

Preparing windows phone 8 apps for international markets

With the release of Windows Phone 8 SDK, Microsoft introduces several changes and improvements to the Windows Phone localization and generalization framework. This guide will cover those changes as well as provide a complete overview on preparing a Windows phone 8 app for international markets.

Introduction



Starting with the release of Windows Phone 7.5 (Mango), which reduced the minimal requirements to run the Windows Phone OS, Microsoft had increased their efforts to improve the Windows Phone OS distribution. With the raising popularity of Windows Phone in Europe and the far east, and the trend of developing of low cost devices (such as: Nokia Lumia 510 and the Nokia Lumia 610) the OS reach is expected to be diverse. With the wide distribution of Windows Phones, ignoring localization might alienate many users who lack the language or simply expect a local app to perform better. By following few technical guidelines, every app can be prepared for localization even if it is not currently planned.

To design an application for international use, there are two different aspects that require attention: Localization and Globalization. Localization is the preparation of the app to work under the user's local language and country based settings. Under localization we handle all the textual translation, handle text direction, choose fonts, change relevant images or videos and even alter the app to meet local standards. Globalization is the preparation of the app to work under the cultural settings of the users. The cultural settings can dictate the way dates, numbers and currency are formatted. An example for such cultural difference might be how dates are displayed. For example, in the US, the date begins first with the month and afterward the day while in other locations the day might appear first and the month afterward. Ignoring such differentiations can cause confusion among app users and hurt their experience.

What is new in Windows Phone 8

In the Windows Phone 8 SDK, Microsoft introduces several changes and improvements in the localization and globalization framework. Among the changes are the support for 26 new languages: Albanian, Arabic, Azerbaijani, Belorussian, Bulgarian, Catalan, Croatian, Estonian, Filipino, Hebrew, Hindi, Kazakh, Latvian, Lithuanian, Macedonian, Persian, Romanian, Serbian, Slovak, Slovenian, Spanish (Mexico), Thai, Turkish, Ukrainian, Uzbek and Vietnamese. Another change introduced in the Windows Phone 8 SDK is the ability to create a right to left (RTL) and bidirectional applications. The ability to define a default culture for an app domain was added. A few visual changes to the visual studio IDE were also performed to allow easier work with localization resources.

Implementing localization

The .NET framework offers a tool called Resources to easily localize .NET applications. This tool was also integrated in the Windows Phone SDK and offers a simple way to localize apps.

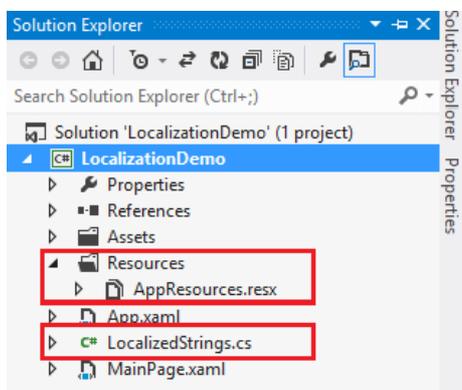
Understanding the resource files

The resource files, are simple settings files. Each file consist of entries, where each entry should represent one resource. A resource could be a simple string (such as App's title), an image (such as an icon), an audio file or any other different file. Each entry is consisted of name, value and a comment fields. The name uses as the identifier of the resource entry, the value as the value and the comment as a remark.

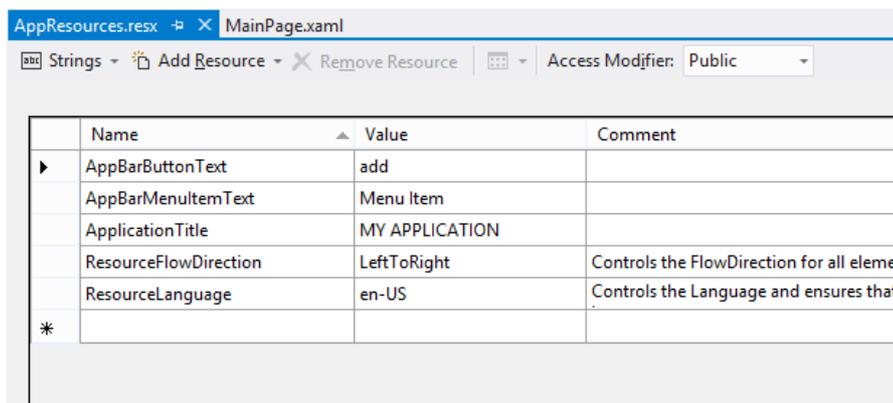
The strength behind the resource files, is their inheritance mechanism. A resource file can be defined in one of three levels. The default top level, resources from this level will be used when no other more appropriate resource was found. The language level, resources from this level will be used when the user's device is configured to the resource's file language and no other appropriate resource file was found. The third and last level is the language and country level, resources in this level will target users with device configured to the language and country of the current resource file. The level of a resource file is determined by the resource file's name. By adding the language or the language and country to the resource file name, we can change the file's level. For example: **AppResource.resx** (Default top level), **AppResource.de.resx** (Language level - German) and **AppResource.de-AT.resx** (Language and country level- German and Austria).

