عنوان: **آموزش ایجاد برنامه های چند زبانه در WPF**
نویسنده: امیر اویسی
تاریخ: ۱۳۹۱/۰۶/۰۹ ۱۵:۱۳
آدرس: www.dotnettips.info
گروه‌ها: WPF

با گسترش استفاده از کامپیوتر در بسیاری از امور روزمره انسان‌ها سازگار بودن برنامه‌ها با سلیقه کاربران به یکی از نیازهای اصلی برنامه‌های کامپیوتری تبدیل شده است. بدون شک زبان و فرهنگ یکی از مهم‌ترین عوامل در ایجاد ارتباط نزدیک بین برنامه و کاربر به شمار می‌رود و نقشی غیر قابل انکار در میزان موفقیت یک برنامه به عهده دارد. از این رو در این نوشته تلاش بر آن است تا یکی از ساده‌ترین و در عین حال کاراترین راه‌های ممکن برای ایجاد برنامه‌های چند زبانه با استفاده از تکنولوژی WPF آموزش داده شود.

مروری بر روش‌های موجود

همواره روش‌های مختلفی برای پیاده سازی یک ایده در دنیای نرم افزار وجود دارد که هر روش را می‌توان بر حسب نیاز مورد استفاده قرار داد. در برنامه‌های مبتنی بر WPF معمولا از دو روش عمده برای این منظور استفاده می‌شود:

1-استفاده از فایل‌های resx

در این روش که برای Win App نیز استفاده می‌شود، اطلاعات مورد نیاز برای هر زبان به شکل جدول هایی دارای کلید و مقدار در داخل یک فایل .resx نگهداری می‌شود و در زمان اجرای برنامه بر اساس انتخاب کاربر اطلاعات زبان مورد نظر از داخل فایل resx خوانده شده و نمایش داده می‌شود. یکی از ضعف هایی که این روش در عین ساده بودن دارد این است که همه اطلاعات مورد نیاز داخل assembly اصلی برنامه قرار می‌گیرد و امکان افزودن زبان‌های جدید بدون تغییر دادن برنامه اصلی ممکن نخواهد بود.

2-استفاده از فایل‌های csv که به فایل‌های dll تبدیل می‌شوند

در این روش با استفاده از ابزارهای موجود در کامپایلر WPF برای هر کنترل یک property به نام Uid ایجاد شده و مقدار دهی می‌شود. سپس با ابزار دیگری (که جزو ابزارهای کامپایلر محسوب نمی‌شود) از فایل csproj پروژه یک خروجی اکسل با فرمت csv ایجاد می‌شود که شامل Uidهای کنترل‌ها و مقادیر آن‌ها است. پس از ترجمه متون مورد نظر به زبان مقصد با کمک ابزار دیگری فایل اکسل مورد نظر به یک net assembly تبدیل می‌شود و داخل پوشه ای با نام culture استاندارد ذخیره می‌شود. (مثلا برای زبان فارسی نام پوشه fa-IR خواهد بود). زمانی که برنامه اجرا می‌شود بر اساس culture ای که در سیستم عامل انتخاب شده است و در صورتی که برای آن culture فایل dll ای موجود باشد، زبان مربوط به آن culture را load خواهد کرد. با وجود این که این روش مشکل روش قبلی را ندارد و بیشتر با ویژگی‌های WPF سازگار است اما پروسه ای طولانی برای انجام کارها دارد و به ازای هر تغییری باید کل مراحل هر بار تکرار شوند. همچنین مشکلاتی در نمایش برخی زبان‌ها (از جمله فارسی) در این روش مشاهده شده است.

روش سوم!

