# Adding multilingual support to your Windows Phone application using the Multilingual App Toolkit

This article explains how to localize your Windows Phone application using the Multilingual App Toolkit.

21 Jul 2013

## Introduction

Windows Phone Store is currently available in 191 countries. This allows your application to have an almost global customer reach. This also means a lot of customer diversity. Language is one of these differences. Building an application that can target multiple languages can help providing a great user experience for as many users around the world as possible.

The Multilingual App Toolkit helps you in this process. It allows you to easily localize your app, either by yourself or by using Microsoft Translator. You can also share the translation files with someone and then import them into your project.

If you have a similar Windows Store App targeting the Windows 8 platform you can import these translation files into a Windows 8 application, too.

Note: Windows Phone 8 is well integrated with the Multilingual App Toolkit and when you create a Windows Phone 8 project most of the work to enable localization is done for you. The tools also work on Windows Phone 7 provided that you first follow these steps before getting started.

## Prerequisites

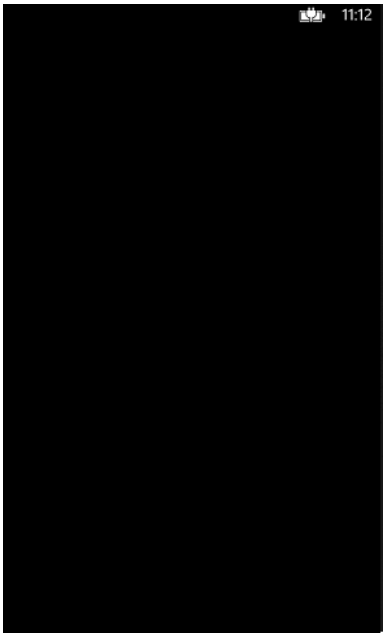
In order to use the Multilingual App Toolkit your application must be built or modified in order to support localization. This means that all your localized resources are separated from the code. MSDN has a great article on How to build a localized app for Windows Phone.

## Multilingual App Toolkit

You can download the Multilingual App Toolkit here: Multilingual App Toolkit for Visual Studio 2012

## Example application

For this article we'll create a simple Hello world! application based on the Windows Phone App template. It is a simple page with a `Button` and a `TextBlock` which displays a message when you tap the button.




Below is the code for the **MainPage.xaml**.

```
<StackPanel x:Name="TitlePanel" Grid.Row="0" Margin="12,17,0,28">
 <TextBlock Text="{Binding Path=LocalizedResources.ApplicationTitle,
Source={StaticResource LocalizedStrings}}" Style="{StaticResource PhoneTextNormalStyle}"
Margin="12,0"/>
 <TextBlock Text="{Binding Path=LocalizedResources.MainPageName, Source={StaticResource
LocalizedStrings}}" Margin="9,-7,0,0" Style="{StaticResource PhoneTextTitle1Style}"/>
</StackPanel>

<!--ContentPanel - place additional content here-->
<Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">
```


(C) Copyright Nokia 2014. All rights reserved.

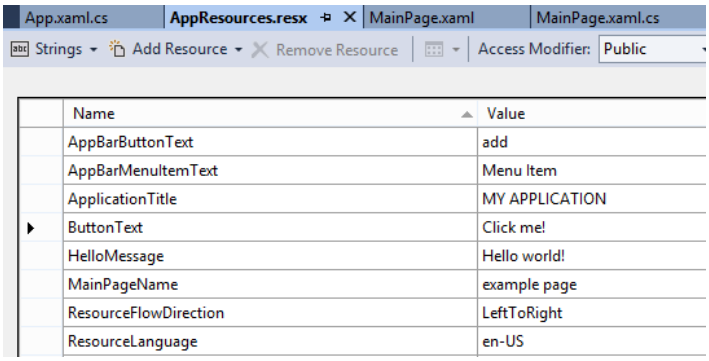
```
 <Button Content="{Binding Path=LocalizedResources.ButtonText, Source={StaticResource
LocalizedStrings}}" HorizontalAlignment="Center" VerticalAlignment="Top"
Click="Button_Click"/>
 <TextBlock x:Name="tbMessage" HorizontalAlignment="Center" VerticalAlignment="Center"
FontSize="72" />
</Grid>
```

Note: `LocalizedResources` is a property in **LocalStrings.cs** (from the app template) populated with the current app language. `LocalizedStrings` can be thought of as a model for localized resources - see **App.xaml**.

