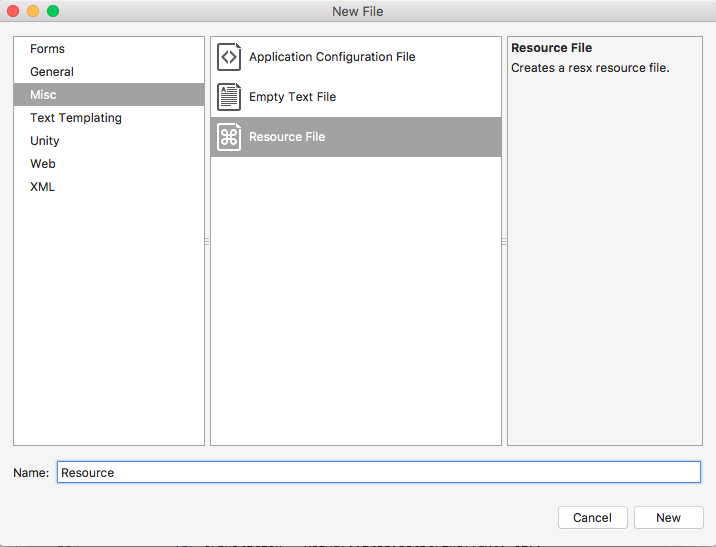
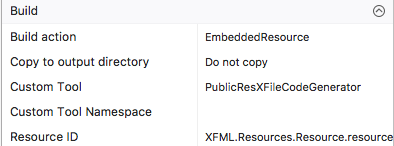
Xamarin.Forms Multi Lenguage

1. Crea un proyecto PCL.
2. Adicione una carpeta llamada **Resources** y dentro de esta crea un archivo de recursos llamado **Resource**:



1. Cambie la propiedad del archivo **Custom Tool** de **ResXFileCodeGenerator** a **PublicResXFileCodeGenerator**:



1. Adicione los valores que quiere ver en varios idiomas, acá un ejemplo:

<data name="Key1" xml:space="preserve">  
    <value>Key One</value>  
</data>

1. Copie el archivo en los diferentes idiomas, por ejemplo: **Resource.es**, para español y **Resource.pt** para portugués. Traduzca los literales:

Para español:

<data name="Key1" xml:space="preserve">  
    <value>Llave Uno</value>  
</data>

Para portugués:

<data name="Key1" xml:space="preserve">  
    <value>Uma Chave</value>  
</data>

1. Cree la carpeta **Interfaces** y dentro de esta la interfaz **ILocalize**:

namespace XFML.Interfaces  
{  
    using System.Globalization;  
  
    public interface ILocalize  
    {  
        CultureInfo GetCurrentCultureInfo();  
        void SetLocale(CultureInfo ci);  
    }  
}

1. Cree la carpeta **Helpers** y dentro de esta la clase **PlatformCulture**:

namespace XFML.Helpers  
{  
    using System;  
      
    public class PlatformCulture  
    {  
        public PlatformCulture(string platformCultureString)  
        {  
            if (string.IsNullOrEmpty(platformCultureString))  
            {  
                throw new ArgumentException("Expected culture identifier", "platformCultureString"); // in C# 6 use nameof(platformCultureString)  
            }  
  
            PlatformString = platformCultureString.Replace("\_", "-"); // .NET expects dash, not underscore  
            var dashIndex = PlatformString.IndexOf("-", StringComparison.Ordinal);  
            if (dashIndex > 0)  
            {  
                var parts = PlatformString.Split('-');  
                LanguageCode = parts[0];  
                LocaleCode = parts[1];  
            }  
            else  
            {  
                LanguageCode = PlatformString;  
                LocaleCode = "";  
            }  
        }  
  
        public string PlatformString { get; private set; }  
        public string LanguageCode { get; private set; }  
        public string LocaleCode { get; private set; }  
          
        public override string ToString()  
        {  
            return PlatformString;  
        }  
    }  
}

1. En la misma carpeta cree la clase: **Lenguages**

namespace XFML.Helpers  
{  
    using Xamarin.Forms;  
    using Interfaces;  
    using Resources;  
  
    public static class Lenguages  
    {  
        static Lenguages()  
        {  
            var ci = DependencyService.Get<ILocalize>().GetCurrentCultureInfo();  
            Resource.Culture = ci;  
            DependencyService.Get<ILocalize>().SetLocale(ci);  
        }  
  
        public static string Key1  
        {  
            get { return Resource.Key1;  }  
        }  
    }  
}

1. Implemente la interfaz en iOS:

using Xamarin.Forms;  
  
