routing rules and will contain "lang" parameter
                if (string.Equals(ctx.Request.HttpMethod, "GET", StringComparison.OrdinalIgnoreCase))
                {
                    var localisedUri = LocalizationHelper.GetLocalisedUrl(ctx.Request.Url, _controllersNamesList,
ctx.Request.UserLanguages);
                    if (!string.IsNullOrEmpty(localisedUri))
                        // Perform redirect action to changed url if it exists
                        ctx.Response.Redirect(localisedUri);
                }
            }
            catch (Exception)
            {
                // Some logging logic could be here
            }
        }

        public void Dispose() { }
    }
}
```

Logic of getting changed target url is in method "*GetLocalisedUrl*" of static class *LocalizationHelper*.

Hide Shrink Copy Code

```
        /// <summary>
        /// Get request url corrected according to logic of routing with locale
        /// </summary>
        /// <param name="initialUri"></param>
        /// <param name="controllersNames"></param>
        /// <param name="userLangs"></param>
        /// <returns></returns>
        public static string GetLocalisedUrl(Uri initialUri, IList<string> controllersNames, IList<string> userLangs)
        {
            var res = string.Empty;

            var supportedLocales = GetSupportedLocales();

            var origUrl = initialUri;

            // Dicide requested url to parts
            var cleanedSegments = origUrl.Segments.Select(X => X.Replace("/", "")).ToList();

            // Check is already supported locale defined in route
            // cleanedSegments[0] is empty string, so lang parameter will be in [1] url segment
            var isLocaleDefined = cleanedSegments.Count > 1 && supportedLocales.Contains(cleanedSegments[1]);

            // does request need to be changed
            var isRequestPathToHandle =
                // Url has controller's name part
                (cleanedSegments.Count > 1 && cleanedSegments.Intersect(controllersNames).Count() > 0) ||
                // This condition is for default (initial) route
                (cleanedSegments.Count == 1) ||
                // initial route with lang parameter that is not supported -> need to change it
                (cleanedSegments.Count == 2 && !supportedLocales.Contains(cleanedSegments[1]));

            if (!isLocaleDefined && isRequestPathToHandle)
            {
                var langVal = "";
                // Get user preffered language from Accept-Language header
                if (userLangs != null && userLangs.Count > 0)
                {
                    // For our locale name approach we'll take only first part of lang-locale definition
                    var splitted = userLangs[0].Split(new char[] { '-' });
                    langVal = splitted[0];
                }

                // If we don't support requested language - then redirect to requested page with default language
                if (!supportedLocales.Contains(langVal))
                    langVal = supportedLocales[0];

                var normalisedPathAndQuery = origUrl.PathAndQuery;
                if ((cleanedSegments.Count > 2 &&
                    !controllersNames.Contains(cleanedSegments[1]) &&
                    controllersNames.Contains(cleanedSegments[2])) ||
                    (cleanedSegments.Count == 2) && (!controllersNames.Contains(cleanedSegments[1])))
                {
                    // Second segment contains lang parameter, third segment contains controller name
                    cleanedSegments.RemoveAt(1);

                    // Remove wrong locale name from initial Uri
                    normalisedPathAndQuery = string.Join("/", cleanedSegments) + origUrl.Query;
                }

                // Finally, create new uri with language loocale
                res = string.Format("{0}://{1}:{2}/{3}{4}", origUrl.Scheme, origUrl.Host, origUrl.Port, langVal.ToLower(),
normalisedPathAndQuery);
            }

            return res;
        }
```

Comments are presented in code.

This method seems to be not trivial, so I've added the project with unit tests to cover all the possible situations. Not all of course :), but at least - situations that has a place in practice.

Now, we need to activate our module by editing *web.config* file like shown below.

Hide Copy Code

```
  <system.webServer>
    <modules>
        <add name="LangQueryAppenderModule" type="RoutedLocalizationExample.HttpModules.LangQueryAppenderModule" />
    </modules>
  </system.webServer>
```

Finally, let's try our new localization logic.

Now, for initial route "*ru*" value of *lang* parameter is added by our module. OK.

Further, let's try a little more interesting case. Simulate situation of requesting some page from extarnal link with no *lang* parameter. Create new browaser tab with url "*http://localhost:64417/Home/LangFromRouteInActionFilter*".

And we will receive redirect to "http://localhost:64417/ru/Home/LangFromRouteInActionFilter".

Exactly what we need!

If your prefered browser language is English, then you will be redirected to english variant of page.

What about not supported locale in initial url?

Requested url "*http://localhost:64417/qwerty/Home/LangFromRouteInActionFilter*" will be redirected to "*http://localhost:64417/ru/Home/LangFromRouteInActionFilter*" again.

And finally. We don't want to not web-browsers clients will receive 404 error. So, let's simulate case when request doesn't provide value of "*Accept-Language*" value. We'll use Fiddler (more info about application can be found here) for this purpose.

Pay attention to response chain - first - code 302 (redirection). Then - english version of requested page. Why English? Because in created request we haven't provide "*Accept-Language*" header. So, our *LangQueryAppenderModule* inserts value of project's default locale ("*en*").

Change current site locale from interface

Supporting of localization in routing mechanism is good. But I think that user will prefer to change it in graphical interface. So, let's add language selector into the main menu of site.

Let's add file *HeaderPartial.cshtml* to *Views/Shared* catalogue.

Hide Shrink ▲ Copy Code