And the code behind:

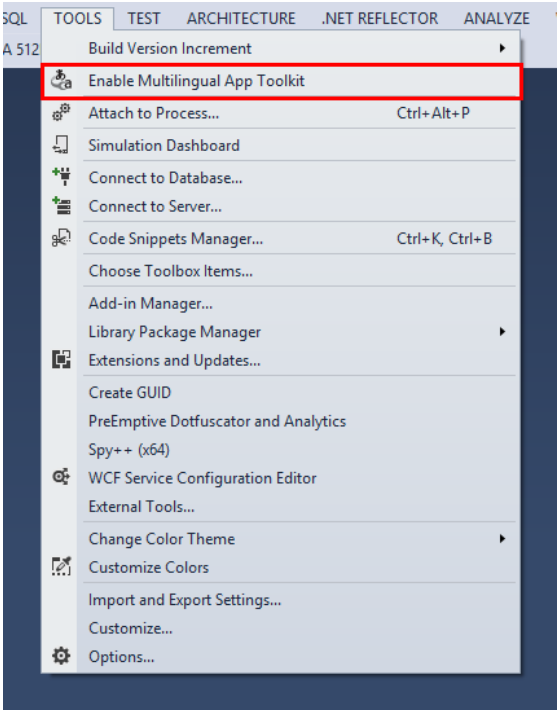
```
private void Button_Click(object sender, RoutedEventArgs e)
{
 tbMessage.Text = AppResources.HelloMessage;
}
```

A few entries were added to the **AppResources.resx** (the file where the *default* resources are stored). It looks like this:




## Enable the Multilingual App Toolkit

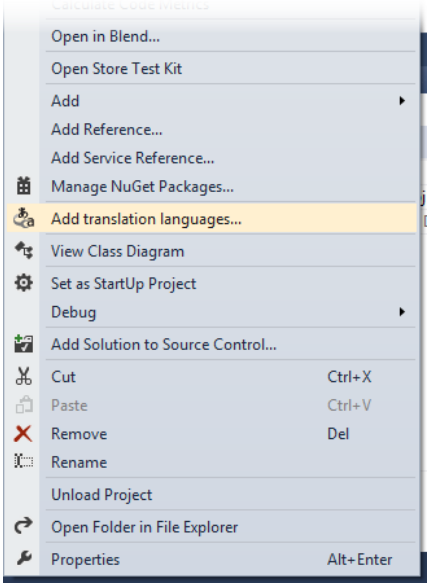
To begin using the Multilingual App Toolkit you must first enable it. Open your project in Visual Studio, open **Tools** menu and select **Enable Multilingual App Toolkit**.



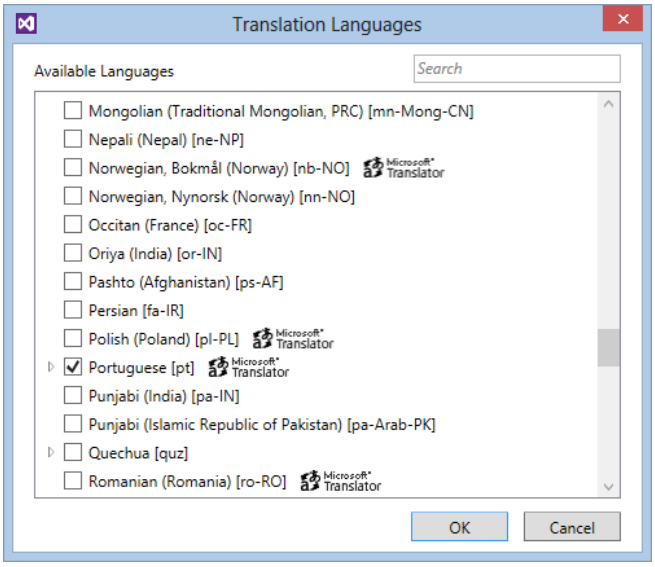

## Add new languages

To add a new language, go to the Solution Explorer and right-click your project. Select **Add translation languages...**.


