Flickr: Authentication using Hammock & finding own's and others Info

This article shows how to implement a Flickr OAuth 1.0 authentication process in your Windows Phone app using Hammock library. It also covers how to fetch your own and other persons information as well from Flickr.

Introduction

Flickr API uses OAuth 1.0 standard for its users authentication. The best advantage OAuth provides is that it never lets the application access and store user's credentials. This way, security of user credentials has become far more powerful than earlier. To know more about OAuth, you may refer the official OAuth documentation here.

Implementing OAuth process is a trivial task as it requires generation of signature key, signature method key, nonce key, timestamp key, etc. To getting rid of all this work, we have a .Net library named Hammock which simplifies consuming and wrapping RESTful services. In this article we will use Hammock library to implement the Flickr authentication process.

Prerequisites

- Windows Phone development environment
- Hammock Library - get it here
- SimpleJson Library (to parse Json strings)

Registering App in Flickr

To work with Flickr, we need to register our app at Flickr.

- Log in with your Flickr account, go to *Explore* | *App Garden* | *Create an App* | *Request an API Key*.
- Fill the details of our app.
- After registering the app, *Consumer key* and *Consumer Secret* Key would be provided.
- Save these two keys as we will require them later in the project.

Click to enlarge the screenshots for the above steps.

Installing the Hammock library

Please read this article which explains how to install Hammock library in WP project.

Installing the SimpleJson library

To install SimpleJson library, we can choose either of the possible way: the UI or the Package Console.

Via Package Console Manager

To install it via Package Manager Console, simply open the Package Manager Console (Tools| Library Package Manager| Package Manager Console) and type the following command:

```
Install-Package SimpleJson
```

Via UI

To install it via UI, simply go to Tools| Library Package Manager| Manage NuGet Packages For Solution | Online. Search for "SimpleJson" and install it.

Creating UI

The UI of the project will have a single screen and three menu items: **Sign In, Find Person and Sign Out**. *Find Person* and *Sign Out* menu items will be initially disabled. The functionality of *Sign In* & *Sign Out* menu items is obvious. On successfully completing *Sign In* functionality, the user will be greeted with his/her Flickr screen name. The other two menu items will be enabled at this time.

 (C) Copyright Nokia 2013. All rights reserved.

On pressing *Find Person* menu item, a text box and a search image will be shown. In that text box, we may fill the email id of the person whose info we want to fetch. On pressing the Search button, the search functionality will be carried out. When the Search functionality will be over successfully, the resultant user's id, real name and his/her location will be displayed on screen. This process of finding person's info may be repeated if required.

