Twitter: OAuth 1.0 Authentication using Hammock

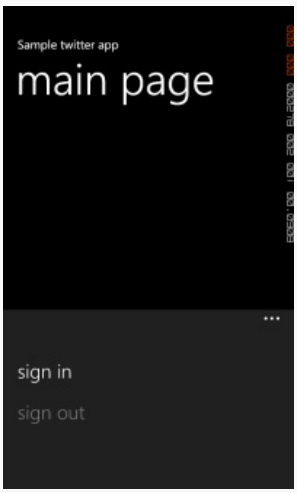
This article will help you in making a Twitter OAuth 1.0 authentication process in your Windows Phone application using Hammock library

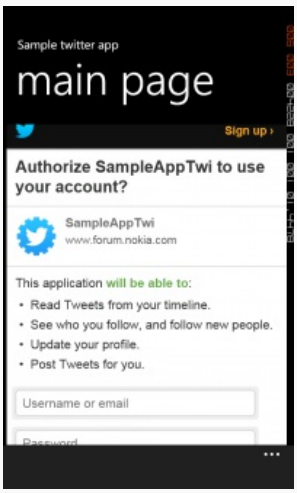
Introduction

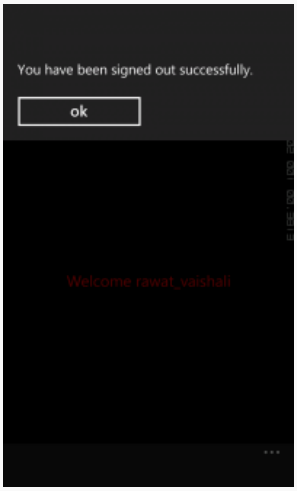
Twitter API uses OAuth 1.0 standard for its users authentication. The best advantage OAuth provides is that it never lets the application keep and store the users credentials. It means a developer has no direct access to the credentials of the users. This way Security of users credentials has become 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 a lot of stuff generation like signature key, signature method key, nonce key, timestamp key etc. To getting rid of all this work, we have a library named Hammock which is a .Net library that simplifies consuming and wrapping RESTful services. So, in this article we will use Hammock library to implement the twitter authentication process.

The screens in this project are as shown below.







