TMS Cloud Pack for .NET
DEVELOPERS GUIDE

Dec 2016

Copyright © by tmssoftware.com bvba
Web: http://www.tmssoftware.com
Email: info@tmssoftware.com

Index

Availability

TMS Cloud Pack is a set of .NET components for Windows and ASP.NET application development and is available for Visual Studio 2010, 2012, 2013, 2015.

Online references

TMS software website:
http://www.tmssoftware.com

TMS Cloud Pack page:
http://www.tmssoftware.com/site/tmscloudnet.asp

Purchase a license

The TMS Cloud Pack for .NET is separately available via:
http://www.tmssoftware.com/site/tmscloudnet.asp

And is also included in the bundle TMS Cloud Studio with support for VCL, LCL, FMX, IntraWeb and .NET: http://www.tmssoftware.com/site/tmscloudstudio.asp

There is also a version of TMS Cloud Pack for VCL, FMX, LCL and IntraWeb:

- TMS VCL Cloud Pack : http://www.tmssoftware.com/site/cloudpack.asp
- TMS FMX Cloud Pack: http://www.tmssoftware.com/site/tmsfmxcloudpack.asp
- TMS IntraWeb Cloud Pack: http://www.tmssoftware.com/site/tmsiwcloud.asp
- TMS LCL Cloud Pack: http://www.tmssoftware.com/site/tmslclcloudpack.asp

Terms of use

With the purchase of TMS Cloud Pack for .NET, you are entitled to our consulting and support services to integrate the Google GDrive, Microsoft SkyDrive, DropBox, Box, Flickr, Google Calendar, Google Contacts, Google Picasa, Microsoft Live Calendar, Microsoft Live Contacts, Facebook, Twitter, LinkedIn, PushOver, FourSquare service in .NET applications and with this consulting and support comes the full source code needed to do this integration.
As TMS Cloud Pack uses these services, you're bound to the terms of these services that can be found at:

http://www.google.com/apps/intl/en/terms/user_terms.html
http://windows.microsoft.com/en-US/windows-live/microsoft-service-agreement?SignedIn=1
http://windows.microsoft.com/en-AU/windows-live/code-of-conduct
https://www.dropbox.com/terms
http://developers.facebook.com/policy/
http://twitter.com/tos
https://m.box.com/static/html/terms.html
http://info.yahoo.com/legal/us/yahoo/utos/utos-173.html
http://developer.linkedin.com/documents/linkedin-apis-terms-use
https://pushover.net/terms
https://foursquare.com/legal/terms

TMS Cloud Pack for .NET includes components for accessing CalDAV, CardDAV servers including the iCloud contacts & iCloud calendar. You're bound to the terms of each of these specific servers and/or services.

TMS software is not responsible for the use of TMS Cloud Pack for .NET components. The purchase of TMS Cloud Pack for .NET does not include any license fee that you might possibly be required to pay to these services. It will depend on your type of usage of these services whether a license fee needs to be paid.

It is the sole responsibility of the user or company providing the application that integrates these services to respect the respective terms and conditions. TMS software does not take any responsibility nor indemnifies any party violating the service terms & conditions of these services.

We cannot guarantee that a 3rd party cloud service will approve or allow the use of the services used in TMS Cloud Pack components in your application(s) now or at any time in the future. This is up to the 3rd party cloud service provider to decide and not under our control. In case the 3rd party cloud service provider makes changes to the conditions for using the service, the API itself or anything else that causes an incompatibility with our component implementations, tmssoftware.com will do its best to accommodate the changes but cannot be forced in any way or within any timeframe to do so and might be technically limited to do so.

Limited warranty

TMS software cannot guarantee the current or future operation & uptime of these services.
TMS software offers the consulting and support for TMS Cloud Pack for .NET in good faith that theses are reliable and future-proof services.
In no case, TMS software shall offer refunds or any other compensation in case the respective service terms/operation changes or stops.

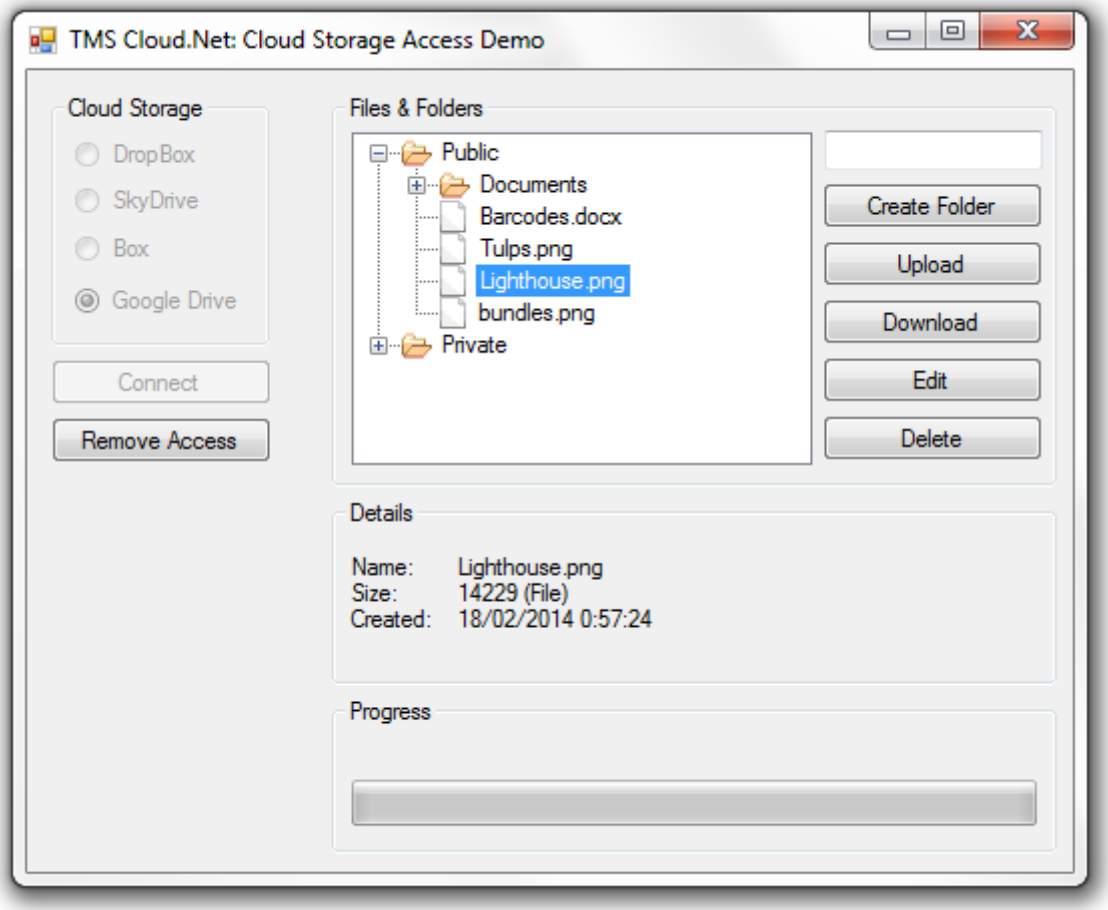
Main features

- Set of .NET components to offer easy access from Windows or ASP.NET applications to cloud services
- Component to get access to DropBox storage
- Component to get access to Box storage
- Component to get access to Google Drive storage
- Component to get access to Microsoft SkyDrive storage
- Component to get access to Facebook API
- Component to get access to Twitter API
- Component to get access to Google Contacts API
- Component to get access to Google Calendar API
- Component to get access to Google Picasa API
- Component to get access to Flickr API
- Component to get access to Windows Live Contacts API
- Component to get access to Windows Live Calendar API
- Component to get access to LinkedIn API
- Component to get access to FourSquare API
- Component to send push messages to iOS devices running the PushOver client
- Components to access CalDAV, CardDAV servers
- Components to access iCloud Contacts and iCloud Calendar
- Built-in support for OAuth 1.0 & 2.0 handling
- Built-in support for use of refresh tokens for use with one time authentication

Registering your application

A first step will be to register your application so you can obtain an application key and secret at the different cloud services. Please refer to our online documentation.

Getting started with cloud storage access



Once your application is registered and you have an application ID or client ID and application secret or client secret, you can get started to use the SkyDrive, DropBox, BoxDrive and GoogleDrive components to access your cloud storage. All 4 components work in a similar way.

1) Drop the component and an oAuthWinFormsPanel on the form.

2) Assign the .Authenticator.OAuthPanel.

3) Setup the client ID, client secret and callback url via the .Authenticator.OAuthApplication.Key, .Authenticator.OAuthApplication.Secret, .Authenticator.CallBackUrl properties.

4) Call the .Open method.

Code:

```
googleDrive1.Authenticator.OAuthPanel = oAuthWinformsPanel1;
googleDrive1.Authenticator.OAuthApplication.Key = "xxxxxxxxxxxx";
googleDrive1.Authenticator.OAuthApplication.Secret = "yyyyyyyyyyyyy";
googleDrive1.Authenticator.CallBackUrl = "zzzzzzzzzzzzz";
googleDrive1.Open();
```

Alternatively an OAuth.cfg file can be used. This file should be located in the application folder and should contain the following:

```
<?xml version="1.0"?>
<oauth>
  <googledrive>
    <key>xxxxxxxxxxxxxxxx</key>
    <secret>yyyyyyyyyyy</secret>
    <callback>zzzzzzzzzzzzz</callback>
  </googledrive>
</oauth>
```

The following services require a working callback url:
Google Drive, Google Calendar, Google Contacts, Picasa, FourSquare, Flickr, LinkedIn

Example:
For testing purposes only, the following url may be used:
http://www.tmssoftware.net/oauth/

The following services require a working local callback url or a callback url using the HTTPS protocol:
Dropbox, Box

Example:
Dropbox: https://www.google.com
Box: https://app.box.com

The following services only require a local callback url:
Twitter, Facebook

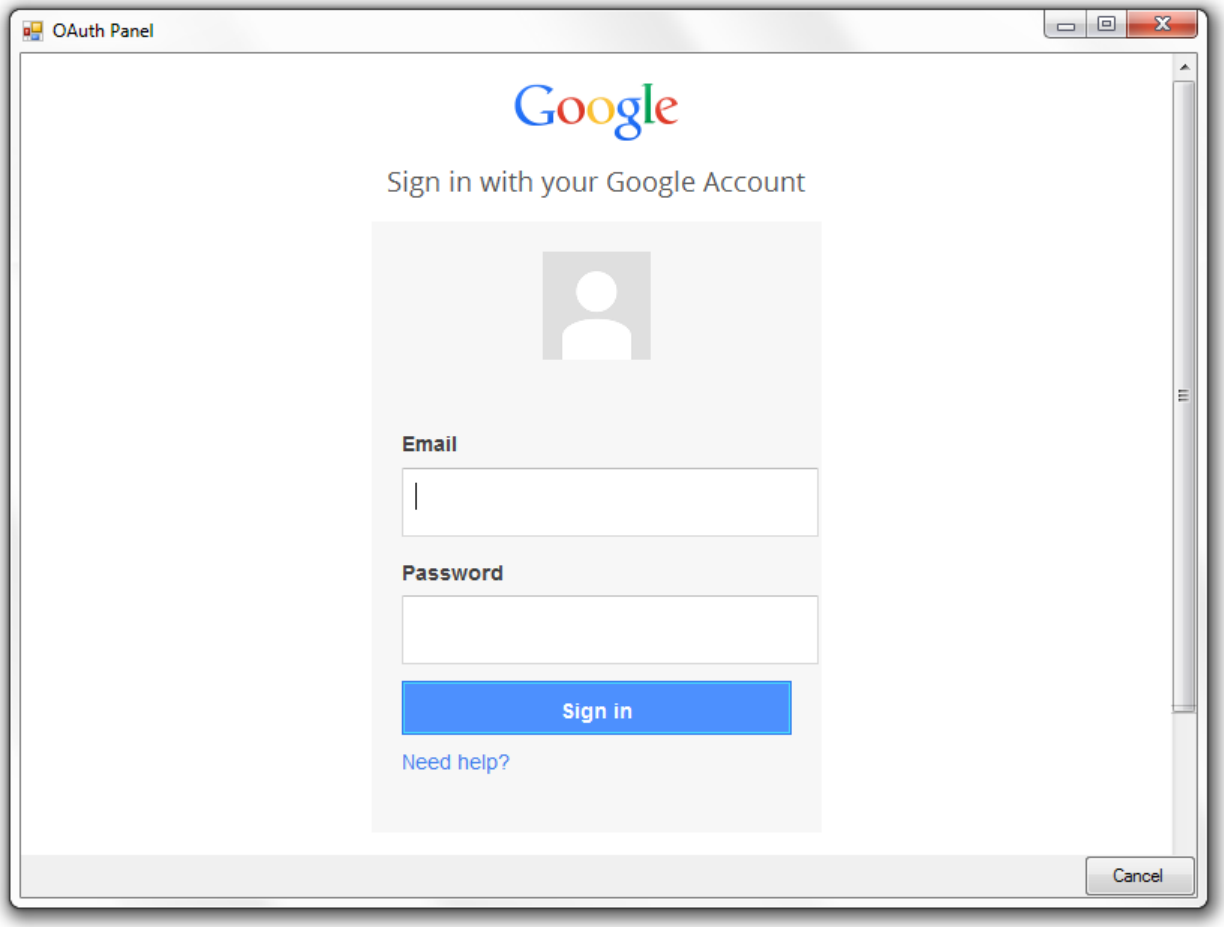
Example:
http://127.0.0.1/

If the file is found the Key, Secret and CallBackUrl values will be loaded automatically.
The name of the .cfg file can be changed using TMS.Common.Settings.DefaultOAuthFileName.