روش سوم اما کاملا بر پایه WPF و در اصطلاح WPF-Native می‌باشد. ایده از آنجا ناشی شده است که برای ایجاد skin در برنامه‌های WPF استفاده می‌شود. در ایجاد برنامه‌های Skin-Based به این شیوه عمل می‌شود که skinهای مورد نظر به صورت style هایی در داخل ResourceDictionary ها قرار می‌گیرند. سپس آن ResourceDictionary به شکل dll کامپایل می‌شود. در برنامه اصلی نیز همه کنترل‌ها style هایشان را به شکل dynamic resource از داخل یک ResourceDictionary مشخص شده load می‌کنند. حال کافی است برای تغییر skin فعلی، ResourceDictionary مورد نظر از dll مشخص load شود و ResourceDictionary ای که در حال حاضر در برنامه از آن استفاده می‌شود با ResourceDictionary ای که load شده جایگزین شود. کنترل‌ها مقادیر جدید را از ResourceDictionary جدید به شکل کاملا خودکار دریافت خواهند کرد.
به سادگی می‌توان از این روش برای تغییر زبان برنامه نیز استفاده کرد با این تفاوت که این بار، به جای Style ها، Stringهای زبان‌های مختلف را درون resourceها نگهداری خواهیم کرد.

یک مثال ساده

در این قسمت نحوه پیاده سازی این روش با ایجاد یک نمونه برنامه ساده که دارای دو زبان انگلیسی و فارسی خواهد بود آموزش

داده می‌شود.

ابتدا یک پروژه WPF Application در Visual Studio 2010 ایجاد کنید. در MainWindow سه کنترل Button قرار دهید و یک ComboBox که قرار است زبان‌های موجود را نمایش دهد و با انتخاب یک زبان، نوشته‌های درون Buttonها متناسب با آن تغییر خواهند کرد.

توجه داشته باشید که برای Buttonها نباید به صورت مستقیم مقداری به Content شان داده شود. زیرا مقدار مورد نظر از داخل ResourceDictionary که خواهیم ساخت به شکل dynamic گرفته خواهد شد. پس در این مرحله یک ResourceDictionary به پروژه اضافه کرده و در آن resource هایی به شکل string ایجاد می‌کنیم. هر resource دارای یک Key می‌باشد که بر اساس آن، Button مورد نظر، مقدار آن Resource را load خواهد کرد. فایل ResourceDictionary را Culture_en-US.xaml نامگذاری کنید و مقادیر مورد نظر را به آن اضافه نمایید.

```
<ResourceDictionary xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
                    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
                    xmlns:system="clr-namespace:System;assembly=mscorlib">
    <system:String x:Key="button1">Hello!</system:String>
    <system:String x:Key="button2">How Are You?</system:String>
    <system:String x:Key="button3">Are You OK?</system:String>

</ResourceDictionary>
```

دقت کنید که namespace ای که کلاس string در آن قرار دارد به فایل xaml اضافه شده است و پیشوند system به آن نسبت داده شده است.

با افزودن یک ResourceDictionary به پروژه، آن ResourceDictionary به MergedDictionary کلاس App اضافه می‌شود. بنابراین فایل App.xaml به شکل زیر خواهد بود:

```
<Application x:Class="BeRMOoDA.WPF.LocalizationSample.App"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             StartupUri="MainWindow.xaml">
    <Application.Resources>

        <ResourceDictionary>
            <ResourceDictionary.MergedDictionaries>
                <ResourceDictionary Source="Culture_en-US.xaml"/>
            </ResourceDictionary.MergedDictionaries>
        </ResourceDictionary>
```

```
    </Application.Resources>
</Application>
```

برای اینکه بتوانیم محتوای Buttonهای موجود را به صورت داینامیک و در زمان اجرای برنامه، از داخل Resourceها بگیریم، از DynamicResource استفاده می‌کنیم.

```
<Button Content="{DynamicResource ResourceKey=button1}" />
<Button Content="{DynamicResource ResourceKey=button2}" />
<Button Content="{DynamicResource ResourceKey=button3}" />
```

