Generating Live tiles for Windows Phone from Live Data

A cool new feature of Windows Phone 8 is new wide live tiles for your apps on the home screen. This article explains how to deliver live images and setup the client side to periodically update the tile images without having to use push notifications and an expensive cloud host.

Introduction

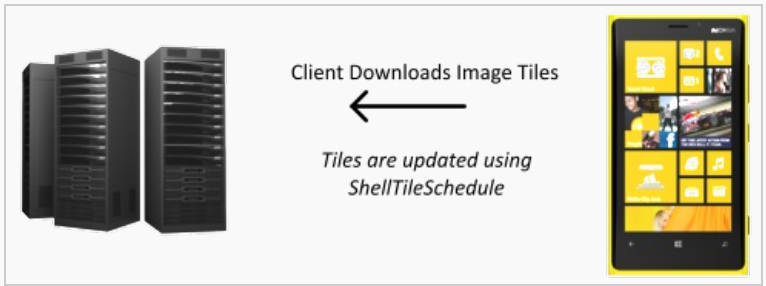
Windows Phone lets you create live tiles with text and/or graphics, and the focus of this article is to use a live graphic data feed (such as weather or traffic data) and generate multiple sizes of images that will be used as background for live tiles. There are two options to update tiles with Windows Phone.

Push Notifications

Push Notifications

The first option is to use Push Notifications for Windows Phone. To update a tile using push notifications, your app has to register with the push service and you need to setup your server side to generate the notifications when necessary. The advantage of this approach is that you can update the tiles whenever necessary (within the limits of the push service). To actively generate notifications, you need to keep a process running on the server to generate the notifications, such as an azure worker. Also it means that your app requires cloud hosting instead of a simple web hosting, and there can be quite an order of magnitude in the pricing. ASP.NET web hosting start at couple of dollars a month (flat rate), whereas cloud hosting is at least 5 times that amount (and rate depends on usage).

Client side updates



Client side updates

The second option is to rely on client side updates. Luckily, Windows Phone provides a ShellTileSchedule class for this purpose. *ShellTileSchedule* lets you schedule tile updates to update tiles periodically and is a perfect way to keep your tiles up to date. With this option, all you need is a web server that can deliver images in specific dimensions.

Setting up the server side

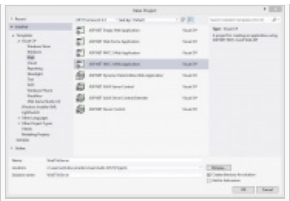
Finding a host

All your need is an ASP.NET host. The asp.net website has a long list of options that start at couple of dollars a month.

The MVC4 Project

To provide the live tiles, you need a web server that will serve the tiles to client. .NET provides many options for web projects, and we will use here the MVC model because it makes it easy to configure URLs and send images using code. The first step is to create a new MVC 4 project that will be published to your web server.

(C) Copyright Nokia 2014. All rights reserved.



1. Creating the MVC 4 project

2. Configuring MVC

In the options, all you need is an empty template.

ImageResizer

Because the live tiles have to be delivered in specific sizes, the original data source has to be transformed and possibly converted (e.g. Windows Phone doesn't like GIFs). In this example, we will use **ImageResizer**.

ImageResizer is a powerful library for manipulating images on the server side. It lets you do all kinds of operations such as cropping, resizing and even adding watermarks.

Adding ImageResizer to your project

The easiest way to add ImageResizer to your project is to use NuGet and search for the ImageResizer package online, or use the Package Manager and type

```
PM> Install-Package ImageResizer
```

The live source

To test this app, we will use a live image from the camera at *MHB-25 Manh S URW @ Twr Walkwy' in New York that can be accessed at http://207.251.86.238/cctv391.jpg*

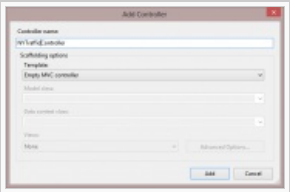
To deliver it as live tile, we will have to crop it and resize it in two sizes:

- 336 x 336 for small and medium tiles
- 336 x 961 for the wide tile

And according to this article, the image size can NOT exceed 80KB, and download time can NOT exceed 60 sec.

