TMS VCL Cloud Pack
DEVELOPERS GUIDE

June 2020

Copyright © 2012-2020 by tmssoftware.com bvba
Web: https://www.tmssoftware.com
Email: info@tmssoftware.com

Index

Availability

TMS VCL Cloud Pack is a set of VCL components for Win32 & Win64 application development and is available for Embarcadero Delphi & C++Builder XE, XE2, XE3, XE4, XE5, XE6, XE7, XE8, 10 Seattle, 10.1 Berlin, 10.2 Tokyo, 10.3 Rio and 10.4 Sydney.

Online references

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

TMS VCL Cloud Pack page:
https://www.tmssoftware.com/site/cloudpack.asp

Purchase a license

The TMS VCL Cloud Pack is available separately and also as part of the TMS VCL Subscription:

- TMS VCL Cloud Pack: https://www.tmssoftware.com/site/cloudpack.asp
- TMS VCL Subscription: https://www.tmssoftware.com/site/vdsub.asp

There is also a version of TMS FMX Cloud Pack, IntraWeb and Visual Studio .NET & ASP.NET:

- TMS FMX Cloud Pack: https://www.tmssoftware.com/site/tmsfmxcloudpack.asp
- TMS IntraWeb Cloud Pack: https://www.tmssoftware.com/site/tmsiwcloud.asp
- TMS VCL Cloud Pack for .NET: https://www.tmssoftware.com/site/tmscloudnet.asp

Terms of use

With the purchase of TMS VCL Cloud Pack, you are entitled to our consulting and support services to integrate the Amazon Cloud Drive, Apple iCloud, Google GDrive, Microsoft SkyDrive, DropBox, Box, Flickr, Google Calendar, Google Contacts, Google Picasa, Google Photos, Google Mail, Google Sheets, Google Analytics, Microsoft Live Calendar, Microsoft Live Contacts, Wunderground weather, Google Location Lookup, Google Search Lookup, Google DataStore, Facebook, Twitter, LinkedIn, PushOver, Instagram, FourSquare, Twilio, Esendex, BulkSMS, FreeGEOIp, Esendex, YouTube, Pryv, CloudConvert, Barcodes4me, PayPal, Hubic, HiDrive, myCloudData, Exceptionless, Outlook, Microsoft Cognitive, Imgur, Cloudinary, Yandex service in Delphi applications and with this consulting and support comes the full source code needed to do this integration. As TMS VCL Cloud Pack uses the Amazon Cloud Drive, Apple iCloud, Google GDrive, Microsoft OneDrive, DropBox, Box, Flickr, Google Calendar, Google Contacts, Google Picasa, Google Photos, Google DataStore, Microsoft Live Calendar, Microsoft Live Contacts, Wunderground weather, Google Location Lookup, Google Search Lookup, Facebook, Twitter, LinkedIn, PushOver, Instagram, FourSquare, Twilio, Esendex, BulkSMS, FreeGEOIp, Esendex, YouTube, Pryv, CloudConvert, PayPal, Hubic, HiDrive, Trello, myCloudData, Exceptionless, Outlook Calendar, Outlook Contacts, Outlook Mail, Microsoft Cognitive, Imgur, Cloudinary service, Yandex 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
http://www.wunderground.com/members/tos.asp
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
http://instagram.com/legal/terms/
https://foursquare.com/legal/terms
https://www.twilio.com/legal/tos
http://www.telapi.com/legal/terms-of-service
http://www.bulksms.com/int/w/terms_and_conditions.htm
http://www.esendex.co.uk/reference/terms-and-conditions
https://www.youtube.com/static?template=terms
http://pryv.com/terms-of-use/
http://www.amazon.com/gp/help/customer/display.html/?nodeId=201376540&ref_=cd_tou_fp
https://cloudconvert.com/terms
https://www.paypal.com/webapps/mpp/ua/legalhub-full?country.x=US&locale.x=en_US
https://hubic.com/en/contracts/Contrat_hubiC_2014.pdf
https://dev.strato.com/hidrive/terms-of-service
https://trello.com/legal
http://www.myclouddata.net/#/tos

https://exceptionless.com/terms/
https://msdn.microsoft.com/en-US/cc300389
http://research.microsoft.com/en-us/um/legal/CognitiveServicesTerms20160804.htm
https://imgur.com/tos
http://cloudinary.com/tos
https://yandex.com/legal/disk_termsofuse/

TMS VCL Cloud Pack includes components for accessing WebDAV, CalDAV, CardDAV servers as well as 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 VCL Cloud Pack components. The purchase of TMS VCL Cloud Pack does not include any license fee that you might possibly be required to pay to Amazon, Apple, Google, Microsoft, DropBox, Box, Flickr, Wunderground, Facebook, Twitter, LinkedIn, PushOver, Instagram, FourSquare, Twilio, Esendex, BulkSMS, TelAPI, YouTube, Pryv, CloudConvert, Barcodes4me, PayPal, Hubic, HiDrive, Trello, myCloudData, Exceptionless, Outlook Calendar, Outlook Contacts, Outlook Mail, Microsoft Cognitive, Imgur, Cloudinary, Yandex. 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 the Amazon, Apple, Google, Microsoft, DropBox, Box, Flickr, Facebook, Twitter, Wunderground, LinkedIn, PushOver, Instagram, FourSquare, Twilio, Esendex, BulkSMS, TelAPI, YouTube, Pryv, CloudConvert, Barcodes4me, PayPal, Hubic service to respect the Google, Microsoft, DropBox, Facebook, Twitter, Wunderground, LinkedIn, PushOver, Instagram, Twilio, Esendex, BulkSMS, TelApi, YouTube, Pryv, CloudConvert, myCloudData, Exceptionless, Outlook Calendar, Outlook Contacts, Outlook Mail, Microsoft Cognitive, Imgur, Cloudinary, Yandex terms and conditions. TMS software does not take any responsibility nor indemnifies any party violating the Google, Microsoft, DropBox, Box, Flickr, Facebook, Twitter, Wunderground, LinkedIn, PushOver, Instagram, FourSquare, Twilio, Esendex, BulkSMS, TelAPI, YouTube, Pryv, CloudConvert, Barcodes4me, PayPal, Hubic, HiDrive, Trello, myCloudData, Exceptionless, Outlook Calendar, Outlook Contacts, Outlook Mail, Microsoft Cognitive, Imgur, Cloudinary, Yandex service terms & conditions.