بسیار خوب! اکنون باید شروع به ایجاد یک ResourceDictionary برای زبان فارسی کنیم و آن را به صورت یک فایل dll کامپایل نماییم.
برای این کار یک پروژه جدید در قسمت WPF از نوع User control ایجاد می‌کنیم و نام آن را Culture_fa-IR_Farsi قرار می‌دهیم. لطفا شیوه نامگذاری را رعایت کنید چرا که در ادامه به آن نیاز خواهیم داشت.
پس از ایجاد پروژه فایل UserControl1.xaml را از پروژه حذف کنید و یک ResourceDictionary با نام Culture_fa-IR.xaml اضافه کنید. محتوای آن را پاک کنید و محتوای فایل Culture_en-US.xaml را از پروژه قبلی به صورت کامل در فایل جدید کپی کنید.
دو فایل باید ساختار کاملا یکسانی از نظر key برای Resourceهای موجود داشته باشند. حالا زمان ترجمه فرا رسیده است!
رشته‌های دلخواه را ترجمه کنید و پروژه را build نمایید.
پس از ترجمه فایل Culture_fa-IR.xaml به شکل زیر خواهد بود:

```
<ResourceDictionary xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
                    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
                    xmlns:system="clr-namespace:System;assembly=mscorlib">
    <ResourceDictionary.MergedDictionaries>
        <ResourceDictionary Source="Culture_fa-IR_Farsi.xaml"/>
    </ResourceDictionary.MergedDictionaries>
    <system:String x:Key="button1">سلام!</system:String>
    <system:String x:Key="button2">حالت چطوره؟</system:String>
    <system:String x:Key="button3">خوبی؟</system:String>
</ResourceDictionary>
```

خروجی این پروژه یک فایل با نام Culture_fa-IR_Farsi.dll خواهد بود که حاوی یک ResourceDictionary برای زبان فارسی می‌باشد.

در ادامه میخواهیم راهکاری ارئه دهیم تا بتوان فایل‌های dll مربوط به زبان‌ها را در زمان اجرای برنامه اصلی، load کرده و نام زبان‌ها را در داخل ComboBox ای که داریم نشان دهیم. سپس با انتخاب هر زبان در ComboBox، محتوای Buttonها بر اساس زبان انتخاب شده تغییر کند.
برای سهولت کار، نام فایل‌ها را به گونه ای انتخاب کردیم که بتوانیم ساده‌تر به این هدف برسیم. نام هر فایل از سه بخش تشکیل شده است:

```
Culture_[standard culture notation]_[display name for this culture].dll
```

یعنی اگر فایل Culture_fa-IR_Farsi.dll را در نظر بگیریم، Culture نشان دهنده این است که این فایل مربوط به یک culture می‌باشد. fa-IR نمایش استاندارد culture برای کشور ایران و زبان فارسی است و Farsi هم مقداری است که می‌خواهیم در ComboBox برای این زبان نمایش داده شود.
پوشه ای با نام Languages در کنار فایل اجرایی برنامه اصلی ایجاد کنید و فایل Culture_fa-IR_Farsi.dll را درون آن کپی کنید.
تصمیم داریم همه dllهای مربوط به زبان‌ها را داخل این پوشه قرار دهیم تا مدیریت آن‌ها ساده‌تر شود.
برای مدیریت بهتر فایل‌های مربوط به زبان‌ها یک کلاس با نام CultureAssemblyModel خواهیم ساخت که هر instance از آن نشانگر یک فایل زبان خواهد بود. یک کلاس با این نام به پروژه اضافه کنید و propertyهای زیر را در آن تعریف نمایید:

```
public class CultureAssemblyModel
    {
        //the text will be displayed to user as language name (like Farsi)
        public string DisplayText { get; set; }
        //name of .dll file (like Culture_fa-IR_Farsi.dll)
        public string Name { get; set; }
        //standar notation of this culture (like fa-IR)
        public string Culture { get; set; }
        //name of resource dictionary file inside the loaded .dll (like Culture_fa-IR.xaml)
        public string XamlFileName { get; set; }
    }
```

اکنون باید لیست cultureهای موجود را از داخل پوشه languages خوانده و نام آنها را در ComboBox نمایش دهیم.
برای خواندن لیست cultureهای موجود، لیستی از CultureAssmeblyModelها ایجاد کرده و با استفاده از متد LoadCultureAssmeblies، آن را پر می‌کنیم.

```
//will keep information about loaded assemblies
public List<CultureAssemblyModel> CultureAssemblies { get; set; }

//loads assmeblies in languages folder and adds their info to list
 void LoadCultureAssemblies()
 {
      //we should be sure that list is empty before adding info (do u want to add some cultures more
than one? of course u dont!)
      CultureAssemblies.Clear();
      //creating a directory represents applications directory\languages
      DirectoryInfo dir = new
DirectoryInfo(System.IO.Path.GetDirectoryName(Assembly.GetExecutingAssembly().Location) +
"\\languages");
      //getting all .dll files in the language folder and its sub dirs. (who knows? maybe someone keeps
each culture file in a seperate folder!)
      var assemblies = dir.GetFiles("*.dll", SearchOption.AllDirectories);
      //for each found .dll we will create a model and set its properties and then add to list  for
(int i = 0; i < assemblies.Count(); i++)
      {

                  string name = assemblies[i].Name;

                  CultureAssemblyModel model = new CultureAssemblyModel() { DisplayText = name.Split('.',
'_')[2], Culture = name.Split('.', '_')[1], Name = name  , XamlFileName =name.Substring(0,
name.LastIndexOf(".")) + ".xaml" };
                  CultureAssemblies.Add(model);
      }
 }
```

