Best way to implement multi-language/globalization in large .NET project

Asked 12 years, 9 months ago Active 24 days ago Viewed 109k times

90

I'll soon be working on a large c# project and would like to build in multi-language support from the start. I've had a play around and can get it working using a separate resource file for each language, then use a resource manager to load up the strings.

Are there any other good approaches that I could look into?

64

c# .net localization multilingual

Share Edit Follow

edited Jul 21 '09 at 19:27
Rich Seller
80.3k 22 168 174

asked Dec 17 '08 at 1:23

tjjohnson
2,862 3 28 33

11 Answers

Active | Oldest | Votes

105

Use a separate project with Resources

I can tell this from out experience, having a current solution with ~~12~~ **24** projects that includes API, MVC, Project Libraries (Core functionalities), WPF, UWP and Xamarin. It is worth reading this long post as I think it is the best way to do so. With the help of VS tools easily exportable and importable to sent to translation agencies or review by other people.

EDIT 02/2018: Still going strong, converting it to a .NET Standard library makes it possible to even use it across .NET Framework and NET Core. I added an extra section for converting it to JSON so for example angular can use it.

EDIT 2019: Going forward with Xamarin, this still works across all platforms. E.g. Xamarin.Forms advices to use resx files as well. (I did not develop an app in Xamarin.Forms yet, but the documentation, that is way to detailed to just get started, covers it: Xamarin.Forms Documentation). Just like converting it to JSON we can also convert it to a .xml file for Xamarin.Android.

EDIT 2019 (2): While upgrading to UWP from WPF, I encountered that in UWP they prefer to use another filetype `.resw`, which is is in terms of content identical but the usage is different. I found a different way of doing this which, in my opinion, works better then the default solution.

EDIT 2020: Updated some suggestions for larger (modulair) projects that might require multiple language projects.

So, lets get to it.

Pro's

- Strongly typed almost everywhere.
- In WPF you don't have to deal with `ResourceDirectories`.
- Supported for ASP.NET, Class Libraries, WPF, Xamarin, .NET Core, .NET Standard as far as I have tested.
- No extra third-party libraries needed.
- Supports culture fallback: en-US -> en.
- Not only back-end, works also in XAML for WPF and Xamarin.Forms, in .cshtml for MVC.
- Easily manipulate the language by changing the `Thread.CurrentThread.CurrentCulture`
- Search engines can Crawl in different languages and user can send or save language-specific urls.

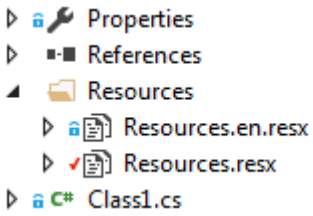
Con's

- WPF XAML is sometimes buggy, newly added strings don't show up directly. Rebuild is the temp fix (vs2015).
- UWP XAML does not show intellisense suggestions and does not show the text while designing.
- Tell me.

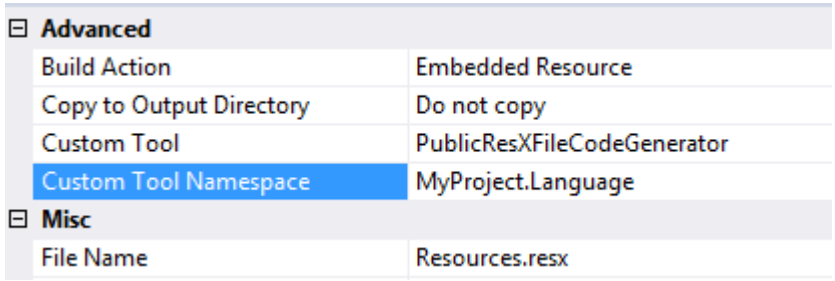
Setup

Create language project in your solution, give it a name like *MyProject.Language*. Add a folder to it called Resources, and in that folder, create two Resources files (.resx). One called **Resources.resx** and another called **Resources.en.resx** (or .en-GB.resx for specific). In my implementation, I have NL (Dutch) language as the default language, so that goes in my first file, and English goes in my second file.