We cannot guarantee that a 3rd party cloud service will approve or allow the use of the services used in TMS VCL 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 the Amazon, Google Drive, Microsoft OneDrive, Microsoft Live Calendar, Microsoft Live Contacts, DropBox, Box, Flickr, Facebook, Twitter, Wunderground, Google Contacts, Google Calendar, Google Picasa, Google Photos, LinkedIn, PushOver, Instagram, Twilio, Esendex, BulkSMS, TelAPI, FreeGEOIP, YouTube, Pryv, CloudConvert, Barcodes4me, PayPal, Hubic, HiDrive, Trello, myCloudData, Exceptionless, Outlook Calendar, Outlook Contacts, Outlook Mail, Microsoft Cognitive, Imgur, Cloudinary, Yandex services.
TMS software offers the consulting and support for TMS VCL Cloud Pack in good faith that the Amazon, Apple, Google Drive, Google DataStore, Microsoft OneDrive, Microsoft Live Calendar, Microsoft Live Contacts, DropBox, Box, Flickr, Facebook, Twitter, Wunderground, Google Contacts, Google Calendar, Google Picasa, Google Photos, Google Mail, Google Sheets, Google Analytics, LinkedIn, PushOver, Instagram, Twilio, Esendex, BulkSMS, TelAPI, FreeGEOIP, YouTube, Pryv, CloudConvert, Barcodes4me, PayPal, Hubic, HiDrive, Trello, myCloudData, Exceptionless, Outlook Calendar, Outlook Contacts, Outlook Mail, Microsoft Cognitive, Imgur, Cloudinary, Yandex service is a reliable and future-proof service.
In no case, TMS software shall offer refunds or any other compensation in case the Amazon, Apple, Google Drive, Microsoft OneDrive, Microsoft Live Calendar, Microsoft Live Contacts, DropBox, Box, Flickr, Facebook, Twitter, Wunderground, Google Contacts, Google Calendar, Google Picasa, Google Photos, LinkedIn, PushOver, Instagram, Twilio, Esendex, BulkSMS, TelAPI, FreeGEOIP, YouTube, Pryv, CloudConvert, Barcodes4me, PayPal, Hubic, HiDrive, Trello, myCloudData, Exceptionless, Outlook Calendar, Outlook Contacts, Outlook Mail, Microsoft Cognitive, Imgur, Cloudinary, Yandex service terms/operation changes or stops.

Main features

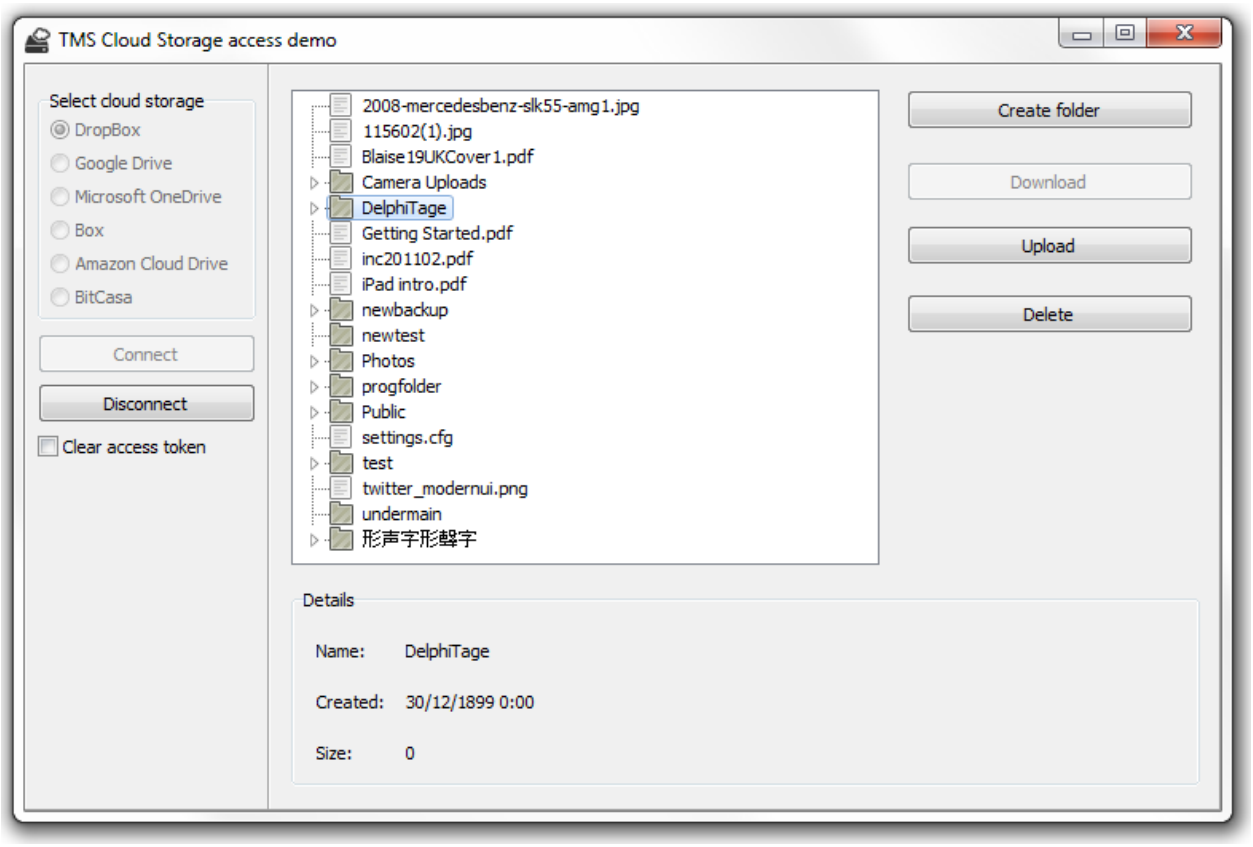
- Set of VCL components to offer easy access from Windows applications to cloud services
- Component to get access to Amazon cloud drive
- Component to get access to Apple CloudKit
- Component to get access to DropBox storage
- Component to get access to Box storage
- Component to get access to HiDrive storage
- Component to get access to Hubic storage
- Component to get access to CloudConvert API
- 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 Google Photos API
- Component to get access to Google Places API
- Component to get access to Google Tasks API
- Component to get access to Google DataStore API
- Component to get access to Google Mail API
- Component to get access to Google Sheets API
- Component to get access to Google Analytics API
- Component to get access to Google Firebase Database 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 Wunderground weather forecast service
- Component to get access to LinkedIn API
- Component to get access to Instagram API
- Component to get access to FourSquare API
- Component to get access to YouTube API
- Component to get access to DropBox DataStore API
- Component to get access to Pryv API
- Component to get access to Barcodes4me API
- Component to get access to PayPal API
- Component to get access to Trello API
- Component to get access to MyCloudData API
- Component to get access to Exceptionless API
- Component to get access to Outlook Calendar API
- Component to get access to Outlook Contacts API
- Component to get access to Outlook Mail API
- Component to get access to Microsoft Computer Vision API
- Component to get access to Microsoft Emotion API
- Component to get access to Microsoft Bing Speech API
- Component to get access to Imgur API
- Component to get access to Cloudinary API
- Component to get access to Yandex Disk API
- Component to display images (JPG, PNG, GIF, BMP) from an URL