Localizing Windows Phone apps

After creating a Windows Phone 8, a resources folder is automatically added to the solution. The resources folder should contain one file called **AppResources.resx**. In addition a file **LocalizedString.cs** should also be automatically added to the solution.



When clicking on the AppResources.resx, a window with the managed resources editor should open with showing the content of the AppResources file. By default the **AppResources.resx** file should include several entries: ApplicationTitle, ResourceFlowDirection and the ResourceLanguage. By default, the ResourceFlowDirection determines the textual direction (LTR or RTL) of the app and the ResourceLanguage determines the default app language.



An additional entry can be added by simply clicking on the new row and adding a new name and value.

Unlike other .NET frameworks, such as ASP.NET, accessing a resource file in the Windows Phone 8 SDK is done through a supporting class that wraps the resource file. This is done to ease the use of localized string inside XAML. By default, such file called **LocalizedStrings.cs** is generated to wrap the **AppResources.resx**. When opening this file, you can see that the LocalizedString simply creates a static instance of AppResources and exposes it for later use.

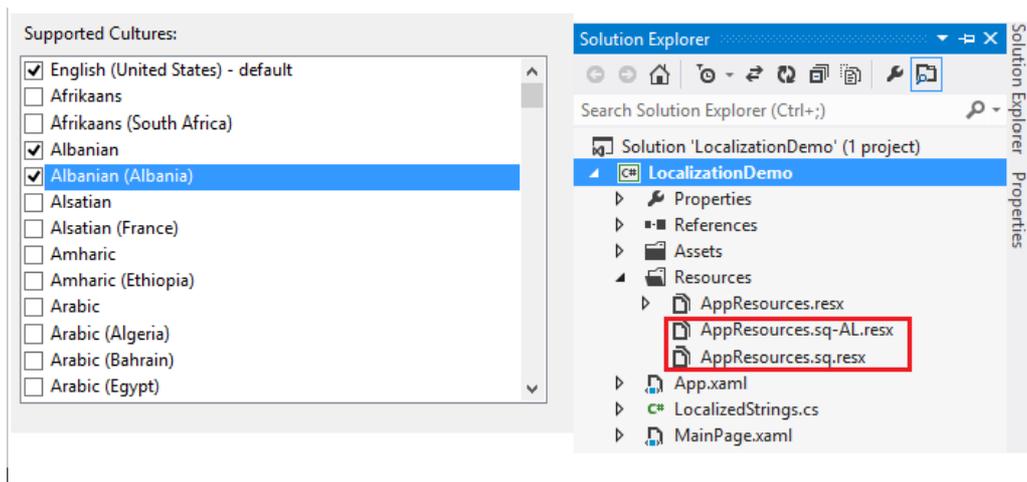
```
public class LocalizedStrings
{
    private static AppResources _localizedResources = new AppResources();
    public AppResources LocalizedResources { get { return _localizedResources; } }
}
```

In order to use the new resources, defined in the AppResources.resx, we should let the XAML know that a call to the resource file should be done to retrieve the relevant resource. The call consists of two parameters, the binding path which determines the name of the resource to be used and source which determines the source in which the resource exists.

```
<TextBlock Text="{Binding Path=LocalizedResources.TestResource, Source={StaticResource LocalizedStrings}}" />
```

In this example, we told the TextBlock to receive it's textual value from an entry in the **AppResource.resx** file.

To add an additional **AppResources** files for other languages, we can use the property page of the Windows Phone project. Under the project's properties, we can define which supported cultures will the app support. After the selection of the relevant cultures, the relevant AppResources files will be automatically generated. When selecting which cultures to support, you should keep in mind, that the supported culture list contains cultures both in language specific or language and country specific resources. In the following example both the Albanian and Albanian (Albania) cultures were added.