Example:

```
public CloudStorageDemo()
{
    Settings.DefaultOAuthFileName = "OAuth.cfg";
    InitializeComponent();
}
```

And this will show the Google login screen:



The size of the login screen and the caption are controlled by the properties:
FormText (string)
Height (integer)
Width (integer)

When the user has provided the correct credentials, the event AfterOpen will be triggered and from that moment, the component has access to the online cloud APIs.

File organisation

The file structure of a cloud storage service typically has a hierarchical organization in folders and files. This is represented in the cloud access component as a collection of the type ICloudStorageItem. The common structure of this collection is:

```
public interface ICloudStorageItemCollection:IList
```

In this collection are items of the type:

```
public interface ICloudStorageItem
```

with properties:

string Name
Holds the filename of a file or folder

ICloudStorageItemCollection Folder
When the ItemType is Folder, this contains in turn a collection of files and folders

Int64 Size
Holds the size of the item

ItemType ItemType
Defines whether the item is a file or folder (ItemType.File, ItemType.Folder)

DateTime CreationDateTime
Holds the creation date of the file or folder

DateTime LastModifiedDateTime
Holds the timestamp when the file was last modified

These are the common file/folder properties. Note that for different cloud storage services, there might be different extra properties available.

Calling the method GetAccountInfo will use the cloud storage API to query the list of all files and will store this hierarchically in the Folders collection property.

A helper method is available to immediately visualize this file structure in a treeview. This can be done by calling WinFormsBase.FillTreeView(TreeView, ICloudStorageFolderCollection);

File operations

Following operations are available:

Create a folder

Delete a file or folder

Download a file

Upload a file

Creating a folder

Creating a folder is simple. You can either create a folder in the root by calling:

```
Folders[].Items.CreateDir(string folderName);
```

Or create a subfolder in a specific folder. To do this, you need the instance of the ICloudStorageItem representing the folder.

This function returns a new CloudStorageItemBase instance representing the created folder.

Delete a file or folder

Deleting a file can be done by calling the function ICloudStorageItem.Delete.

Download a file

Downloading a file is equally simple. Call the function ICloudStorageItem.SaveToPath(string path);. Note that the progress of the download can be tracked via the event WorkingProgress.

Upload a file

Uploading a file means creating the file on the cloud storage and uploading its data. A new ICloudStorageItem should first be inserted into the ICloudStorageItemsCollection:

var item = ICloudStorageItem.Items.InsertFile(string FileName);

The function that is used for upload is:

item.Post();

The local file that will be uploaded is set via the FileName parameter. The progress during the upload can also be tracked via the WorkingProgress event.

Public shared files

Cloud storage controls have a built-in function to create a public share URL for a file on the cloud storage.

To create & get the share URL, you can use:

```
ICloudStorageItem si = (CloudStorageItemBase) treeView1.SelectedNode.Tag;

TextBox1.Text = si.GetShare().Link;
```

The ShareResult.Link instance contains a public accessible HTTP URL to download the file.

Twitter

Usage

Twitter is a component that provides access to the Twitter API service. It allows to Tweet messages with an optional image, Retweet messages from other users and retrieve existing messages, followers & friends from a Twitter account.

Organisation

Classes:

User: Class property that contains the Twitter profile information for a Twitter user.

ScreenName: string; The screen name, handle, or alias that this user identifies themselves with.
Id: integer; The unique identifier for this user.
CreatedAt: DateTime; The UTC datetime that the user account was created on Twitter.
ListedCount: Int64; The number of public lists that this user is a member of.
Name: The name of the user, as they've defined it.
FavoritesCount: Int64; The number of tweets this user has set as favorite in the account's lifetime.
FollowersCount: Int64; The number of followers this account currently has.
FriendsCount: Int64; The number of users this account is following.
GeoEnabled: Boolean; When true, indicates that the user has enabled the possibility of geotagging their Tweets.
ProfileImageUrl: string; An URL pointing to the user's avatar image.
Protected: Boolean; When true, indicates that this user has chosen to protect their Tweets.
Status: Tweet: The user's most recent tweet or retweet.
StatusesCount: integer; The number of tweets (including retweets) issued by the user.
TimeZone: string; A string describing the timezone this user declares themselves within.
URL: string; A URL provided by the user in association with their profile.

Tweet: Contains a Twitter Status message details.

User: User; The user who posted this Tweet.
Text: string; The actual text of the status update.
Favorited: Boolean; Indicates whether this Tweet has been set as a favorite by the user.
CreatedAt: TDateTime; Time when this Tweet was created.
Id: Int64; The unique identifier for this Tweet.
InReplyToID: Int64; If the represented Tweet is a reply, this field will contain the original Tweet's author ID.
InReplyToScreenName: string; If the represented Tweet is a reply, this field will contain the screen name of the original Tweet's author.
InReplyToStatusID: Int64; If the represented Tweet is a reply, this field will contain the original Tweet's ID.
Medias: MediaCollection; Contains a list of Urls to the images that are connected to this

Tweet (if any).
Source: string; Utility used to post the Tweet, as an HTML-formatted string.

Methods:

Void Tweet.Post()
Create a new Tweet on the user's timeline.
Create a new Tweet with included image on the user's timeline.

Example 1:

```
User user = twitter1.GetAccountInfo();
TweetCollection currentStatus = user.Statuses.GetHome();
currentStatus.InsertItem(string message).Post();
```

Example 2: (Tweet with media)

```
if (openFileDialog1.ShowDialog() == DialogResult.OK)
{
    foreach (var s in openFileDialog1.FileNames)
        Attachments.Add(s);

    if (Attachments.Count > 0)
    {
        User user = twitter1.GetAccountInfo();
        TweetCollection currentStatus = user.Statuses.GetHome();

        var x = new MediaCollection();
        foreach (var m in Attachments)
            x.AddMedia(m);
        currentStatus.InsertItem(string message, x).Post();
    }
}
```

```
Tweet User.Statuses.Retweet(Int64 id)
```
Retweet an existing Tweet on the user's timeline.

```
Message User.DirectMessages.SendDirectMessage(string message, string screenName)
```
Send a direct message to another Twitter user.

```
UserCollection User.GetFollowers()
```
Fill the list of User.Followers.

```
UserCollection User.GetFriends()
```
Fill the list of User.Friends.

```
int GetStatusList(ref TweetCollection tweetList, StatusType statusType, int count,
Int64? pageNumber, Int64? sinceId, Int64? maxId, Int64? userId)
```

Get a list of statuses for a specific user defined by the userId.
The type of statuses is defined by the statusType.

Home: Returns the list of statuses on the user's timeline
Mentions: Returns a list of mentions
User: Returns of user statuses
RewteetsOfMe: Returns a list of retweeted user statuses

```
TweetCollection Search(string query)
```
Return a list of Status messages that contain the Query string value.

```
User GetAccountInfo()
```
Get the user's Profile information.

```
User GetProfileInfo(int id)
User GetProfileInfo(string screenName)
```
Get a user profile based on a profile Id or ScreenName.

```
bool GetProfileListInfo(UserCollection profileList)
```
Fill a list user profiles based on the profile list Ids.

Example:

```
AdvTwitter1.GetFollowers;
AdvTwitter1.GetProfileListInfo(AdvTwitter1.Followers);
```

Facebook

Usage

Facebook is a component that provides access to the Facebook API service. It allows to post a status update (with optional URL and image), upload an image, retrieve a list of friends, retrieve a user's Feed and Like/Unlike a feed item.

Organisation

Classes:

UserProfile: Class property that contains the Facebook profile info for the currently authenticated user.

Id: string; The user's Facebook Id.
FirstName: string; The user's first name.
LastName: string; The user's last name.
MiddleName: string; The user's middle name.
Gender: Gender; The user's gender.
Link: string; The URL of the profile for the user on Facebook.
UserName: string; The user's Facebook username.
BirthDay: string; The user's birthday.
Email: string; The proxied or contact email address granted by the user.
TimeZone: double; The user's timezone.
Locale: string; The user's locale.
Verified: Boolean; The user's account verification status.
UpdatedTime: DateTime; The last time the user's profile was updated.
PictureUrl: string; The URL of the user's profile picture.
HomeTown: ResultObject; The user's hometown.
Location: ResultObject; The user's current city.
RelationShipStatus: string; The user's relationship status.
SignificantOther: ResultObject; The user's significant other.
Website: string; The URL of the user's personal website.
Likes: UserActivityCollection; List of the Facebook pages the user has liked.

FBPost: Class property that contains a Facebook post info.

Id: string; The post Id.
User: ProfileBase; Information about the user who posted the message.
Message: string; The actual message text.
Picture: string; If available, a link to the picture included with this post.
Link: string; The link attached to this post.
Name: string; The name of the link.
Caption: string; The caption of the link.
Description: string; A description of the link.

CreatedTime: DateTime; The time the post was initially published.
Story: string; Text of stories not intentionally generated by users.
ObjectID: string; The Facebook object id for an uploaded photo or video.
UpdatedTime: TDateTime; The time of the last comment on this post.
Likes: UserActivityCollection; List of users that liked this post.
ToUsers: ProfileBaseCollection; List of profiles mentioned or targeted in this post.
Comments: CommentCollection; List of comments add to this post.

Id: string; The Id of the comment.
From: ProfileBase; The user that created the comment.
Message: string; The comment text.
CreatedTime: DateTime; The time the comment was created.
Likes: UserActivityCollection; List of users that liked this comment.
UserLikes: boolean; Indicates if the currently authenticated user has liked the comment.

Album: Facebook album info.

Id: string; The Id of the album.
Name: string; The name of the album.
From: ApplicationBase; The profile that created this album.
Link: string; A link to this album on Facebook.
Description: string; The description of the album.
CoverPhoto: string; The album cover photo.
CreatedTime: TDateTime; The time the photo album was initially created.
UpdatedTime: TDateTime; The last time the photo album was updated.

Photo: Facebook Photo info.

Id: string; The Id of the picture.
Name: string; The user provided caption given to this picture.
From: ProfileBase; The profile (user or page) that posted this picture.
Source: string; A direct URL link to the picture on Facebook.
CreatedTime: TDateTime; The time the picture was initially published.
UpdatedTime: TDateTime; The last time the picture or its caption was updated.

Page: Facebook Page info.

Id: string; The page Facebook Id.
AccessToken: string; The access token that must be used to post to this page.
Name: string; The page name.
CategoryList: string; The page categories.
Permissions: string; The list of permissions that are available for the currently authenticated user.

Methods:

```
UserProfile GetProfile()
```
Get the user's Profile.

```
ResultObjectCollection UserProfile.ReadFriends()
```
Get the user's Facebook Friends.

```
UserProfile GetProfile(string userId)
```
Get a Facebook Profile for a specific user id.

```
AlbumCollection ReadAlbums(string userId, Dictionary<string, string>
localAuthenticationValues = null)
```
Fill a TFacebookProfile.Albums list with the Facebook Photo Albums connected to the provided Id.

```
PhotoCollection Album.ReadPhotos()
```
Get a collection of photos connected to a Facebook Album.

```
PostCollection UserProfile.ReadFeed()
```
Fill a Profile.Feed with a list of Facebook Feed messages for the Facebook Profile.

```
CommentCollection FBPost.ReadComments()
```
Fill a FBPost.Comments with a list of comments for the Facebook Post.