Setup should look like this:



The properties for Resources.resx must be:



Make sure that the custom tool namespace is set to your project namespace. Reason for this is that in WPF, you cannot reference to `Resources` inside XAML.

And inside the resource file, set the access modifier to Public:

If you have such a large application (let's say different modules) you can consider creating multiple projects like above. In that case you could prefix your Keys and resource classes with the particular Module. Use the best language editor there is for Visual Studio to combine all files into a single overview.

Using in another project

Reference to your project: Right click on References -> Add Reference -> Prjects\Solutions.

Use namespace in a file: `using MyProject.Language;`

Use it like so in back-end: `string someText = Resources.orderGeneralError;` If there is something else called Resources, then just put in the entire namespace.

Using in MVC

In MVC you can do however you like to set the language, but I used parameterized url's, which can be setup like so:

RouteConfig.cs Below the other mappings

```
routes.MapRoute(
    name: "Locolized",
    url: "{lang}/{controller}/{action}/{id}",
    constraints: new { lang = @"(\w{2})|(\w{2}-\w{2})" },   // en or en-US
    defaults: new { controller = "shop", action = "index", id = UrlParameter.Optional }
);
```

FilterConfig.cs (might need to be added, if so, add `FilterConfig.RegisterGlobalFilters(GlobalFilters.Filters);` to the `Application_start()` method in `Global.asax`

```
public class FilterConfig
{
    public static void RegisterGlobalFilters(GlobalFilterCollection filters)
    {
        filters.Add(new ErrorHandler.AiHandleErrorAttribute());
        //filters.Add(new HandleErrorAttribute());
        filters.Add(new LocalizationAttribute("nl-NL"), 0);
    }
}
```

LocalizationAttribute

```
public class LocalizationAttribute : ActionFilterAttribute
{
    private string _DefaultLanguage = "nl-NL";
    private string[] allowedLanguages = { "nl", "en" };

    public LocalizationAttribute(string defaultLanguage)
    {
        _DefaultLanguage = defaultLanguage;
    }

    public override void OnActionExecuting(ActionExecutingContext filterContext)
    {
        string lang = (string) filterContext.RouteData.Values["lang"] ??
_DefaultLanguage;
        LanguageHelper.SetLanguage(lang);
```

```
    }
}
```

LanguageHelper just sets the Culture info.

```
//fixed number and date format for now, this can be improved.
public static void SetLanguage(LanguageEnum language)
{
    string lang = "";
    switch (language)
    {
        case LanguageEnum.NL:
            lang = "nl-NL";
            break;
        case LanguageEnum.EN:
            lang = "en-GB";
            break;
        case LanguageEnum.DE:
            lang = "de-DE";
            break;
    }
    try
    {
        NumberFormatInfo numberInfo = CultureInfo.CreateSpecificCulture("nl-
NL").NumberFormat;
        CultureInfo info = new CultureInfo(lang);
        info.NumberFormat = numberInfo;
        //later, we will if-else the language here
        info.DateTimeFormat.DateSeparator = "/";
        info.DateTimeFormat.ShortDatePattern = "dd/MM/yyyy";
        Thread.CurrentThread.CurrentUICulture = info;
        Thread.CurrentThread.CurrentCulture = info;
    }
    catch (Exception)
    {

    }
}
```

Usage in .cshtml

```
@using MyProject.Language;
<h3>@Resources.w_home_header</h3>
```

or if you don't want to define usings then just fill in the entire namespace OR you can define the namespace under /Views/web.config:

```
<system.web.webPages.razor>
<host factoryType="System.Web.Mvc.MvcWebRazorHostFactory, System.Web.Mvc,
Version=5.2.3.0, Culture=neutral, PublicKeyToken=31BF3856AD364E35" />
<pages pageBaseType="System.Web.Mvc.WebViewPage">
  <namespaces>
    ...
    <add namespace="MyProject.Language" />
  </namespaces>
</pages>
</system.web.webPages.razor>
```