- Component to shorten URLs based on Google URL shortener service
- Component to send push messages to iOS devices running the PushOver client
- Components to access WebDAV, CalDAV, CardDAV servers
- Components to access iCloud Contacts and iCloud Calendar
- Component to send SMS messages via the Twilio service
- Component to send SMS messages via the Esendex service
- Component to send SMS messages via the TelAPI service
- Component to send SMS messages via the BulkSMS service
- Component to determine machine location via the FreeGEOIP service
- WebDAV, CalDAV, CardDAV local storage, synchronization & filtering components
- Edit control with lookup capabilities based on web services lookup data
- 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 TAdvSkyDrive/TAdvOneDrive, TAdvDropBox, TAmazonCloudDrive, TAdvBoxNetDrive, TAdvGDrive, TAdvHubic, TAdvHiDrive, TAdvYandexDisk components to access your cloud storage. All storage components work in a similar way:

1) Drop the component on the form.

2) Setup the client ID, client secret via the .App.Key and .App.Secret property.

3) Call the .DoAuth method.

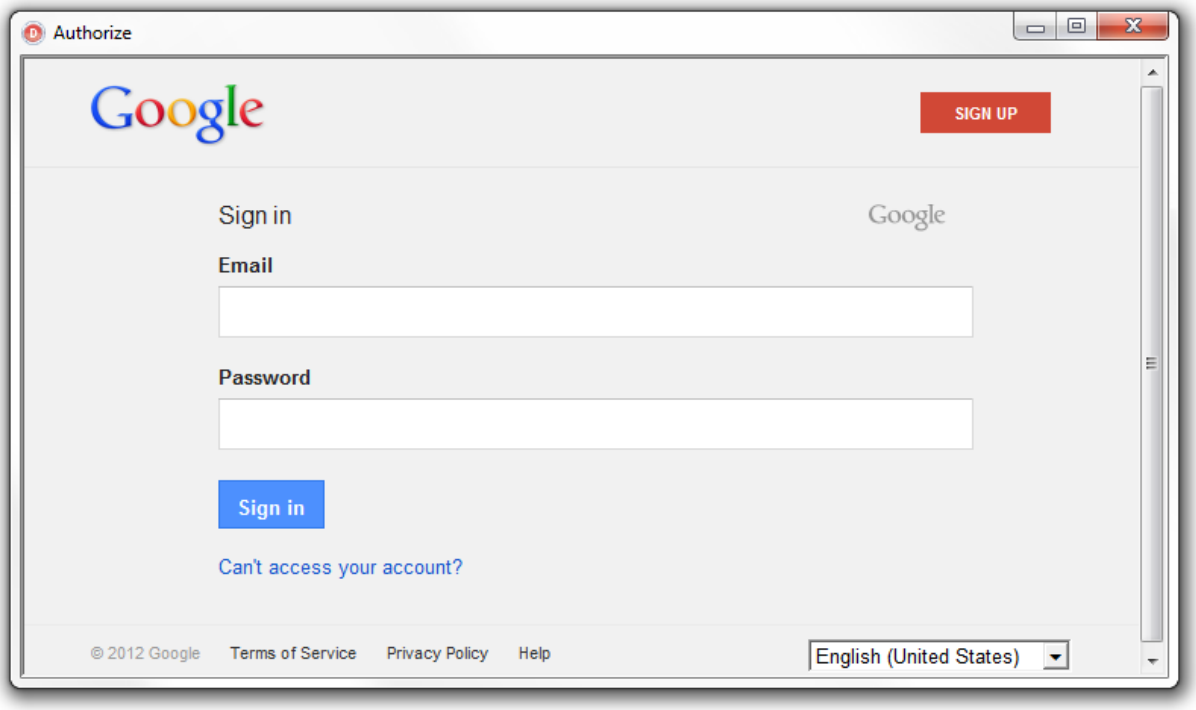
Code:

```
AdvGDrive1.App.Key := 'xxxxxxxxx.apps.googleusercontent.com';

AdvGDrive1.App.Secret := 'yyyyyyyyyyyyyyyy';

AdvGDrive1.DoAuth;
```

And this will show the Google login screen:



The size of the login screen and the caption are controlled by the properties:

```
AuthFormSettings.Caption: string;
AuthFormSettings.Height: integer;
AuthFormSettings.Width: integer;
```

When the login screen has been closed without a successful authentication, the event OnAuthFormClose will be triggered.
Alternatively, it is also possible to use any other TWebBrowser instance as login screen. To do this, just assign this TWebBrowser instance to TAdvXXXDrive.AuthBrowser: TWebBrowser.

When the user has provided the correct credentials, the event OnReceivedAccessToken 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 TCloudItems. The common structure of this collection is:

TCloudItems = class(TCollection)

in this collection are items of the type:

TCloudItem = class(TCollectionItem)

Properties:

property FileName: string;
Holds the filename of a file or folder

property Folder: TCloudItems;
When the ItemType is itFolder, this contains in turn a collection of files and folders

property Size: int64;
Holds the size of the item

property ItemType: TCloudItemType;
Defines whether the item is a file or folder (ciFile, ciFolder)

property CreationDate: TDateTime;
Holds the creation date of the file or folder

property ModifiedDate: TDateTime;
Holds the timestamp when the file was last modified

property Tag: integer;
Integer property that can be freely used.