GetImageURL(const ImageID: string): string;
Return a direct URL to a TFacebookPicture.ImageID.

```
string UserCreatePost(string objectId, string message)
```
Post a message on the user's Facebook Feed with an optional URL link and image URL.

```
string PageCreatePhoto(string pageId, string sourceFilename, Stream sourceStream,
string message, FeedTargeting targeting, FeedTargeting feedTargeting, bool published,
TimeSpan scheduledPublishTime, bool NoStroy)
```
Post a message on the user's Facebook Page with an optional URL link and image URL.

```
string CreateComment(string objectId, string message,
Dictionary<string, string> localAuthenticationValues = null)
```
Post a comment on a Facebook object.

```
string UserCreatePhoto(string userId, string sourceFileName, Stream sourceStream,
string message = null, string place = null, bool noStory = false)
```
Post a photo to a user's wall. The Id of the newly created photo is returned.
By default the photo is posted to the Facebook Album that is connected to your application registered with Facebook. (If the Facebook Album does not exist yet, it will automatically be created)

```
bool UserProfile.Feed[].Like()
```
Like a Facebook feed item.

```
bool UserProfile.Feed[].UnLike()
```
Unlike a Facebook feed item.

Example:

Post a message that contains an image from a Facebook Album to the user's feed:
First upload an image to a Facebook album then retrieve the direct remote URL of the image and use this to post a new message.

```
ImageID := AdvFacebook1.PostImage('Description', OpenDialog1.FileName);
ImageURL := AdvFacebook1.GetImageURL(ImageID);
AdvFacebook1.Post('Message', 'http://www.mywebsite.com', ImageURL);
```

Flickr

Usage

Flickr is a component that facilitates access to the Flickr service. It allows retrieving your galleries, sets, and photostream as well as creating / deleting sets, uploading / downloading photos to sets and to your photostream. Photos can be uploaded / updated with a title / description and a geo location. Comments on sets and photos can also be retrieved.

Organisation

Properties:

Profile: The profile information for the authenticated user, loaded after calling GetProfile().
Photos: The collection of photos from a specific Set, loaded after calling GetPhotos().
Sets: The collection of sets, loaded after calling GetSets().
Sizes: The available sizes for a photo, loaded after calling GetSizes().
Info: The extended information for a photo, loaded after calling GetInfo().
Location: The location information for a photo, loaded after calling GetLocation().
Comments: The comments for a photo, loaded after calling GetComments().

Classes:

Profile Properties:

UserID: String: The ID retrieved after login, used to access user related content.
UserName: String The user name retrieved after login.

Set Properties:

Id: String: The Id of the set, used to access the flickr api for sets specifically.
Primary: String: The ID of the primary photo inside the set.
Title: String: The title of the set.
Description: String: The description of the set.

Photo Properties:

Id: String: The Id of the photo, used to access the flickr api for photos specifically.
Owner: String: The User ID of the owner of the photo.
Title: String: The title of the photo
Description: String: The description of the photo.

Tag Properties:

Id: String: The Id of the tag.
Author: String: The User Id of the author of the tag.
Description: String: The value of the tag.

Comment Properties:

Id: String: The Id of the comment.
AuthorID: String: The User Id of the author of the comment.
Author: String: The User Name of the author of the comment.
Content: String: The value of the comment.
Created: String: The date the comment was created.

Size Properties:

Label: String: Identifier for the size in which the photo can be downloaded.
Width: Integer: The width of the photo for the specific size.
Height: Integer: The height of the photo for the specific size.
Url: String: The URL of the photo for the specific size to display in a browser window.

Methods:

```
void CreateSet(string title, string description, string primaryPhotoId)
```
Creates with a specific title, description and primary photo ID. The Primary Photo ID is required when creating a new set and can be retrieved by first uploading a photo to the photo stream.

```
void DownloadFile(string url, string fileName)
```
Downloads a photo to a target location. The APhotoURL is retrieved after calling GetSizes for a photo object and selecting the DownloadURL depending on the size in the sizes collection.

```
Profile GetProfile()
```
Sets the User ID and User Name properties necessary for API access such as loading the sets / galleries. This function needs to be called after the authorization is complete to make sure subsequent calls function correctly.

```
List<PhotoSet> GetSets(string userId)
```
Fills the Sets collection. If no userId is provided the Sets for the currently authenticated user are retrieved.

```
List<Photo> GetPhotoStream(string userId)
```
Fills the PhotoStream collection.

```
void AddPhoto(string photoId, string photoSetId)
```
Adds a Photo with a specific ID to a set.

```
PhotoUploadResult UploadPhoto(String filename, string title = null, string description
= null, Privacy? privacy = null, SafetyLevel? safety = null, ContentType? contentType
= null, bool? hidden = false)
```
Adds a photo to the set and uploads the photo to the photostream. A set cannot be empty and needs a primary photo to exist. The photo is first uploaded to the photo stream and then added to the set.

```
PhotoUploadResult ReplacePhoto(string photoId,string fileName)
```
Replaces an existing photo.

```
List<Comment> GetComments(string photoId)
```
Retrieves the comments and fills the comment collection for a set.

```
List<Photo> GetPhotos(string photoSetId)
```
Retrieves the photos and fills the photo collection for a set.

```
string DeleteSet(string photoSetId)
```
Remove a set.

```
PhotoLocation GetLocation(string photoId)
```
Gets the geo location and sets the Latitude and Longitude properties of the Location class.

```
PhotoInfo GetInfo(string photoId)
```
Gets the information and fills the properties of the Info class.

```
List<Size> GetSizes(string photoId)
```
Gets the downloadable sizes of the photo and fills the sizes collection.

```
List<Comment> GetComments(string photoId)
```
Gets the comments and fills the comments collection of a photo.

```
string DeletePhoto(string photoId)
```
Remove a photo.

```
void SetTags(string photoId, List<string> tags)
```
Updates / sets the tags on a photo, found in the Tags collection.

FourSquare

Usage

AdvFourSquare is a component that facilitates access to the FourSquare service. It allows searching venues by keyword, category and location. Venue details can be retrieved as well as venue photos, tips and specials. The component also enables retrieving a FourSquare user's profile information and a list of places the authenticated user has checked in to.

Organisation

Properties:

Categories: Contains a list of FourSquare categories and sub-categories.

ID: string; The category ID.
Name: string; The name of the category.
PluralName: string; The plural name of the category.
ShortName: string; The short name of the category.
Icon.URL: string; The URL of the icon image for the category.
Primary: Boolean; Indicates if this is the primary category for the parent venue object.
SubCategories; Contains a list of sub-categories.

Location: Contains information about the geographical location where the venues in the Venues collection are located.

Name: string; The description of the location.
CountryCode: string; The country code related to the location.
Center: Coordinates; The center latitude and longitude coordinates for the location.
Bounds: Bounds; The North East and South West coordinates for the location. All venues currently listed in Venues are located within these bounds.

UserProfile: Contains the profile information for a FourSquare user.

ID: string; The user ID.
FirstName: string; The first name of the user.
LastName: string; The last name of the user.
Photo.ImageURL: string; The URL for the profile image of the user.
FriendsCount: integer; The number of friends the user has on FourSquare.
CheckInsCount: integer; The number of times the user has checked in at a venue.
City: string; The home city of the user.
Gender: FourSquareGender; The gender of the user.
Bio: The description of the user.

Contact: The contact information of the user.
Email: The email address of the user.
Phone: The phone number of the user.
Facebook: The Facebook ID of the user.
Twitter: The Twitter ID of the user.

CheckIns: Contains a list of locations where the user has checked in.

ID: string; The CheckIn ID.
Created: DateTime; The time when this CheckIn was created.
Shout: string; The comment text for this CheckIn.
Venue: The venue related to this CheckIn.

Venues: Collection that contains a list of FourSquare venues.

ID: string; The venue ID.
Name: string; The name of the venue.

Location: The location information of the venue.
Address: The address of the venue.
CrossStreet: string; The main street nearby the venue.
PostalCode: string; The postal code of the venue.
City: string; The city of the venue.
State: string; The state of the venue.
Country: string; The country of the venue.
CountryCode: string; The country code of the venue.
Latitude: double; The latitude coordinate of the venue.
Longitude: double; The longitude coordinate of the venue.
Contact: The contact information of the venue.
Email: The email address of the venue.
Phone: The phone number of the venue.
Facebook: The Facebook ID of the venue.
Twitter: The Twitter ID of the venue.

URL: string; The FourSquare URL of the venue.
Categories: TFourSquareVenueCategories; A list of FourSquare categories related to the venue.
Website: string; The website URL of the venue.
CheckInsCount: integer; The number of times a user has checked in at the venue.
UsersCount: integer; The number of unique users that have checked in at the venue.
TipCount: integer; The number of tips that have been posted for the venue.
HereNowCount: integer; The number of users that is currently checked in at the venue.

Photos: A list of photos related to the venue.

ID: string; The photo ID.
Created: DateTime; The time the photo was created.
ImageURL: string; The URL for the image.

Hours: Contains information about the opening hours of the venue.

Status: string; Describes when the venue will open (when IsOpen = fiOpen) or when the venue will close (when IsOpen = fiClosed).
IsOpen: FourSquareIsOpen; Indicates if the venue is currently open, closed or if opening hours are unknown.

Rating: double; The FourSquare rating of the venue.
Specials: A list of FourSquare specials that are available at the venue.

ID: string; The special ID.
Title: string; The title of the special.
Message: string; The message text of the special.
SpecialType: FourSquareSpecialType; The type of the special.
Summary: The description of the special.
Description: The description of how to unlock the special.

Classes:

Tips: A list of tips that has been posted for the venue.

ID: string; The tip ID.
Text: string; The tip content text.
Created: DateTime; The time the tip was posted.
ImageURL: string; The URL for the image related to the tip.
User; The user who has posted the tip.
Likes: string; The number of likes this tip has received.
Liked: Boolean; Indicates if the currently authenticated user has liked the tip.

Methods:

```
User GetProfile(string userId);
```
Gets the user profile information for the userId or for the authenticated user if no id is provided.

```
List<CheckIn> GetCheckIns()
```
Gets a list of CheckIns for the currently authenticated user. Fills up the CheckIns collection.

```
List<Category> GetCategories()
```
Gets a list of FourSquare Venue Categories and Sub-Categories. Fills up the Categories collection.

```
Venues GetNearbyVenues(double latitude, double longitude,
            string location = "", string keyword = "", string categoryId = "",
            int resultCount = 10)
```
Gets a list of nearby venues based on Latitude and Longitude coordinates or Location. Fills up the Venues collection.

```
Venue GetVenue(Venue venue);
```

Gets extra information about a venue using the Venue.ID value.

```
SimilarVenues GetSimilarVenues(string VenueID);
```
Gets similar venues (in the same location) based on the VenueID value. Fills up the SimilarVenues collection.

```
List<Tip> GetTips(Venue venue, FourSquareSort sort = FourSquareSort.Recent, int
resultCount = 10, int resultOffset = 0)
```
Gets the Tips that have been posted for a venue.

```
List<Photo> GetPhotos(Venue venue, int resultCount, int resultOffset);
```
Gets the Photos that have been posted for a venue.

```
CheckIn CheckIn(string venueId, string comment)
```

Performs a CheckIn (with optional comment text) for the currently authenticated user a the venue specified through the venueId.

GoogleCalendar

Usage

GoogleCalendar is a component that provides access to the Google calendar service. It allows to retrieve a list of Google calendars and read, create, update & delete Google calendar events.

Organisation

Properties:

Calendars: Class that contains a list of Google calendars.

Id: string; The calendar Id.
Description: string; The description of the calendar.
Location: string; The location of the calendar.
Summary: string; The name of the calendar.
TimeZone: string; The time zone of the calendar.

Calendar: Class that contains a list of Google calendar events.