This mvc implementation source tutorial: Awesome tutorial blog

Using in class libraries for models

Back-end using is the same, but just an example for using in attributes

```
using MyProject.Language;
namespace MyProject.Core.Models
{
    public class RegisterViewModel
    {
        [Required(ErrorMessageResourceName = "accountEmailRequired",
ErrorMessageResourceType = typeof(Resources))]
        [EmailAddress]
        [Display(Name = "Email")]
        public string Email { get; set; }
    }
}
```

If you have reshaper it will automatically check if the given resource name exists. If you prefer type safety you can use T4 templates to generate an enum

Using in WPF.

Ofcourse add a reference to your *MyProject.Language* namespace, we know how to use it in back-end.

In XAML, inside the header of a Window or UserControl, add a namespace reference called `lang` like so:

```
<UserControl x:Class="Babywatcher.App.Windows.Views.LoginView"
         xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
             xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
             xmlns:local="clr-namespace:MyProject.App.Windows.Views"
              xmlns:lang="clr-namespace:MyProject.Language;assembly=MyProject.Language"
<!--this one-->
             mc:Ignorable="d"
            d:DesignHeight="210" d:DesignWidth="300">
```

Then, inside a label:

```
    <Label x:Name="lblHeader" Content="{x:Static lang:Resources.w_home_header}"
TextBlock.FontSize="20" HorizontalAlignment="Center"/>
```

Since it is strongly typed you are sure the resource string exists. You might need to recompile the project sometimes during setup, WPF is sometimes buggy with new namespaces.

One more thing for WPF, set the language inside the `App.xaml.cs`. You can do your own implementation (choose during installation) or let the system decide.

```
public partial class App : Application
{
    protected override void OnStartup(StartupEventArgs e)
    {
        base.OnStartup(e);
        SetLanguageDictionary();
    }

    private void SetLanguageDictionary()
    {
        switch (Thread.CurrentThread.CurrentCulture.ToString())
        {
            case "nl-NL":
                MyProject.Language.Resources.Culture = new
System.Globalization.CultureInfo("nl-NL");
                break;
            case "en-GB":
                MyProject.Language.Resources.Culture = new
System.Globalization.CultureInfo("en-GB");
                break;
```

```
            default://default english because there can be so many different system
language, we rather fallback on english in this case.
                MyProject.Language.Resources.Culture = new
System.Globalization.CultureInfo("en-GB");
                break;
        }

    }
}
```

Using in UWP

In UWP, Microsoft uses [this solution], meaning you will need to create new resource files. Plus you can not re-use the text either because they want you to set the `x:Uid` of your control in XAML to a key in your resources. And in your resources you have to do `Example.Text` to fill a `TextBlock` 's text. I didn't like that solution at all because I want to re-use my resource files. Eventually I came up with the following solution. I just found this out today (2019-09-26) so I might come back with something else if it turns out this doesn't work as desired.

Add this to your project:

```
using Windows.UI.Xaml.Resources;

public class MyXamlResourceLoader : CustomXamlResourceLoader
{
    protected override object GetResource(string resourceId, string objectType, string
propertyName, string propertyType)
    {
        return MyProject.Language.Resources.ResourceManager.GetString(resourceId);
    }
}
```

Add this to `App.xaml.cs` in the constructor:

```
CustomXamlResourceLoader.Current = new MyXamlResourceLoader();
```

Where ever you want to in your app, use this to change the language:

```
ApplicationLanguages.PrimaryLanguageOverride = "nl";
Frame.Navigate(this.GetType());
```

The last line is needed to refresh the UI. While I am still working on this project I noticed that I needed to do this 2 times. I might end up with a language selection at the first time the user is starting. But since this will be distributed via Windows Store, the language is usually equal to the system language.