Prerequisites

- Windows Phone development environment.
- Hammock library

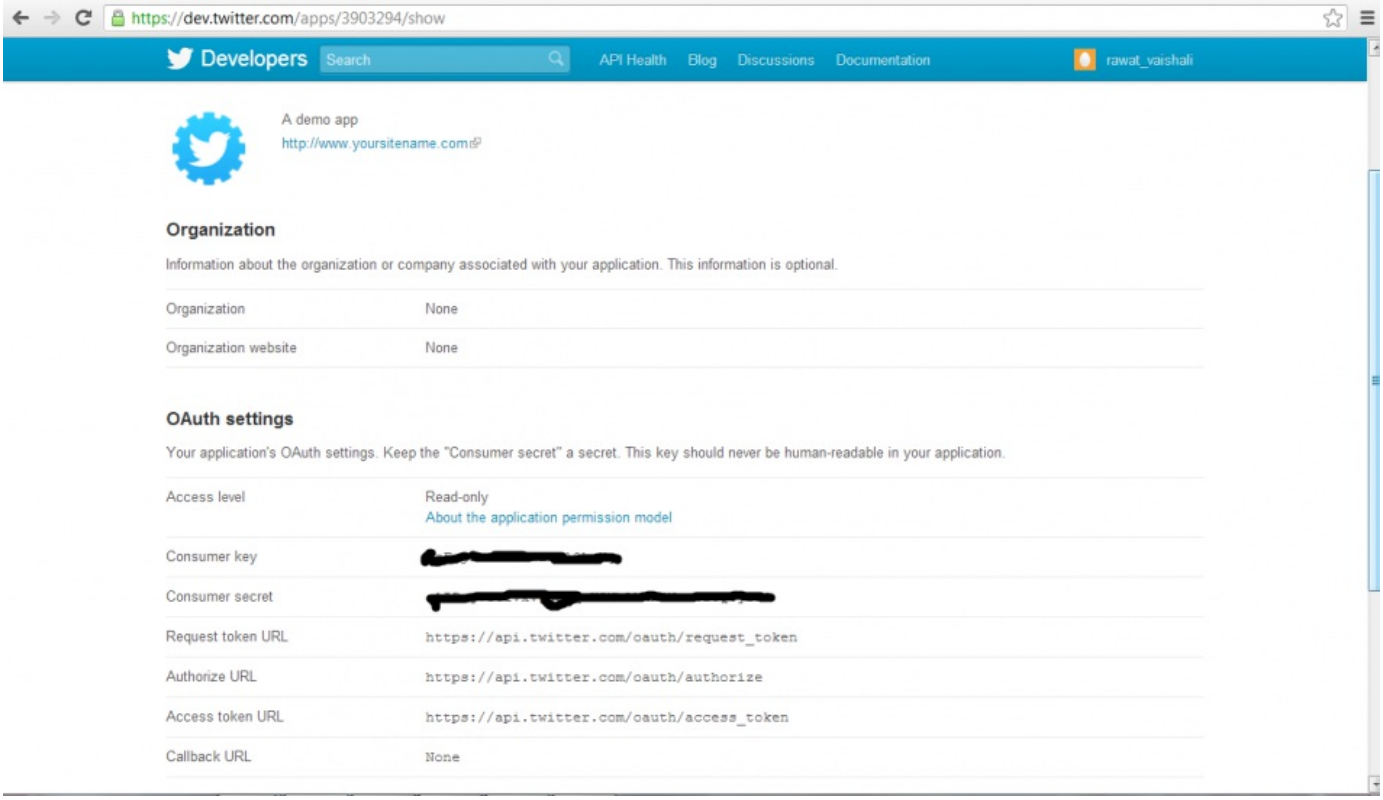
Registering App in Twitter

To work with Twitter, we have to register our app at Twitter Developer Site. To do so, after we successfully logged in in Twitter, we've to go to **My Applications | Create a New Application**. Now we have to fill the details of our app.

After registering the app, we will be provided with some URLs that we will be using in making OAuth requests and Consumer key and Consumer Secret

(C) Copyright Nokia 2013. All rights reserved.

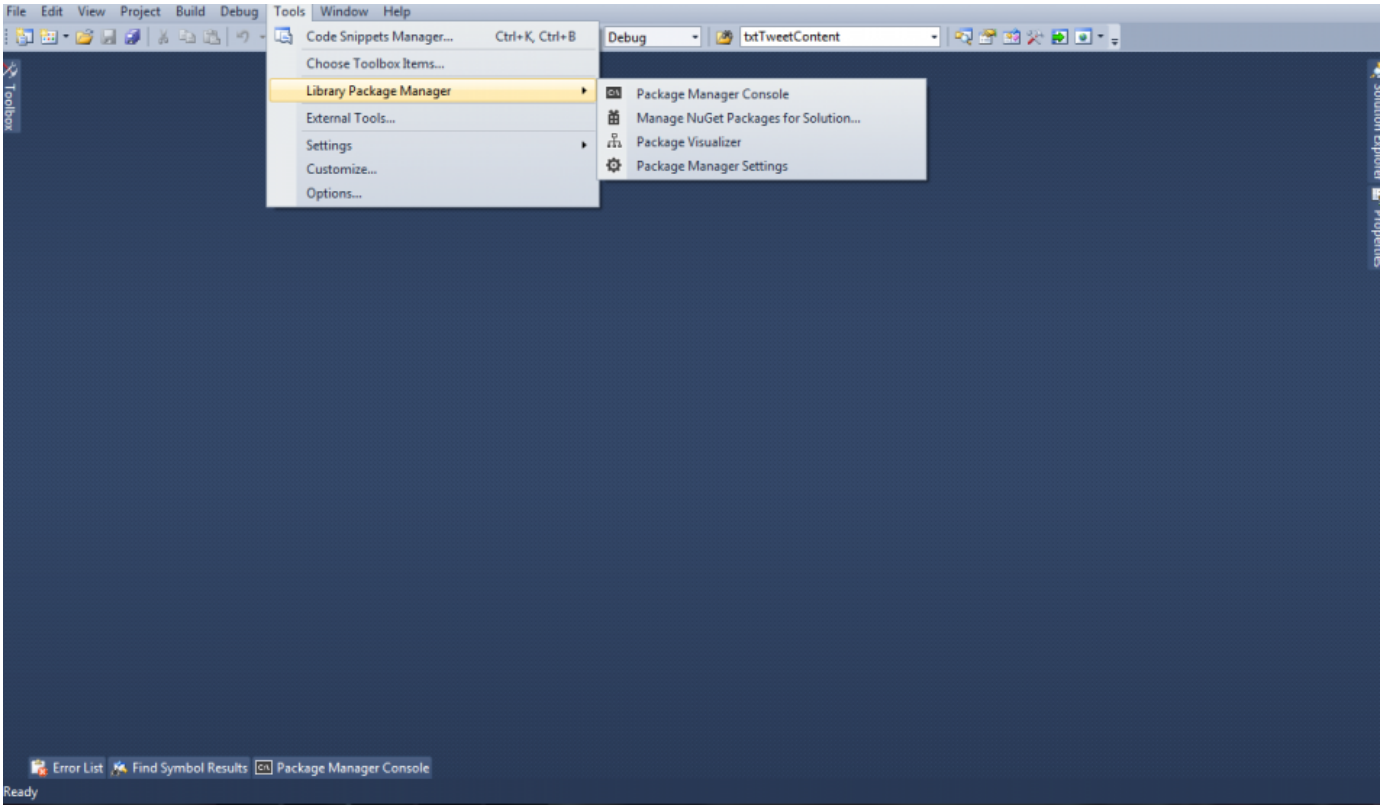
Key. Save the two keys somewhere as we will require them later in the project.