(C) Copyright Nokia 2014. All rights reserved.




In the *Translation Languages* window select the language(s) you want to add. In this example we'll add Portuguese localization to our application.




A new file called **AppResources.<*culture-code*>.xlf** is added to the *Resources* folder. This is a XLIFF file where localizable data is stored. You can export and import this file into your project. This allows you to share this file with a translator and later import it into your project.

These **.xlf** files are converted into **.resx** files during compilation. These are are the same **.resx** files you would be using when localizing an application without the Multilingual App Toolkit.

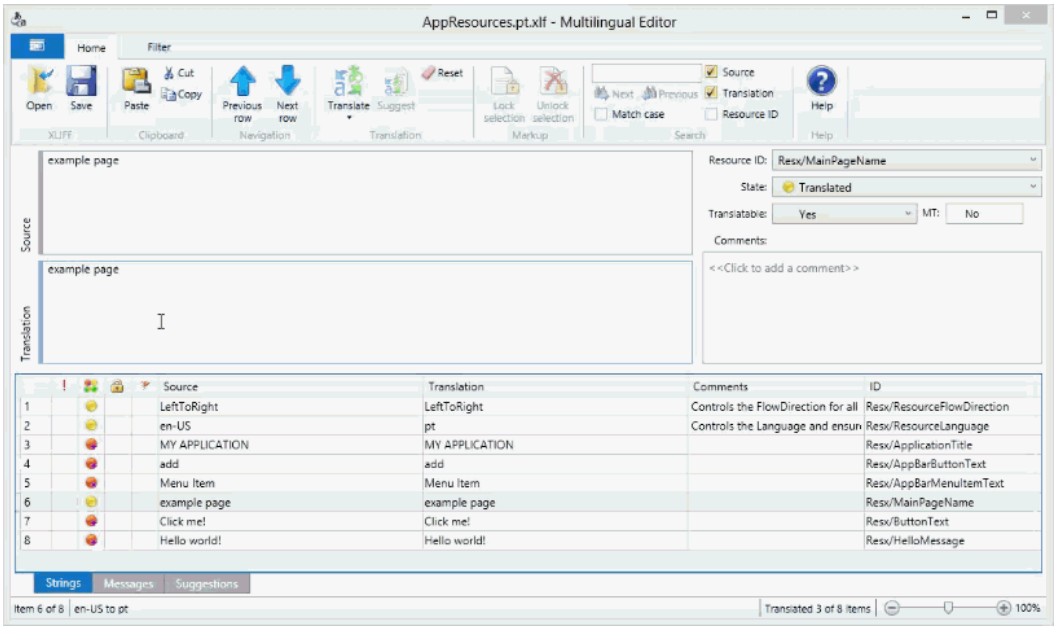
## Multilingual Editor tool

Double click the file that was created in the previous step. The Multilingual Editor window will open. You can use this tool to translate your application in multiple ways:

### Translate your application by yourself

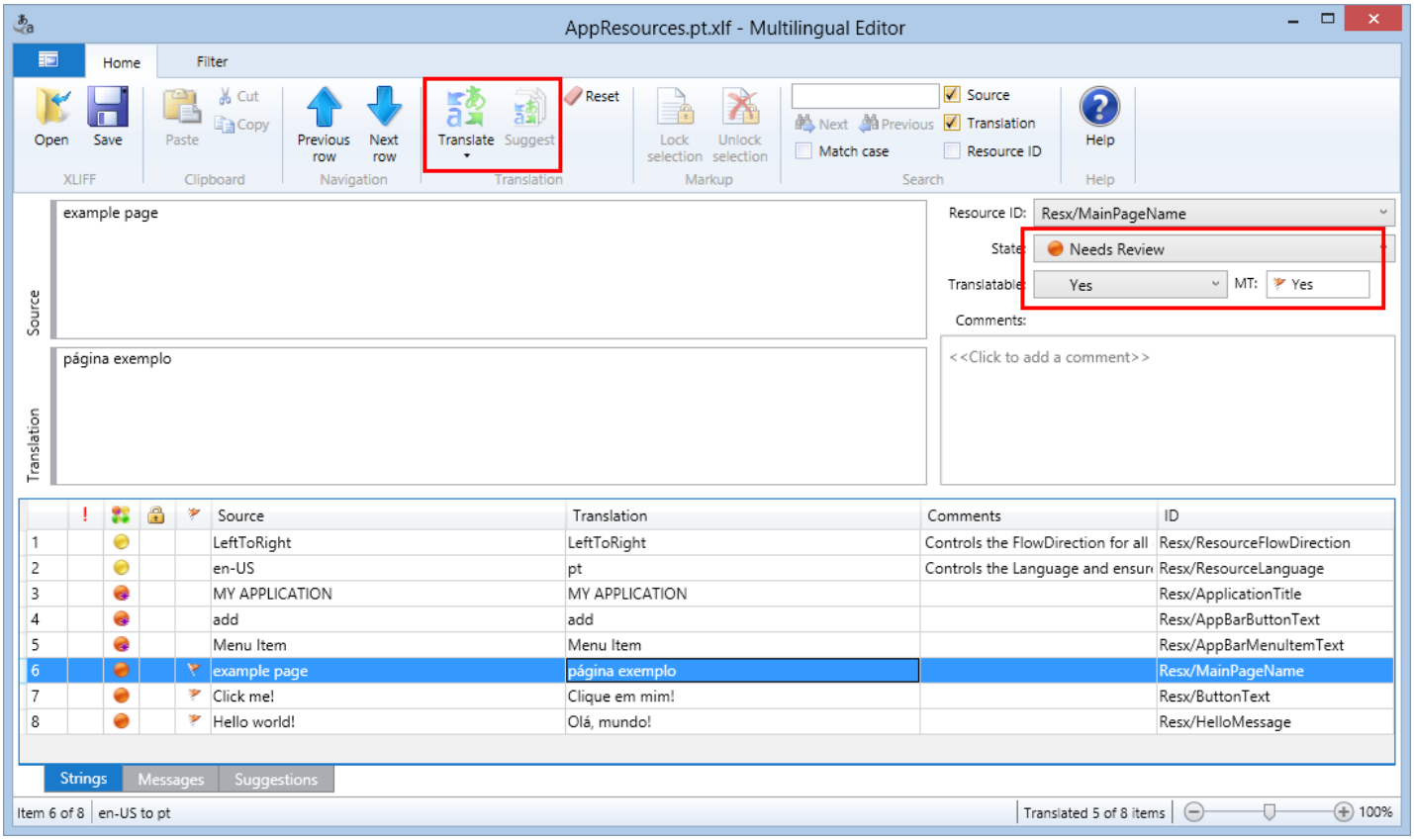
If you happen to speak the language you're adding to your application, you can translate it all by yourself. Simply go through the list of strings at the bottom and type the translation for each item in the *Translation* field. Notice that each string has an icon to its left. This represents the state of each translation. You can use this to, for example, mark some string for later review. This feature is really useful when sharing translation files with other people.


(C) Copyright Nokia 2014. All rights reserved.




## Translate using Microsoft Translator Service

The Microsoft Translator service can also be used to help you in your localization process, either by providing suggestions for each translation or by completely replacing a string with its translation. To use the this service, simply select the string you want to translate and click on the options in the toolbar.



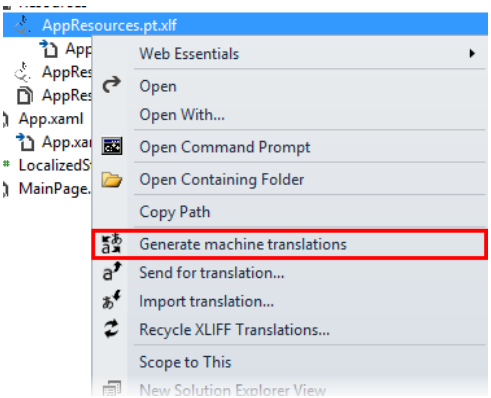

As you may know, machine translation is not perfect due to the complexity of the human language. A word may have a few different meanings, depending on the context it is. Since there is no way you can give this information to the Microsoft Translator, some results may sometimes be inaccurate.