Then use in XAML:

```
<TextBlock Text="{CustomResource ExampleResourceKey}"></TextBlock>
```

Using it in Angular (convert to JSON)

Now days it is more common to have a framework like Angular in combination with components, so without cshtml. Translations are stored in json files, I am not going to cover how that works, I would just highly recommend ngx-translate instead of the angular multi-translation. So if you want to convert translations to a JSON file, it is pretty easy, I use a T4 template script that converts the Resources file to a json file. I recommend installing T4 editor to read the syntax and use it correctly because you need to do some modifications.

Only 1 thing to note: It is not possible to generate the data, copy it, clean the data and generate it for another language. So you have to copy below code as many times as languages you have and change the entry before '//choose language here'. Currently no time to fix this but probably will update later (if interested).

Path: MyProject.Language/T4/CreateLocalizationEN.tt

```
<#@ template debug="false" hostspecific="true" language="C#" #>
<#@ assembly name="System.Core" #>
<#@ assembly name="System.Windows.Forms" #>
<#@ import namespace="System.Linq" #>
<#@ import namespace="System.Text" #>
<#@ import namespace="System.Collections.Generic" #>
<#@ import namespace="System.Resources" #>
<#@ import namespace="System.Collections" #>
<#@ import namespace="System.IO" #>
<#@ import namespace="System.ComponentModel.Design" #>
<#@ output extension=".json" #>
<#
```

```
var fileNameNl = "../Resources/Resources.resx";
var fileNameEn = "../Resources/Resources.en.resx";
var fileNameDe = "../Resources/Resources.de.resx";
var fileNameTr = "../Resources/Resources.tr.resx";

var fileResultName = "../T4/CreateLocalizationEN.json";//choose language here
var fileResultPath = Path.Combine(Path.GetDirectoryName(this.Host.ResolvePath("")),
"MyProject.Language", fileResultName);
//var fileDestinationPath = "../../MyProject.Web/ClientApp/app/i18n/";

var fileNameDestNl = "nl.json";
var fileNameDestEn = "en.json";
var fileNameDestDe = "de.json";
var fileNameDestTr = "tr.json";

var pathBaseDestination =
Directory.GetParent(Directory.GetParent(this.Host.ResolvePath("")).ToString()).ToString()


string[] fileNamesResx = new string[] {fileNameEn }; //choose language here
string[] fileNamesDest = new string[] {fileNameDestEn }; //choose language here

for(int x = 0; x < fileNamesResx.Length; x++)
{
    var currentFileNameResx = fileNamesResx[x];
    var currentFileNameDest = fileNamesDest[x];
    var currentPathResx =
Path.Combine(Path.GetDirectoryName(this.Host.ResolvePath("")), "MyProject.Language",
currentFileNameResx);
    var currentPathDest =pathBaseDestination + "/MyProject.Web/ClientApp/app/i18n/" +
currentFileNameDest;
    using(var reader = new ResXResourceReader(currentPathResx))
    {
        reader.UseResXDataNodes = true;
#>
        {
<#
            foreach(DictionaryEntry entry in reader)
            {
                var name = entry.Key;
                var node = (ResXDataNode)entry.Value;
                var value = node.GetValue((ITypeResolutionService) null);
                 if (!String.IsNullOrEmpty(value.ToString())) value =
value.ToString().Replace("\n", "");
                 if (!String.IsNullOrEmpty(value.ToString())) value =
value.ToString().Replace("\r", "");
#>
            "<#=name#>": "<#=value#>",
```

```
<#


            }
#>
        "WEBSHOP_LASTELEMENT": "just ignore this, for testing purpose"
        }
<#
    }
    File.Copy(fileResultPath, currentPathDest, true);
}


#>
```

If you have a modulair application and you followed my suggestion to create multiple language projects, then you will have to create a T4 file for each of them. Make sure the json files are logically defined, it doesn't have to be `en.json`, it can also be `example-en.json`. To combine multiple json files for using with ngx-translate, follow the instructions here

Use in Xamarin.Android

As explained above in the updates, I use the same method as I have done with Angular/JSON. But Android uses XML files, so I wrote a T4 file that generates those XML files.