The controller code

We need a MVC controller to generate the images. The default MVC project contains a *Controller* folder, and the context menu has a wizard to create a new controller.



3. Creating the controller

That class needs one method for the medium icon and another one for the large:

```
    public class NYTrafficController : Controller
    {
        public FileResult Icon()
```


(C) Copyright Nokia 2014. All rights reserved.

```
        {
            return Resize(336, 336);
        }

        public FileResult IconLarge()
        {
            return Resize(336, 691);
        }

        public FileResult Resize(int height, int width)
        {
            var uri = "http://207.251.86.238/cctv391.jpg";
            var rsSettings = new ResizeSettings() { Format = "jpg", MaxHeight = height,
MaxWidth = width };
            rsSettings.Mode = FitMode.Crop;
            rsSettings.Scale = ScaleMode.Both;
            Config c = Config.Current;

            var wc = new WebClient();
            wc.CachePolicy = new
System.Net.Cache.RequestCachePolicy(System.Net.Cache.RequestCacheLevel.NoCacheNoStore);
            var dt = wc.DownloadData(uri);
            using (var ms = new MemoryStream(dt))
            {
                var mStream = new MemoryStream();
                c.CurrentImageBuilder.Build(ms, mStream, rsSettings);
                mStream.Seek(0, SeekOrigin.Begin);
                return new FileStreamResult(mStream, "image/jpeg");
            }
        }

    }
```

The final URLs for these icons will be:

- http://mywebsite/NYTraffic/Icon and,
- http://mywebsite/NYTraffic/IconLarge

The actual resizing and cropping are done by the **Resize** method. The live URL is configured there and the WebClient is configured to download the image every time and disable caching. The default settings of the jpeg will generate an image of about 45k for the wide one and about 26k for the medium one which should download fast and are under the limits.

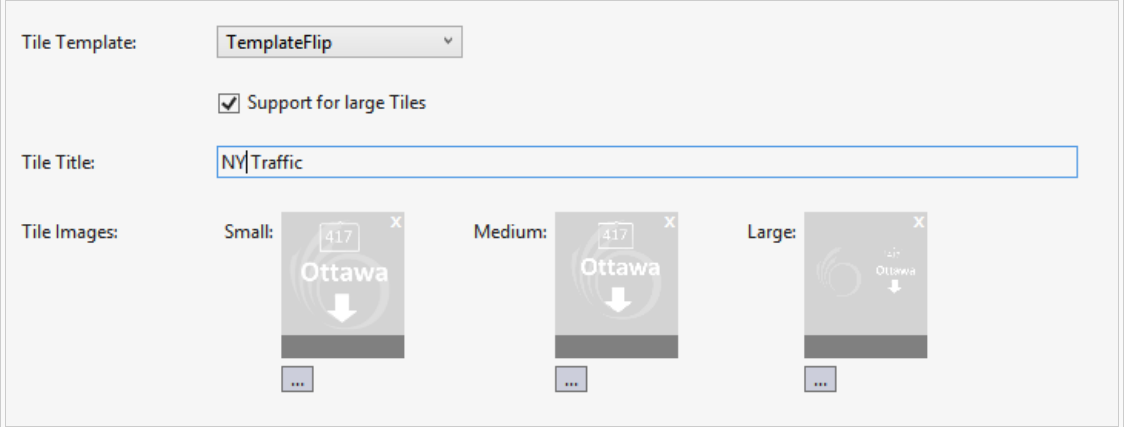
Setting up your app

Now that you have your website ready to deliver live data, all that is left is to configure one or many tiles for your app.

Manifest configuration

The first step is to set your app with FlipTiles. It seems that only FlipTiles do support remote URIs for the images. Also you need to provide icons for each tile size.