The Multilingual Editor has a way of dealing with this issue. When you use this service to translate a string (or a group of strings), you'll notice that they are flagged as being MTed (Machine Translated) and that their state is set to *Needs Review*.

You can later accept this translation or make changes to suit your style.

To translate all the strings in a file using the Microsoft Translator service, click the arrow below the **Translate** button and select **Translate All**. You can also right click the .xlf file in the Solution Explorer and select **Generate machine translations**.


(C) Copyright Nokia 2014. All rights reserved.



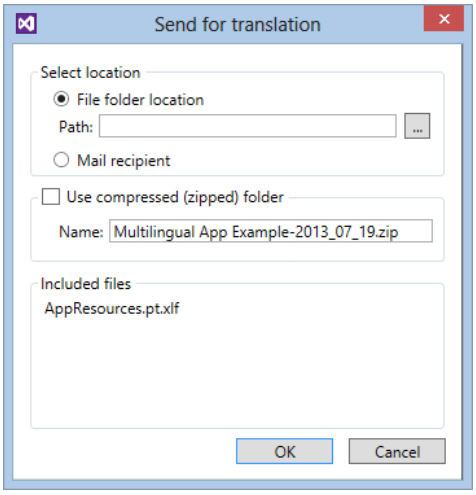

## Share a translation files

Sometimes you may want to add support for a language that you may not know very well. In this case you can share the XLIFF file with another person (or service) so that they can translate it for you. XLIFF is a format that is supported by a lot of applications. You can also open this type of files using the Multilingual Editor. It can be used standalone. Just look for it in your Start Menu.

### Exporting a file for translation

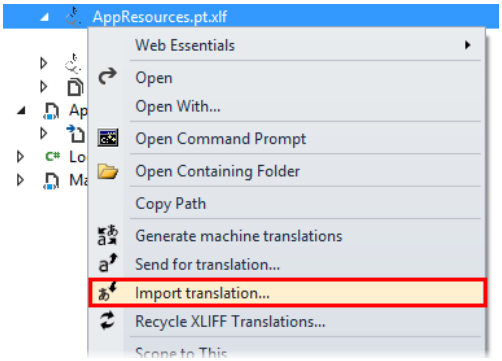
Right click the file you want to export and select **Send for translation...**. Select **File folder location** if you want to save the **.xlf** file to a local folder or select **Mail recipient** if you want to send the **.xlf** by email. Select **Use compressed (zipped) folder** if you want to compress the file before saving or sending.

*Note that you can export multiple files at once. Select the files you want to export in the Solution Explorer while pressing* Ctrl *and follow the steps above.*




### Import a translation file

Right click the matching .xlf file in your solution and select **Import translation...**. If the languages for each file are not the same, the importing will fail.




## Test a localized application

You can test a different language by changing the display language on your phone (or emulator). Check this article for more information: How to test a localized app for Windows Phone

You can alternatively force your application to use a different language by adding the following line to the beginning of the App constructor in **App.xaml.cs**:

```
using System.Globalization;
using System.Threading;
```

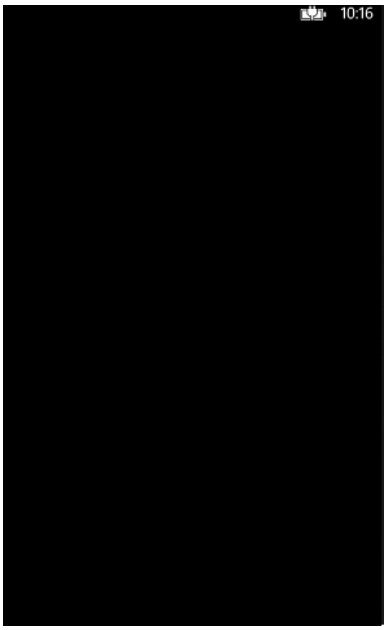

(C) Copyright Nokia 2014. All rights reserved.

```
...

Thread.CurrentThread.CurrentUICulture = new CultureInfo("pt");
```

Replace the **pt** string with the language code you want to test.