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

Methods:

GetDriveInfo;
Use the cloud storage API to query the list of all files and folders, then store this hierarchically in the Drive: TCloudItems collection property.

FillTreeView(TreeView: TTreeView);
A helper method to immediately visualize the file structure retrieved by GetDriveInfo in a Treeview control.

GetFolderList(Folder: TCloudItem): TCloudItems;
Get the list of all files and folders of the Folder without recursion into subfolders. Returns the collection of TCloudItems. If no Folder is specified the root folder is retrieved. For TAdvGDrive, TAdvAmazonCloudDrive, TAdvOneDrive, TAdvBoxNetDrive only the Folder.ID property is required. For TAdvDropBox, TAdvHiDrive only the Folder.Path property is required. (This method is not supported for TAdvHubic due to API limitations)

GetFolderListHierarchical(Folder: TCloudItem; IsRootFolder: boolean): TCloudItems;
See GetFolderList for usage information. This method will add the retrieved list of files and folders to the Drive collection hierarchically. It is required that the specified Folder is already part of the

Drive collection unless the optional parameter IsRootFolder is set to True. If IsRootFolder is True, the Folder will appear as the root folder in the Drive collection. (This method is not supported for TAdvHubic due to API limitations)

A helper component TCloudTreeviewAdapter is available to automatically and progressively visualize the file structure in a Treeview control.

SearchList(Query: string; ExactMatch: Boolean; Folder: TCloudItem): TCloudItems;
Only supported in TAdvDropBox, TAdvGDrive, TAdvOneDrive and TAdvBoxNetDrive.
Performs a search for the Query value in the File and Folder names available on the cloud storage. A list of TCloudItems with the results of the search action is returned. ExactMatch is only supported in TAdvGDrive. Specify a Folder to limit the search to a specific folder (Only supported in TAdvDropBox, TAdvGDrive).

File operations

Following operations are available:

Create a folder

Delete a file or folder

Download a file

Upload a file

Rename a file

Creating a folder

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

TAdvXXXDrive.CreateFolder(nil, 'New folder'): TCloudItem;

or create a subfolder in a specific folder. To do this, you need the instance of the TCloudItem representing the folder and use it as first parameter of the CreateFolder() call.

TAdvXXXDrive.CreateFolder(ParentFolderItem, 'New folder'): TCloudItem;

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

Delete a file or folder

Deleting a file can be done by calling the function TAdvXXXDrive.Delete(CloudItem): Boolean. This function returns true on a successful delete. The parameter is the TCloudItem instance that represents the file or folder to be deleted

Download a file

Downloading a file is equally simple. Call the function TAdvXXXDrive.Download(CloudItem, TargetFileName): Boolean. For a successful download, this function returns true. Note that the progress of the download can be tracked via the event OnDownloadProgress.

Upload a file

Uploading a file means creating the file on the cloud storage and uploading its data. The function that is used for upload is: TAdvXXXDrive.Upload(Folder: TCloudItem; FileName:string): TCloudItem.

The file is uploaded in the root (when Folder parameter is nil) or in the folder as specified by the Folder TCloudItem. The local file that will be uploaded is set via the FileName parameter. When successful, this function returns an instance of the new created file. Note that this item will also automatically be added in the TAdvXXXDrive.Drive collection. The progress during the upload can also be tracked via the OnUploadProgress event.

TAdvDropBox only:
property UploadMode:
When set to umAdd, if a file is uploaded with a filename that already exists. The new file is automatically renamed and uploaded.
When set to umOverWrite, if a file is uploaded with a filename that already exists. The original file is automatically replaced with the new file.

Rename a file

Renaming files is currently only available in TAdvGDrive.
Deleting a file can be done by calling the function TAdvGDrive.Rename(CloudItem, FileName): Boolean. This function returns true on a successful rename. The parameters are the TCloudItem instance that represents the file or folder to be renamed and the new FileName.

Public shared files

TAdvDropBox, TAdvSkyDrive/TAdvOneDrive, TAdvBoxNetDrive, TAdvGDrive, TAdvHubic, TAdvHiDrive and TAdvYandexDisk 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 either:

property TCloudItem.Share: string

Property returning a public accessible HTTP URL to download the file.

or

function TCloudStorage.GetShare(CloudItem): string

With the property

property TCloudItem.Shared: Boolean

it can be retrieved whether the file is shared or not.

Notes:

- **TADVGDrive:**
  For TAdvGDrive, the permissions are organized in a slightly different way:

  TGDriveItem.PublicShare := true;
  Set this item as public accessible.

  TGDriveItem.PublicShare := false;
  Remove the public accessible permission for this item.

  The property **TGDriveItem.Shared: boolean** will return true when the file can be shared with other users.
  The property **TGDriveItem.DownloadURL: string** can return the HTTP URL for such shared file.
  The property **TGDriveItem.AlternateURL: string** can return a link for opening the file using a relevant Google editor or viewer.
  The property **TGDriveItem.WebContentURL: string** can return a link for downloading the content of the file. In cases where the content is shared publicly, the content can be downloaded without credentials.

- **TAdvHiDrive:**
  For TAdvHiDrive, shared URLs can expire after the shared file has been downloaded a specific number of times or after a specific amount of time. The number of times a shared file can be downloaded or the amount of time before the shared link expires depends on the HiDrive

account type of the authenticated user.
To re-enable an expired share, first delete the existing share and then create a new share for the same THiDriveItem.

function THiDriveItem.DeleteShare: Boolean

or

function TAdvHiDrive.DeleteShare(ShareURL: string): Boolean

Function to delete an existing share. The share will be deleted if it is expired or not.

Account Info

Retrieve specific information about the currently authenticated account including drive space quota. (Only available in TAdvDropBox, TAdvGDrive and TAdvOneDrive)

function GetAccountInfo;
Function to f fill the Info properties:

TAdvDropBox:
- UserName
- Country
- Quota (allocated space)
- QuotaUsed (used space)

TAdvGDrive:
- UserName
- Quota (allocated space)
- QuotaUsed (used space)

TAdvOneDrive:
- UserName
- Quota (allocated space)
- QuotaAvailable (available space)
- QuotaUsed (used space)

CloudStorage specific settings

TAdvDropBox

TAdvDropBox has a few extra settings to take control of specific behavior of the DropBox service.

The public property ExternalBrowser: boolean is added. When setting this to true, the authentication/authorization step can be done via the default browser.