The whole **Mainpage.xaml** file contents is shown below:

```
        <StackPanel x:Name="TitlePanel" Grid.Row="0" Margin="12,17,0,28">
            <TextBlock x:Name="ApplicationTitle" Text="Sample Flickr" Style="{StaticResource PhoneTextNormalStyle}"/>
        </StackPanel>

        <!--ContentPanel - place additional content here-->
        <Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">
            <Grid.RowDefinitions>
                <RowDefinition Height=".15*"/>
                <RowDefinition Height="*"/>
            </Grid.RowDefinitions>

            <phone:WebBrowser Grid.Row="0" Grid.RowSpan="2" Margin="-6,3,0,1" Name="loginBrowserControl" Visibility="Collapsed"
                              Navigated="loginBrowserControl_Navigated" Navigating="loginBrowserControl_Navigating"
                              IsScriptEnabled="True"/>

            <Grid x:Name="MainPanel" Grid.Row="0" Visibility="Visible">
                <Grid.RowDefinitions>
                    <RowDefinition Height=".25*"/>
                </Grid.RowDefinitions>

                <TextBlock Grid.Row="0" x:Name="txtUserName" VerticalAlignment="Top" HorizontalAlignment="Center" FontSize="26" Font
            </Grid>

            <Grid x:Name="EmailPanel" Grid.Row="1" Visibility="Collapsed">
                <Grid.RowDefinitions>
                    <RowDefinition Height=".08*"/>
                    <RowDefinition Height=".20*"/>
                    <RowDefinition Height="*"/>
                </Grid.RowDefinitions>

                <TextBlock Grid.Row="0" Grid.ColumnSpan="2" Text="Enter Email Id" VerticalAlignment="Center" HorizontalAlignment="Le
                <StackPanel Name="stackEmailId" Orientation="Horizontal" Grid.Row="1">
                    <TextBox Name="txtEmailId" TextWrapping="Wrap" Text="" Height="78" FontSize="24" FontFamily="Segoe WP Bold" Widt
                    <Image VerticalAlignment="Center" Name="imgQuestionMark"  Source="/SampleFlickrApp;component/Images/search.png"
                </StackPanel>
            </Grid>

            <Grid x:Name="UserPanel" Grid.Row="1" Visibility="Collapsed">
                    <Grid.RowDefinitions>
                        <RowDefinition Height=".20*"/>
                        <RowDefinition Height=".20*"/>
                        <RowDefinition Height=".20*"/>
                        <RowDefinition Height="*"/>
                    </Grid.RowDefinitions>
                    <Grid.ColumnDefinitions>
                        <ColumnDefinition Width=".60*"/>
                        <ColumnDefinition Width="*"/>
                    </Grid.ColumnDefinitions>

                    <TextBlock Grid.Row="0" Grid.Column="0" Text="User Id" VerticalAlignment="Center" HorizontalAlignment="Center" F
                    <TextBlock Grid.Row="0" Grid.Column="1" x:Name="txtUserId" VerticalAlignment="Center" HorizontalAlignment="Cente
                    <TextBlock Grid.Row="1" Grid.Column="0" Text="Real Name" VerticalAlignment="Center" HorizontalAlignment="Center"
                    <TextBlock Grid.Row="1" Grid.Column="1" x:Name="txtUserRealName" VerticalAlignment="Center" HorizontalAlignment=
                    <TextBlock Grid.Row="2" Grid.Column="0" Text="Location" VerticalAlignment="Center" HorizontalAlignment="Center"
                    <TextBlock Grid.Row="2" Grid.Column="1" x:Name="txtLocation" VerticalAlignment="Center" HorizontalAlignment="Cen
                </Grid>
        </Grid>

    </Grid>

    <!--Sample code showing usage of ApplicationBar-->
    <phone:PhoneApplicationPage.ApplicationBar>
        <shell:ApplicationBar IsVisible="True" IsMenuEnabled="True">
            <shell:ApplicationBar.MenuItems>
                <shell:ApplicationBarMenuItem Text="Sign In"     Click="MenuItemSignIn_Click"/>
                <shell:ApplicationBarMenuItem Text="Find Person" Click="MenuItemFindPerson_Click" IsEnabled="False"/>
                <shell:ApplicationBarMenuItem Text="Sign Out"    Click="MenuItemSignOut_Click" IsEnabled="False"/>
            </shell:ApplicationBar.MenuItems>
        </shell:ApplicationBar>
    </phone:PhoneApplicationPage.ApplicationBar>
```

Code Behind

In this article, we have created some .cs files to kept some of our functions and other stuff. These are:

- **AppSettings.cs** - to keep the Flickr related URLs and other keys.
- **MainUtils.cs** - to set/get keys in/from Isolated Storage.
- **OAuthUtil.cs** - to fetch the request token query which would be of *OAuthWebQuery* type (defined in Hammock), inherited from WebQuery class.
- **UserIdentity.cs** - to save the result of current (logged in) user
- **UserInfo.cs** - to save the result of any other user

AppSettings.cs File's contents

This file has following declaration.

```
public class AppSettings
    {
        public static string RequestTokenUri = "http://www.flickr.com/services/oauth/request_token";
        public static string AuthorizeUri = "http://www.flickr.com/services/oauth/authorize";
        public static string AccessTokenUri = "http://www.flickr.com/services/oauth/access_token";
        public static string CallbackUri = "http://www.google.com";

        #error TODO REGISTER YOUR APP WITH FLICKR TO GET YOUR KEYS AND FILL THEM IN HERE
        public static string consumerKey = "";
        public static string consumerKeySecret = "";
```