Id: string; The event Id.
ETag: string; The ETag of the event.
CalendarID: string; The ID of the calendar this event belongs to.
Description: string; The description of the event.
Summary: string; The name of the event.
StartTime: DateTime; The start time of the event.
EndTime: DateTime; The end time of the event.
IsAllDay: Boolean; Indicates if this is an all-day event.
Location: string; The location of the event.
Visibility: Visbility; Indicates if the event is public, private or confidential.
Recurrence: string; The recurrency rule for a recurring event. (Not available for instances of the recurring event)
RecurrenceID: string; For an instance of a recurring event, this is the event ID of the parent event.
Sequence: integer; Indicates the number of times this event has been updated.
TimeZone: string; The time zone of the event. Required when adding a new recurring event if IsAllDay is false.
Attendees: List of attendees for the event.

Name: string; Name of the attendee.
Email: string; Email of the attendee.
Status: ResponseStatus; Indicates if the attendee has responded to the invitation and if the invitation was declined, tentatively accepted or accepted.

Methods:

```
List<Item> GetCalendars(string userId)
```
Fill the list of calendars. If no userId is provided the calendars from the currently authenticated user are retrieved.

```
Calendar GetCalendar(string calendarId, int maxResults)
Calendar GetCalendar(string calendarId, DateTime changedSince,
int maxResults)
Calendar GetCalendar(string calendarId, DateTime startDate, DateTime endDate,
int maxResults)
Calendar GetCalendar(string calendarId, DateTime startDate, DateTime endDate,
DateTime changedSince, int maxResults)
```
Fill the Items list with calendar events from a Google calendar for a certain timespan. If no calendarId is provided, the events are retrieved from the default Google calendar. If no FromDate/ToDate is specified the events are retrieved for the default timespan. When ChangedSince is used, only events that changed since the specified date/time are retrieved.

```
Event Add(Event anEvent)
```
Add a new event to the calendar as specified by Item.CalendarID.

```
Event Update(Event anEvent)
```
Update an existing event.

Example:

```
Event ev;

Event.Description = “Event Description”;
Event.Summary = “Event Subject”;
googleCalendar1.Update(ev);
```

```
void DeleteEvent(Event calendarEvent)
```
Delete an event.

GoogleContacts

Usage

GoogleContacts is a component that provides access to the Google contacts service. It enables to read, update, create and delete contacts. It also allows to read, update, create and delete contact groups.

Organisation

Properties:

Contacts: Contains a list of Google Contact items.

Id: string; The Id of the contact.
Title: string; The title of the contact.
Birthday: DateTime; The birthday of the contact.
Organization: Contains a list of organizations the contact belongs to.

CompanyName: string; The company name related to the contact.
Title: string; The job title of the contact.
NamePrefix: string; The name prefix of the contact.
NameSuffix: string; The name suffix of the contact.

UserDefinedField: Contains a list of custom item data.

Key: string; The id for the custom item.
Value: string; The value for the custom item.

Emails: Contains a list of email addresses for the contact.

Address: string; The email address.
CustomType: string; The type description if EmailType is set to ftCustom.
lType: GFieldType; The type of email address.
Primary: boolean; Indicates if this is the contact's primary email address.

Name: Contains name information for the contact.

GivenName: string; The first name of the contact.
MiddleName: string; The middle name of the contact.
FamilyName: string; The last name of the contact.
FullName: string; The full name of the contact.

NickName: string; The nickname of the contact.

Groups: Contains a list of Google Groups this contact belongs to.

ID: string; The ID of the group that the contact is assigned to.

ImageURL: string; The image for the contact.
InstantMessenger: Contains a list of instant messenger IDs for the contact.

CustomType: string; The type description if InstantMessengerType is set to itCustom.
ID: string; The ID for this instant messenger type.
Type: GIMType; The type of instant messenger.

Notes: string; The notes for the contact.
PhoneNumbers: Contains a list of phone numbers for the contact.

CustomType: string; The type description if PhoneType is set to ptCustom.
Number: string; The phone number.
Type: GPhoneType; The type of phone number.
Primary: boolean; Indicates if this is the contact's primary phone number.

PostalAddresses: Contains a list of postal addresses for the contact.

Address: string; The full postal address.
Type: GFieldType; The type of postal address.
City: string; The city value of the address.
Country: string; The country value of the address.
CustomType: string; The type description if AddressType is set to ftCustom.
Neigborhood: string; The neighborhood value of the address.
POBox: string; The PO Box value of the address.
PostCode: string; The post code value of the address.
Primary: Boolean; Indicates if this is the contact's primary postal address.
Region: string; The region value of the address.
Street: string; The street value of the address.

Relations: TGRelations: Contains a list of relations of the contact.

CustomRelation: string; The type description if Relation is set to grCustom.
Type: GRelationType; The type of relation.
Value: string; The value text (Name) of the relation.

Title: string; The title of the contact.
Websites: Contains a list of websites for this contact.

CustomType: string; The type description if WebsiteType is set to wtCustom.
URL: string; The URL of the website.
Type: GWebsiteType; The type of website.

Groups: Contains a list of Google Group items.

ID: string; The ID of the group.
Summary: string; the group description.

Methods:

```
List<Group> GetGroups()
```
Fill the list of groups.

```
List<Contact> GetContacts()
```
Fill the list of contacts.

```
Contact AddContact(Contact contact)
```
Add a new Google contact item.

Example:

```
Contact co = new Contact();
co.Name.GivenName.Value = tbFirstName.Text;
co.Name.FamilyName.Value = tbLastName.Text;
googleContacts1.Add(co);
```

```
Contact UpdateContact(Contact contact)
```
Update an existing Google contact item.

```
void DeleteContact(Contact contact)
```
Delete a Google contact item.

```
Group AddGroup(Group group)
```
Add a new Google group item.

```
Group UpdateGroup(Group group)
```
Update an existing Google group item.

```
void DeleteGroup(Group group)
```
Delete a Google group item.

Picasa

Usage

Picasa is a component that facilitates access to the Google Picasa service. It allows retrieving your albums and photos as well as creating / deleting albums, uploading / downloading photos to albums. Photos can be uploaded / updated with a title / description and a geo location. Comments on photos can also be retrieved. The component also enables searching photos in public albums of other users.

Organisation

Properties:

Albums: Collection that contains a list of Picasa albums.

Id: string; The album Id.
Title: string; The title of the album.
Media: The list of links to the image connected to this album.

Url: string; The link to the image file of the photo.

Summary: string; The summary of the album.
Updated: DateTime; The time when the album was last updated.

Photos: Collection that contains a list of Picasa photos.

Id: string; The photo Id.
AlbumId: string; The Id of the album the photo belongs to.
Author: The author of the photo.
Image: string; The link to the image file of the photo
Latitude: double; The latitude value of the geolocation of the photo.
Longitude: double; The longitude value of the geolocation of the photo.
Summary: string; The description of the photo.
Media: List of image files associated with the photo.

Tags: string; The tags that have been set for the photo.
Thumbnail.URL: string; The link to the thumbnail image file of the photo.

Updated: DateTime; The time when the photo was last updated.

Comments: Collection that contains a list of Picasa comments.

Id: string; The comment Id.
Author: The author of the comment.
Published: DateTime: The time when the comment was published.
Content: string; The content text of the comment.
Title: string; The title of the comment.

Classes:

Author: Class that contains user information.

Id: string; The user Id.
Name: string; The full name of the user.
NickName: string; The nickname of the user.

Methods:

```
List<Album> GetAlbums(string userId)
```
Fill the list of albums, accessible via Picasa.Albums. If no userId is provided the albums from the currently authenticated user are retrieved.

```
List<Photo> GetPhotos(string userId, string albumId, PicasaPhotoSize maxPhotoSize)
```
Fill the list of Picasa photos assigned to the album, accessible via AdvPicasa.Photos. If no userId is provided it is assumed the albumId belongs to the currently authenticated user.

```
Album CreateAlbum(string title, string description)
```
Create a new Picasa album with properties set via Album parameter on the Picasa web service.

Example:
```
picasa1.CreateAlbum("Album Title", "Album Description";
```

```
void DeleteAlbum(Album album)
```
Delete an existing Picasa album and all photos assigned to the album.

```
void DeletePhoto(Photo photo)
```
Delete an existing Picasa photo.

```
void UploadPhoto(string albumId, string fileName, string title, string summary, string tags, double latitude, double longitude)
```
Upload a new photo to an existing Picasa album.

Example:
```
if (openFileDialog1.ShowDialog() == DialogResult.OK)
{
      picasa1.UploadPhoto(albumId, openFileDialog1.FileName, "Title", "Description", "Tag A, Tag B", 0, 0);
}
```

```
void UpdatePhoto(string photoId, string albumId, string title, string summary, string tags, double latitude, double longitude)
```
Update an existing Picasa photo.

```
List<Photo> SearchPhotos(string keywords, int page = 0, int pageSize = 0)
```
Search in the available public Picasa albums for photos that match the provide keyword(s).

```
List<Comment> GetComments(Photo photo)
```

Fill the list of Picasa comments that have been made for the photo.

```
void DownloadFile(string url, string fileName)
```
Download a photo to a local folder.

LinkedIn

Usage

LinkedIn is a component that provides access to the LinkedIn API service. It allows posting an activity, sharing a message and retrieving a list of connections. It also enables searching for People, Companies and Jobs that are listed on LinkedIn.

Organisation

Properties:

Person UserProfile: Class that contains the Facebook profile info for the currently authenticated user.

Id: string; The user's LinkedIn Id.
FormattedName: string; The user's formatted name.
FirstName: string; The user's first name.
LastName: string; The user's last name.
MaidenName: string; The user's maiden name.
EmailAddress: string; The contact email address granted by the user.
NumberOfConnections: string; The number of connections of the user.
NumberOfConnectionsCapped: boolean; Indicates if the NumberOfConnections value is capped. (LinkedIn won't communicate if a user has 500+ connections)
Positions: PositionCollection; List of the user's job positions.

Id: string; The Id for this position.
Company: CompanyBase; The company related to this position.
EndDate: LinkedInDateTime; The date indicating when the position ended.
IsCurrent: Boolean; Indicates if this is the current position of the user.
Title: string; The job title held at this position, as indicated by the user.
Summary: string; The summary of the user's position.
StartDate: LinkedInDateTime; The date indicating when the position began.

Location: Location; The location of the user.
CurrentShare: ShareResult; The current share of the user.

Id: string; The Id for this share.
Author: PersonBase; The author for this share.
Comment: string; The comment for this share.
Content: ShareResultContent; The description for this share.

Title: string; The title of this share.
ResolvedUrl: string; The URL for this share.

TimeStamp: string; The time this share was posted.

PublicProfileURL: string; The URL for the user's LinkedIn profile.
Distance: integer; The degree distance of the fetched profile from the user who fetched the profile.
LastModifiedTimeStamp: DateTime; The time when the user's profile was last modified.
Associations: string; The associations of the user.
HonorsAwards: HonorsAwardsCollection; The honors and awards of the user.
ProposalComments: string; The description of how the user approaches proposals.
Publications: PublicationCollection; A list of the user's publications.

Id: string; The Id for this publication.
Date: string; The time this publication was published.
Title: string; The title of this publication.

Patents: PatentCollection; A list of the user's patents.

Id: string; The Id for this patent.
Date: string; The time related to this patent.
Title: string; The title of this patent.

Languages: LanguageCollection; A list of the user's languages.

Id: string; The Id for this language.
LanguageName; The name of this language.

Skills: SkillCollection: A list of the user's skills.

Id: string; The Id for this skill.
Name: The name of this skill.

Certifications: CertificationCollection; A list of the user's certifications.

Id: string; The Id for this certification.
Name: The name of this certification.

Educations: EducationCollection; A list of the user's educations.

Id: string; The Id for this education.
Degree: string; The description of the degree received at this institution.
SchoolName: string; The name of the school as indicated by the user.

Courses: CourseCollection; A list of the user's courses.

Id: string; The Id for this course.
Name: string; The name of this course.
Number: string; The course number assigned, as entered by the user.

VolunteerExperiences: VolunteerExperienceCollection; A list of the user's volunteer experiences.

Id: string; The Id of this volunteer experience.
OrganizationName; The name of the organization the user has volunteered with.
Role: string; The role the member has performed as a volunteer.

Recommendations: RecommendationCollection; A list of the recommendations the user has received.

Id: string; The Id of this recommendation.
RecommendationText: string; The text of the recommendation received.
RecommendationType: string; Indicates the type of recommendation that was selected by the person making the recommendation.
Recommender: PersonBaseCollection; The profile of the person who made the recommendation.

