Use new tiles on Windows Phone apps using reflection

The article shows how to use reflection to create an app that can use the new tiles in Windows Phone 8, while still remaining compatible with Windows Phone 7.5

Introduction

Windows Phone 8 as well as Windows Phone 7.8 features new Tile sizes. If you want to maintain an app that targets Windows Phone 7.5, you are able to use reflection to access Windows Phone 8 APIs that showcase the new Tile functionality.

This way, if your Windows Phone 7.5 app runs on a Windows Phone that supports the new tile (WP7.8 and WP8), your application will be able to use them!

Advantages/disadvantages of this solution

Advantages:

- Only one application to maintain between Windows Phone 7 and 8
- Application will take advantage of the new tiles on Windows Phone 8 devices
- Solution is compatible with Windows Phone 7.8

Disadvantages:

- Your application will not take advantage of new controls like the native panorama or native pivot
- Your application will be displayed with a black band on 720p screen phones
- You will not be able to use the new Windows Phone 8 features such as in-app purchase, WinPRT, etc.

How do I know if my application runs on a Windows Phone 7.8/8 device ?

Before we can implement our new tiles, we first need to test the system version

7.10.8858 is the official version of Windows Phone 7.8 OS.

You only need to compare it to the version of the system, as follow :

```
    private static readonly Version _targetedVersion78 = new Version(7, 10, 8858);
    public static bool CanUseLiveTiles
    {
     get { return Environment.OSVersion.Version >= _targetedVersion78; }
    }
```

We are now able to test the version of the device and to create Windows Phone 7 or 8 tiles:

```
if(CanUseLiveTiles)
 CreateApolloTiles();
else
 CreateMangoTiles();
```

How to use reflection

To use reflection to create an object, you need first to retrieve the type of the object:

```
var mytype=Type.GetType("assembly-qualified name");
```

Then retrieve and invoke the constructor of this type:

```
var mynewtile = mytype.GetConstructor(new Type[] { }).Invoke(null);
```

"new Type[] { }" indicates that the constructor has no parameters.

Once the object is created, it's possible to assign values to its properties.

First retrieve the type of the properties of your object and ask for the property using it's name. You will get a `PropertyInfo` object containing all the information about this property. Call the accessor method `GetSetMethod()` to get the information about the accessor.

```
var setMethod = instance.GetType().GetProperty("MyProperty").GetSetMethod();
```

Finally, invoke this property using this information using the original object as a parameter and the value that you want to assign

(C) Copyright Nokia 2013. All rights reserved.

```
setMethod.Invoke(instance, new object[] { value });
```

This is quite complicated, so we'll create a method to facilitate our work:

```
internal static void SetProperty(object instance, string name, object value)
{
  var setMethod = instance.GetType().GetProperty(name).GetSetMethod();
  setMethod.Invoke(instance, new object[] { value });
}

internal static object GetProperty(object instance, string name)
{
  return instance.GetType().GetProperty(name).GetValue(instance, new Object[] { });
}
```

Using new live tiles

First, retrieve the type of `FlipTileData` and create a new instance:

```
Type flipTileDataType = Type.GetType("Microsoft.Phone.Shell.FlipTileData, Microsoft.Phone");
var mynewtile = flipTileDataType.GetConstructor(new Type[] { }).Invoke(null);
```

Then set properties:

```
SetProperty(mynewtile, "Title", title);
SetProperty(mynewtile, "Count", count);
SetProperty(mynewtile, "BackTitle", backTitle);
SetProperty(mynewtile, "BackContent", backContent);
SetProperty(mynewtile, "SmallBackgroundImage", smallBackgroundImage);
SetProperty(mynewtile, "BackgroundImage", backgroundImage);
SetProperty(mynewtile, "BackBackgroundImage", backBackgroundImage);
SetProperty(mynewtile, "WideBackgroundImage", wideBackgroundImage);
SetProperty(mynewtile, "WideBackBackgroundImage", wideBackBackgroundImage);
SetProperty(mynewtile, "WideBackContent", wideBackContent);
```

Finally, update our tile as usual:

```
ShellTile.ActiveTiles.First().Update((ShellTileData)mynewtile);
```

And that's it! The code compiles on Windows Phone 7, despite the absence of the `FlipTileData` type and will work properly under Windows Phone 8 and Windows Phone 7.8 !