 (C) Copyright Nokia 2013. All rights reserved.

```
        public static string oAuthVersion = "1.0";

        //general Strings
        public static string TEXT_WELCOME = "Welcome ";
        public static string SIGNED_OUT_SUCCESSFULLY = "You have been signed out successfully.";
    }
```

MainUtil.cs File's contents

This file has following declaration.

```
 public class MainUtil
    {
        public static Dictionary<string, string> GetQueryParameters(string response)
        {
            Dictionary<string, string> nameValueCollection = new Dictionary<string, string>();
            string[] items = response.Split('&');

            foreach (string item in items)
            {
                if (item.Contains("="))
                {
                    string[] nameValue = item.Split('=');
                    if (nameValue[0].Contains("?"))
                        nameValue[0] = nameValue[0].Replace("?", "");
                    nameValueCollection.Add(nameValue[0], System.Net.HttpUtility.UrlDecode(nameValue[1]));
                }
            }
            return nameValueCollection;
        }

        internal static T GetKeyValue<T>(string key)
        {
            if (IsolatedStorageSettings.ApplicationSettings.Contains(key))
                return (T)IsolatedStorageSettings.ApplicationSettings[key];
            else
                return default(T);
        }

        internal static void SetKeyValue<T>(string key, T value)
        {
            if (IsolatedStorageSettings.ApplicationSettings.Contains(key))
                IsolatedStorageSettings.ApplicationSettings[key] = value;
            else
                IsolatedStorageSettings.ApplicationSettings.Add(key, value);
            IsolatedStorageSettings.ApplicationSettings.Save();
        }
    }
```

In this class, the method of get/set keys in Isolated Storage defined. Also, one method is also defined to fetch the key pair values from a String.

OAuthUtil.cs File's contents

This file has following declaration.

```
public class OAuthUtil
    {
        internal static OAuthWebQuery GetRequestTokenQuery()
        {
            var oauth = new OAuthWorkflow
            {
                ConsumerKey = AppSettings.consumerKey,
                ConsumerSecret = AppSettings.consumerKeySecret,
                SignatureMethod = OAuthSignatureMethod.HmacSha1,
                ParameterHandling = OAuthParameterHandling.HttpAuthorizationHeader,
                RequestTokenUrl = AppSettings.RequestTokenUri,
                Version = AppSettings.oAuthVersion,
                CallbackUrl = AppSettings.CallbackUri
            };

            var info = oauth.BuildRequestTokenInfo(WebMethod.Get);
            var objOAuthWebQuery = new OAuthWebQuery(info, false);
            objOAuthWebQuery.HasElevatedPermissions = true;
            objOAuthWebQuery.SilverlightUserAgentHeader = "Hammock";
            return objOAuthWebQuery;
        }
    }
```

UserIdentity.cs File's contents

We will make use of this class when we will get the current user's information. This file has following declaration.

```
public class UserIdentity
    {
        public string userId { get; set; }
        public string userNSID { get; set; }
        public string userName { get; set; }
    }
```

UserInfo.cs File's contents

We will make use of this class when we will get the info of any other user. The structure of both the API's methods response is different, so we've created two files here. This file has following declaration.

```
public class UserInfo
    {
```

 (C) Copyright Nokia 2013. All rights reserved.

```
        public string userId { get; set; }
        public string userNSID { get; set; }
        public string userName { get; set; }
        public string userRealName { get; set; }
        public string userLocation { get; set; }
        public string userProfileUrl { get; set; }
    }
```

OAuth Authentication Process

- A request is sent to server to fetch the Request Token and Request Token Secret Keys.
- By Request Token and it's key, Flickr Login page is opened for the user to enter the credentials.
- If the user logged in successfully then a Verifier Pin is returned in a HTML response which needs to be parsed and pin needs to be saved.
- A request is sent using Request Token, Request Token Secret Key and the Verifier Pin, this time to fetch the permanent Access Token and Access Token Key. In case all goes well, then the two keys are returned back to the user.
- Flickr APIs methods can be invoked by using the Access Token and Access Token Key.

