sharpbox a cloud storage programming interface

SharpBox Quick Start Guide

Version 1.2

Table of Contents

Document:	Quick Start Guide (Tutorial)
Author:	Dirk Eisenberg (dirk dot Eisenberg at gmail dot com)
License:	MIT as part of the SharpBox Project

Who should use SharpBox?

SharpBox is an abstraction library which supports access to service providers who offer cloud based block or file storage. The major idea is to provide a unique interface on all platforms where the .NET Framework (Mono as well) is available to make it as easy as possible to write platform independent software. The current SharpBox support is focused on the following platforms and providers:

Platforms:

- Windows XP, Vista, 7 or higher
- Windows Server 2003 or higher
- MAC OS X (with Mono required)
- iPhone OS 4 or higher (Monotouch required)
- Android (Experimental, Monodroid required)
- Windows Phone 7 (Experimental)

Providers:

- DropBox (includes DropBox for Teams)
- Box.NET
- StoreGate
- 1&1 SmartDrive
- Any WebDAV provider (with Standard Authentication)

If you are planning to build an application which needs access to cloud storage for one or more platforms described above and you want to achieve the goal to be provider independent, SharpBox will be your first choice. You will get a stable API which abstracts the complexity of the vendor specific interfaces and gives you an unique access path to your files.

What can SharpBox do for you?

The SharpBox API offers an unique access path to your cloud files based on a stable interface. This fact makes it possible to write applications which are using the same piece of code to get access to DropBox and/or StoreGate and/or Box.NET. Complex storage algorithms, e.g. compression can be implemented totally platform independent without the need to know how the cloud provider implements his interface. SharpBox also supports special features, e.g. metadata and hash codes in an abstract way.

Where do you find the things you need?

As an open source project the home of SharpBox is a subpage at the codeplex platform. Additional resources and downloads are available at sharpbox.codeplex.com. The discussion board and the issue tracker can be used to send feedback to the SharpBox development team.

Heads up – First steps with SharpBox

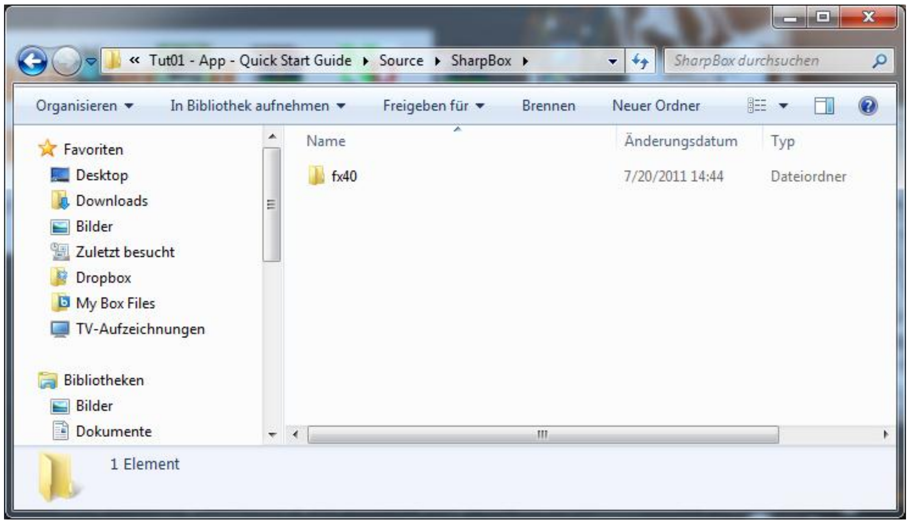
Prerequisites

This tutorial is based on Visual Studio 2010 Professional but can also be done in the Visual Studio Express Edition. Every sample is linked against the .NET Framework 4.0 Full Profile; currently the Client Profile is not supported. In addition to this a valid developer account at DropBox is necessary to try the examples. Please create a new application and a new API key/secret pair in the DropBox developer portal. This information will be used in the examples of this tutorial. If you own an existing API key/secret pair it can be reused for the examples as well.

