Dropbox with Windows Phone

This article explains how to connect Windows Phone to DropBox cloud service using SharpBox

Introduction

SharpBox is an open source project that provides access to *Dropbox* cloud services on Windows Phone. This article covers basic functionality of Sharpbox library: how to use it and how to wrap its synchronous functions to work with Windows Phone. This provides a useful complement or addition to Microsoft's SkyDrive cloud services.

Note: SharpBox also provides access to other cloud services like *CloudMe*. However at time of writing this article these services work on Desktop only (not Windows Phone)

Using SharpBox

Getting SharpBox

SharpBox is available as a NuGet package or for download from codeplex here. If you download a Sharpbox release, extract the downloaded zip file. In Visual Studion add **AppLimit.CloudComputing.SharpBox.dll** and **Newtonsoft.Json.Silverlight.dll** files from **sl3-wp** folder as reference to your project.

Warning: The newer release (1.2) has few bugs when logging and authenticating to Dropbox service. Mainly, token exchanging uses synchronous functions which do not work in Windows Phone. Earlier versions have functions to log in with credentials, those will work only with older Dropbox v0 API. If you have Dropbox key & secret for older API, older library works just fine. But if you create Dropbox application now, you will get Dropbox v1 key and secret and those do not work with older API.

Prerequisites

SharpBox needs a valid Dropbox application key and secret (for release, not required during testing and developing).These can be obtained in the developer section of the Dropbox website.

SharpBox should not run in UI thread

The SharpBox library has both synchronous and and asynchronous functions for many operations. Both forms are useful for desktop usage, but the synchronous versions cannot be used directly in Windows Phone apps because these calls block the running UI thread and therefore the whole application.

The asynchronous functions can be used in the UI thread. If needed functions only exist in synchronous variants it is possible to run these in another (non UI) thread and return the results using a callback.

The following snippet shows how to use Dispatcher to call `parseFilesAndDirectories` function - a normal private function in application thread. Example function `CallbackFunction()` is asynchronous callback function.

```
void CallbackFunction(IAsyncResult result)
{
    Deployment.Current.Dispatcher.BeginInvoke(() =>
    {
    if (fs != null)
        {
         parseFilesAndDirectories(fs);
        }
    });
}
```

Logging to Dropbox

This is pretty easy, create credentials and configuration object. Then call `BeginOpenRequest()` and wait for callback.

Dropbox API v0, Sharpbox 1.1 and older

```
public void ConnectCloud(string username, string password)
    {
            DropBoxCredentials creds = new DropBoxCredentials();
            creds.ConsumerSecret = APP_SECRET;
            creds.ConsumerKey = APP_KEY;
            creds.UserName = username;
            creds.Password = password;

            m_dropBox.BeginOpenRequest(LoginCallback, mCloudConfig, creds);
    }
```

Dropbox API v1, Sharpbox 1.2

From Sharpbox 1.2, usage of **DropBoxTokenIssuer.exe** has been deprecated. Instead the access token is generated in code.

Getting Security Token

Generate a request token

In the first phase a request token has to be generated based on the app key and app secret issued by Dropbox in your developer account interface. The interface will be available after registration under developer console.

Create an application if you haven't already.

```
 //create a request token
 DropBoxRequestToken requestToken =
DropBoxStorageProviderTools.GetDropBoxRequestToken(config,
 <<YOUR APP KEY>>,
 <<YOUR APP SECRET>>);
```

Generate the authorization URL and visit it to allow access

In phase 2 the generated authorization URL has to be visited in a web browser control. The URL will be returned from SharpBox with the following code fragment.

```
// call the authorization url via WebBrowser Plugin
String AuthorizationUrl = DropBoxStorageProviderTools.GetDropBoxAuthorizationUrl(config,
requestToken);
```

Note: For testing purpose you can put a breakpoint at `AuthorizationUrl` and manually browse to it to allow or deny access.