So, next we'll write code to do all the above process.

MainPageXaml.cs File's contents

- Using Directives:

```
using Hammock.Web;
using System.IO;
using Hammock.Authentication.OAuth;
using Hammock;
using System.Text;
using SampleFlickrApp.data;
```

- Variables Declared:

```
        string OAuthTokenKey = string.Empty;
        string tokenSecret = string.Empty;
        string accessToken = string.Empty;
        string accessTokenSecret = string.Empty;
        string userScreenName = string.Empty;

        UserInfo userObj;
```

When a user presses **Sign In**, then we are just instantiating an asyncronous request to fetch the Request Token and Request Token Secret Key.

```
 private void MenuItemSignIn_Click(object sender, EventArgs e)
        {
            accessToken = MainUtil.GetKeyValue<string>("AccessToken");
            accessTokenSecret = MainUtil.GetKeyValue<string>("AccessTokenSecret");
            userScreenName = MainUtil.GetKeyValue<string>("ScreenName");

            if (string.IsNullOrEmpty(accessToken) || string.IsNullOrEmpty(accessTokenSecret))
            {
                var requestTokenQuery = OAuthUtil.GetRequestTokenQuery();
                requestTokenQuery.RequestAsync(AppSettings.RequestTokenUri, null);
                requestTokenQuery.QueryResponse += new EventHandler<WebQueryResponseEventArgs>(requestTokenQuery_QueryResponse);
            }
            else
            {
                Dispatcher.BeginInvoke(() =>
                {
                    var SignInMenuItem = (Microsoft.Phone.Shell.ApplicationBarMenuItem)this.ApplicationBar.MenuItems[0];
                    SignInMenuItem.IsEnabled = false;

                    var FindPersonMenuItem = (Microsoft.Phone.Shell.ApplicationBarMenuItem)this.ApplicationBar.MenuItems[1];
                    FindPersonMenuItem.IsEnabled = true;

                    var SignOutMenuItem = (Microsoft.Phone.Shell.ApplicationBarMenuItem)this.ApplicationBar.MenuItems[2];
                    SignOutMenuItem.IsEnabled = true;

                    MainPanel.Visibility = System.Windows.Visibility.Visible;
                    txtUserName.Text = AppSettings.TEXT_WELCOME + userScreenName;
                });
            }
        }

        void requestTokenQuery_QueryResponse(object sender, WebQueryResponseEventArgs e)
        {
            try
            {
                StreamReader reader = new StreamReader(e.Response);
                string strResponse = reader.ReadToEnd();
                var parameters = MainUtil.GetQueryParameters(strResponse);
                OAuthTokenKey = parameters["oauth_token"];
                tokenSecret = parameters["oauth_token_secret"];
                var authorizeUrl = AppSettings.AuthorizeUri + "?oauth_token=" + OAuthTokenKey;

                Dispatcher.BeginInvoke(() =>
                {
                    this.loginBrowserControl.Navigate(new Uri(authorizeUrl, UriKind.RelativeOrAbsolute));
                });
            }
            catch (Exception ex)
            {
                Dispatcher.BeginInvoke(() =>
                {
                    MessageBox.Show(ex.Message);
                });
            }
        }
```

Using the above code, we've created and fired a async request to get the request token and request token secret key. If the response is successfully received then those keys are saved in the Isolated storage. Next, the browser is redirected to the authorize url by passing the request token key. On the web browser's onNavigating event, we will receive the Verifier pin which we will save and then use to fetch the access token key and access token secret key. If the request is made successfully, then along with the token key and secret key, we'll also get the user name which we will save.

 (C) Copyright Nokia 2013. All rights reserved.