Creating a Visual Studio project which uses SharpBox

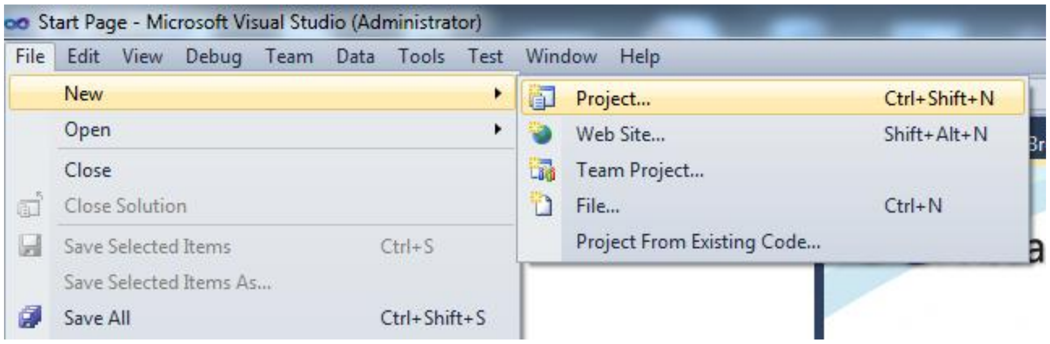
To create a SharpBox based projects the following steps are necessary:

1. Download the latest stable version of SharpBox from sharpbox.codeplex.com and extract the archive into a folder of your choice. The result should look like this picture:

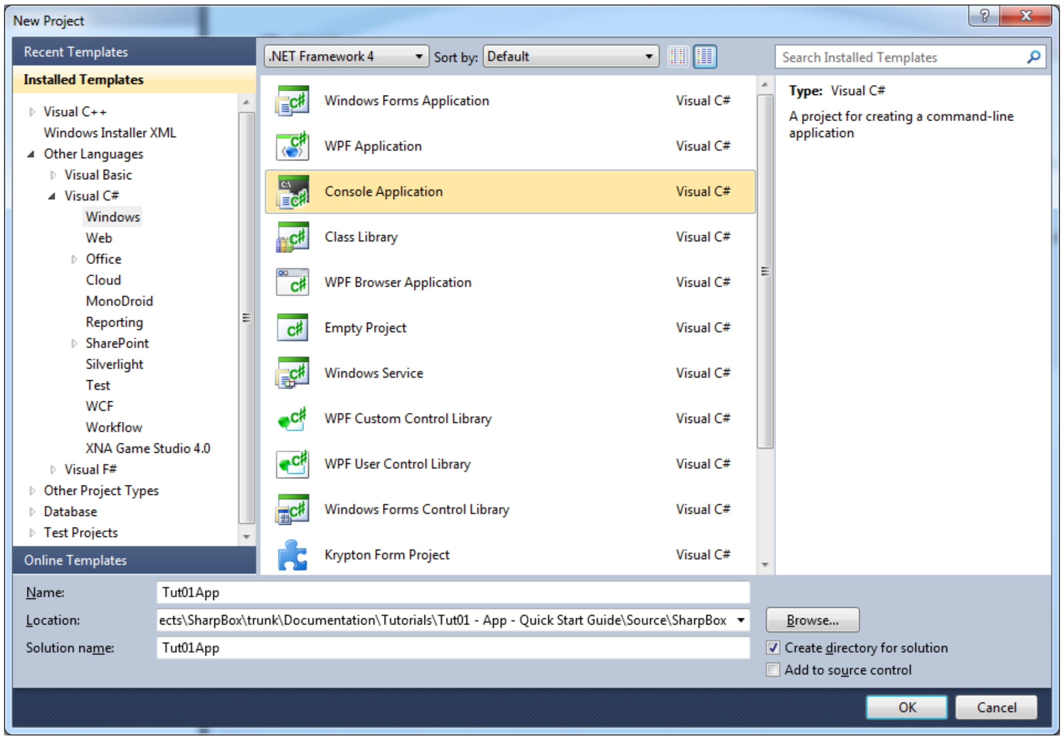


2. Start visual studio and create a new project, e.g. Windows Forms, WPF or Console Application. SharpBox is compatible with all project templates and can be used for most types of applications. In this tutorial a Console Application will be generated as follows:

   a. Create a new project in Visual Studio:


Start Page - Microsoft Visual Studio (Administrator)
File Edit View Debug Team Data Tools Test Window Help
New
Open
Close
Close Solution
Save Selected Items Ctrl+S
Save Selected Items As...
Save All Ctrl+Shift+S
Project... Ctrl+Shift+N
Web Site... Shift+Alt+N
Team Project...
File... Ctrl+N
Project From Existing Code...