Following: PersonBaseNormalCollection; A list of the items the user is following.

JobBookmarks: A list of jobs that the user has bookmarked.

ID: string; The ID for this job bookmark.
Company: The company this job is listed for.
Description: string; The description of this job.
IsApplied: boolean; Indicates if the user has applied for the job.
IsActive: boolean; Indicates if the job listing is active.
PositionTitle: string; The title of the position.
SavedTimeStamp: TDateTime; The time this job bookmark was created.

GroupMemberships: GroupMembershipCollection; A list of LinkedIn groups the user is a member of.

Id: string; The Id for this group.
Group; The LinkedIn group.
MembershipState: StateHolder; The state of the user's membership to the specified group.

DateOfBirth: string; The user's birth date.
Resources: UrlCollection; A list of URLs the user has chosen to share on their LinkedIn profile.

Name: string; The name of the resource.
URL: string; The resource url.

PhoneNumbers: PhoneCollection; A list of the user's phone numbers.

PhoneNumber: string; The phone number.
PhoneType: string; The type of this phone number.

IMAccounts: IMAccountCollection; A list of the user's instant messaging accounts.

AccountName: string; The name of this IM account.
AccountType: string; The type of this IM account.

MainAddress: string; The user's address.
Headline: string; The user's headline.
Industry: string; The industry the LinkedIn member has indicated their profile belongs to.
Interests: string; A description of the user's interests.
PictureUrl: string; An url to the user's profile picture.
Specialties: string; A description of the user's specialties.
Summary: string; A description of the user's professional profile.

Classes:

PersonConnection: User profile the currently authenticated user is connected with.

Company:

Id: string; The Id for the company.
BlogRssURL: string The URL for the company blog.
Name: string; The name of the company.
Type: string; The type of the company.
Description: string; The description of the company.
LogoURL: string; The Url of the company logo image.
Industries; A list containing the names of the company's industries.
Locations: CompanyLocationCollection; A list of the company's locations.

Street1: string; The first line of the street address.
Street2: string; The second line of the street address.
City: string; The city for this location.

State: string; The state for this location.
PostalCode: string; The postal code for this location.
RegionCode: string; The region code for this location.
CountryCode: string; The country code for this location.
Phone1: string; The company phone number for this location.
Phone2: string; The second company phone number for this location.
Description: string; The company location description.

Size: string; The number range of employees at the company.
Specialties: StringList; A list of the company's specialties.
StockExchange; The stock exchange the company is in.
Ticker: string; The company ticker identification for the stock exchange.
TwitterId: string; The Id for the company Twitter feed.
WebsiteUrl: string; The company website address Url.

Job:

Id: string; The Id for this job.
Company; The hiring company for this job.
CountryCode: string; The country code for this job.
Description: string; The description for the position.
IsActive: boolean; Indicates of the job listing is active.
SiteJobUrl: string; The Url for the job listing.
JobPoster: TLinkedInProfile; The user who posted the job listing.
Location: string; The location for this job.
Position; The description of the job position.
Industries: LinkedInObjects; A list of industries related to this position.

PostingTimeStamp: string; The date of the job listing.
Salary: string; The salary for this job listing.
SkillsAndExperience: string; The description of the skills and experience needed for the listed position.

Methods:

```
bool PostActivity(string activity,string userId="~")
```
Post an activity message on the stream of the user.

```
Person GetAccountInfo()
```
Fill the DefaultProfile class with the profile information of the user.

```
PersonCollection GetConnections(string memberId, int? start, int? count,
ConnectionModifyType? modified, string modifiedSince)
```
Fill the PersonCollection with the connections of the user.

```
Person GetProfileById(string memberId, bool returnSecureUrl = true)
```
Get the profile information of a user based on the member ID.

```
CompanySearchResult SearchForCompany(string keyWords, bool? hqOnly = false,
CountriesEnum? countryCode=null,int? industryCode=null, string sort = null, uint?
start = null, uint? count = null)
```
Search for companies based on the parameters.

Returns a list of companies.

```
JobSearchResult SearchForJobs(string keyWords = "", string jobTitle = "", string
companyName = "", string countryCode = "",uint? count=null,uint? start=null)
```
Search for job listings based on the parameters.

Returns a list of jobs.

```
PeopleSearchResult SearchForPeople(string keywords = null, string firstName = null,
string lastName = null, string companyName = null, bool? currentCompany = null,
string title = null, bool? currentTitle = false, string schoolName = null,
bool? currentSchool = null, string countryCode = null,
string postalCode = null, int? distance = null, uint? start = null,
uint? count = null, string facetCode = null, Facet values = null,
string sort = "connections")
```
Search for people based on the parameters.

Returns a list of people.

```
ShareResponse Share(string comment,string title,string submittedUrl,string
submittedImageUrl=null,string description=null,ShareVisibility
visibility=ShareVisibility.Anyone)
```
Share a message on the stream of the user.

Example:

```
  linkedIn1.Share("comment", tbShare.Text, tbLink.Text, tbImage.Text,
tbShareDesc.Text);
```

LiveCalendar

Usage

LiveCalendar is a component that provides access to the Windows Live calendar service. It allows to retrieve a list of Windows Live calendars and read, create, update & delete Windows Live calendar events.

Organisation

Properties:

Calendars: Contains a list of Live calendars.

ID: string; The calendar ID.
Name: string; The name of the calendar.
Description: string; The description of the calendar.
ReadOnly: Boolean; Indicates if events can be added/updated for this calendar.

Calendar: Contains a list Live calendar events.

ID: string; The event ID.
CalendarID: string; The ID of the calendar this event belongs to.
Name: string; The name of the event.
Created: DateTime: The date and time the event was created.
Updated: DateTime: The date and time the event was last updated.
Description: string; The description of the event.
StartTime: DateTime; The start time of the event.
EndTime: DateTime; The end time of the event.
IsAllDay: Boolean; Indicates if this is an all-day event.
Location: string; The location of the event.
Availability: Indicates if the event availability: free, busy, tentative, outofoffice.
Visibility: Indicates if the event is public, private or confidential.
IsRecurrent: Boolean; Indicates if this is a recurring event.
Recurrence: string; Description of the event recurrence.

Methods:

```
List<Calendar> GetCalendars(string userId)
```
Fill the list of Calendars. If no userId is provided the Calendars for the currently authenticated user are retrieved.

```
List<Event> GetCalendar(string calendarId, DateTime startDate, DateTime endDate)
```
Fill the Items list with calendar events from a Live Calendar for a certain timespan. If no ID is provided, the events are retrieved from the default Live Calendar. If no FromDate/ToDate is specified the events are retrieved for the default timespan.

```
Event AddEvent(Event anEvent)
```
Add a new event to the Calendar as specified by Item.CalendarID.

```
Event UpdateEvent(Event anEvent)
```
Update an existing event.

Example:
```
Event Event;

Event.Name = tbSubject.Text;
Event.Description = tbDescription.Text;
Event.Location = tbLocation.Text;
liveCalendar1.UpdateEvent(ev);
```

```
void DeleteEvent(Event anEvent)
```
Delete an event.

LiveContacts

Usage

LiveContacts is a component that provides access to the Windows Live contacts service. It enables to read and create contacts.

Organisation

Properties:

Items: Contains a list of Live Contact items.

Id: string; The Id of the contact.
FirstName: string; The first name of the contact.
LastName: string; The last name of the contact.
FullName: string; The full name of the contact.
Gender: Gender; The gender of the contact.
IsFriend: Boolean; Indicates if the contact is a friend.
IsFavorite: Boolean; Indicates if the contact has been added as a favorite.
UserID: string; The user ID of the contact.
BirthDay: integer; The day part of the contact's birthday.
BirthMonth: integer; The month part of the contact's birthday.

Methods:

```
ContactCollection GetContacts(string userId)
```
Fill the list of Contacts. If no userId is provided the contacts for the currently authenticated user are retrieved.

```
Contact Add(Contact contact)
```
Add a new Live contact item.

Example:

```
Contact co = new Contact();
co.FirstName = "First Name";
co.LastName = "Last Name";

liveContacts1.Contacts.Items.Add(co);
liveContacts1.Add(co);
```

PushOver

Usage

PushOver is a component that sends push messages to the PushOver client running on iOS devices. This service allows to send free messages to one or more devices with the PushOver client from a Windows application.

Organisation

PushOver descends from CloudBase and exposes following properties and methods:

Properties

PushOverMessage

Class giving access to all settings associated with a PushOver message:

string User;
PushOver ID of the user.

string Title;
Title of the message to send. Can be up to 50 characters.

string Device;
Device name of the user where to send the message. Only required when the user has multiple devices and the message needs to be sent to a single device only.

string Message;
Content of the message itself. Can be up to 500 characters.

URL: string;
Optional URL to associate with the message.

string URLTitle;
Optional title to set for the URL that will be sent with the message

MessagePriority Priority;
Can be any of following values:

public enum MessagePriority = {mpNone,mpQuiet,mpHighPriority,mpConfirmation}

mpNone is the default value for normal priority push message
mpQuiet is message without popup & sound on the device
mpHighPriority set when message needs to be sent with high priority
mpConfirmation sends a push message with receipt confirm prompt

DateTime TimeStamp;
When different from zero, sends a timestamp different from the server timestamp when the message is sent.

MessageSound Sound;
Select the sound to associate with the push message. The value can be:

public enum MessageSound = {msDefault,msBike,msBugle,msCashRegister,msClassical,msCosmic,

msFalling,msGamelan,msIncoming,msIntermission,msMagic,msMechanical,msPianoBar,

msSiren,msSpaceAlarm,msTugBoat,msAlien,msClimb,msPersistent,msEcho,msUpDown,msNone}

Methods

public bool PushMessage(string user, string message);

Sends a simple text message to user with PushOver ID AUser.

public bool PushMessage(string user, string title, string message);

Sends a simple title and text message to user with PushOver ID AUser.

public bool PushMessage(string user, string device, string title, string message);

Sends a simple title and text message to device ADevice belonging to user with PushOver ID AUser

public bool PushMessage(PushOverMessage message);

Sends a message with all properties as defined by AMessage.

Sample code

```
PushOver pt = new PushOver();
// set the PushOver application ID
pt.Authenticator.OAuthApplication.Key = APPID;
PushOverMessage msg = new PushOverMessage();

// set the PushOver user ID to send the message to
msg.User = USERID;
msg.Title = “Hello world”;
msg.Message = “My first message to PushOver”;
msg.Sound = MessageSound.Bike;
pt.PushMessage(msg);
```

CardDAV

Usage

CardDAV is a component that retrieves vCard (contacts) information from a CardDAV server.

Properties

Bool Active
Set Active to true or call CardDAV.Open() to connect to the server

bool IgnoreCertificateError
When true, ignores errors raised when the server certificate is not valid or recognized.

CardDavItemCollection Items
List of CardDavItem items. A TCardDAVItem contains the information of a contact and the information is available via CardDavItem.vCard.

string Password
Account password

string URL: string
Sets the URL of the CardDAV server

string Username
Account username

string UserAgent
Sets the name of the agent used to identify the client to the server

CardDavItem object

The CarDavItem object represents a vCard item on the CardDav server and has following properties:

string DisplayText
Summary text for the contact

vCard vCard
Access to the vCard for the contact

vCard object

The vCard contains a collection of vContacts. In typical CardDav exchange, the information for the contact is in vCard.vContacts[0]

vContact object

string Id
Unique identifier of the contact

Image Image
Image associated with the contact

string FirstName
First name of the contact

string LastName
Last name of the contact

string MiddleName
Middle name of the contact

string NamePrefix
Optional prefix for the contact name

string NameSuffix
Optional suffix for the contact name

string FullName
Full name of the contact

string NickName
Nick name of the contact

string Company
Company name where the contact works

string JobTitle
Job title of the contact

DateTime BirthDay
Date of birth of the contact

string Profession
Profession of the contact

string WebSiteUrl
URL of the contact website

vTypeCollection<vCardEmail>Emails
Collection of email addresses associated with the contact

vTypeCollection<vCardPhone> Phones
Collection of phone numbers associated with the contact

vTypeCollection<vCardAddress> Addresses
Collection of addresses associated with the contact

vGeoLocation GeoLocation
Geolocation of the contact

StringList Note
Additional information for the contact

string PhotoUrl
Optional Url to a photo for the contact

string TimeZone
Timezone where contact is

string SortString
String to use for sort order of the contact when it should be different from sorting on last name

Methods

bool CardDAV.Open()
Establishes a connection to the server and retrieves the contacts data. This is equivalent to setting Active = true.

bool CardDAV.Close()
Closes the connection to the server.

void CardDAV.Sync()
Performs a synchronization of local data with data on the server.