Exchange the request token into an access token

The last phase converts the request token (which is only be usable during the authorization process) into an issued access token which can be stored on a local cache and has to be reused during login phase when the user comes back with your application. The following code performs the exchange process:

```
// create the access token
ICloudStorageAccessToken accessToken =
DropBoxStorageProviderTools.ExchangeDropBoxRequestTokenIntoAccessToken(config, <<YOUR
APP KEY>>, <<YOUR APP SECRET>>, requestToken);
```

Saving access token for future use

The generated token can be stored in the *IsolatedStorage* for use in all future transactions to connect to Dropbox.

```
private void SaveAccessTokenToIsolatedStorage(ICloudStorageAccessToken accessToken)
{
 using (var store = IsolatedStorageFile.GetUserStoreForApplication())
 {
  using (var stream = new IsolatedStorageFileStream(FileName, FileMode.Create,
FileAccess.Write, store))
  {
   Stream accessTokenStream;
   accessTokenStream = cloudStorage.SerializeSecurityToken(accessToken);
   stream.Flush();
   byte[] accessTokenBytes;
   using (var streamReader = new MemoryStream())
   {
    accessTokenStream.CopyTo(streamReader);
    accessTokenBytes = streamReader.ToArray();
    stream.Write(accessTokenBytes, 0, accessTokenBytes.Length);
   }
  }
 }
}
```

Note: `CloudStorage.Open()` must be called before attempting to serialize access token due to some issue with SharpBox library.

Reading Access token from IsolatedStorage

```
Use below snippet to read back access token from ''IsolatedStorage''.
private ICloudStorageAccessToken LoadAccessTokenFromIsolatedStorage()
{
 ICloudStorageAccessToken accessToken = null;
 using (var store = IsolatedStorageFile.GetUserStoreForApplication())
 {
  using (var stream =
     new IsolatedStorageFileStream(FileName, FileMode.OpenOrCreate, FileAccess.Read,
store))
  {
   //using (var reader = new StreamReader(stream))
   {
    var cloudStorage = new CloudStorage();
    accessToken =
(ICloudStorageAccessToken)cloudStorage.DeserializeSecurityToken(stream);
   }
  }
 }
 return accessToken;
}
```

Login callback

This is the callback function for the open request. The Dropbox connection token is extracted from the asynchronous function's result. This token also determines whether the connection succeeded; if the value is null, connection failed and with other values

it succeeded.

The token is required because without it the the SharpBox library does not know there is a service and will not continue.

```
void LoginCallback(IAsyncResult result)
    {
            ICloudStorageAccessToken token = m_dropBox.EndOpenRequest(result);
            if (token != null)
            {
                m_dropBox.BeginGetRootRequest(RootCallback);
            }
            else if (m_dropBox.IsOpened)
            {
                m_dropBox.BeginGetRootRequest(RootCallback);
            }
    }
```

Get Dropbox file listing

After successful connection, we can proceed to reading files from Dropbox. As seen in `LoginCallback()`, `RootCallback()` function is callback for `BeginGetRootRequest()` asynchronous function. Before reading any file from Dropbox, you need to know root where start. After having root, it's possible to read files from anywhere. So reading root folder is important only in the beginning. And from reading root callback, again jump to another async callback function.

```
private void RootCallback(IAsyncResult result)
    {
     ICloudDirectoryEntry root = m_dropBox.EndGetRootRequest(result);
            if (root != null)
            {
                m_dropBox.BeginGetChildsRequest(ChildCallback, root);
            }
     }
```

After having child objects of root, any file or directory in topmost level, start parsing results.

```
private void ChildCallback(IAsyncResult result)
    {

    List<ICloudFileSystemEntry> fs = m_dropBox.EndGetChildsRequest(result);
    Deployment.Current.Dispatcher.BeginInvoke(() =>
    {
    if (fs != null)
        {
        parseFilesAndDirectories(fs);
        while (m_directories.Count > 0)
            {
            ICloudDirectoryEntry e = m_directories[0];
            m_directories.RemoveAt(0);
            m_dropBox.BeginGetChildsRequest(ChildCallback, e);
            }
        }
    });
    }
```