   b. Select "Console Application" as project type:



Type: Visual C#

A project for creating a command-line application

Name: Tut01App

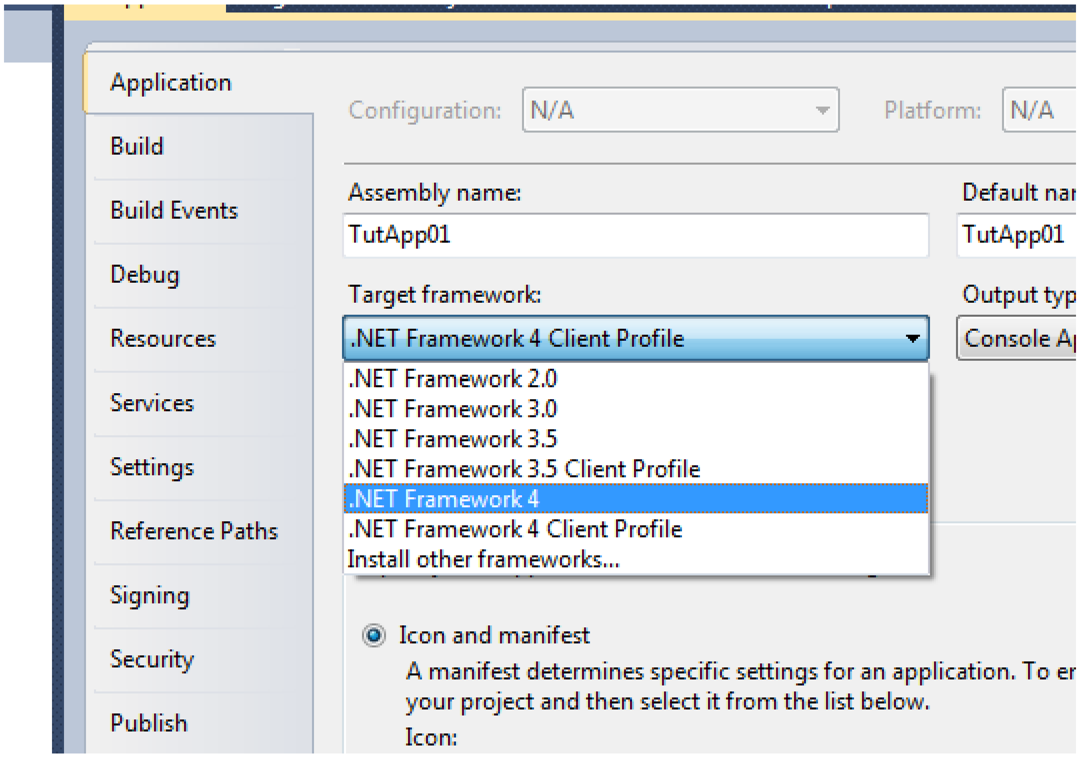
Location: ects\SharpBox\trunk\Documentation\Tutorials\Tut01 - App - Quick Start Guide\Source\SharpBox

Solution name: Tut01App

[✓] Create directory for solution

[ ] Add to source control

3. After creating the project with Visual Studio 2010 the client profile of the .NET Framework 4.0 is used. SharpBox needs a full profile because of using different network specific classes which are not available in the Client Profile. The profile type has to be changed in the properties of the newly created project:



Configuration: N/A Platform: N/A

Assembly name: TutApp01

Default nar TutApp01

Target framework: .NET Framework 4 Client Profile

Output typ Console Ap

.NET Framework 2.0
.NET Framework 3.0
.NET Framework 3.5
.NET Framework 3.5 Client Profile
.NET Framework 4
.NET Framework 4 Client Profile
Install other frameworks...