Secondly, a property TAdvDropBox.FileLimit: integer is added. This controls how many files the DropBox API can return for a folder file listing. The default value is 10000. If you have more than 10000 files in a DropBox folder, this can be increased to a maximum value of 25000. The maximum value of 25000 is a limitation of the DropBox API itself.

Note:
The TAdvDropBox component now supports the DropBox API v2. In v2 the FileLimit setting is no longer used, therefore this property is also no longer used. The DropBox API v2 does not specifiy a file limit number.

TAdvHubic:
The TAdvHubic component is derived from the TAdvCustomOpenStack class which partially implements the OpenStack Object Storage API v1.

Information about the OpenStack Object Storage API can be found here:
http://developer.openstack.org/api-ref-objectstorage-v1.html

TCloudTreeViewAdapter / TCloudAdvTreeViewAdapter

Usage

A helper control to automatically and progressively visualize the file structure from CloudStorage in a TreeView control.
After calling CloudStorage.Connect, the TreeView will automatically be filled based on the selected InitMethod.
After calling CloudStorage.Disconnect, the TreeView will automatically be cleared.

TCloudTreeViewAdapter
Can be used in combination with a TTreeView component, included in RAD Studio by default.

TCloudAdvTreeViewAdapter
Can be used in combination with a TAdvTreeView component, included in the TMS Component Pack

(available separately).
Please note that the TCloudAdvTreeViewAdapter is only included with the registered version of the TMS VCL Cloud Pack, it is not available in the trial version. The component needs to be installed separately because it depends on the TAdvTreeView component from the TMS Component Pack.

Organisation

Properties:

TreeView: TTreeView; (TCloudTreeViewAdapter only)
TreeView: TAdvTreeView; (TCloudAdvTreeViewAdapter only)
The Treeview control used to visuzalize the file structure.

CloudStorage: TCloudStorage;
The cloud storage component (TAdvSkyDrive/TAdvOneDrive, TAdvDropBox, TAmazonCloudDrive, TAdvBoxNetDrive, TAdvGDrive, TAdvHiDrive, TAdvHubic) used to retrieve the file structure.

InitMethod: TCloudAdapterInitMethod;
Defines how the TreeView is initialized. If set to cmRoot the InitRoot method is used, if set to cmDrive the InitDrive method is used and if set to cmFolder the InitFolder method is used. (TAdvHubic only supports cmDrive due to API limiations)

Information about the different methods is provided below. When set to cmFolder, the folder to use can be assigned to the Folder property.

Folder: TCloudItem;
Defines which folder to use when the InitMethod is set to cmFolder.

Options: TCloudAdapterOptions; (TCloudAdvTreeViewAdapter only)
Select which columns to display in the TAdvTreeView. Available columns are: Name (File or Folder name), CreationDate, ModifiedDate, Size. By default all columns are displayed.

Methods:

InitRoot;
Fills the Treeview with all files and folders (without recursion) of the root folder retrieved through the assigned CloudStorage component. When the user selects a folder in the Treeview the underlying files and folders (without recursion) are automatically retrieved and visualized hierarchically in the Treeview. (Not supported for TAdvHubic due to API limiations)

InitDrive;
Fills the TreeView hierarchically with all files and folders retrieved through the assigned CloudStorage component.

InitFolder(Folder: TCloudItem);
Fills the Treeview with all files and folders (without recursion) contained in the specified folder,

retrieved through the assigned CloudStorage component. When the user selects a folder in the Treeview the underlying files and folders are automatically retrieved and visualized hierarchically in the Treeview. (Not supported for TAdvHubic due to API limiations)

InitTreeView;
Fills the Treeview hierarchically with all files and folders found in the Drive collection of the CloudStorage. There is no data retrieved from the cloud storage, only data already present in the Drive collection is displayed.

ClearTreeView;
Removes all items from the TreeView.

TAdvTwitter

Usage

TAdvTwitter 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 Tweets, followers & friends from a Twitter account. It's also possible to send and retrieve direct messages.

Organisation

Properties:

Profile: TTwitterProfile; Class property that contains the Twitter profile information for the currently authenticated user.

ScreenName: string; The screen name, handle, or alias that this user identifies themselves with.
ID: integer; The unique identifier for this user.
CreatedAt: TDateTime; The UTC datetime that the user account was created on Twitter.
ListedCount: integer; The number of public lists that this user is a member of.
Name: The name of the user, as they've defined it.
StatusCount: integer; The number of tweets this user has set as favorite in the account's lifetime.
FollowersCount: integer; The number of followers this account currently has.
FriendsCount: integer; The number of users this account is following.
GeoEnabled: Boolean; When true, indicates that the user has enabled the possibility of geotagging their Tweets.
ImageUrl: 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: The user's most recent tweet or retweet.
StatusCount: 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.

Followers: TProfileList; Contains the list of Twitter followers.
Friends: TProfileList; Contains a list of Twitter friends.
Statuses: TStatusList; Contains a list of Twitter Status messages.

User: TTwitterProfile; 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: uint64; The unique identifier for this Tweet.
InReplyToID: integer; 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: integer; If the represented Tweet is a reply, this field will contain the original Tweet's ID.
MediaURL: string; Link to the image that is connected to this Tweet (if any).
MediaURLs: TStringList; List of images that are connected to this Tweet (if any). Can be used when the tweet contains multiple images.
Source: string; Utility used to post the Tweet, as an HTML-formatted string.

Mentions: TStatusList; Contains a list of Twitter mentions.
UserTimeLine: TStatusList; Contains a list of Twitter messages that are on the user's timeline.

DirectMessages: TDirectMessageList; Contains a list of Direct Messages retrieved by the authenticated account. (See GetDirectMessages)

Sender: TTwitterProfile; The Twitter Profile of the user that sent the message.
Text: string; The text of the message.
CreatedAt: TDateTime; Time when the message was received.
ID: string; The unique identiefier for this message.

Configuration: TTwitterConfiguration; Contains various configuration settings for the Twitter API. (See GetConfiguration)

MediaLength: integer; The number of reserved characters in a Tweet when an image is included.
DMMaxLength: integer: The maximum number of characters allowed in a direct message text.
ShortURLLengthHTTP: integer; The number of reserved characters in a Tweet for each HTTP URL included.
ShortURLLengthHTTPS: integer; The number of reserved characters in a Tweet for each HTTPS URL included.