[assembly: Dependency(typeof(XFML.iOS.Localize))]  
  
namespace XFML.iOS  
{  
    using System.Globalization;  
    using System.Threading;  
    using Foundation;  
    using Helpers;  
    using Interfaces;  
  
    public class Localize : ILocalize  
    {  
        public CultureInfo GetCurrentCultureInfo()  
        {  
            var netLanguage = "en";  
            if (NSLocale.PreferredLanguages.Length > 0)  
            {  
                var pref = NSLocale.PreferredLanguages[0];  
                netLanguage = iOSToDotnetLanguage(pref);  
            }  
            // this gets called a lot - try/catch can be expensive so consider caching or something  
            System.Globalization.CultureInfo ci = null;  
            try  
            {  
                ci = new System.Globalization.CultureInfo(netLanguage);  
            }  
            catch (CultureNotFoundException e1)  
            {  
                // iOS locale not valid .NET culture (eg. "en-ES" : English in Spain)  
                // fallback to first characters, in this case "en"  
                try  
                {  
                    var fallback = ToDotnetFallbackLanguage(new PlatformCulture(netLanguage));  
                    ci = new System.Globalization.CultureInfo(fallback);  
                }  
                catch (CultureNotFoundException e2)  
                {  
                    // iOS language not valid .NET culture, falling back to English  
                    ci = new System.Globalization.CultureInfo("en");  
                }  
            }  
            return ci;  
        }  
             
        public void SetLocale(CultureInfo ci)  
        {  
            Thread.CurrentThread.CurrentCulture = ci;  
            Thread.CurrentThread.CurrentUICulture = ci;  
        }  
  
        string iOSToDotnetLanguage(string iOSLanguage)  
        {  
            var netLanguage = iOSLanguage;  
            //certain languages need to be converted to CultureInfo equivalent  
            switch (iOSLanguage)  
            {  
                case "ms-MY":   // "Malaysian (Malaysia)" not supported .NET culture  
                case "ms-SG":   // "Malaysian (Singapore)" not supported .NET culture  
                    netLanguage = "ms"; // closest supported  
                    break;  
                case "gsw-CH":  // "Schwiizertüütsch (Swiss German)" not supported .NET culture  
                    netLanguage = "de-CH"; // closest supported  
                    break;  
                    // add more application-specific cases here (if required)  
                    // ONLY use cultures that have been tested and known to work  
            }  
  
            return netLanguage;  
        }  
  
        string ToDotnetFallbackLanguage(PlatformCulture platCulture)  
        {  
            var netLanguage = platCulture.LanguageCode; // use the first part of the identifier (two chars, usually);  
            switch (platCulture.LanguageCode)  
            {  
                case "pt":  
                    netLanguage = "pt-PT"; // fallback to Portuguese (Portugal)  
                    break;  
                case "gsw":  
                    netLanguage = "de-CH"; // equivalent to German (Switzerland) for this app  
                    break;  
                    // add more application-specific cases here (if required)  
                    // ONLY use cultures that have been tested and known to work  
            }  
  
            return netLanguage;  
        }  
    }  
}

1. Debes agregar al **Info.plist** el idioma por defecto y la lista de los idiomas que se van a utilizar:

<key>CFBundleLocalizations</key>

<array>

<string>es</string>

<string>pt</string>

</array>

<key>CFBundleDevelopmentRegion</key>

<string>en</string>

1. Ahora implementamos la interfaz en Android:

using Xamarin.Forms;  
  
[assembly: Dependency(typeof(XFML.Droid.Localize))]  
  
namespace XFML.Droid  
{  
    using System.Globalization;  
    using System.Threading;  
    using Helpers;  
    using Interfaces;  
  
    public class Localize : ILocalize  
    {  
        public CultureInfo GetCurrentCultureInfo()  
        {  
            var netLanguage = "en";  
            var androidLocale = Java.Util.Locale.Default;  
            netLanguage = AndroidToDotnetLanguage(androidLocale.ToString().Replace("\_", "-"));  
            // this gets called a lot - try/catch can be expensive so consider caching or something  
            System.Globalization.CultureInfo ci = null;  
            try  
            {  
                ci = new System.Globalization.CultureInfo(netLanguage);  
            }  
            catch (CultureNotFoundException e1)  
            {  
                // iOS locale not valid .NET culture (eg. "en-ES" : English in Spain)  
                // fallback to first characters, in this case "en"  
                try  
                {  
                    var fallback = ToDotnetFallbackLanguage(new PlatformCulture(netLanguage));  
                    ci = new System.Globalization.CultureInfo(fallback);  
                }  
                catch (CultureNotFoundException e2)  
                {  
                    // iOS language not valid .NET culture, falling back to English  
                    ci = new System.Globalization.CultureInfo("en");  
                }  
            }  
            return ci;  
        }  
  
        public void SetLocale(CultureInfo ci)  
        {  
            Thread.CurrentThread.CurrentCulture = ci;  
            Thread.CurrentThread.CurrentUICulture = ci;  
        }  
  