Installing Hammock Library

NuGet is a Visual Studio extension that helps us install and use third party libraries and tools easily in Visual Studio. To install Hammock library, we should have the option of *Library Package Manager* in our **Visual Studio | Tools**. This option doesn't come by default with the Visual Studio version which comes with WP7 SDK. So, first of all we need to install NuGet itself. NuGet Package Manger can be downloaded from here.

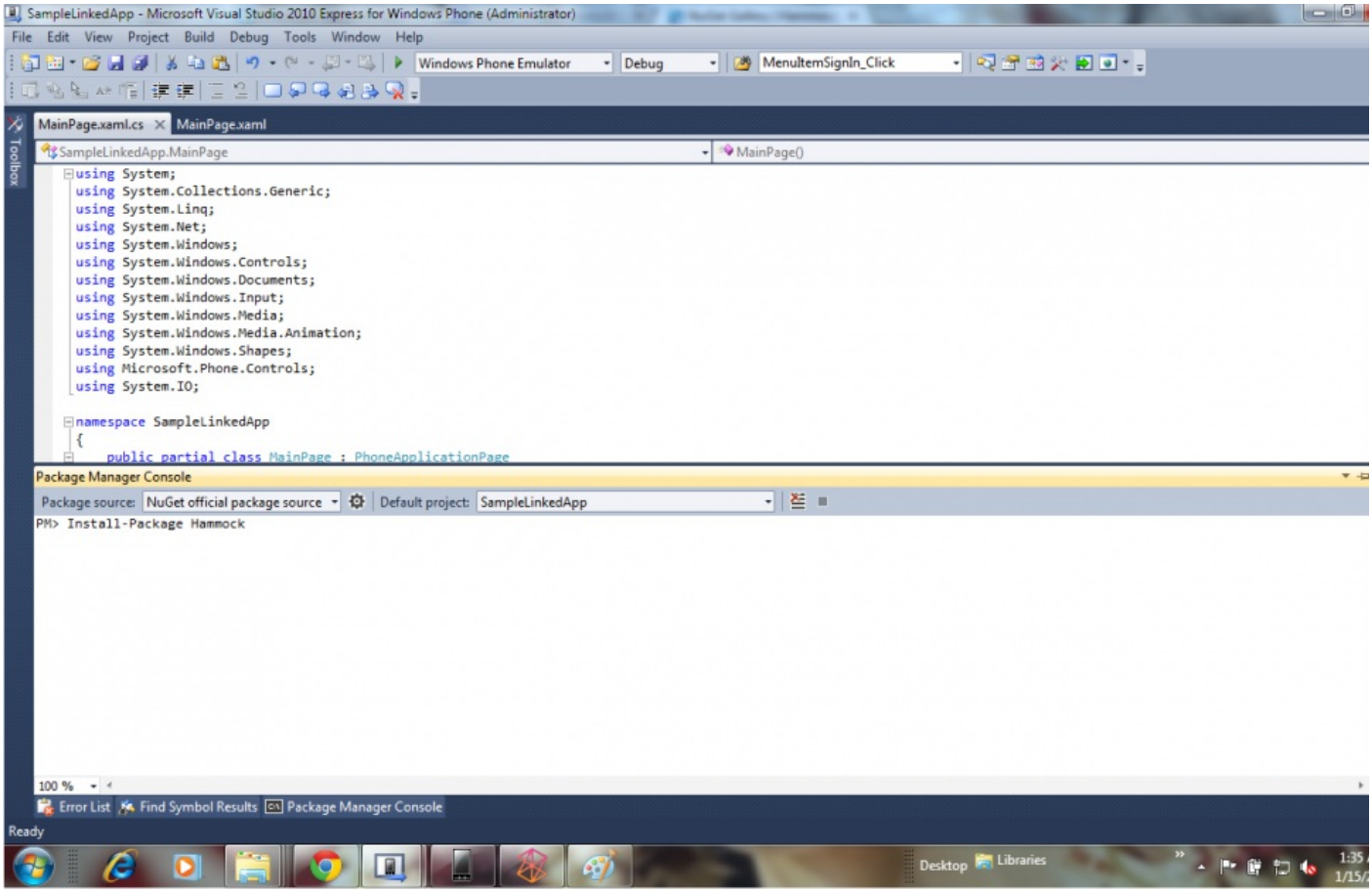
After downloading NuGet .exe file, we need to install it and restart our Visual Studio (in case it's already running). Now the option of **Tools | Library Package Manager** should exist in our Visual Studio.