Methods:

Tweet(const Msg: string): string;
Create a new Tweet on the user's timeline.

Example:

```
AdvTwitter1.Tweet('Hello World!');
```

TweetWithMedia(const Msg, FileName: string): string; (deprecated)
Create a new Tweet with included image on the user's timeline.

TweetWithMedia(const Msg, MediaIDs: TStringList): string;
Create a new Tweet with included one or more images (with a maximum of 4 or 1 animated GIF) on the user's timeline. Note that images must first be uploaded with the "UploadMedia" call. The result of the call contains the ID that should be added to the MediaIDs list.

Retweet(const ID: Int64): string;
Retweet an existing Tweet on the user's timeline.

DeleteTweet(const ID: Int64): string;
Delete an existing Tweet on the user's timeline.

DirectMessage(const Msg: string; ScreenName: string): string; (deprecated)
Send a direct message to another Twitter user.

DirectMessage(const Msg: string; ID: Int64): string;
Send a direct message to another Twitter user based on the provided ID.

DirectMessageWithMedia(const Msg: string; ID: Int64; MediaID: string): string;
Send a direct message with image included to another Twitter user based on the provided ID. Note that the image must first be uploaded with the "UploadMedia" call. The result of the call contains the ID that should be assigned to the MediaID parameter.

GetConfiguration: Boolean;
Retrieve the values for the Configuration properties.

GetDirectMessages: Boolean;
Fill the list of DirectMessages (both sent and received) within the last 30 days.

GetFollowers: integer;
Fill the list of Followers.

GetFriends(UserID: string = ''): integer;
Fill the list of Friends.

GetMentions(Count: integer = 100; SinceID: Int64 = -1; MaxID: Int64 = -1): integer;
Fill the list of Mentions.

GetStatuses(Count: integer = 100; SinceID: Int64 = -1; MaxID: Int64 = -1; UserID: integer = -1; UserName: string = ''): integer;
Fill the list of Statuses. If a UserID or UserName is provided a list of Statuses from the specific user is retrieved.

GetTweetLength(Msg: string; IncludeImage: Boolean = false): integer;
Returns the character count of a Tweet based on the Msg parameter text. The length is calculatedby using the values of the Configuration properties. Set IncludeImage to true if an image is to be included with the Tweet. The maximum number of allowed characters is 140.

Search(const Query: string; Count: integer = 100; SinceID: integer = -1): TStatusList;
Return a list of Status messages that contain the Query string value.

GetAccountInfo: boolean;
Fill the user's Profile.

GetProfileInfo(const ID: integer; Profile: TTwitterProfile): boolean; overload;
GetProfileInfo(const ID: string; Profile: TTwitterProfile): boolean; overload;

GetProfileInfo(Profile: TTwitterProfile): boolean; overload;
Fill a TTwitterProfile based on the profile ID.

GetProfileListInfo(ProfileList: TProfileList): boolean;
Fill a list TTwitterProfiles based on the profile IDs.

Example:

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

MuteUser(ScreenName: string): string;
MuteUser(ID: Int64): string;
Mute a Twitter user based on the account's ScreenName or ID.

BlockUser(ScreenName: string): string;
BlockUser(ID: Int64): string;
Block a Twitter user based on the account's ScreenName or ID.

UnMuteUser(ScreenName: string): string;
UnMuteUser(ID: Int64): string;
Remove the mute setting on a Twitter user based on the account's ScreenName or ID.

UnBlockUser(ScreenName: string): string;
UnBlockUser(ID: Int64): string;
Unblock a Twitter user based on the account's ScreenName or ID.

UploadMedia(FileName: string): string;
Uploads an image to Twitter and returns a MediaID. The MediaID can be used to include an image with a tweet (See "TweetWithMedia") or a direct message (See "DirectMessageWithMedia").

TAdvFacebook

Note: Unfortunately, due to changes and restrictions in the Facebook API beyond our control, posting to user's feed, Facebook Page, Facebook Group or uploading image files is no longer functional.

Usage

TAdvFacebook 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

Properties:

APIVersion: TFacebookAPIVersion; Indicates which version of the Facebook API should be used (av10 for v1.0, av21 for v2.1, or av21 for v2.4).

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

ID: string; The user's Facebook ID.
FullName: string; The user's full name.
FirstName: string; The user's first name.
LastName: string; The user's last name.
MiddleName: string; The user's middle name.
Gender: TFacebookGender; 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: integer; The user's timezone.
Locale: string; The user's locale.
Verified: Boolean; The user's account verification status.
UpdatedTime: TDateTime; The last time the user's profile was updated.
ImageURL: string; The URL of the user's profile pic.
HomeTown: TFacebookObject; The user's hometown.
Location: TFacebookObject; The user's current city.
RelationShip: string; The user's relationship status.
SignificantOther: TFacebookProfile; The user's significant other.
Website: string; The URL of the user's personal website.
Likes: TFacebookObjectList; List of the Facebook pages the user has liked.

Feed: TFeed; List of the user/page/group's feed items.

ID: string; The post ID.
User: TFacebookProfile; Information about the user who posted the message.
Text: string; The actual message text.
ImageURL: string; If available, a link to the picture included with this post.
Link: string; The link attached to this post.
Summary: string; The name of the link.
Caption: string; The caption of the link.
Description: string; A description of the link.
CreatedTime: TDateTime; 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: TFacebookProflieList; List of users that liked this post.
ToUsers: TFacebookProfileList; List of profiles mentioned or targeted in this post.
Comments: TFacebookCommentList; List of comments add to this post.

ID: string; The ID of the comment.
User: TFacebookProfile; The user that created the comment.
Text: string; The comment text.
CreatedTime: TDateTime; The time the comment was created.
Likes: TFacebookProfileList; List of users that liked this comment.
UserLikes: boolean; Indicates if the currently authenticated user has liked the comment.

Albums: TFacebookAlbumList; List of the user's album items.

ID: string; The ID of the album.
Title: string; The title of the album.
From: TFacebookProfile; The profile that created this album.
Link: string; A link to this album on Facebook.
Description: string; The description of the album.
CoverPhotoID: string; The album cover photo ID.
CreatedTime: TDateTime; The time the photo album was initially created.
UpdatedTime: TDateTime; The last time the photo album was updated.

Pictures: TFacebookPictureList; List of pictures that are in this album.