The whole code of above mentioned process is given below:

```
private void loginBrowserControl_Navigated(object sender, System.Windows.Navigation.NavigationEventArgs e)
        {
            this.loginBrowserControl.Visibility = Visibility.Visible;
            this.loginBrowserControl.Navigated -= loginBrowserControl_Navigated;
        }

        private void loginBrowserControl_Navigating(object sender, NavigatingEventArgs e)
        {
            if (e.Uri.ToString().StartsWith(AppSettings.CallbackUri))
            {
                var AuthorizeResult = MainUtil.GetQueryParameters(e.Uri.ToString());
                var VerifyPin = AuthorizeResult["oauth_verifier"];
                this.loginBrowserControl.Visibility = Visibility.Collapsed;

                getAccessToken(VerifyPin);
            }
        }

        private void getAccessToken(String aVerifyPin)
        {
            var credentials = new OAuthCredentials
            {
                Type = OAuthType.AccessToken,
                SignatureMethod = OAuthSignatureMethod.HmacSha1,
                ParameterHandling = OAuthParameterHandling.HttpAuthorizationHeader,
                ConsumerKey = AppSettings.consumerKey,
                ConsumerSecret = AppSettings.consumerKeySecret,
                Token = this.OAuthTokenKey,
                TokenSecret = this.tokenSecret,
                Verifier = aVerifyPin,
                Version = "1.0"
            };

            var client = new RestClient
            {
                Authority = "http://www.flickr.com/services/oauth",
                HasElevatedPermissions = true
            };

            var request = new RestRequest
            {
                Credentials = credentials,
                Path = "/access_token",
                Method = WebMethod.Post
            };

            client.BeginRequest(request, new RestCallback(AccessTokenRequestCallback));
        }

        private void AccessTokenRequestCallback(RestRequest request, RestResponse response, object obj)
        {
            try
            {
                string respContent = response.Content;
                var parameters = MainUtil.GetQueryParameters(respContent);
                accessToken = parameters["oauth_token"];
                accessTokenSecret = parameters["oauth_token_secret"];
                userScreenName = parameters["username"];

                MainUtil.SetKeyValue<string>("AccessToken", accessToken);
                MainUtil.SetKeyValue<string>("AccessTokenSecret", accessTokenSecret);
                MainUtil.SetKeyValue<string>("ScreenName", userScreenName);

                Dispatcher.BeginInvoke(() =>
                {
                    var SignInMenuItem = (Microsoft.Phone.Shell.ApplicationBarMenuItem)this.ApplicationBar.MenuItems[0];
                    SignInMenuItem.IsEnabled = false;

                    var FindPersonMenuItem = (Microsoft.Phone.Shell.ApplicationBarMenuItem)this.ApplicationBar.MenuItems[1];
                    FindPersonMenuItem.IsEnabled = true;

                    var SignOutMenuItem = (Microsoft.Phone.Shell.ApplicationBarMenuItem)this.ApplicationBar.MenuItems[2];
                    SignOutMenuItem.IsEnabled = true;

                    MainPanel.Visibility = System.Windows.Visibility.Visible;
                    txtUserName.Text = AppSettings.TEXT_WELCOME + userScreenName;
                });
            }
            catch (Exception ex)
            {
                Dispatcher.BeginInvoke(() =>
                {
                    MessageBox.Show(ex.Message);
                });
            }
        }
```

Finding Person's info

According to this article, finding a person's info is a two step process (you can just use the first one if you want only the name and NSID of person).

1. We can make use of Flickr API's flickr.people.findByEmail method to find a person by his/her email Id. In response, we will get the person's name and his/her NSID. Further this NSID can be used to fetch other detailed info.
2. To fetch the detailed info of a user, we will have to make use of Flickr API's flickr.people.getInfo method. This method will give us the user's Id, NSID, screen name, real name, location, profile URL etc.

Note: There is another possible way to find a person which is using flickr.people.findByUsername. You may use this method as well rather than finding by an email id.