Path: MyProject.Language/T4/CreateAppLocalizationEN.tt

```
#@ template debug="false" hostspecific="true" language="C#" #>
<#@ assembly name="System.Core" #>
<#@ assembly name="System.Windows.Forms" #>
<#@ import namespace="System.Linq" #>
<#@ import namespace="System.Text" #>
<#@ import namespace="System.Collections.Generic" #>
<#@ import namespace="System.Resources" #>
<#@ import namespace="System.Collections" #>
<#@ import namespace="System.IO" #>
<#@ import namespace="System.ComponentModel.Design" #>
<#@ output extension=".xml" #>
<#
var fileName = "../Resources/Resources.en.resx";
var fileResultName = "../T4/CreateAppLocalizationEN.xml";
var fileResultRexPath = Path.Combine(Path.GetDirectoryName(this.Host.ResolvePath("")),
"MyProject.Language", fileName);
```

```
var fileResultPath = Path.Combine(Path.GetDirectoryName(this.Host.ResolvePath("")),
"MyProject.Language", fileResultName);

    var fileNameDest = "strings.xml";

    var pathBaseDestination =
Directory.GetParent(Directory.GetParent(this.Host.ResolvePath("")).ToString()).ToString()


    var currentPathDest =pathBaseDestination +
"/MyProject.App.AndroidApp/Resources/values-en/" + fileNameDest;

    using(var reader = new ResXResourceReader(fileResultRexPath))
    {
        reader.UseResXDataNodes = true;
        #>
        <resources>
        <#

                foreach(DictionaryEntry entry in reader)
                {
                    var name = entry.Key;
                    //if(!name.ToString().Contains("WEBSHOP_") &&
!name.ToString().Contains("DASHBOARD_"))//only include keys with these prefixes, or the
country ones.
                    //{
                    //  if(name.ToString().Length != 2)
                    //  {
                    //      continue;
                    //  }
                    //}
                    var node = (ResXDataNode)entry.Value;
                    var value = node.GetValue((ITypeResolutionService) null);
                     if (!String.IsNullOrEmpty(value.ToString())) value =
value.ToString().Replace("\n", "");
                     if (!String.IsNullOrEmpty(value.ToString())) value =
value.ToString().Replace("\r", "");
                     if (!String.IsNullOrEmpty(value.ToString())) value =
value.ToString().Replace("&", "&amp;");
                     if (!String.IsNullOrEmpty(value.ToString())) value =
value.ToString().Replace("<<", "");
                     //if (!String.IsNullOrEmpty(value.ToString())) value =
value.ToString().Replace("'", "\'");
#>
             <string name="<#=name#>">"<#=value#>"</string>
<#
                }
#>
           <string name="WEBSHOP_LASTELEMENT">just ignore this</string>
```

```
<#
        #>
        </resources>
        <#
        File.Copy(fileResultPath, currentPathDest, true);
    }

#>
```

Android works with `values-xx` folders, so above is for English for in the `values-en` folder. But you also have to generate a default which goes into the `values` folder. Just copy above T4 template and change the folder in the above code.

There you go, you can now use one single resource file for all your projects. This makes it very easy exporting everything to an excl document and let someone translate it and import it again.

Special thanks to this amazing VS extension which works awesome with `resx` files. Consider donating to him for his awesome work (I have nothing to do with that, I just love the extension).