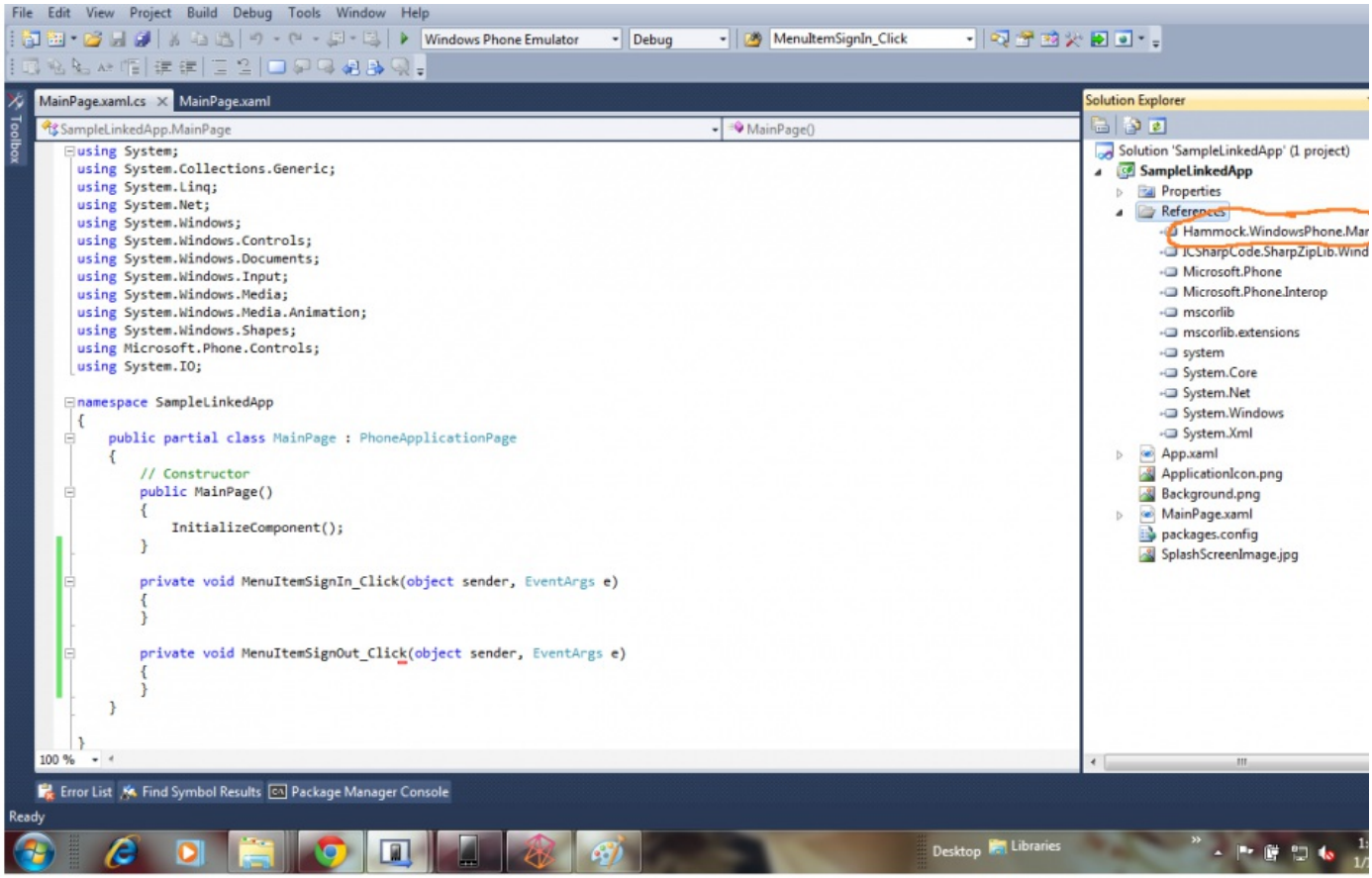
To install Hammock, go to **Tools | Library Package Manager| Package Manager Console**. Run the following command on the console screen, make sure that any of your solution should be opened corresponding to which Hammock will be installed.

```
Install-Package Hammock
```