On pressing "Find Person" menu item, we will display a text box and a search button. On entering email id and pressing search button, we will carry out the functionality of above mentioned step 1.

 (C) Copyright Nokia 2013. All rights reserved.

The code is:

```
    private void MenuItemFindPerson_Click(object sender, EventArgs e)
        {
            //fetching User's NSID first
            UserPanel.Visibility = System.Windows.Visibility.Collapsed;
            EmailPanel.Visibility = System.Windows.Visibility.Visible;
        }

        private void imgQuestionMark_Tap(object sender, System.Windows.Input.GestureEventArgs e)
        {
            String aEmailId = txtEmailId.Text.Trim();
            if (aEmailId != "")
            {
                FindPersonNSIDByEmailID(aEmailId);
            }
            else
            {
                MessageBox.Show("Enter the Email Id to be searched", "Message", MessageBoxButton.OK);
                txtEmailId.Focus();
            }
        }

        private void FindPersonNSIDByEmailID(String aEmailId)
        {
            WebClient client = new WebClient();
            client.DownloadStringCompleted += new DownloadStringCompletedEventHandler(client_DownloadStringUserNSIDCompleted);
            string URL = string.Format("http://www.flickr.com/services/rest/?method=flickr.people.findByEmail&api_key={0}&format=jso
                AppSettings.consumerKey, aEmailId);
            client.DownloadStringAsync(new Uri(URL));
        }

        private void client_DownloadStringUserNSIDCompleted(object sender, DownloadStringCompletedEventArgs e)
        {
            if (e.Error != null)
            {
                MessageBox.Show("error");
            }
            else
            {
                String response = e.Result.ToString();
                parseUserNSIDInfo(fetchCorrectJson(response));
            }
        }
```

As you can see above, we are sending the API key of our project and the required person's email id as parameter. Also, we are mentioning that we want the response in JSON format. You may set it to *XML* as well.

The response string we receive after making API's functions hits is not the JSON string, it contains some extra text at both ends. So we need to first fetch the JSON string out of that response string. For that, we have declared a function named fetchCorrectJson which will trim the unnecessary characters from both sides of the response string for us. The code used is:

```
 private string fetchCorrectJson(string aString)
        {
            int totalResplength = aString.Length;
            int firstPos = aString.IndexOf('{');
            int secPos = aString.LastIndexOf('}');

            int desiredLen = totalResplength - firstPos - (totalResplength - secPos - 1);
            return (aString.Substring(firstPos, desiredLen));
        }
```

After finding the JSON string out of the total response string, we'll fetch the JSON string and save the required values. The code used to carry out this purpose is mentioned below:

```
  private void parseUserNSIDInfo(string aRespString)
        {
            UserIdentity userObj = new UserIdentity();

            IDictionary<string, object> rootObj = (IDictionary<string, object>)SimpleJson.DeserializeObject(aRespString);
            string statResult = (string)rootObj["stat"];

            if (statResult.Equals("ok"))
            {
                IDictionary<string, object> user = (IDictionary<string, object>)rootObj["user"];

                userObj.userId = (string)user["id"];
                userObj.userNSID = (string)user["nsid"];

                IDictionary<string, object> userName = (IDictionary<string, object>)user["username"];
                userObj.userName = (string)userName["_content"];

                fetchUserInfo(userObj.userNSID);
            }
            else
            {
                string errorMsg = (string)rootObj["message"];
                MessageBox.Show(errorMsg);
            }
        }
```

As shown above, as soon as we are saving the user's info, we are calling another function to get the detailed info of the user. For this, we are passing the NSID value of the user. The further code is:

```
  private void fetchUserInfo(String aNSID)
        {
            WebClient client = new WebClient();
            client.DownloadStringCompleted += new DownloadStringCompletedEventHandler(client_DownloadStringUserInfoCompleted);
            string URL = string.Format("http://www.flickr.com/services/rest/?method=flickr.people.getInfo&api_key={0}&format=json&us
                AppSettings.consumerKey, aNSID);
            client.DownloadStringAsync(new Uri(URL));
        }

        private void client_DownloadStringUserInfoCompleted(object sender, DownloadStringCompletedEventArgs e)
        {
```