Share Edit Follow

edited Jun 24 '20 at 21:36

answered Mar 5 '16 at 11:47

CularBytes
8,635 7 67 94

1. how do you add more languages after deployment?, by adding more resource to the Mylanguage and recompile and then redeploy? (2) Is there any IDE or Tool you used for translation(I have over 40 forms in winform), can you share? (3) I tried it and it didn't get the translation when i changed the culture to Arabic, i will create a question and let you know – Smith Mar 1 '18 at 16:04

5 1: Just dublicate Resources.xx.resx and change xx and everywhere where I have the switch statements add the language. 2 I Use ResxManager (by TomEnglert) extension, can be installed via Tools -> Extensions, nice way to have the languages next to each other but actual translation is done by someone else (easy export and import to/from excel with ResxManager). 3. Encountered this last time as well by adding one as well, check all files that are listed above, but some breakpoints. I didn't use this in WinForms though. – CularBytes Mar 1 '18 at 20:22

13 Can I upvote my own answer? I just had to stupidly check my own answer to fix a bug for switching between languages in WPF -.- – CularBytes Jul 3 '18 at 14:36

1 @TiagoBrenck Hi, only in this specific situation, if english is your default language you could create a resources.en-gb.resx file and add the 1% changes in there. By default, if the language en-gb is selected and words are not translated, it uses the default language (resources.resx). – CularBytes Sep 13 '18 at 10:01

1 How have you structured the keys of the Resource file? Since I see you only use one Resource file and, according to you, you have a huge app, how have you splitted the keys? Something like "Feature.MyMessage", or "Area.Feature.MyMessage" – Francesco Venturini Nov 22 '19 at 14:34

21

I've seen projects implemented using a number of different approaches, each have their merits and drawbacks.

- One did it in the config file (not my favourite)
- One did it using a database - this worked pretty well, but was a pain in the you know what to maintain.
- One used resource files the way you're suggesting and I have to say it was my favourite approach.
- The most basic one did it using an include file full of strings - ugly.

I'd say the resource method you've chosen makes a lot of sense. It would be interesting to see other people's answers too as I often wonder if there's a better way of doing things like this. I've seen numerous resources that all point to the using resources method, including one right here on SO.

Share Edit Follow

edited May 23 '17 at 12:34
Community ♦
1 1

answered Dec 17 '08 at 1:45

BenAlabaster
36.9k 20 100 147

5

I don't think there is a "best way". It really will depend on the technologies and type of application you are building.

Webapps can store the information in the database as other posters have suggested, but I recommend using seperate resource files. That is **resource files seperate from your source**. Seperate resource files reduces contention for the same files and as your project grows you may find localization will be done seperatly from business logic. (Programmers and Translators).

Microsoft WinForm and WPF gurus recommend using seperate resource assemblies customized to each locale.

WPF's ability to size UI elements to content lowers the layout work required eg: (japanese words are much shorter than english).

If you are considering WPF: I suggest reading this msdn article To be truthful I found the WPF localization tools: msbuild, locbaml, (and maybe an excel spreadsheet) tedious to use, but it does work.

Something only slightly related: A common problem I face is integrating legacy systems that send error messages (usually in english), not error codes. This forces either changes to legacy systems, or mapping backend strings to my own error codes and then to localized strings...yech. **Error codes are localizations friend**

Share Edit Follow

answered Dec 17 '08 at 4:07

MW_dev

2,147 1 26 37

+1 Database

4

Forms in your app can even re-translate themselves on the fly if corrections are made to the database.

We used a system where all the controls were mapped in an XML file (one per form) to language resource IDs, but all the IDs were in the database.

Basically, instead of having each control hold the ID (implementing an interface, or using the tag property in VB6), we used the fact that in .NET, the control tree was easily discoverable through reflection. A process when the form loaded would build the XML file if it was missing. The XML file would map the controls to their resource IDs, so this simply needed to be filled in and mapped to the database. This meant that there was no need to change the compiled binary if something was not tagged, or if it needed to be split to another ID (some words in English which might be used as both nouns and verbs might need to translate to two different words in the dictionary and not be re-used, but you might not discover this during initial assignment of IDs). But the fact is that the whole translation process becomes completely independent of your binary (every form has to inherit from a base form which knows how to translate itself and all its controls).