Supporting right-to-left

With the introduction of Arabic, Hebrew and Persian languages in Windows Phone 8, the need to support right-to-left (RTL) arose. Supporting RTL languages can be done by changing the `ResourceFlowDirection` in the relevant **AppResources** file. The reason we define the `ResourceFlowDirection` in the **AppResource.XXXX.resx**, and not in any other resource file, is due to the default implementation of the `InitializeLanguage()` function inside the **App.xml**. The default implementation uses the `ResourceFlowDirection` to determine the app's text direction. This implementation can be overridden in special cases where we prefer to determine the text direction flow by other means such as external logic.

```
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.
        //
        // 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
        // ResourceLanguage 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;
    }
}
```

In some cases, there is a need to change the text direction for only part of the app. Using the `FlowDirection` property we can change the textual direction of only certain parts of the application layout. It is important to keep in mind that layout controls inherit this property from their parent.

```
<StackPanel x:Name="TitlePanel" Grid.Row="0" Margin="12,17,0,28" FlowDirection="RightToLeft">
  <TextBlock Text="{Binding Path=LocalizedResources.TestResource, Source={StaticResource LocalizedStrings}}" Style="{StaticResource LocalizedStrings}" />
</StackPanel>
```

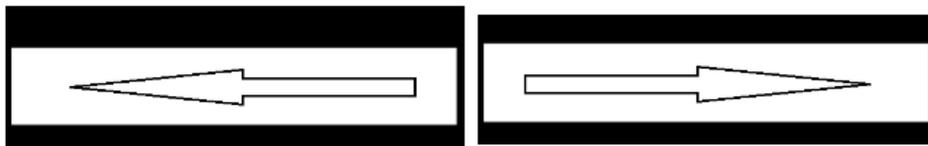
Another special case, is when there is a need to display bidirectional content. Such case might occur when using a right-to-left language mixed with a left-to-right content. Using the `<Run>` element, we can separate the elements that should be displayed with different textual direction. By setting the `FlowDirection` under the `<Run>` tag, we will be able to change the flow direction only to those elements.

Localizing images

It is not uncommon to find applications that use textual content or location relevant content inside their graphical content. When localizing an application, these graphical content should be localized as well. The easiest way to localize images is by inserting the image URI into the relevant resource file and then in a similar manner to textual content, binding the resource to the image source property.

```
<Image Source="{Binding Path=LocalizedResources.ImageURI, Source={StaticResource LocalizedStrings}}" Style="{StaticResource LocalizedStrings}" />
```

When working in an RTL environment, the Windows Phone 8 SDK offers the capability to reverse the image's flow direction (Flip horizontally). Unlike other layout controls, the image control doesn't automatically inherit the `FlowDirection` property. By changing the `FlowDirection` property, an image will automatically flip horizontally.



How to use globalization

The .NET framework implements generalization through the `CultureInfo` class. The `CultureInfo` class already contains the relevant formats for many different cultures. After initializing the `CultureInfo` with the relevant language and country code (For example: en-US for English-United States), the instance is ready to be used for all the formatting local aware strings (Dates, Time and Currency).

```
CultureInfo culture = new CultureInfo("de-DE"); // Initing the cultureinfo (German)
DateTime.Now.ToString("d", culture.DateTimeFormat); // Printing date time in local format
int number = 5000;
number.ToString("C", culture.NumberFormat); // Formating currency
```

To access or change the current culture, use the `Thread.CurrentThread.CurrentCulture` and `Thread.CurrentThread.CurrentUICulture`. The

CurrentCulture represents the user's culture while the CurrentUICulture represents the culture used to access the resource files. In the [Printed on: 2013-06-19](#) we will change the current culture to German (Germany).

```
CultureInfo culture = new CultureInfo("de-DE"); // Initing the cultureinfo (German)
Thread.CurrentThread.CurrentCulture = culture; // Changing the thread culture to German
Thread.CurrentThread.CurrentUICulture = culture;
```

Testing localization

Testing an application localization can be done by changing the device/emulator display language setting. The display language settings can be found under settings → region + language → Display language. After changing the language, you will be prompted to restart the device/emulator to complete the process.

Localizing app's store product page

The last step before releasing a Windows Phone application to marketplace is localizing it's marketplace page. During the publishing process, the Windows Phone marketplace offers an insertion of language specific: title, description and images. Localizing the app store's content lets potential users find your application when they search using their native language. In addition localizing the app store's content lets users know that your app supports their native language.

Common pitfalls when localizing

This list summarize common pitfalls that might occur when localizing an application.

- String length might vary depending on the language. It is important to keep in mind that the same content might have a different length in different languages and therefore beside testing the app in different localization you should also leave a buffer for length increases.
- Always remember to convert time to a user's current time zone. When using a server to distribute information between different clients, always remember to convert the server time to the local user time.
- When creating a resource entry, always remember to create an entry for each variation of that entry. For example, for a string showing the number of new messages ("5 new messages") the following entries should be created: When no messages exists, when only one exists and when more than one exists.
- Never reuse resources. Although reusing resources might save a bit of time and space, it might also create problems when translating the app to different language where the re-usage isn't appropriate.
- Don't forget to localize dates. Different countries use different format.

Summary

Localizing an app is easy and recommended procedure. By designing an app to support localization the apps code become more portable and more readable. Localizing an app to different markets can offer an easy and affordable way to increase exposure and revenues.

