Dropbox with Windows Phone

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

This article needs to be updated: If you found this article useful, please fix the problems below then delete the {{ArticleNeedsUpdate}} template from the article to remove this warning.

Reasons: hamishwillee
(14 Jun 2013)
As per the article comments there are issues with using this article. Would be good to get it verified against current platform and versions of 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 libary: 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.

SharpBox prerequisities

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();
```

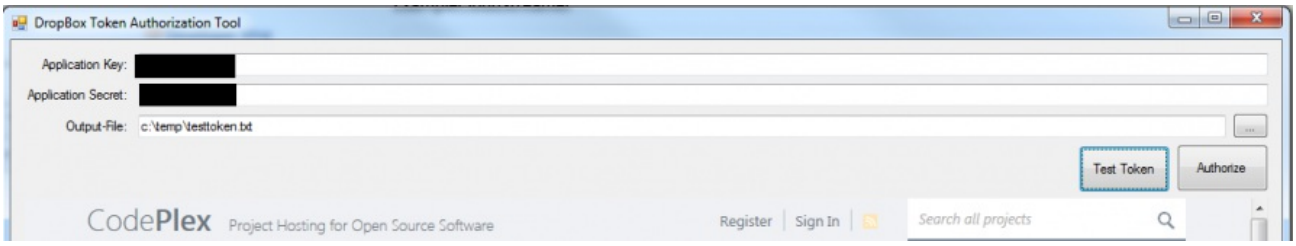
```
            creds.ConsumerSecret = APP_SECRET;
            creds.ConsumerKey = APP_KEY;
            creds.UserName = username;
            creds.Password = password;

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

Dropbox API v1, Sharpbox 1.2

DropBox wants application linked to your account and the user must allow or deny access to it. At the same time, tokens can be exchanged. For exchanging, SharpBox provides small utility tool which can be found from folder: **<your project>\packages\AppLimit.CloudComputing.SharpBox.1.2.0.542\lib\net40-full**

The tool is a desktop application called **DropBoxTokenIssuer.exe**. The tool will ask for DropBox application key and secret, output is plain text and you need to provide an empty text file and put its path to output file section. Click the **authorize button**, in-app browser goes to DropBox and asks your credentials. Log in and link your application. After that you have token in TXT-file.



Add the token text file to your project as resource. The `Login()` function is bit different, now you don't need credentials but you will load token and authorize SharpBox with that.

```
public void ConnectCloud(string username, string password)
    {
    ICloudStorageAccessToken token = null;
    StreamResourceInfo resource = Application.GetResourceStream(new Uri(@"/DropBoxImages;component/token.txt", UriKind.Relative));
    if (resource != null)
        {
        token = m_dropBox.DeserializeSecurityToken(resource.Stream);
        if (token != null)
            {
            m_dropBox.BeginOpenRequest(LoginCallback, mCloudConfig, token);
            }
        }
    }
```

Login callback

And the callback function for open request. Token of DropBox connection is extracted from asynchronous functions result. This token also determines did connection succeed; if value is null, connection failed and in other values it succeeded. Without having token value, cannot continue because SharpBox library does not know is service there.

```
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;
            });
    }
```

Sample code

Sample code is using SharpBox 1.2. To try it out, create token and add it to project. Change token file name to token.txt or change code in **Cloudhandler.cs** to load it with different name. Build and run, click login. For older Sharpbox releases, look `ConnectCloud` function in **Cloudhandler.cs**, change commented code. Build and run, type your credentials and hit login.

Download sample code

File:DropBoxImages.zip

Useful links

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