The only ones where the app gets more involved is when a phase with insertion points is used.

The database translation software was your basic CRUD maintenance screen with various workflow options to facilitate going through the missing translations, etc.

Share Edit Follow

answered Dec 17 '08 at 2:47

Cade Roux

84.3k 39 172 260

I like your method, can you tell me how the xml file is generated and then mapped to database? – Smith Jul 25 '18 at 20:52

In Form_Load or whatever, the translation function was invoked on the form instance. The XML file for that form was loaded. If it didn't exist it was created. I don't have the schema off-hand, but basically, it mapped control name on a form to a translation ID. So any control on the form that was not in the XML file would get an entry with no translation ID. Because they would get created on demand, you could simply create your form, run the app which would build or update the XML file for any missing controls. Then fill in the translation IDs for the items. – Cade Roux Jul 26 '18 at 15:18

4

I´ve been searching and I´ve found this:

If your using WPF or Silverlight your aproach could be use WPF LocalizationExtension for many reasons.

IT´s Open Source It´s FREE (and will stay free) is in a real stabel state

In a Windows Application you could do someting like this:

```
public partial class App : Application
{
     public App()
     {
     }

     protected override void OnStartup(StartupEventArgs e)
     {
         Thread.CurrentThread.CurrentCulture = new
System.Globalization.CultureInfo("de-DE"); ;
         Thread.CurrentThread.CurrentUICulture = new
System.Globalization.CultureInfo("de-DE"); ;

          FrameworkElement.LanguageProperty.OverrideMetadata(
              typeof(FrameworkElement),
              new FrameworkPropertyMetadata(

XmlLanguage.GetLanguage(CultureInfo.CurrentCulture.IetfLanguageTag)));
          base.OnStartup(e);
     }
}
```

And I think on a Wep Page the aproach could be the same.

Good Luck!

Share Edit Follow

edited Jun 20 '20 at 9:12
Community ♦
1 1

answered Dec 2 '10 at 8:51
JxXx
111 1 7

2

I'd go with the multiple resource files. It shouldn't be that hard to configure. In fact I recently answered a similar question on setting a global language based resource files in conjunction with form language resource files.

Localization in Visual Studio 2008

I would consider that the best approach at least for WinForm development.

Share Edit Follow

edited May 23 '17 at 12:26
Community ♦
1 1

answered Dec 17 '08 at 2:21
KMessenger
81 1 3

A drawback of resources is that you have to restart to switch languages. It's probably acceptable for most, but it's my pet peeve... – Roman Starkov Jan 13 '10 at 0:17

2

You can use commercial tools like Sisulizer. It will create satellite assembly for each language. Only thing you should pay attention is not to obfuscate form class names (if you use obfuscator).

Share Edit Follow

answered Dec 17 '08 at 8:59
Davorin
1,268 10 14

0

I highly recommend ResXManager tool which works with ResourceDirectories (resx) and {x:static} markup.

ResXManager works as addin for visual studio or standalone application.

Share Edit Follow

answered Aug 27 at 10:42
Marcin Sulecki
11 3

We use a custom provider for multi language support and put all texts in a database table. It works well except we sometimes face caching problems when updating texts in the database without updating the web application.

0

Share Edit Follow

answered Dec 17 '08 at 8:53

Cossintan

114 4 12

0

Most opensource projects use GetText for this purpose. I don't know how and if it's ever been used on a .Net project before.

Share Edit Follow

answered Dec 17 '08 at 3:13

Vasil

33.2k 26 85 113

0

Standard resource files are easier. However, if you have any language dependent data such as lookup tables then you will have to manage two resource sets.

I haven't done it, but in my next project I would implement a database resource provider. I found how to do it on MSDN:

http://msdn.microsoft.com/en-us/library/aa905797.aspx

I also found this implementation:

DBResource Provider

Share Edit Follow

edited Feb 13 '10 at 18:46

answered Dec 17 '08 at 5:57

Ben Dempsey

350 1 6