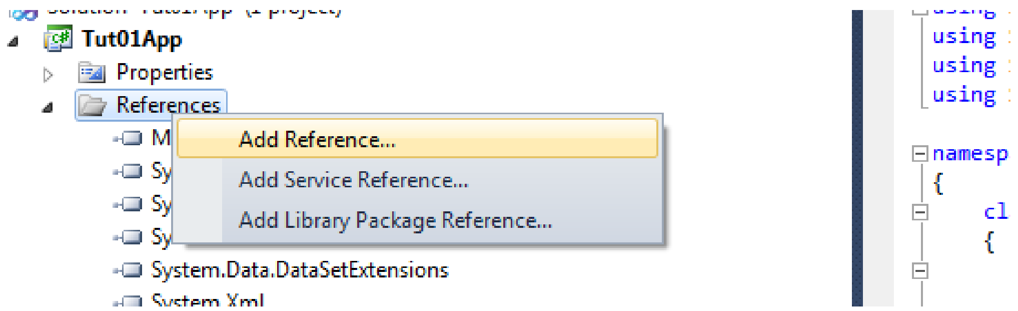
Icon and manifest

A manifest determines specific settings for an application. To er your project and then select it from the list below.

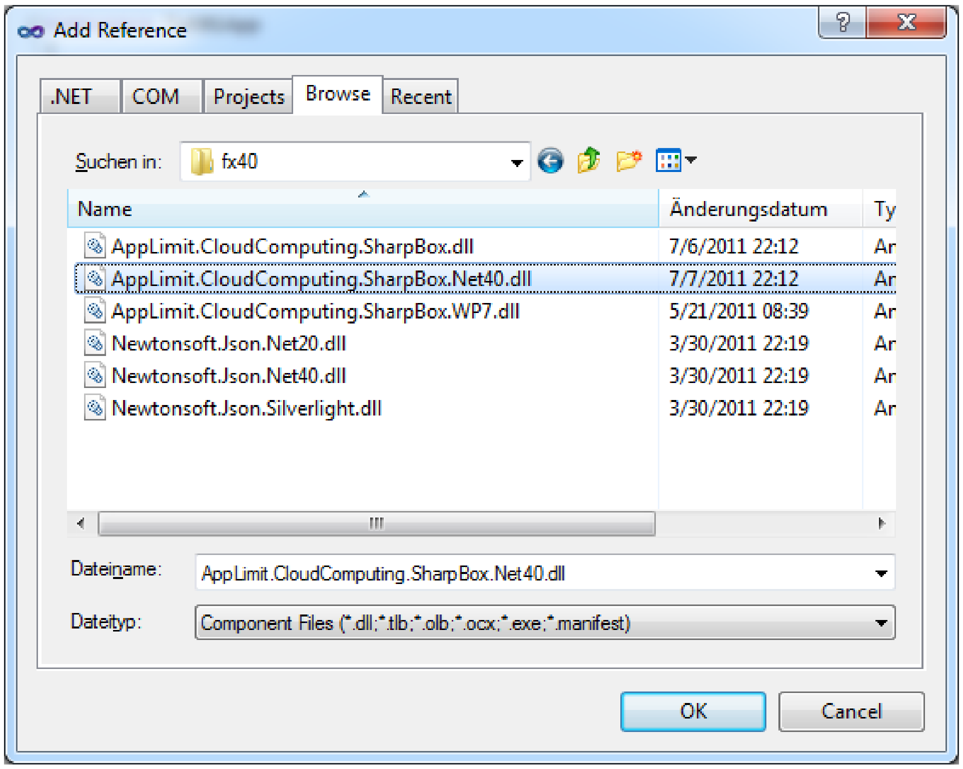
Icon:

4. Last but not least a reference to the SharpBox assembly has to be added to the reference list of the created Console Application project

   a. Select “Add Reference” in Solution Manager:



   b. Select the .NET 4.0 SharpBox library via Browse-Tab:

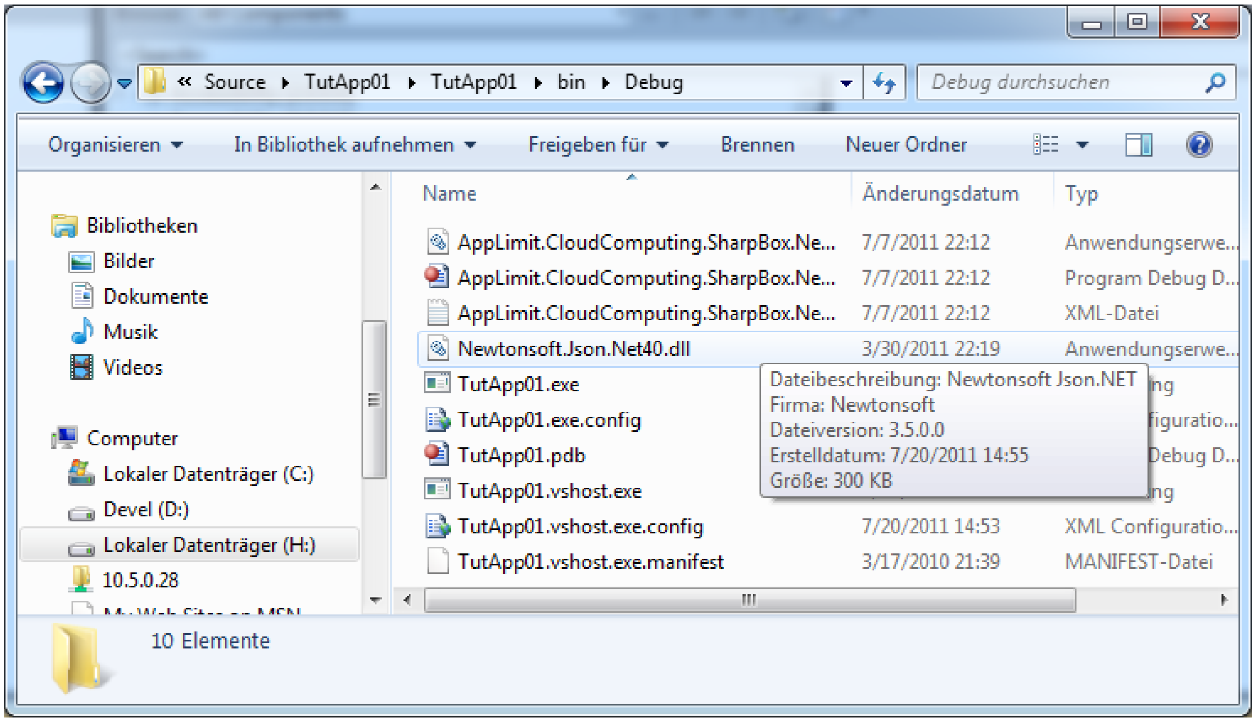


Name	Änderungsdatum	Ty
AppLimit.CloudComputing.SharpBox.dll	7/6/2011 22:12	Ar
AppLimit.CloudComputing.SharpBox.Net40.dll	7/7/2011 22:12	Ar
AppLimit.CloudComputing.SharpBox.WP7.dll	5/21/2011 08:39	Ar
Newtonsoft.Json.Net20.dll	3/30/2011 22:19	Ar
Newtonsoft.Json.Net40.dll	3/30/2011 22:19	Ar
Newtonsoft.Json.Silverlight.dll	3/30/2011 22:19	Ar

Dateiname: AppLimit.CloudComputing.SharpBox.Net40.dll

Dateityp: Component Files (*.dll;*.tlb;*.olb;*.ocx;*.exe;*.manifest)

5. Try to compile and start the application. The compiler output should look like this:



« Source ▸ TutApp01 ▸ TutApp01 ▸ bin ▸ Debug

Debug durchsuchen

Organisieren ▾ In Bibliothek aufnehmen ▾ Freigeben für ▾ Brennen Neuer Ordner

Bibliotheken
Bilder
Dokumente
Musik
Videos
Computer
Lokaler Datenträger (C:)
Devel (D:)
Lokaler Datenträger (H:)
10.5.0.28

Name	Änderungsdatum	Typ
AppLimit.CloudComputing.SharpBox.Ne...	7/7/2011 22:12	Anwendungserwe...
AppLimit.CloudComputing.SharpBox.Ne...	7/7/2011 22:12	Program Debug D...
AppLimit.CloudComputing.SharpBox.Ne...	7/7/2011 22:12	XML-Datei
Newtonsoft.Json.Net40.dll	3/30/2011 22:19	Anwendungserwe...
TutApp01.exe		ng
TutApp01.exe.config		figuratio...
TutApp01.pdb		Debug D...
TutApp01.vshost.exe		ng
TutApp01.vshost.exe.config	7/20/2011 14:53	XML Configuratio...
TutApp01.vshost.exe.manifest	3/17/2010 21:39	MANIFEST-Datei

Dateibeschreibung: Newtonsoft Json.NET
Firma: Newtonsoft
Dateiversion: 3.5.0.0
Erstelldatum: 7/20/2011 14:55
Größe: 300 KB

10 Elemente

Congratulations! After these simple steps your first SharpBox application is ready to run. The next steps of this tutorial will give you an idea how to use SharpBox in more complex applications. All next steps are based on the created empty tutorial project.