And how to loop results list and populate file list. Example code looks only image files which are ending to JPG.

```
private void parseFilesAndDirectories(List<ICloudFileSystemEntry> direntry)
    {
    foreach (ICloudFileSystemEntry entry in direntry)
            {
                System.Diagnostics.Debug.WriteLine("entry: " + entry.Name);
                if (entry is ICloudDirectoryEntry)
                {
                    m_directories.Add((ICloudDirectoryEntry)entry);
                }

                if (entry is ICloudFileSystemEntry && entry.Name.EndsWith(".jpg",
StringComparison.CurrentCultureIgnoreCase))
                {
                    Files.Add(new CloudItem() { Name = entry.Name, Entry = entry });
                }
    }
}
```

Downloading files

Although SharpBox has synchronous function to get file system objects from Dropbox, those won't work in Windows Phone because again, synchronous functions will block UI-thread and therefore whole application. To fix this problem, implement and asynchronous wrapper for the synchronous function. This wrapping code will run the synchronous function in separate thread.

Implement helper functions, `GetFileUri` is entry point to start work. This function constructs request and sets callback function to thread, what is sent to `ThreadPool`.

```
public void GetFileUri(AsyncCallback callback, ICloudFileSystemEntry entry)
    {
    BackgroundRequest request = new BackgroundRequest();
    request.callback = callback;
    request.result = new AsyncResultEx(request);
    request.fileEntry = entry;
    ThreadPool.QueueUserWorkItem(GetFileUriCallback, request);
    }
```

After a while, callback function will be fired and we can try to get Dropbox file object URL. Note that this is not in your application (UI-thread) anymore. Use dispatcher if you need to communicate the URL to your application.

```
private void GetFileUriCallback(object state)
    {
    BackgroundRequest req = state as BackgroundRequest;

            try
            {
                req.OperationResult =
m_dropBox.GetFileSystemObjectUrl(req.fileEntry.Name, req.fileEntry.Parent);
            }
            catch (Exception e)
            {
                var openRequest = req.result.AsyncState as BackgroundRequest;
                openRequest.OperationResult = null;
                openRequest.errorReason = e;
            }
```

```
    req.callback(req.result);
    }
```

Also a final function is needed to actually return result of thread:

```
public Uri EndGetFileUri(IAsyncResult result)
    {
    BackgroundRequest req = result.AsyncState as BackgroundRequest;
    return req.OperationResult as Uri;
    }
```

Now `GetFileUri()` can be used to get file URL.

```
protected override void OnNavigatedTo(System.Windows.Navigation.NavigationEventArgs e)
    {
            base.OnNavigatedTo(e);
            CloudHandler ch = Application.Current.Resources["CloudHandler"] as
CloudHandler;
            if (ch != null)
            {
                imageProgress.IsIndeterminate = true;
                ch.GetFileUri(UrlCallback, ch.CurrentItem.Entry);
            }
    }
```

Almost there! The last callback function can provide more handling to URL what you get. This example code gets the URL and then downloads image file, later shows it.

```
void UrlCallback(IAsyncResult result)
    {
            Uri fileUri = result as Uri;

            Deployment.Current.Dispatcher.BeginInvoke(() =>
            {
                CloudHandler ch = Application.Current.Resources["CloudHandler"] as
CloudHandler;
                fileUri = ch.EndGetFileUri(result);
                BitmapImage bi = new BitmapImage();
                bi.DownloadProgress += new
EventHandler<DownloadProgressEventArgs>(bi_DownloadProgress);
                bi.UriSource = fileUri;
                currentImage.Source = bi;
            });
    }
```

Download sample code

The attached code was create using SharpBox 1.2: File:DropBoxImages.zip

To try it out, firts build and run, then click login.

For older Sharpbox releases, look `ConnectCloud` function in **Cloudhandler.cs**, change commented code. Also uncomment code to display text boxes for username and password. Send values of these fields to `ConnectCloud` function in **Cloudhandler.cs**. Build and run, type your credentials and hit login.

Useful links

- Dispatcher documentation (MSDN)
- Sharpbox website
- Dropbox developer docs
- NuGet