After translating our example application to Portuguese, this is what it looks like:

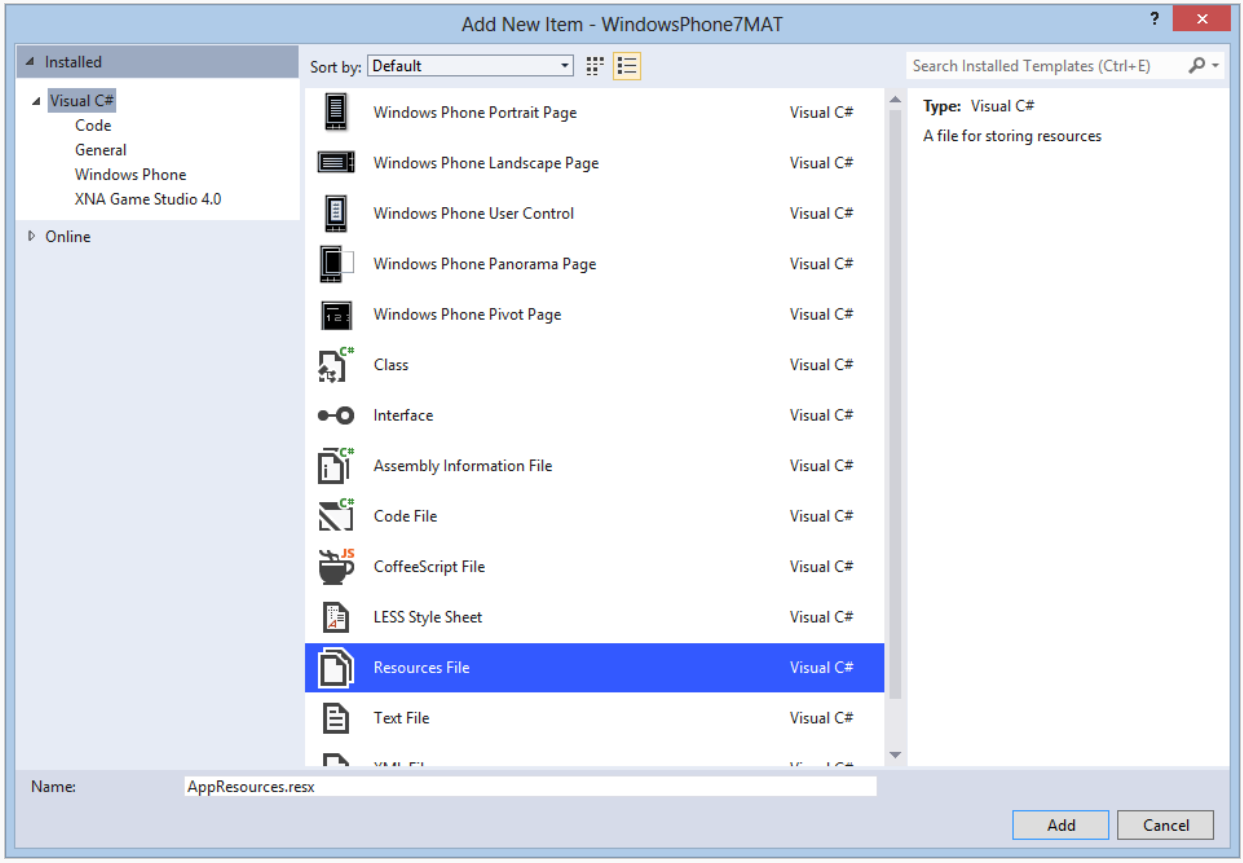



## Using Multilingual App Toolkit with Windows Phone 7 apps

When you create a new Windows Phone 8 project, you'll notice that some of the wiring required to localize your application is already done for you. However, this can be implemented as well in a Windows Phone 7 application. Here's how to do it is in a few simple steps:

### Add a resource file for the default language

Right-click your project file in the Solution Explorer and select **Add > New Folder**. Name this folder **Resources**. Now, right-click this folder you created and select **Add > New Item...**. Select **Resources File** and change the name to **AppResources.resx**.