 (C) Copyright Nokia 2013. All rights reserved.

```
            if (e.Error != null)
            {
                MessageBox.Show("error");
            }
            else
            {
                String response = e.Result.ToString();
                parseUserInfo(fetchCorrectJson(response));
            }
        }
```

On successfully receiving the response, we need to again parse the JSON string to fetch the required information. The code for that is:

```
private void parseUserInfo(string aRespString)
        {
            userObj = new UserInfo();

            IDictionary<string, object> rootObj = (IDictionary<string, object>)SimpleJson.DeserializeObject(aRespString);
            string statResult = (string)rootObj["stat"];

            if (statResult.Equals("ok"))
            {
                IDictionary<string, object> person = (IDictionary<string, object>)rootObj["person"];

                userObj.userId = (string)person["id"];
                userObj.userNSID = (string)person["nsid"];

                IDictionary<string, object> userName = (IDictionary<string, object>)person["username"];
                userObj.userName = (string)userName["_content"];

                IDictionary<string, object> userRealName = (IDictionary<string, object>)person["realname"];
                userObj.userRealName = (string)userRealName["_content"];

                IDictionary<string, object> location = (IDictionary<string, object>)person["location"];
                userObj.userLocation = (string)location["_content"];

                IDictionary<string, object> profileUrl = (IDictionary<string, object>)person["profileurl"];
                userObj.userProfileUrl = (string)profileUrl["_content"];

                // updating UI
                EmailPanel.Visibility = System.Windows.Visibility.Collapsed;
                UserPanel.Visibility = System.Windows.Visibility.Visible;
                txtUserId.Text = userObj.userId;
                txtUserRealName.Text = userObj.userRealName;
                txtLocation.Text = userObj.userLocation;
            }
            else
            {
                string errorMsg = (string)rootObj["message"];
                MessageBox.Show(errorMsg);
            }
        }
```

The *Sign Out* menu item's code is mentioned below where we are just clearing the values stored in Isolated Storage and setting the other un-required panels visibility to *Collapsed* again.

```
 private void MenuItemSignOut_Click(object sender, EventArgs e)
        {
            MainUtil.SetKeyValue<string>("AccessToken", string.Empty);
            MainUtil.SetKeyValue<string>("AccessTokenSecret", string.Empty);
            MainUtil.SetKeyValue<string>("ScreenName", userScreenName);

            Dispatcher.BeginInvoke(() =>
            {
                var SignInMenuItem = (Microsoft.Phone.Shell.ApplicationBarMenuItem)this.ApplicationBar.MenuItems[0];
                SignInMenuItem.IsEnabled = true;

                var FindPersonMenuItem = (Microsoft.Phone.Shell.ApplicationBarMenuItem)this.ApplicationBar.MenuItems[1];
                FindPersonMenuItem.IsEnabled = false;

                var SignOutMenuItem = (Microsoft.Phone.Shell.ApplicationBarMenuItem)this.ApplicationBar.MenuItems[2];
                SignOutMenuItem.IsEnabled = false;

                MainPanel.Visibility = System.Windows.Visibility.Collapsed;
                EmailPanel.Visibility = System.Windows.Visibility.Collapsed;
                UserPanel.Visibility = System.Windows.Visibility.Collapsed;
                txtUserId.Text = "";
                txtUserRealName.Text = "";
                txtLocation.Text = "";

                MessageBox.Show(AppSettings.SIGNED_OUT_SUCCESSFULLY);
            });
        }
```

Build and Run

Now you may build the app and try to run it.

References

- http://nuget.org/
- http://oauth.net/documentation/getting-started/
- http://nuget.org/packages/Hammock
- http://nuget.org/packages/simplejson
- http://www.flickr.com/services/api/

 (C) Copyright Nokia 2013. All rights reserved.

 (C) Copyright Nokia 2013. All rights reserved.