Connecting to DropBox or another storage provider

The basic entry point into the SharpBox API is the `CloudStorage` class. This class contains all methods to communicate with the storage provider. Establishing a connection to the selected storage provider is the first task which has to be achieved as follows:

1. Creating a specific service configuration which contains all important information about the target service. Typically this information is the service URL and some protocol meta data. Out of the box SharpBox delivers standard configurations for all supported services or supports an easy way to derive a specific configuration from an existing configuration class.

2. Creating specific user credentials which has to be used to become an authenticated user at the target service. The different credential objects are derived from the technical needs of the target service. DropBox for instance is using a set of API-Key & API-Secret and an end user Username & Password. Traditional WebDAV services are only using an Username & Password pair.

3. Using the configuration and the credentials to get a connection to the cloud storage provider. After preparing the two important parts it's just calling a simple method.

The following walkthrough ends up with a running sample which has the capability to get data from your DropBox account:

1. Add using statements for more convenient work in the main program:

```
using System.Linq;
using System.Text;
using AppLimit.CloudComputing.SharpBox;
using AppLimit.CloudComputing.SharpBox.StorageProvider.DropBox;

namespace TutApp01
{
```

2. Create a new `Cloudstorage` object which will be used for establishing the connection:

```
static void Main(string[] args)
{
  // Creating the cloudstorage object
  CloudStorage dropBoxStorage = new CloudStorage();
```

3. Generate the right service configuration for DropBox. All other configuration, e.g. for Box.NET or StoreGate might be used as well:

   ```
   // get the configuration for dropbox
   var dropBoxConfig =
   CloudStorage.GetCloudConfigurationEasy(nSupportedCloudConfigurations.DropBox);
   ```

4. Creating the service specific credentials is necessary to let SharpBox know which user tries to use the targeted cloud storage service:

   ```
   // building credentials for the service
   var dropBoxCredentials = new DropBoxCredentials();

   // set the information you got from the dropbox team
   dropBoxCredentials.ConsumerKey = "<<YOURAPIKEY>>";
   dropBoxCredentials.ConsumerSecret = "<<YOURAPISECRET";

   // set the information you got from the enduser
   dropBoxCredentials.UserName = "<<ENDUSER DROPBOX ACCOUNT>>";
   dropBoxCredentials.Password = "<<ENDUSER DROPBOX PASSWORD>>";
   ```

5. Calling the open method starts the network communication to establish a connection to the targeted cloud service.

   ```
   // open the connection
   var storageToken = dropBoxStorage.Open(dropBoxConfig, dropBoxCredentials);

   //
   // do what ever you want to
   //
   ```

6. After finishing using the service the close method cleans up all used resources:

   ```
   // close the connection
   dropBoxStorage.Close();
   ```

Connection options to other providers are described in the Appendix

Enumerating files and directories

SharpBox abstracts the virtual filesystem of every cloud storage service similar to an UNIX filesystem structure; the root element has always the name "/". To enumerate all files or folders in the cloud storage service a connection to a specific folder has to be established. After that all child elements can be used by normal .NET enumeration because the class `ICloudDirectoryEntry` implements the `IEnumerable` interface. The following step-by-step sequence walks through the sample of getting all childs from the DropBox folder /Public:

1. Open the created tutorial sample and find the following position in the code:

   ```
   //
   // do what ever you want to
   //
   ```

   This code has to be replaced with the lines of code described in the next steps.

2. Receive the "/Public" folder information from the DropBox service:

   ```
   // get a specific directory in the cloud storage, e.g. /Public
   var publicFolder = dropBoxStorage.GetFolder("/Public");
   ```

3. Enumerating all childs (files and folders) of the "/Public"-Folder. Every file element implements the interface `ICloudFileSystemEntry`, every folder implements the interface `ICloudDirectoryEntry` in addition:

   ```
   // enumerate all child (folder and files)
   foreach (var fof in publicFolder)
   {
     // check if we have a directory
     Boolean bIsDirectory = fof is ICloudDirectoryEntry;

     // output the info
     Console.WriteLine("{0}: {1}", bIsDirectory ? "DIR" : "FIL", fof.Name );
   }
   ```