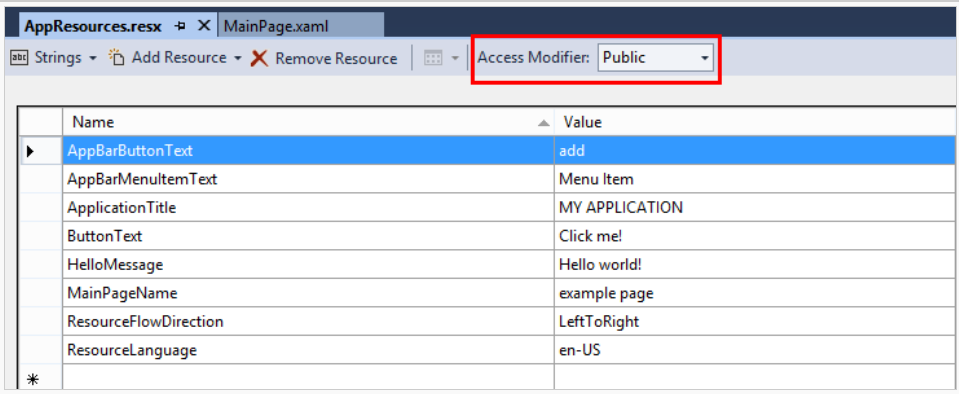
Adding the default language resource file

### Edit the default language file


(C) Copyright Nokia 2014. All rights reserved.

First, make sure that the **Access Modifier** field is set to **Public**. Next, add a string called *ResourceFlowDirection* and set it to *LeftToRight* (or *RightToLeft*, if your language is written from right to left, like Arabic). This will set the FlowDirection for all the elements in the RootFrame, so they can adapt to the language. Now add a string called *ResourceLanguage* and set it to *en-US* (or any other culture code, like *pt-PT* for Portuguese (Portugal)). This ensures that the font for all elements in the RootFrame aligns with the app's language.

Now you can add all the resources for your default language.



Default language resource file

## Integrate the Resources with your application

Add a new class to the root of your project called **LocalizedStrings.cs**:

```
using <Your_App_Namespace>.Resources;
...
/// <summary>
/// Provides access to string resources.
/// </summary>
public class LocalizedStrings
{
 private static AppResources _localizedResources = new AppResources();

 public AppResources LocalizedResources { get { return _localizedResources; } }
}
```

Open App.xaml and add this line in the section:

```
<local:LocalizedStrings xmlns:local="clr-namespace:<Your_App_Namespace>"
x:Key="LocalizedStrings"/>
```

Finally, open App.xaml.cs and add the following method:

```
using System.Windows.Markup;
using <Your_App_Namespace>.Resources;
...
private void InitializeLanguage()
{
 try
 {
  // Set the font to match the display language defined by the
  // ResourceLanguage resource string for each supported language.
  //
  // Fall back to the font of the neutral language if the Display
  // language of the phone is not supported.
  //
  // If a compiler error is hit then ResourceLanguage is missing from
  // the resource file.
  RootFrame.Language = XmlLanguage.GetLanguage(AppResources.ResourceLanguage);

  // Set the FlowDirection of all elements under the root frame based
  // on the ResourceFlowDirection resource string for each
  // supported language.
  //
```


(C) Copyright Nokia 2014. All rights reserved.

```
  // If a compiler error is hit then ResourceFlowDirection is missing from
  // the resource file.
  FlowDirection flow = (FlowDirection)Enum.Parse(typeof(FlowDirection),
AppResources.ResourceFlowDirection);
  RootFrame.FlowDirection = flow;
 }
 catch
 {
  // If an exception is caught here it is most likely due to either
  // ResourceLangauge not being correctly set to a supported language
  // code or ResourceFlowDirection is set to a value other than LeftToRight
  // or RightToLeft.

  if (Debugger.IsAttached)
  {
   Debugger.Break();
  }

  throw;
 }
}
```

Finally add the method call below `InitializePhoneApplication();`

```
// Language display initialization
InitializeLanguage();
```

Now your application is ready to be localized.

## Summary

As said before, by having an app in the Windows Phone Store you can reach a lot of different cultures. English may be a globally spoken language, but if you can adapt your application to better target a specific culture, you will be providing your users a better experience. And if a user likes your application, he will certainly recommend it to other people, giving you a larger user base. Tools like the Multilingual App Toolkit can help you achieve this.


(C) Copyright Nokia 2014. All rights reserved.