```
@helper langSelector() {
    string curLang = "en"; ;
    if (this.ViewContext.RouteData.Values["lang"] != null)
    {
        curLang = this.ViewContext.RouteData.Values["lang"].ToString();
    }

    var enabledLangsList = RoutedLocalizationExample.Utils.LocalizationHelper.GetSupportedLocales();

    var targetPath = string.Format("/{0}/{1}{2}",
        ViewContext.RouteData.Values["controller"].ToString(),
        ViewContext.RouteData.Values["action"].ToString(),
        Request.Url.Query);
    var hostRoot = string.Format("{0}://{1}", Request.Url.Scheme, Request.Url.Authority);
    var targetUrlMask = string.Format("{0}/{{0}}/{1}", hostRoot, targetPath);

    <li class="dropdown special" style="margin-left: 15px;">
        <a href="#" class="dropdown-toggle" data-toggle="dropdown" role="button" aria-haspopup="true" aria-expanded="false">
            @(curLang)
            <span class="caret"></span>
        </a>
        <ul class="dropdown-menu lang-selector">
            @foreach (var lang in enabledLangsList)
            {
                <li><a href="@(string.Format(targetUrlMask, lang))">@lang</a></li>
            }
        </ul>
    </li>
}

<div class="navbar navbar-inverse navbar-fixed-top">
    <div class="container">
        <div class="navbar-header">
            <button type="button" class="navbar-toggle" data-toggle="collapse" data-target=".navbar-collapse">
                <span class="icon-bar"></span>
                <span class="icon-bar"></span>
                <span class="icon-bar"></span>
            </button>
            @Html.ActionLink("Application name", "Index", "Home", new { area = "" }, new { @class = "navbar-brand" })
        </div>
        <div class="navbar-collapse collapse">
            <ul class="nav navbar-nav navbar-left">
                <li>@Html.ActionLink("Home", "Index", "Home")</li>
                <li>@Html.ActionLink("About", "About", "Home")</li>
                <li>@Html.ActionLink("Contact", "Contact", "Home")</li>
            </ul>
            <ul class="nav navbar-nav navbar-right" style="margin-right: -5px;">
                @langSelector()
            </ul>
        </div>
    </div>
</div>
```

We've added a helper that will contain all the language selector logic and visual elements.

And, finally, try to change language from site menu.

Localization and caching - using together

Another thing to note when you are working with ASP.NET MVC project localization is that you have to be very carefully when using caching mechanism, including ***OutputCacheAttribute***. The reason is that your localization logic should comply to limitations and logic of caching.

Most common negative scenario is when the client will receive cached page that will not match to requested locale. This problem is resolved in localization logic described above in this article.

But you can try to create conditions to reproduce it. In the case of out-of-box ASP.NET MVC caching configuration key condition to reproduce bad result is that pages for different locales will be requested by the same url. Then, site will return view with locale, that was called at first.

When we've added "*lang*" route parameter - we've gone around this problem.

You can see working example of localization approach described above here for example.

Points of Interest

One of the most interesting part of localizaton scheme described in this article is to imagine all the combinations of url segments for all possible real life scenarios.

For example, correct handling of requests with no "*Accept-Language*" header provided. I've found these requests only during analysis of application's and IIS's logs. Because it received 404 response. :)

This situation once again have proved the importance and benefits of using unit tests in your projects...

History

2016-04-27 - initial state.

License

Share

About the Author

DmitriyArh88

Software Developer Bank of Russia
Russian Federation

No Biography provided

Comments and Discussions

You must Sign In to use this message board.

Search Comments

Spacing Relaxed ▼ Layout Normal ▼ Per page 25 ▼ Update

First Prev Next

How to ignore some route from this localization?	**Yorlen Guirado**	**24-Jul-19 4:36**
My vote of 5	**xuanhoang_itdlu**	**29-Aug-17 3:11**
Little misredirection	**DJmRek**	**24-Aug-17 0:05**
I thought it was on: "user-friendly localized routes in ASP.NET"	**Red Feet**	**29-Apr-16 0:06**

Re: I thought it was on: "user-friendly localized routes in ASP.NET"	DmitriyArh88	29-Apr-16 0:23
Re: I thought it was on: "user-friendly localized routes in ASP.NET"	Red Feet	29-Apr-16 0:25
Re: I thought it was on: "user-friendly localized routes in ASP.NET"	Member 12198480	2-May-16 23:57

Last Visit: 10-Mar-20 2:16 Last Update: 10-Mar-20 3:32 Refresh 1

General News Suggestion Question Bug Answer Joke Praise Rant Admin

Use Ctrl+Left/Right to switch messages, Ctrl+Up/Down to switch threads, Ctrl+Shift+Left/Right to switch pages.