Create additional tiles

With Windows Phone 7.5, the following method is normally used to add a new tile:

```
ShellTile.Create(new Uri("/test.xaml",UriKind.Relative),newtile);
```

But it doesn't work with the new tiles. In fact, Windows Phone 8 added a third parameter (`bool supportsWideTile`) to this method. We therefore need to use reflection once again to access its method:

```
public static class ShellTileExt
{
 public static void Create(Uri uri, ShellTileData tiledata, bool usewide)
 {
  Type shellTileType = Type.GetType("Microsoft.Phone.Shell.ShellTile, Microsoft.Phone");
  var createmethod = shellTileType.GetMethod("Create", new Type[] { typeof(Uri), typeof(ShellTileData), typeof(bool) });
  createmethod.Invoke(null, new object[] { uri, tiledata, usewide });
 }
}
```

So, to create a additional live tile with a new template, simply write:

```
ShellTileExt.Create(new Uri("/Test.xaml", UriKind.Relative), mytiledata, true);
```

Warning: When you call the `Create()` method, you are not allowed to use remote image. You will be able to update the tile with remote data - this limitation is only for the `Create()` method.

Simplify it

Reflection is not easy to use. So let's try to simplify to this.

In this purpose, we will create the following fake objects: `FlipTileData`, `CyclicTileData`, `IconicTileData` to abstract the use of reflection. The final purpose is to be able to write:

```
var mytile = new IconicTileData()
{
 Title = "iconic tile",
 Count = 8,
```


(C) Copyright Nokia 2013. All rights reserved.

```
  BackgroundColor = Color.FromArgb(255, 200, 10, 30),
  IconImage = new Uri("/Assets/logo202x202.png", UriKind.Relative),
  SmallIconImage = new Uri("/Assets/logo110x110.png", UriKind.Relative),
  WideContent1 = "This is an example",
  WideContent2 = "for Nokia",
  WideContent3 = "Wiki"
 };

 ShellTileExt.Create(new Uri("/Test.xaml", UriKind.Relative), mytile, true);
```

The only difference between this code and a native Windows Phone 8 code is the use of `ShellTileExt` instead of `ShellTile`.

Create fake IconicTileData object

First create the "proxy" class `IconicTileData` owning a real `IconicTileData`:

```
public class IconicTileData
{
 private readonly object _shelltiledata;

 public IconicTileData()
 {
  Type tileDataType = Type.GetType("Microsoft.Phone.Shell.IconicTileData, Microsoft.Phone");
  _shelltiledata = tileDataType.GetConstructor(new Type[] { }).Invoke(null);
 }
}
```

`_shelltiledata` is the real `IconicTileData` built with reflection. Create now the fake property to mimic the real `IconicTileData` object:

```
 public string Title
 {
  get { return (string)Utils.GetProperty(_shelltiledata, "Title"); }
  set { Utils.SetProperty(_shelltiledata, "Title", value); }
 }
```

Repeat it for each property. We need also to be able to implicit convert this object in `ShellTileData` in order to use it with `ShellTile.Update`.

```
   public static implicit operator ShellTileData(IconicTileData data)
   {
  return (ShellTileData)data._shelltiledata;
   }
```

And it's all ! We have now a fake IconicTileData linked with a real one that can be used with ShellTile.Update !

Mangopollo library

You can find sources on codeplex : http://mangopollo.codeplex.com

And you can use Nuget to add the library to your WP7 project: PM> Install-Package Mangopollo.Light

Conclusion

With reflection and this library, we are now able to use the new live tiles on a Windows Phone 8 device without porting the project. Better, even if 7.8 SDK doesn't yet exist, your application is now ready to use the new tiles.

(C) Copyright Nokia 2013. All rights reserved.

(C) Copyright Nokia 2013. All rights reserved.