Events

AfterOpen
Event triggered when a connection to the server was established

BeforeOpen
Event triggered just before is connection will be established

Example

This code snippet shows how to connect to the server and retrieve contacts information:

1) connect to the server

```
    cardDav1.Username = YourUserName;
```

```
cardDav1.Password = YourPassword;
cardDav1.UrlAddress = "https://caldav.icloud.com"  //A CalDav server
cardDav1.UserAgent = "TMS Cloud Pack for .NET"
cardDav1.Open();
```

2) retrieve the contacts

```
//add the contact information in a listview
listView1.Items.Clear();
foreach (var c in cardDav1.Items)
{
  var addedItem = listView1.Items.Add(c.DisplayText);

  if (c.vCard.vContacts.Count > 0)
  {
    vContact contact = c.vCard.vContacts[0];

    addedItem.SubItems.Add(contact.FirstName);
    addedItem.SubItems.Add(contact.LastName);
    addedItem.SubItems.Add(contact.Company);
    addedItem.SubItems.Add(contact.JobTitle);
    addedItem.SubItems.Add(contact.Note.Text);

    if (contact.Phones.Count > 0)
    {
    addedItem.SubItems.Add(contact.Phones[0].PhoneNumber);
    }
    else
    addedItem.SubItems.Add("");

    if (contact.Emails.Count > 0)
    {
    addedItem.SubItems.Add(contact.Emails[0].EmailAddress);
    }
    else
    addedItem.SubItems.Add("");
  }

}
```

The CardDavItem has the methods Delete, Update, Post to perform CRUD actions.
Thus, to delete some contact, use:

```
cardDav1.Items[CardIndex].Delete();
cardDav1.Items[CardItems[0].Index].Post();
```

to update an item, use:

```
var item = cardDav1.Items.UpdateItem(CardIndex);
vContact contact;
if (item.vCard.Contacts.Count > 0)
```

```
    contact = item.vCard.Contacts[0];
  else
    contact = item.vCard.Contacts.Add();
  contact.FirstName = SomeFirstName;
  contact.LastName = SomeLastName
  contact.FullName = SomeLastName + " " + SomeFirstName;
  contact.Company = SomeCompany;
  contact.Note.Text = SomeNotesText;
  item.Post();
```

Finally, to create a new contact, use the code:

```
  var item = cardDav1.Cards[0].Items.InsertItem();
  var contact = item.vCard.Contacts.Add();
  contact.FirstName = SomeFirstName;
  contact.LastName = SomeLastName
  contact.FullName = SomeLastName + " " + SomeFirstName;
  contact.Company = SomeCompany;
  contact.Note.Text = SomeNotesText;
  item.Post();
```

CalDAV

Usage

CalDAV is a component that retrieves vCalendar information from a CalDAV server.

Properties

bool Active
Set Active to true or call CalDAV.Open() to connect to the server

bool IgnoreCertificateError
When true, ignores errors raised when the server certificate is not valid or recognized.

CalendarCollection Calendars
List of calendars in the CalDAV server. A CalDavCalendar contains the information of events in the calendar and this information is available via CalDavCalendar.Items. CalDavCalendar.Items is a CalDavItemsCollection containing CalDavItem instances.

string Password
Account password

string URL
Sets the URL of the CalDAV server

string Username
Account username

string UserAgent
Sets the name of the agent used to identify the client to the server

CalDavItem object

The CalDavItem object represents a vCalendar item for a CalDav event and has following properties:

string DisplayText
Summary text for the event

vCalendar vCalendar
Access to full vCalendar information for the event. vCalendar contains a collection of vEvents, vTodos, vJournals and vTimeZones

vEvent object

StringList Description
Stringlist hold the description text for an event

DateTime DtStart
Start time of the event

DateTime DtEnd
End time of the event

TimeSpan DtStamp
Timespan of the event

vOrganizer Organizer
Organizer of the event

string Id
Unique identifier of the event

string Summary
Summary text for the event

vAttendees Attendees
Collection of attendees for the event

vRequestStatus RequestStatus
Status of event invitation request

Methods

bool CalDAV.Open()
Establishes a connection to the server and retrieves the calendar data. This is equivalent to setting Active = true. Returns false when opening fails.

bool CalDAV.Close()
Closes the connection to the server.

void CalDAV.Sync();
Performs a synchronization of local data with data on the server.

Events

AfterOpen
Event triggered when a connection to the server was established

BeforeOpen
Event triggered just before is connection will be established

Example

This code snippet shows how to connect to the server and retrieve calendar information:

1) connect to the server

```
calDav1.Username = YourUserName;
calDav1.Password = YourPassword;
calDav1.UrlAddress = "https://caldav.icloud.com"  //A CalDav server
calDav1.UserAgent = "TMS Cloud Pack for .NET"
calDav1.Open();
```

2) retrieve the calenders

```
//add the calendar names to a listbox
listBox1.Items.Clear();
foreach (var c in calDav1.Calendars)
{
  listBox1.Items.Add(c.DisplayText);
}
```

3) retrieve the events from a selected calendar

```
//add the calendar events to a listview

listView1.Items.Clear();
foreach(var c in calDav1.Calendars[folderIndex].Items)
{
   var addedItem = listView1.Items.Add(c.DisplayText);
   // in case it is an event type, add start & end datetime
   if (c.vCalendar.vEvents.Count>0)
   {
     addedItem.SubItems.Add(c.vCalendar.vEvents[0].DtStart.ToString());
     addedItem.SubItems.Add(c.vCalendar.vEvents[0].DtEnd.ToString());
   }
}
```

The CalDAVItem has the methods Delete, Update, Post to perform CRUD actions.

Thus, to delete some contact, use:

```
calDav1.Calendars[CalendarIndex].Items[ItemIndex].Delete();
calDav1.Calendars[CalendarIndex].Items[ItemIndex].Post();
```

to update an item, use:

```
var insertedItem = calDav1.Calendars[CalendarIndex].Items.UpdateItem(ItemIndex);
var anEvent = insertedItem.vCalendar.vEvents[0];
anEvent.Summary = newsummarytext;
anEvent.DtEnd = newendtime;
anEvent.DtStart = newstarttime;
anEvent.Description.Text = newdescriptiontext;
insertedItem.Post();
```

Finally, to create a new calendar event, use the code:

```
var insertedItem = calDav1.Calendars[CalendarIndex].Items.InsertItem();
var anEvent = insertedItem.vCalendar.vEvents.Add();
anEvent.Summary = newsummarytext;
anEvent.DtEnd = newendtime;
anEvent.DtStart = newstarttime;
anEvent.Description.Text = newdescriptiontext;
insertedItem.Post();
```

myCloudData

Usage

MyCloudData is a component that provides seamless access to the **myCloudData.net** service that allows you to create and design your tables and store and retrieve your data with a very limited amount of code.

A user can have access to one or more tables.
After login, the collection of tables available to the user is returned with `MyCloudData.GetTables()` or accessible via `MyCloudData.Tables`.

A table on myCloudData.net is represented by the class `MyCloudDataTable`.
A table has **metadata**, **entities**, **filters**, **sortorder** and **shares**.

The table entities are represented by the class `MyCloudDataEntity`. The shares are represented by the class`MyCloudDataTableShare`. The meta data for a table is retrieved via `Table.MetaData`. The entities are retrieved via the `Table.Query` function and the list of shares is retrieved with `Table.GetShares`. The filter is accessible via the collection `Table.Filters` and the sort ordering can be setup via the collection `Table.Sorting`.

MyCloudData

`MyCloudData` is the class that wraps the entire access to the myCloudData.net service.

Properties

```
MyCloudDataTableCollection Tables { get; }
```
Returns the collection of tables for the current user

Methods

```
MyCloudDataTableCollection GetTables();
```
Returns the collection of tables for the current user

```
MyCloudDataTable CreateTable(string tableName);
```
Creates a new table with a given tableName and returns an instance to the table

```
void DeleteTable(int tableId);
```
Deletes a table based on its unique ID

```
void UpdateTable(MyCloudDataTable table);
```
Updates table info, such as name, permissions based on an existing MyCloudDataTable class

```
void TruncateTable(int tableId);
```
Clear all records from a table

```
MyCloudDataTableMetaData GetTableMetaData(int tableId);
```
Retrieves the metadata for the table specified by a tableId

```
void AddMetaDataField(MyCloudDataFieldMetaData fieldToAdd);
```
Adds a new metadata field to a table. The tableId is specified on the MyCloudDataFieldMetaData instance.

```
void DeleteMetaDataField(int tableId, string fieldName);
```
Removes a metadata field from a given table

```
void UpdateMetaDataField(MyCloudDataFieldMetaData metaData);
```
Updates a metadata field on a table. The tableId is specified on the MyCloudDataFieldMetaData instance.

```
MyCloudDataEntityQueryResult Query(MyCloudDataEntityQuery query);
```
Retrieves a set of entities from a table specified by the paremeters set in the MyCloudDataEntityQuery instance.

```
int GetQueryCount(MyCloudDataEntityQuery query);
```
This method takes the same arguments as the Query method but only returns the number of records that match the query conditions.

```
long AddEntity(MyCloudDataEntity entity);
```
Inserts a new intity and returns its unique ID.

```
void DeleteEntities(int tableId, IEnumerable<long> entityIds);
```
Removes a set of entities from a table.

```
void UpdateEntity(MyCloudDataEntity entity);
```
Updates an entity.

```
void UpdateMultipleEntities(int tableId, IEnumerable<long> entityIds, IDictionary<string, object>fieldsToUpdate);
```
Update multiple entities in a table specified by tableId. Only entities with an ID specified in entityIds will be updated. Only fields specified in fieldsToUpdate will be updated.

```
void UpdateAllEntities(int tableId, IDictionary<string, object> fieldsToUpdate);
```
Update all entities in a table specified by tableId. Only fields specified in fieldsToUpdate will be updated.

```
Stream GetBlob(int tableId, long entityId, string blobFieldName);
```
Download a blob value in a stream from a specific entity in a specific table and fieldname

```
void SaveBlob(int tableId, long entityId, string blobFieldName, Stream fileStream);
```
Upload a file to a blob field from a specific entity in a specific table and fieldname

```
void SaveShare(int tableId, MyCloudDataTableShare share);
```
Share a table with specific permissions with another myCloudData.net user

```
IEnumerable<MyCloudDataTableShare> GetTableShares(int tableId);
```
Returns the a collection of shares for a given table

MyCloudDataTable

This class represents the table in the cloud storage and is part of the set of tables in the collection `MyCloudData.Tables`.

Properties

```
int TableId { get; }
```
Read-only property returning the unique identifier of the table

```
string TableName { get; set; }
```
Gets or sets the name of the table

```
int OwnerId { get; }
```
Read-only property returning the unique owner identifier of the table

```
bool IsOwner { get; }
```
Read-only property returns true when the logged in user owns the table

```
MyCloudDataTablePermissions Permissions { get; }
```
Permissions the current user has on the table. Permissions are: CRUD, i.e. create/read/update/delete

```
MyCloudDataTableMetaData MetaData { get; set; }
```
Access to the metadata of the table

```
MyCloudDataEntityFilterCollection Filters { get; set; }
```
Access to the filter conditions for a query

```
MyCloudDataEntitySortDirectionCollection Sorting { get; set; }
```
Access to the sort order settings for a query

```
int PageSize { get; set; }
```
Access to the page size setting for a query

```
int PageIndex { get; set; }
```
Access to the page index setting for a query

Methods

```
void SaveMetaData()
```
Looks for changes on the metadata of the table and sends the updates to the myCloudData API.

```
void InvalidateCachedMetaData()
```
The metadata is loaded and cached on first request, this method can be used to remove it from the cache.

```
MyCloudDataEntityQueryResult Query()
```
Executes a query on the table.

```
MyCloudDataEntityQueryResult Query(List<string> fields)
```
Query with specifier of selection of fields to return

```
MyCloudDataEntity CreateEntity()
```
Creates a new entity on this table, the entity is not saved until the SaveEntities() method is called.

```
void UpdateEntity(MyCloudDataEntity entity)
```
Updates an entity on the table, the entity is not saved until the SaveEntities() method is called.

```
void RemoveEntity(long entityId)
```
Remove an entity from the table, the entity is not deleted until the SaveEntities() method is called.

```
void SaveEntities()
```
This method bundles all changes made on the entities of the table and sends the requests to the myCLoudData API.

```
IEnumerable<MyCloudDataTableShare> GetShares()
```
Retrieve the list of email addresses and permissions with who the table was shared

```
void SaveShare(string email, MyCloudDataTablePermissions permissions)
```
Add or modify a users permission on this table

```
void RemoveShare(string email)
```
Remove a users permission on this table

```
LookupDataCollection GetFieldLookupData(string fieldName)
```
Retrieves the lookup data for a given field. It is required that the fieldmetadata contains valid values for the LookupTable, LookupField and LookupKeyField properties in order to be able to retrieve lookup data.

```
void InvalidateCachedLookupData()
```
Invalidate the cached lookup data for all fields on the table.