پس از دریافت اطلاعات cultureهای موجود، زمان نمایش آن‌ها در ComboBox است. این کار بسیار ساده است، تنها کافی است ItemsSource آن را با لیستی از CultureAssmeblyModelها که ساختیم، مقدار دهی کنیم.

```
comboboxLanguages.ItemsSource = CultureAssemblies;
```

البته لازم به ذکر است که برای نمایش فقط نام هر CultureAssemblyModel در ComboBox، باید ItemTemplate مناسبی برای ComboBox ایجاد کنیم. در مثال ما ItemTemplate به شکل زیر خواهد بود:

```
<ComboBox HorizontalAlignment="Left" Margin="10" VerticalAlignment="Top" MinWidth="100"
Name="comboboxLanguages">
            <ComboBox.ItemTemplate>
                <DataTemplate>
                    <Label Content="{Binding DisplayText}"/>
                </DataTemplate>
            </ComboBox.ItemTemplate>
</ComboBox>
```

توجه داشته باشید که با وجود اینکه فقط نام را در ComboBox نشان می‌دهیم، اما باز هم هر آیتم از ComboBox یک instance از نوع CultureAssemblyModel می‌باشد.

در مرحله بعد، قرار است متدی بنویسیم که اطلاعات زبان انتخاب شده را گرفته و با جابجایی ResourceDictionary ها، زبان برنامه را تغییر دهیم.
متدی با نام LoadCulture در کلاس App ایجاد می‌کنیم که یک CultureAssemblyModel به عنوان ورودی دریافت کرده و ResourceDictionary داخل آن را load می‌کند و آن را با ResourceDictionary فعلی موجود در App.xaml جابجا می‌نماید.
با این کار، Button هایی که قبلا مقدار Content خود را از Resourceهای موجود دریافت می‌کردند، اکنون از Resourceهای جابجا شده خواهند گرفت و به این ترتیب زبان انتخاب شده بر روی برنامه اعمال می‌شود.

```
//loads selected culture
 public void LoadCulture(CultureAssemblyModel culture)
 {
     //creating a FileInfo object represents .dll file of selected cultur
     FileInfo assemblyFile = new FileInfo("languages\\" + culture.Name);
     //loading .dll into memory as a .net assembly
     var assembly = Assembly.LoadFile(assemblyFile.FullName);
     //getting .dll file name
     var assemblyName = assemblyFile.Name.Substring(0, assemblyFile.Name.LastIndexOf("."));
     //creating string represents structure of a pack uri (something like this:
/{myassemblyname;component/myresourcefile.xaml}
     string packUri = string.Format(@"/{0};component/{1}", assemblyName, culture.XamlFileName);
     //creating a pack uri
     Uri uri = new Uri(packUri, UriKind.Relative);
     //now we have created a pack uri that represents a resource object in loaded assembly
     //and its time to load that as a resource dictionary (do u remember that we had resource
dictionary in culture assemblies? don't u?)
     var dic = Application.LoadComponent(uri) as ResourceDictionary;
     dic.Source = uri;
     //here we will remove current merged dictionaries in our resource dictionary and add recently-
loaded resource dictionary as e merged dictionary
     var mergedDics = this.Resources.MergedDictionaries;
     if (mergedDics.Count > 0)
          mergedDics.Clear();
     mergedDics.Add(dic);
 }
```