ID: string; The ID of the picture.
Summary: string; The user provided caption given to this picture.
From: TFacebookProfile; The profile (user or page) that posted this picture.
ImageURL: string; A direct URL link to the picture on Facebook.
Link: string; A 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.

FriendList: TFacebookProfileList; Contains a list friend profiles.

FriendsCount: Integer; The total count of friends

PageList: TFacebookPageList; Contains a list of Facebook pages the currently authenticated user has administrative rights to.

ID: string; The page Facebook ID.
AccessToken: string; The access token that must be used to post to this page.
Summary: string; The page name.
Category: string; The page category.
Permissions: TStringList; The list of permissions that are available for the currently authenticated user.
Likes: integer; The number of likes the Page has received. (See GetLikes)
Feed: TFeed; The list of feed items for this group (See GetFeed)

GroupList: TFacebookGroupList; Contains a list of Facebook groups the currently authenticated user has administrative rights to.

ID: string; The group Facebook ID.
Summary: string; The group name.
Privacy: string; The group's privacy setting (Open, Closed, Secret).
Feed: TFeed; The list of feed items for this group (See GetFeed)

Methods:

GetUserInfo;
Fill the user's "Profile".

GetFriends;
Fill the "FriendList" with the user's Facebook Friends.

GetProfileInfo(const ID: string; Profile: TFacebookProfile): boolean;
Fill a TFacebookProfile based on the profile ID.

GetLikes(Profile: TFacebookProfile): boolean; overload;
Fill a TFacebookProfile.Likes list with the Facebook Pages that the user has "liked".

GetLikes(FeedItem: TFacebookFeedItem): boolean; overload;
Fill a TFacebookFeedItem.Likes list with TFacebookProfiles that have "liked" the TFacebookFeedItem.

GetLikes(Comment: TFacebookComment): boolean; overload;
Fill a TFacebookComment.Likes list with TFacebookProfiles that have "liked" the TFacebookComment.

GetLikes(Page: TFacebookPage): boolean; overload;
Set a TFacebookPage.Likes property with the total number of likes the Page has received. (requires the "read_insights" scope)

GetAlbums(Profile: TFacebookProfile): boolean;
Fill a TFacebookProfile.Albums list with the Facebook Photo Albums connected to the TFacebookProfile.

GetAlbums(ID: string): boolean;
Fill a TFacebookProfile.Albums list with the Facebook Photo Albums connected to the provided ID. The ID can be of a Facebook Profile or a Facebook Page.

GetPages(): boolean;
Fill “PageList” with the Facebook Pages the user has administrative access to.

GetGroups(): boolean;
Fill “GroupList” with the Facebook Groups the user has administrative access to.

GetPictures(Album: TFacebookAlbum): boolean;
Fill a TFacebookAlbum.Pictures list with the Pictures connected to the TFacebookAlbum.

GetFeed(Profile: TFacebookProfile; Count: integer = 100; Offset: integer = 0;Since: TDateTime = 0): integer;
Fill a TFacebookProfile.Feed with a list of Facebook Feed messages for the TFacebookProfile.

GetFeed(Page: TFacebookPage; Count: integer = 100; Offset: integer = 0;Since: TDateTime = 0): integer;
Fill a TFacebookPage.Feed with a list of Facebook Feed messages for the TFacebookPage.

GetFeed(Group: TFacebookGroup; Count: integer = 100; Offset: integer = 0;Since: TDateTime = 0): integer;
Fill a TFacebookProfile.Group with a list of Facebook Feed messages for the TFacebookGroup.

GetComments(FeedItem: TFacebookFeedItem): boolean;
Fill a TFacebookFeedItem.Comments with a list of comments for the TFacebookFeedItem.

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

Post(const Msg, Link, Image: string): string; (deprecated)
Post a message on the user’s Facebook Feed with an optional URL link and image URL.

Post(const Msg, Link, Image: string; Page: TFacebookPage): string; (deprecated)
Post a message on the user’s Facebook Page with an optional URL link and image URL.

PostComment(ID: string;const Msg: string): string; (deprecated)
Post a comment on one of the user’s Facebook feed items based on a TFacebookFeedItem ID.

PostImage(const Msg, FileName: string): string; overload; (deprecated)
Post an image to the Facebook Album that is connected to your application registered with Facebook. (If the Facebook Album does not exist, it will automatically be created.)

PostImage(const Msg, FileName: string; Album: TFacebookAlbum): string; overload; (deprecated)
Post an image to the TFacebookAlbum.

PostImage(const Msg, FileName: string; Album: TFacebookAlbum; Page: TFacebookPage): string; overload; (deprecated)
Post an image to the TFacebookAlbum of a specific Facebook Page.

Like(ID: string): string;
Like a Facebook Feed message/comment based on a TFacebookFeedItem / TFacebookComment ID.

UnLike(ID: string): string;
Unlike a Facebook Feed message/comment based on a TFacebookFeedItem / TFacebookComment ID.

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);
```

TAdvFlickr

Usage

TAdvFlickr 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:

property UserID: String: The ID retrieved after login, used to access user related content.
property UserName: String The user name retrieved after login.
property Galleries: TFlickrGalleries: The collection of galleries, loaded after calling GetGalleries.
property PhotoStream: TFlickrPhotos: The collection of photos from the photostream, loaded after calling GetPhotoStream.
property Sets: TFlickrSets: The collection of sets, loaded after calling GetSets.

Gallery Properties:
property ID: String: The ID of the set, used to access the flickr api for galleries specifically.
property URL: String: The direct URL of the gallery, to display in a browser window.
property Owner: String: The User ID of the owner of this gallery.
property DateCreate: String: The date the gallery is created.
property Title: String: The title of the gallery.
property Description: String: The description of the gallery.
property Photos: TFlickrPhotos: The collection of photos inside the gallery.

Set Properties:
property ID: String: The ID of the set, used to access the flickr api for sets specifically.
property Primary: String: The ID of the primary photo inside the set.
property Title: String: The title of the set.
property Description: String: The description of the set.
property Photos: TFlickrPhotos: The collection of photos inside the set.
property Comments: TFlickrComments: The comments of the set.

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

property Sizes: TFlickrSizes: The various sizes in which the photo exists and can be downloaded.
property Latitude: Double: The latitude of the geo location of the photo.
property Longitude: Double: The longitude of the geo location of the photo.
property Tags: TFlickrTags: The tags of the photo.
property Comments: TFlickrComments: The comments of the photo.

Tag Properties:

property ID: String: The ID of the tag.
property Author: String: The User ID of the author of the tag.
property Value: String: The value of the tag.

Comment Properties:

property ID: String: The ID of the comment.
property Author: String: The User ID of the author of the comment.
property AuthorName: String: The User Name of the author of the comment.
property Value: String: The value of the comment.
property DateCreate: String: The date the comment was created.

Size Properties:

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

Methods:

```
function AddFolderToSet(AFolder, ATitle: String; ADescription: String): TFlickrSet;
```

Adds a folder of images to a new or existing set with a specific title and description.