MyCloudDataTableShare

This class represents a share on a table.

Properties

```
string Email { get; }
```
The email address of the user this table is shared with

```
MyCloudDataTablePermissions Permissions { get; set; }
```
The permission this user has on the table

MyCloudDataEntityQuery

This is the class that contains all conditions and parameters for performing a query on a table using the `MyCloudData.Query()` function. This class is used behind the scenes when you perform a query on a table by using `MyTable.Query()`.

Properties

```
int TableId { get; }
```
Gets the unique ID of the table you want to perform your query on.

```
int PageSize { get; set; }
```
Gets or sets the size of the page you want to retrieve.

```
int pageIndex { get; set; }
```
Gets or sets the index of the page you want to retrieve.

```
List<string> Fields { get; set; }
```
Gets or sets set of fields you want to retrieve.

```
IEnumerable<MyCloudDataEntityFilter> pageIndex { get; set; }
```
Gets or sets the the filter conditions for a query.

```
IEnumerable<MyCloudDataEntitySortDirection> pageIndex { get; set; }
```
Gets or sets the sort order settings for a query.

MyCloudDataEntityQueryResult

This is the class that gets returned when performing a query. This is a read-only collection of MyCloudDataEntity objects.

Properties

```
int PageSize { get; }
```
Gets the size of the page.

```
int Pageindex { get; }
```
Gets the index of the page.

```
int TotalRecords { get; }
```
Gets the total amount of results that match the query.

MyCloudDataFieldMetaDataCollection

This class represents the collection of fields that exist on a single table.

Properties

```
int TableId { get; }
```
Gets the unique Id of the table.

Methods

```
MyCloudDataFieldMetaData GetField(string fieldName)
```
Returns a field by name.

```
bool ContainsField(string fieldName)
```
Returns true if the collection contains a field with that name.

```
MyCloudDataFieldMetaData AddField(string fieldName, MyCloudDataFieldDataType dataType,
 int? size = null)
```
Adds a field to the collection.

```
MyCloudDataFieldMetaData AddOrUpdateField(string fieldName, MyCloudDataFieldDataType d
ataType, int? size = null)
```
Adds of updates a field on the collection.

MyCloudDataFieldMetaData

This class holds the information about a single field in the field collection of a table.

Properties

```
string FieldName { get; set; }
```
Gets or sets the field name

```
MyCloudDataFieldDataType DataType { get; set; }
```
Gets or sets the field type. The field type can be any of following values `Integer`, `Float`, `String`, `Boolean`, `DateTime`, `Date`, `Time`, `Binary`

```
int? MaxSize { get; set; }
```
Optionally gets or sets the size of a field (only available for fields of type String)

```
string LabelText { get; set; }
```
Gets or sets the label text associated with the field

```
string DefaultValue { get; set; }
```
Gets or sets the default value associated with the field

```
int? Width { get; set; }
```
Gets or sets the width associated with the field

```
int? Order { get; set; }
```
Gets or sets the order index associated with the field (relative to the other fields in the table)

```
string Mask { get; set; }
```
Gets or sets the field content mask

```
float? Minimum { get; set; }
```
Gets or sets the minimum allowed value associated with the field

```
float? Maximum { get; set; }
```
Gets or sets the maximum allowed value associated with the field

```
DateTime? MinimumDate { get; set; }
```
Gets or sets the minimum allowed date and/or time associated with the field

```
DateTime? MaximumDate { get; set; }
```
Gets or sets the maximum allowed date and/or time associated with the field

```
bool? IsVisible { get; set; }
```
Gets or sets the visibility associated with the field. Default is true

```
bool? IsEnabled { get; set; }
```
Gets or sets the enabled status associated with the field. Default is true

```
bool? IsRequired { get; set; }
```
Gets or sets the required status associated with the field. Default is false

```
string Description { get; set; }
```
Gets or sets the description value associated with the field

```
long? LookupTable { get; set; }
```
Gets or sets the ID of the lookup table associated with the field

```
string LookupField { get; set; }
```
Gets or sets the lookup value field from the LookupTable associated with the field. The value should be identical to one of the field names of the table defined in LookupTable and should be different from the LookupKeyField value.

```
string LookupKeyField { get; set; }
```
Gets or sets the lookup key field from the LookupTable associated with the field. The value should be identical to one of the field names of the table defined in LookupTable and value should be different from the LookupField value.

MyCloudDataEntity

The `MyCloudDataEntity` class is the class that wraps an entity (in database terminology also often referred to as record).

Properties

```
int TableId { get; }
```
The unique ID of the table containing this entity

```
long Id { get; }
```
The unique ID of this entity

```
bool HasChanged { get; }
```
Returns true if the entity has change since it was loaded from the myCloudData.net service.

Methods

```
void SetValue(string fieldName, T value)
```
Sets the value of a field.

```
T GetValue(string fieldName)
```
Gets the value of a field.

```
object GetValue(string fieldName)
```
Gets the value of a field.

```
MyCloudDataBlob GetBlobField(string fieldName)
```
Gets a blob field.

Usage

To set a value for a field within the entity, you can use the `SetValue` method

```
MyEntity.SetValue("MyFieldName", "Field Value");
```

Retrieving the value can be done using the `GetValue` method.
All supported field types: int/float/string/datetime/Boolean/TimeSpan can be get or set this way

```
var fieldValue = MyEntity.GetValue<string>("MyFieldName");
```

The binary blob fields can be accessed as follows:

```
var blobField = MyEntity.GetBlobField("BlobFieldName);
```

This will return an instance of the class `MyCloudDataBlob`. This is explained further in the paragraph covering this class.

MyCloudDataBlob

The `MyCloudDataBlob` class is a wrapper class for binary data stored in a blob in the cloud service.
For performance reasons, blobs are returned via the entity only and retrieved from the cloud storage at the time `GetStream()` is executed.

Note: The blob storage capability is not available for a free account in myCloudData.net but requires a subscription

Properties

```
bool HasData { get; }
```
Returns true if the field contains binary data.

Methods

```
Stream GetStream()
```
Get data from the blob field and save it to a stream.

```
void SetStream(Stream newStream)
```
Load data from the stream into the blob field.

Usage

A typical operation to store some binary data into a blob field in a new entity would be:

```
var dialogResult = openFileDialog1.ShowDialog();
var blobField = MyEntity.GetBlobField("BlobFieldName");
if (dialogResult == DialogResult.OK)
{
    var file = openFileDialog1.FileName;
    using (var fsSource = new FileStream(file, FileMode.Open, FileAccess.Read))
    {
        blobField.SetStream(fsSource);
    }
}
```

myCloudData How-to guides

Creating and designing a table

Getting a table from the myCloudData.net service

You can get the complete list of tables from the service by using the `MyCloudData.GetTables()` method.

To get a specific table you can use the `MyCloudData.GetTableByName()` or the `MyCloudData.GetTableById()` methods.

Creating a table

To create a new table you can use the following snippet.

```
// Determine the name for the table you want to create.
// please try to make the name as "unique" as possible.
var tableName = "MyApplication_Contacts";

// Create the table on myCloudData.net
// if successful this will return a empty table object with a unique TableId
var contactsTable = MyCloudData.CreateTable(tableName);

// Start adding your fields:
contactsTable.MetaData.AddField("ContactID", MyCloudDataFieldDataType.Integer);
contactsTable.MetaData.AddField("FirstName", MyCloudDataFieldDataType.String, 30);
contactsTable.MetaData.AddField("EmailAddress", MyCloudDataFieldDataType.String, 50);
contactsTable.MetaData.AddField("Company", MyCloudDataFieldDataType.String, 50);
contactsTable.MetaData.AddField("PostalCode", MyCloudDataFieldDataType.String, 10);
contactsTable.MetaData.AddField("CountryCode", MyCloudDataFieldDataType.String, 5);

// Save the fields to the myCloudData.net service
contactsTable.SaveMetaData();
```

Please note: The above code will always create a new table, even when there is already a table present with exactly the same name.

Creating or updating a table

Typically, when you start your application you'll want to ensure that the table exists and that the table has all the fields for your application to function correctly.

```
// Get the contacts table
var contactsTable = MyCloudData.GetTableByName("MyApplication_Contacts");

// if the contacts table does not exist => Create it
if (contactsTable == null){
    contactsTable = MyCloudData.CreateTable(tableName);
}
```

```
// Ensure the correct fields are present on the table:
contactsTable.MetaData.AddOrUpdateField("ContactID", MyCloudDataFieldDataType.Integer);
contactsTable.MetaData.AddOrUpdateField("FirstName", MyCloudDataFieldDataType.String, 30);
contactsTable.MetaData.AddOrUpdateField("LastName", MyCloudDataFieldDataType.String, 50);
contactsTable.MetaData.AddOrUpdateField("EmailAddress", MyCloudDataFieldDataType.String,
50);
contactsTable.MetaData.AddOrUpdateField("Company", MyCloudDataFieldDataType.String, 50);
contactsTable.MetaData.AddOrUpdateField("PostalCode", MyCloudDataFieldDataType.String, 10);
contactsTable.MetaData.AddOrUpdateField("CountryCode", MyCloudDataFieldDataType.String, 5);

// Save the fields to the myCloudData.net service
contactsTable.SaveMetaData();
```

Sharing a table

Share a table with another user

In some situations it could be useful to share a table with one or more other myCloudData.net users. See the example below for how to achieve that.

```
// Get the table you want to share
var citiesTable = MyCloudData.GetTableByName("MyApplication_Cities");

// Adding a user that can only read from the table
citiesTable.SaveShare("user@domain.com", MyCloudDataTablePermissions.ReadOnly);

// Adding a user that can read and write data on the table
citiesTable.SaveShare("user@domain.com", MyCloudDataTablePermissions.ReadWrite);

// To a user with custom permissions, you can use the 'FromString' method.
// Options are 'C', 'R', 'U' and 'D' or any combination
var permissions = MyCloudDataTablePermissions.FromString("RU");
citiesTable.SaveShare("user@domain.com", permissions);
```

Removing a share on a table

```
// Option 1:
citiesTable.RemoveShare("user@domain.com");

// Option 2:
citiesTable.SaveShare("user@domain.com", MyCloudDataTablePermissions.None);
```

Querying and filtering entities

Select all entities

To perform a simple query that returns all entities on the table, use:

```
// Return all contacts
var results = contactsTable.Query();
```

Filter entities

To filter data, following code can be used

```
// Clear the filter collection
contactsTable.Filters.Clear();

// Add your filter conditions
contactsTable.Filters.Add("FirstName", ComparisonOperator.Equal, "John");
contactsTable.Filters.Add("Email", ComparisonOperator.Like, "gmail.com",
LogicalOperator.And);

// Optionaly set the page size and index
contactsTable.PageSize = 20;
contactsTable.PageIndex = 0;

// execute the query, this will return the first 20 results matching the conditions above
var results = contactsTable.Query();
```

Note that the ComparisonOperator can be any of the following values: `Equal`, `NotEqual`, `Like`, `Greater`, `GreaterOrEqual`, `Less`, `LessOrEqual`, `StartsWith`, `EndsWith`, `Null`, `NotNull`
The LogicalOperator that sets the logical operation between two sequential filter conditions can be: `And`, `Or`, `None`

Sorting entities

To specify the sort order for a query, the Table.SortOrder collection can be used:

```
// Clear the sorting collection
contactsTable.Sorting.Clear();

// Add your sorting statements
contactsTable.Sorting.Add("Company", SortOrder.Ascending);
contactsTable.Sorting.Add("FirstName", SortOrder.Descending);

// retrieve the results
results = contactsTable.Query();
```

Work with a limited fieldset

This will return and Entities collection with the entities retrieved from the cloud storage.

```
// Create a collection of fields you want to work with
var fieldsToRetrieve = new List<string> { "Company", "CountryCode" }

// Retrieve the entities from the table with only the above fields populated
// The rest of the fields are ignored.
var results = Table.Query(fieldsToRetrieve);
```

Working with entities

Creating entities

Creating a new entity can be done as follows

```
// Get the contacts table
var contactsTable = MyCloudData.GetTableByName("MyApplication_Contacts");

// Create the new entity
var newContact = contactsTable.CreateEntity();

// Populate the fields on the entity
newContact.SetValue("ContactID", 285);
newContact.SetValue("FirstName", "Jean-Pierre");
newContact.SetValue("EmailAddress", "jean-pierre@gmail.com");
newContact.SetValue("PostalCode", "75001");
newContact.SetValue("CountryCode", "FR");

// Save the entity to the myCloudData.net service
myTable.SaveEntities();
```

Please note that nothing will be saved to the myCloudData.Net API until the `SaveEntities()` method is called.