        string AndroidToDotnetLanguage(string androidLanguage)  
        {  
            var netLanguage = androidLanguage;  
            //certain languages need to be converted to CultureInfo equivalent  
            switch (androidLanguage)  
            {  
                case "ms-BN":   // "Malaysian (Brunei)" not supported .NET culture  
                case "ms-MY":   // "Malaysian (Malaysia)" not supported .NET culture  
                case "ms-SG":   // "Malaysian (Singapore)" not supported .NET culture  
                    netLanguage = "ms"; // closest supported  
                    break;  
                case "in-ID":  // "Indonesian (Indonesia)" has different code in  .NET  
                    netLanguage = "id-ID"; // correct code for .NET  
                    break;  
                case "gsw-CH":  // "Schwiizertüütsch (Swiss German)" not supported .NET culture  
                    netLanguage = "de-CH"; // closest supported  
                    break;  
                    // add more application-specific cases here (if required)  
                    // ONLY use cultures that have been tested and known to work  
            }  
            return netLanguage;  
        }  
  
        string ToDotnetFallbackLanguage(PlatformCulture platCulture)  
        {  
            var netLanguage = platCulture.LanguageCode; // use the first part of the identifier (two chars, usually);  
            switch (platCulture.LanguageCode)  
            {  
                case "gsw":  
                    netLanguage = "de-CH"; // equivalent to German (Switzerland) for this app  
                    break;  
                    // add more application-specific cases here (if required)  
                    // ONLY use cultures that have been tested and known to work  
            }  
            return netLanguage;  
        }  
    }  
}

1. Modifique la página principal para poder hacer la prueba:

<?xml version="1.0" encoding="utf-8"?>  
<ContentPage   
    xmlns="http://xamarin.com/schemas/2014/forms"   
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"   
    xmlns:local="clr-namespace:XFML"   
    x:Class="XFML.XFMLPage">  
    <Label  
        x:Name="MyLabel"  
        VerticalOptions="Center"   
        HorizontalOptions="Center" />  
</ContentPage>

Y en el código de la página:

namespace XFML  
{  
    using Xamarin.Forms;  
    using Helpers;  
  
    public partial class XFMLPage : ContentPage  
    {  
        public XFMLPage()  
        {  
            InitializeComponent();  
  
            MyLabel.Text = Lenguages.Key1;  
        }  
    }  
}

1. Ahora puedes hacer la prueba, ejecutando el programa en los tres idiomas y verificando que muestre los literales correctos.
2. Ahora para usar las traducciones directamente desde el XML debemos crear la clase: **TranslateExtension**:

namespace XFML  
{  
    using System;  
    using System.Globalization;  
    using System.Reflection;  
    using System.Resources;  
    using Interfaces;  
    using Xamarin.Forms;  
    using Xamarin.Forms.Xaml;  
  
    [ContentProperty("Text")]  
    public class TranslateExtension : IMarkupExtension  
    {  
        readonly CultureInfo ci;  
        const string ResourceId = "XFML.Resources.Resource";  
  
        static readonly Lazy<ResourceManager> ResMgr =   
            new Lazy<ResourceManager>(() => new ResourceManager(  
                ResourceId,   
                typeof(TranslateExtension).GetTypeInfo().Assembly));  
  
        public TranslateExtension()  
        {  
            ci = DependencyService.Get<ILocalize>().GetCurrentCultureInfo();  
        }  
  
        public string Text { get; set; }  
  
        public object ProvideValue(IServiceProvider serviceProvider)  
        {  
            if (Text == null)   
            {  
                return "";  
            }  
  
            var translation = ResMgr.Value.GetString(Text, ci);  
  
            if (translation == null)  
            {  
#if DEBUG  
                throw new ArgumentException(  
                    String.Format(  
                        "Key '{0}' was not found in resources '{1}' for culture '{2}'.",  
                        Text, ResourceId, ci.Name), "Text");  
#else  
                translation = Text; // returns the key, which GETS DISPLAYED TO THE USER  
#endif  
            }  
  
            return translation;  
        }  
    }  
}

1. Y para usarla, hacemos esta modificación a la vista:

<?xml version="1.0" encoding="utf-8"?>  
<ContentPage   
    xmlns="http://xamarin.com/schemas/2014/forms"   
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"   
    xmlns:i18n="clr-namespace:XFML;assembly=XFML"  
    xmlns:local="clr-namespace:XFML"   
    x:Class="XFML.XFMLPage">  
   <Label  
        x:Name="MyLabel"  
        Text="{i18n:Translate Key1}"   
        VerticalOptions="Center"   
        HorizontalOptions="Center" />  
</ContentPage>

Y en el código de la página:

namespace XFML  
{  
    using Xamarin.Forms;  
  
    public partial class XFMLPage : ContentPage  
    {  
        public XFMLPage()  
        {  
            InitializeComponent();  
        }  
    }  
}

1. Prueba de nuevo, ejecutando el programa en los tres idiomas y verificando que muestre los literales correctos.