Download or upload files

Downloading and uploading data into or from a cloud storage service is supported by SharpBox in many different ways. All methods are aligned with the data stream approach of the .NET framework, this means every upload or download method transfers nothing more than a data stream from a local file into a data stream of a remote file. The easiest way based on the SharpBox comfort methods will be described in the following example:

1. Ensure that the file “test.dat” exists in the %temp% directory of your Windows Machine

2. Add the following code below the code of the former steps to upload the created test-file into the “Public” folder of your DropBox.

```
// upload a testfile from temp directory into public folder of DropBox
String srcFile = Environment.ExpandEnvironmentVariables("%temp%\\test.dat");
dropBoxStorage.UploadFile(srcFile, publicFolder);
```

3. Download the uploaded file into the local temp-Directory back again

```
// download the testfile from DropBox
dropBoxStorage.DownloadFile( publicFolder,
                             "test.dat",
                             Environment.ExpandEnvironmentVariables("%temp%"));
```

That’s it! After these steps you should be able to establish connections to different cloud storage services, enumerate files/folders and up/download content of different files.

Working with progress callback

An upload or download of big files can take a couple of seconds, minutes or hours. Normally the end-user would like to see progress information to get an impression how long it will take in total. To achieve this requirement SharpBox offers a bunch of progress callbacks which can be used to track an up/download process via progress bar or text based progress reports. The upload or download progress callback has to following signature:

```
public delegate bool FileOperationProgressChanged(
  ICloudFileSystemEntry file,
  long currentbytes,
  long sizebytes,
  object progressContext);
```

With SharpBox Version 1.1.1 the signature for this callback has been changed. Please use the following signature:

```
public delegate void FileOperationProgressChanged(
  object sender,
  FileDataTransferEventArgs e);
```

To use the progress callback the following steps are necessary:

1. Create a method in the used tutorial project which matches to the progress callback signature. These code defines the needed callback and implements the output of a dot per callback call. How often a callback will be called is not defined, it depends on the technical parameters of the used network connection and the Windows Version or .NET Framework version:

   ```
   static bool UploadDownloadProgress(ICloudFileSystemEntry file, long currentbytes,
   long sizebytes, object progressContext)
   {
     // print a dot
     Console.Write(".");

     return true;
   }
   ```

   Because of the changed signature in SharpBox Version 1.1.1 the following progress handler has to be implemented:

   ```
   static void UploadDownloadProgress(Object sender, FileDataTransferEventArgs e)
   {
     // print a dot
     Console.Write(".");
   ```

```
  // it's ok to go forward
  e.Cancel = false;
}
```

2. Connect the Up/Download method call with the defined callback so that the method can call this callback a couple times during the operation:

```
// upload a testfile from temp directory into public folder of DropBox
// with progress information
dropBoxStorage.UploadFile(srcFile, publicFolder, UploadDownloadProgress);

// download the testfile from DropBox with progress information
dropBoxStorage.DownloadFile(publicFolder, "test.dat",
Environment.ExpandEnvironmentVariables("%temp%"), UploadDownloadProgress);
```

Summary

After studying this tutorial you should be able to use SharpBox in your applications at a basic level, this means accessing directories and uploading or downloading files are implementable. Check out our other tutorials to get a deeper look into the features SharpBox offers to implement platform & service independent applications.

Appendix

Connecting to Box.NET or other supported WebDAV based provider

SharpBox supports a couple of different providers which are based on the WebDAV protocol. Comparable to the described way to following code fragments has to be used to establish a connection instead of the similar fragments in the document above:

1. Generate the right service configuration for Box.NET. All other configuration might be used as well:

   ```
   // get the configuration for dropbox
   var boxnetConfig =
   CloudStorage.GetCloudConfigurationEasy(nSupportedCloudConfigurations.BoxNet);
   ```

   Valid values are: DropBox, StoreGate, BoxNet, SmartDrive, WebDav, CloudMe

2. Creating the service specific credentials is necessary to let SharpBox know which user tries to use the targeted cloud storage service:

   ```
   // building credentials for the service
   var webDavCredentials = new GenericNetworkCredentials();

   // set the information you got from the enduser
   webDavCredentials.UserName = "<<ENDUSER WEBDAV ACCOUNT>>";
   webDavCredentials.Password = "<<ENDUSER WEBDAV PASSWORD>>";
   ```