(C) Copyright Nokia 2013. All rights reserved.



Now Hammock library should be installed for your project. You may test it via Project Explorer >> **Your Project | References**. It should look like the screen shown below.



After the Hammock installation is done, we may now proceed with further things.

Creating UI

We will just make one screen which will have two menu items named *Sign In* and *Sign Out*. When a user will press Sign In button, a web browser will be

(C) Copyright Nokia 2013. All rights reserved.

shown which will be redirected to the Twitter login page. After successful authentication, we will hide the web browser control and will show a textbox in the middle of the screen to welcome the user by showing his/her name in Twitter. So, the xaml code would be like:

```
        <!--TitlePanel contains the name of the application and page title-->
        <StackPanel x:Name="TitlePanel" Grid.Row="0" Margin="12,17,0,28">
            <TextBlock x:Name="ApplicationTitle" Text="Sample twitter app" Style="{StaticResource PhoneTextNormalStyle}"/>
            <TextBlock x:Name="PageTitle" Text="main page" Margin="9,-7,0,0" Style="{StaticResource PhoneTextTitle1Style}"/>
        </StackPanel>

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

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

            <Grid x:Name="TweetPanel" Grid.Row="0" Visibility="Collapsed">
                <TextBlock x:Name="txtUserName" VerticalAlignment="Center" HorizontalAlignment="Center" FontSize="26" FontFamily="Se
           </Grid>
        </Grid>
```

As shown above, we have kept the web browser visibility to collapsed by default, so that it can be visible on pressing Sign In menu item. We have also defined the Web Browser's events *Navigated* and *Navigating*.

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 Twitter related URLs and other keys.
- **MainUtils.cs** - to set/get keys in/from Isolated Storage.
- **OAuthUtil.cs** - to fetch the request token and access token queries. These queries would be of *OAuthWebQuery* type (defined in Hammock), inherited from WebQuery class.

AppSettings.cs File's contents

This file has following declaration.

```
    public class AppSettings
    {
        // twitter
        public static string RequestTokenUri = "https://api.twitter.com/oauth/request_token";
        public static string AuthorizeUri = "https://api.twitter.com/oauth/authorize";
        public static string AccessTokenUri = "https://api.twitter.com/oauth/access_token";
        public static string CallbackUri = "http://www.google.com";   // we've mentioned Google.com as our callback URL.

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

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

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();
         }
     }
```


(C) Copyright Nokia 2013. All rights reserved.

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;
        }

        internal static OAuthWebQuery GetAccessTokenQuery(string requestToken, string RequestTokenSecret, string oAuthVerificationPi
        {
            var oauth = new OAuthWorkflow
            {
                AccessTokenUrl = AppSettings.AccessTokenUri,
                ConsumerKey = AppSettings.consumerKey,
                ConsumerSecret = AppSettings.consumerKeySecret,
                ParameterHandling = OAuthParameterHandling.HttpAuthorizationHeader,
                SignatureMethod = OAuthSignatureMethod.HmacSha1,
                Token = requestToken,
                Verifier = oAuthVerificationPin,
                Version = AppSettings.oAuthVersion
            };

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

Via above defined functions, we will fetch the Request Token and Access Token queries. For that, Hammock code id used.