Updating entities

Updating entities can be achieved with the following snippet:

```
// Get the contacts table
var contactsTable = MyCloudData.GetTableByName("MyApplication_Contacts");

// Get all the entities on from the table
var allEntities = contactsTable.Query();

// loop over all entities
foreach (var entity in allEntities)
{
    // update a field value
    entity.SetValue("Company", "TMS software");

    // update the entity on the table
    table.UpdateEntity(entity);
}

// Save the entities to the myCloudData.net service
```

```
table.SaveEntities();
```

Deleting entities

```
// Get the contacts table
var contactsTable = MyCloudData.GetTableByName("MyApplication_Contacts");

// Mark the entity as deleted on the table
contactsTable.RemoveEntity(contactToRemove.Id);

// Save the entities to the myCloudData.net service
contactsTable.SaveEntities();
```

MyCloudData Component

MyCloudData is the class that wraps the entire access to the myCloudData.net service.

Methods & properties available in MyCloudData:

Properties

```
public MyCloudDataTableCollection Tables { get; }
```
Returns the collection of tables for the current user.

Methods

Table related methods:

```
MyCloudDataTableCollection GetTables();
```
Retrieves the list of tables for the current user

```
MyCloudDataTable CreateTable(string tableName);
```
Creates a new table with a given tableName and returns an instance to the table

```
void DeleteTable(int tableId);
```
Deletes a table based on its unique ID

```
void UpdateTable(MyCloudDataTable table);
```
Updates table info, such as name, permissions based on an existing MyCloudDataTable class

```
void TruncateTable(int tableId);
Clear all records from a table
```

Metadata related methods:

```
MyCloudDataTableMetaData GetTableMetaData(int tableId);
```
Retrieves the metadata for the table specified by a tableId

```
void AddMetaDataField(MyCloudDataFieldMetaData fieldToAdd);
```
Adds a new metadata field to a table.
The tableId is specified on the MyCloudDataFieldMetaData instance

```
void DeleteMetaDataField(int tableId, string fieldName);
Removes a metadata field from a given table
```

```
void UpdateMetaDataField(MyCloudDataFieldMetaData metaData);
```
Updates a metadata field on a table.
The tableId is specified on the MyCloudDataFieldMetaData instance

Entity related methods:

```
MyCloudDataEntityQueryResult Query(MyCloudDataEntityQuery query);
```
Retrieves a set of entities from a table specified by the paremeters set in the MyCloudDataEntityQuery instance.

```
int GetQueryCount(MyCloudDataEntityQuery query);
```
This method takes the same arguments as the Query method but only returns the number of records that match the query conditions.

```
long AddEntity(MyCloudDataEntity entity);
Inserts a new intity and returns its unique ID.

void DeleteEntities(int tableId, IEnumerable<long> entityIds);
Removes a set of entities from a table.

void UpdateEntity(MyCloudDataEntity entity);
Updates an entity.

void UpdateMultipleEntities(int tableId, IEnumerable<long> entityIds,
IDictionary<string, object> fieldsToUpdate);
```
Update multiple entities in a table specified by `tableId`. Only entities with an ID specified in `entityIds` will be updated. Only fields specified in `fieldsToUpdate` will be updated.

```
void UpdateAllEntities(int tableId, IDictionary<string, object> fieldsToUpdate);
```
Update all entities in a table specified by `tableId`. Only fields specified in `fieldsToUpdate` will be updated.

Blob related methods:

```
Stream GetBlob(int tableId, long entityId, string blobFieldName);
```
Download a blob value in a stream from a specific entity in a specific table and fieldname

```
void SaveBlob(int tableId, long entityId, string blobFieldName, Stream fileStream);
```
Upload a file to a blob field from a specific entity in a specific table and fieldname

Shares related methods:

```
void SaveShare(int tableId, MyCloudDataTableShare share);
```
Share a table with specific permissions with another myCloudData.net user

```
IEnumerable<MyCloudDataTableShare> GetTableShares(int tableId);
```
Returns the a collection of shares for a given table

Authentication persistence

Internally, after authentication with the cloud service, the component has obtained an access token and for some services also a refresh token. This access token and refresh token can be used at a later time to access the cloud service again without the need for authentication. The components can:

Test if the access token is still accepted

Save the tokens in encrypted form in a file

Load the tokens from a file

The component has methods:

Authenticator.LoadTokensFromFile(string fileName):
Load & decrypt access and refresh token from a file

Authenticator.SaveTokensToFile(string FileName):
Encrypt and save access and refresh tokens to a file

TokensValidation TestTokens():
Performs a test if the access token is still accepted

And the property:

String Authenticator.GetTokens(): encrypted string holding all token values

The Authenticator.GetTokens() call can be used to save and load the token values in a DB field for example.

A typical flow is:

```
string fileName = "tokens.txt";

if (File.Exists(fileName))
       facebook1.Authenticator.LoadTokensFromFile(fileName);

facebook1.Open();
```

The access token, refresh token is saved from the AfterOpen event:

```
        private void facebook1_AfterOpen(object sender, ref bool handled)
        {
            Stream stream = File.Create(fileName);
            stream.Close();
```

```
        facebook1.Authenticator.SaveTokensToFile(fileName);
    }
```

The Open() call will automatically test if the currently loaded tokens are still valid. If the tokens are no longer valid, authentication is started to refresh the tokens.
Token validity can be checked manually by using the TestTokens() call.

Example:

```
if (facebook1.TestTokens() != TokensValidation.TokensAreValid)
    facebook1.Open();
```

Cloud.NET for ASP.NET

Usage

The Cloud.NET controls are compatible with ASP.NET. Apart from some initial configuration settings (explained in Getting Started below), the organization and functionality is identical to that of a Windows application.

Getting Started

- If you are using Visual Studio's built-in web server running on localhost:
  - Make sure you are using "http://127.0.0.1" and not "http://localhost" in the browser's address bar to access the web application
  - Make sure the custom CallBackURL contains the correct port number
- Make sure the registered service application is configured correctly to accept the custom CallBackURL
- Create a new ASP.NET Web Application
- Add a Global.asax files to the project
- Add an optional OAuth.cfg which contains your application Key, Secret and CallBackUrl file to the project
- Add OAuthWebFormsPanel and CloudCacheProvider controls to the form
- Set OAuthWebFormspanel.OpenMode to OpenMode.Redirect.
- For the Twitter and Flickr services it is required that OAuthWebFormsPanel.StatelessRedirection is set to true. For other services this should be set to false (default).
- Create a new instance of a cloud control using the CloudCacheProvider
- If no OAuth.cfg file is used, assign the .Authenticator.OAuthApplication.Key, .Authenticator.OAuthApplication.Secret and .Authenticator.CallBackURL property values programmaticaly
- If an OAuth.cfg file is used, load settings from Oauth.cfg
- Assign the AfterOpen event (optional)
- Authenticate with the service

Please note that the Windows Live service does not accept local urls (localhost, 127.0.0.1), a non local url is required for LiveCalendar and LiveContacts.

Example:

OAuth.cfg file contents:

```
<?xml version="1.0"?>
<oauth>
  <googledrive>
    <key>xxxxxxxxxxxxxxxxx</key>
    <secret>yyyyyyyyyyyy</secret>
    <callback>http://127.0.0.1:57115/</callback>
```

```
  </googledrive>
</oauth>
```

Global.asax file contents:

```
    public class Global : System.Web.HttpApplication
    {
        void Application_BeginRequest(object sender, EventArgs e)
        {
            HttpContext ctx = HttpContext.Current;
            if (CloudBase.GetFileName(ctx.Request.Url.AbsolutePath) ==
"oauth_redirected_postback.tms")
            {
                ctx.Response.End();
                return;
            }
            if (CloudBase.GetFileName(ctx.Request.Url.AbsolutePath) ==
"oauth_redirected.tms")
            {
                ctx.Response.Write("<!DOCTYPE html><html><head></head><body>");
                ctx.Response.Write("<script type=\"text/javascript\">\n");
                ctx.Response.Write("//<![CDATA[\n");
                ctx.Response.Write("//]]>\n");
                ctx.Response.Write("</script>\n");
                ctx.Response.Write("</body></html>");
                ctx.Response.End();
                ctx.Response.Cookies.Add(new HttpCookie("query"));
                return;
            }
            if (CloudBase.GetFileName(ctx.Request.Url.AbsolutePath) ==
"oauth_authorization.tms")
            {
                ctx.Response.Redirect(ctx.Request.Url.Query.Substring(1,
ctx.Request.Url.Query.Length - 1));
                ctx.Response.End();
                return;
            }
        }
    }
```

Code (with OAuth.cfg file):

```
public Dropbox dropbox { get; set; }

OAuthWebFormsPanel1.OpenMode = TMS.Cloud.Authentication.OpenMode.Redirect;
//Only required for Twitter and Flickr
//OAuthWebFormsPanel1.StatelessRedirection = true;
googledrive = CloudCacheProvider1.Factory<GoogleDrive>("GoogleDrive");
googledrive.Authenticator.OAuthPanel = OAuthWebFormsPanel1;
googledrive.Authenticator.LoadSettings();
googledrive.AfterOpen += AfterOpen;
googledrive.Open();
```

```
private void AfterOpen(object sender, ref bool handled)
{
	//Add code to be executed after authentication here
}
```

Code (without OAuth.cfg file):

```
public string CallBackURL = "http://127.0.0.1:57115/ ";
public Dropbox dropbox { get; set; }

OAuthWebFormsPanel1.OpenMode = TMS.Cloud.Authentication.OpenMode.Redirect;
//Only required for Twitter and Flickr
//OAuthWebFormsPanel1.StatelessRedirection = strue;
googledrive = CloudCacheProvider1.Factory<GoogleDrive>("GoogleDrive");
googledrive.Authenticator.OAuthPanel = OAuthWebFormsPanel1;
googledrive.Authenticator.OAuthApplication.Key = "xxxxxxxxxxxx";
googledrive.Authenticator.OAuthApplication.Secret = "yyyyyyyyyyyy";
googledrive.Authenticator.CallBackUrl = CallBackURL;
googledrive.AfterOpen += AfterOpen;
googledrive.Open();

private void AfterOpen(object sender, ref bool handled)
{
	//Add code to be executed after authentication here
}
```

Tips and FAQ

Scopes

Many REST APIs implement Scopes as part of the authentication. The scopes specify what parts of the API the client needs access to and thus needs to authenticate for. In the TMS cloud storage access components these scopes are automatically preset to perform full read/write access to the files.

For Microsoft SkyDrive, these scopes are:

wl.signin: consent to login on Live services
wl.basic: basic authentication
wl.offline_access: request a refresh token
wl.skydrive: read access to user SkyDrive files
wl.skydrive_update: write access to user SkyDrive files

For Google Drive the scopes are:

https://www.googleapis.com/auth/drive: read access
https://www.googleapis.com/auth/drive.file: full read/write access

If you want to limit access as read-only for example for the Windows SkyDrive or Google Drive, remove the specifier wl.skydrive_update from the SkyDrive scopes or remove https://www.googleapis.com/auth/drive.file from the GoogleDrive scopes.