برای ارسال زبان انتخاب شده به این متد، باید رویداد SelectionChanged را برای ComboBox مدیریت کنیم:

```
void comboboxLanguages_SelectionChanged(object sender, SelectionChangedEventArgs e)
 {
     var selectedCulture = (CultureAssemblyModel)comboboxLanguages.SelectedItem;
     App app = Application.Current as App;
     app.LoadCulture(selectedCulture);
 }
```

کار انجام شد!
از مزیت‌های این روش می‌توان به WPF-Native بودن، سادگی در پیاده سازی، قابلیت load کردن هر زبان جدیدی در زمان اجرا بدون نیاز به کوچک‌ترین تغییر در برنامه اصلی و همچنین پشتیبانی کامل از نمایش زبان‌های مختلف از جمله فارسی اشاره کرد.

نظرات خوانندگان

نویسنده: رضا
تاریخ: ۱۳۹۱/۰۶/۰۹ ۱۵:۲۸

آیا استفاده زیاد از این Dynamic Resourceها مشکلی در Performance برنامه ایجاد نمیکند؟

نویسنده: امیر اویسی
تاریخ: ۱۳۹۱/۰۶/۰۹ ۱۸:۳۵

Dynamic Resourceها در مقایسه با Static Resourceها دارای performance کمتری هستند اما در مواردی که گرفتن مقدار از Resourceها در زمان اجرا انجام می‌گیرد، باید از Dynamic Resourceها استفاده کرد.
در کل تفاوت Performance در کاربردهای این چنین آنقدر نیست که موجب نگرانی باشد.

نویسنده: S.Roshan
تاریخ: ۱۳۹۱/۱۲/۰۵ ۲۱:۲۰

سلام. ممنون از مقاله مفیدتون.
امااااا....

```
if (mergedDics.Count > 0)   mergedDics.Clear();
```

متاسفانه این خط کد ، کل دیکشنریهای ادغامی رو پاک میکنه.
در صورتیکه ممکنه ما یه سری ریسورس دیگه مرتبط با بخشهای دیگه‌ی برنامه داشته باشیم، که نیازی به پاک کردنشون نباشه.
به نظرم باید این کد تغییر کنه ، فقط ریسورس مربوط به زبان برنامه رو پاک کنه.

فقط نمیدونم چه جوری ؟
بی زحمت کدشو بنویسید . ممنون

نویسنده: بهزاد دات نت
تاریخ: ۱۳۹۲/۱۰/۰۹ ۱۱:۴۵

با تشکر از آموزش خوبتون. میخواستم بدونم Direction صفحات برنامه رو چطور مدیریت کنم. مثلا برای انتخاب زبان فارسی راست به چپ و انگلیسی چپ به راست بشه به صورت خودکار؟ با سپاس

نویسنده: امیر اویسی
تاریخ: ۱۳۹۲/۱۲/۱۴ ۱۹:۴۹

سلام
با عرض پوزش به خاطر تاخیر زیاد در ارسال پاسخ باید عرض کنم که اجباری به حذف کلیه ResourceDictionaryها نیست. شما میتوانید با استفاده از متد ResourceDictionary.Remove یک ResourceDictionary به خصوص را با استفاده از Key آن از لیست MergedDictionaryها حذف کنید.

نویسنده: امیر اویسی
تاریخ: ۱۳۹۲/۱۲/۱۴ ۱۹:۵۸

برای همه تنظیماتی که نیاز دارید در زمان load شدن یک زبان خاص بر روی برنامه اعمال شود میتوانید در داخل فایل Resource آن زبان مقدار مورد نظر را با استفاده از یک کلید به عنوان Resource تعریف کنید و در برنامه خودتان مقدار مورد نیاز را از همان

Resource بخوانید.

مثلا با فرض اینکه میخواهیم با انتخاب زبان فارسی، برنامه ما راست به چپ شود، میتوانید یک Resource از نوع ResourceDictionary زبان فارسی ایجاد کنید و مقدار مورد نظر FrameworkElement.FlowDirection با کلید Direction در داخل را به آن اختصاص دهید. سپس در کنترل هایی که نیاز دارید راست به چپ شوند، مقدار FlowDirection آن‌ها را به صورت DynamicResource به همین Resource ای که تعریف کردید مقدار دهی کنید.