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, Twitter 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.
- Twitter 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;
using System.Text;
using Hammock.Authentication.OAuth;
```

- Variables Declared:

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

In the Main Page's constructor, we are checking if the user is already logged in. If yes, then simply a welcome text is shown else a request is made to access the first two tokens.

```
  public MainPage()
        {
            InitializeComponent();
            if (isAlreadyLoggedIn())
            {
                userLoggedIn();
            }
        }

        private Boolean isAlreadyLoggedIn()
        {
            accessToken = MainUtil.GetKeyValue<string>("AccessToken");
            accessTokenSecret = MainUtil.GetKeyValue<string>("AccessTokenSecret");
```


(C) Copyright Nokia 2013. All rights reserved.

```
            userScreenName = MainUtil.GetKeyValue<string>("ScreenName");

            if (string.IsNullOrEmpty(accessToken) || string.IsNullOrEmpty(accessTokenSecret))
                return false;
            else
                return true;
        }

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

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

                TweetPanel.Visibility = System.Windows.Visibility.Visible;
                txtUserName.Text = "Welcome " + userScreenName;
            });
        }
```

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)
        {
            var requestTokenQuery = OAuthUtil.GetRequestTokenQuery();
            requestTokenQuery.RequestAsync(AppSettings.RequestTokenUri, null);
            requestTokenQuery.QueryResponse += new EventHandler<WebQueryResponseEventArgs>(requestTokenQuery_QueryResponse);
        }

        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);
                });
            }
        }

        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;
                var AccessTokenQuery = OAuthUtil.GetAccessTokenQuery(OAuthTokenKey, tokenSecret, VerifyPin);

                AccessTokenQuery.QueryResponse += new EventHandler<WebQueryResponseEventArgs>(AccessTokenQuery_QueryResponse);
                AccessTokenQuery.RequestAsync(AppSettings.AccessTokenUri, null);
            }
        }

        void AccessTokenQuery_QueryResponse(object sender, WebQueryResponseEventArgs e)
        {
            try
            {
                StreamReader reader = new StreamReader(e.Response);
                string strResponse = reader.ReadToEnd();
                var parameters = MainUtil.GetQueryParameters(strResponse);
                accessToken = parameters["oauth_token"];
                accessTokenSecret = parameters["oauth_token_secret"];
                userID = parameters["user_id"];
                userScreenName = parameters["screen_name"];

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

                userLoggedIn();
            }
            catch (Exception ex)
            {
                Dispatcher.BeginInvoke(() =>
                {
                    MessageBox.Show(ex.Message);
                });
            }
        }
```

Via above code, we have first of all made an asyncronous request to fetch the Request Token and Request Token Secret Key.We have defined an event handler to handle the situation when a response will arrive. From the response arrived, we've tried to fetch the required Token and Secret Token.

After doing so, we have created a Authorization (Login) URL to which the web browser control will be redirected. The web browser control will open the Twitter Login page. After a successful logged in, the control will return back to our application and while doing so, we will fetch the Verifier Pin. This Verifier Pin will be then used to make another asyncronous request to fetch the final and the permanent **Access Token** and **Access Token Key**. When this query's response will come, we will parse the response in key-pair values and find the required ones out of the available. On successfully finding them, we

(C) Copyright Nokia 2013. All rights reserved.

will save them in the Isolated Storage. At this time, we are hiding the web browser control and showing a welcome message to the user by greeting him/her name.

To carry out Sign Out functionality, we just have to clear out the saved values in Isolated Storage like shown below:

```
private void MenuItemSignOut_Click(object sender, EventArgs e)
        {
            MainUtil.SetKeyValue<string>("AccessToken", string.Empty);
            MainUtil.SetKeyValue<string>("AccessTokenSecret", string.Empty);
            Dispatcher.BeginInvoke(() =>
            {
                var SignInMenuItem = (Microsoft.Phone.Shell.ApplicationBarMenuItem)this.ApplicationBar.MenuItems[0];
                SignInMenuItem.IsEnabled = true;

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

                TweetPanel.Visibility = System.Windows.Visibility.Collapsed;

                MessageBox.Show("You have been signed out successfully.");
            });
        }
```

Build and Run

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

References

- http://nuget.org/
- http://nuget.org/packages/Hammock
- http://oauth.net/documentation/getting-started/
- https://dev.twitter.com/docs/auth/oauth

(C) Copyright Nokia 2013. All rights reserved.

(C) Copyright Nokia 2013. All rights reserved.