(C) Copyright Nokia 2014. All rights reserved.



Manifest setup

The secrets of ShellTileSchedule

There is one not well known constructor of **ShellTileSchedule** that takes a ShellTile and a ShellTileData. The **ShellTileData** argument is the only way to define all properties of a tile and including backgrounds with remote URLs. When this constructor is used, the **RemoteImageUri** property of the **ShellTileSchedule** will be ignored.

Limitations

The minimum update interval for the live tile is 1 hour. If your app needs more frequent updates to the background, the only alternative is to use push notifications.

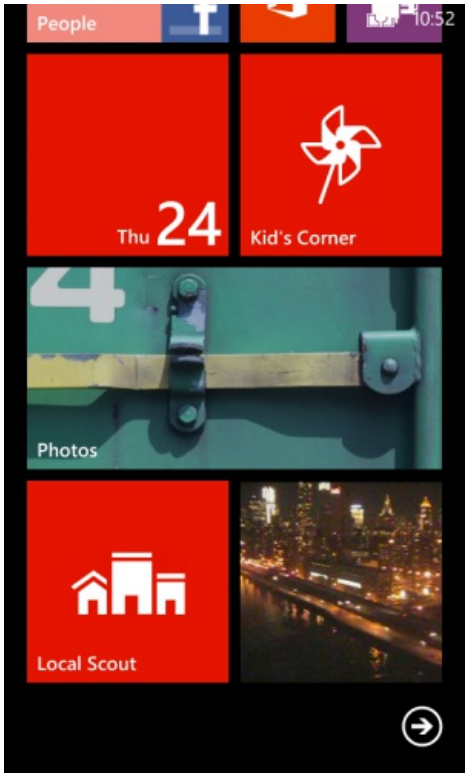
Adding ShellTileSchedule to your app

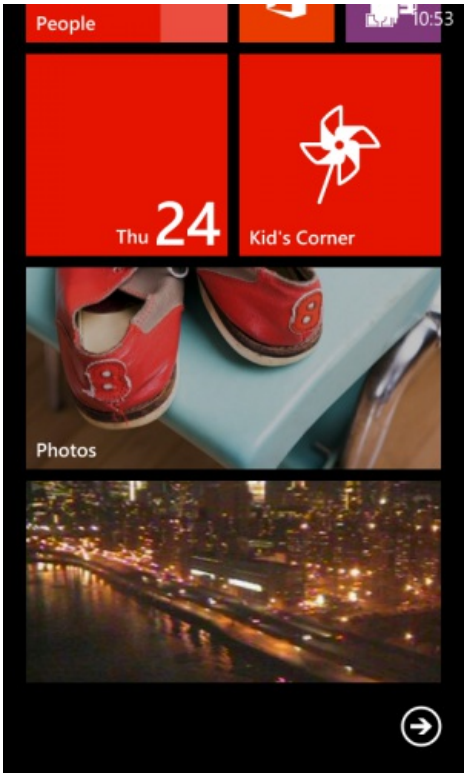
The easiest location to configure the default tile is in the constructor as shown below:

```
        public MainPage()
        {
            InitializeComponent();
            var tileId = ShellTile.ActiveTiles.FirstOrDefault();
            if (tileId != null)
            {
                var tileData = new FlipTileData();
                tileData.BackBackgroundImage= new Uri(@"
http://mywebsite/NYTraffic/Icon");
                tileData.WideBackBackgroundImage = new Uri(@"
http://mywebsite/NYTraffic/IconLarge");
                var trafficTileSchedule = new ShellTileSchedule(tileId, tileData);
                trafficTileSchedule.Recurrence = UpdateRecurrence.Interval;
                trafficTileSchedule.Interval = UpdateInterval.EveryHour;
                trafficTileSchedule.Start();
            }
        }
```

The tiles

(C) Copyright Nokia 2014. All rights reserved.





(C) Copyright Nokia 2014. All rights reserved.