```
Sample:
AdvFlickr1.AddFolderToSet('C:\folder\*.jpg', 'new set', 'set description');
```

```
function AddPhotoToSet(AFileName, ATitle, ADescription: String):
TFlickrSet;
```

Adds an image to a new or existing set with a specific title and description.

```
Sample:
AdvFlickr1.AddPhotoToSet('C:\folder\image1.jpg', 'new set', 'set
description');
```

```
function CreateGallery(ATitle: String; ADescription: String):
TFlickrGallery;
```

Creates and returns a new gallery with a specific title and description.

```
Sample:
AdvFlickr1.CreateGallery('new gallery', 'gallery description');

function CreateSet(ATitle, ADescription, APrimaryPhotoID: String):
TFlickrSet;
```

Creates and returns a new set 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.

```
Sample:
var
  APhoto: TFlickrPhoto;
begin
  APhoto := AdvFlickr1.UploadPhotoToStream('C:\folder\image1.jpg', 'new
image', 'image description');
  if Assigned(APhoto) then
    AdvFlickr1.CreateSet('new set', 'set description', APhoto.ID);

function DownloadPhoto(ATargetFile, APhotoURL: String): Boolean; overload;
```

Downloads a photo to a target location. The APhotoURL is retrieved after calling GetSizes on a photo object and selecting the DownloadURL property depending on the size in the sizes collection.

```
function FindGalleryByID(AGalleryID: String): TFlickrGallery;
```

Returns a gallery by ID.

```
function FindGalleryByTitle(AGalleryTitle: String): TFlickrGallery;
```

Returns a gallery by title.

```
function FindGalleryByDescription(AGalleryDescription: String):
TFlickrGallery;
```

Returns a gallery by description.

```
function FindSetByID(ASetID: String): TFlickrSet;
```

Returns a set by ID.

```
function FindSetByTitle(ASetTitle: String): TFlickrSet;
```

Returns a set by title.

```
function FindSetByDescription(ASetDescription: String): TFlickrSet;
```

Returns a set by description.

```
function FindPhotoByID(APhotoID: String): TFlickrPhoto;
```

Returns a photo by id.

```
function FindPhotoInGalleryByID(APhotoID: String): TFlickrPhoto;
```
Returns a photo by id in the galleries collection.

```
function FindPhotoInSetByID(APhotoID: String): TFlickrPhoto;
```
Returns a photo by id in the sets collection.

```
function FindPhotoInStreamByID(APhotoID: String): TFlickrPhoto;
```
Returns a photo by id in the streams collection.

```
function GetAccountInfo: Boolean;
```
Returns a Boolean if the call has completed correctly and 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.

```
function GetGalleries(AUserID: String = ''): Boolean;
```
Returns a Boolean if the call has completed correctly and fills the Galleries collection.

```
function GetSets(AUserID: String = ''): Boolean;
```
Returns a Boolean if the call has completed correctly and fills the Sets collection.

```
function GetPhotoStream(AUserID: String = ''): Boolean;
```
Returns a Boolean if the call has completed correctly and fills the PhotoStream collection.

```
function LastRequestData: AnsiString;
```
Returns the last requested data string. Can be used for debugging purposes.

```
procedure Logout;
```
This procedure can be used to force a user-logout when the user has checked the 'Keep me signed in' checkbox at the login page.

```
function PerformGetURL(AMethod: String; AParameters: TStringList = nil):
TJSONValue;
```
Performs a Post URL with a given method and an optional parameters list. Returns a TJSONValue object when performed correctly. Can be useful for unsupported flickr api calls in the TAdvFlickr.

Sample:

```
var
  jv: TJSONValue;
  jo, joUser: TJSONValue;
begin
  Result := False;
  jv := PerformGetURL('flickr.test.login');
  if Assigned(jv) then
  begin
    try
      jo := GetJSONValue(jv as TJSONObject, 'user');
```

```
      if Assigned(jo) and (jo is TJSONObject) then
      begin
        joUser := GetJSONValue(jo as TJSONObject, 'id');
        if Assigned(joUser) then
        begin
          FUserID := joUser.Value;
          Result := UserID <> '';
        end;

        joUser := GetJSONValue(jo as TJSONObject, 'username');
        if Assigned(joUser) and (joUser is TJSONObject) then
          FUserName := (joUser as
TJSONObject).Get('_content').JsonValue.Value;
      end;
    finally
      jv.Free;
    end;
  end;
```

```
function PerformPostURL(AMethod: String; AParameters: TStringList = nil):
TJSONValue;
```
Same functionality as the PerformGetURL, but uses a HTTP POST instead.

```
function UploadPhotoToStream(AFileName: String; ATitle: String = '';
ADescription: String = ''):TFlickrPhoto;
```
Uploads and returns photo to the photostream.

Gallery Methods:
```
function AddPhoto(APhotoID: String; ACreate: Boolean = False):
TFlickrPhoto; overload;
```
Adds a Photo with a specific ID to a gallery, if the Photo does not exist in the collection, the ACreate parameter can be used to create an Object of this Photo with the specific Photo ID.

```
function AddPhoto(APhoto: TFlickrPhoto): TFlickrPhoto; overload;
```
Adds a photo object to a gallery, the function internally uses the ID property of the Photo to upload.

```
procedure DownloadToFolder(const AFolder: string);
```
Download all the photos from a gallery to a specific folder.

```
function GetPhotos: Boolean;
```
Returns a Boolean if the call has completed correctly and fills the Galleries collection.

Set Methods:
```
function AddAndUploadPhoto(const AFileName: String; ATitle: String = '';
ADescription: String = ''): TFlickrPhoto;
```

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.

```
procedure AddFolder(const AFolder: String);
```
Adds a list of images from a folder to a set.

```
function AddPhoto(const APhotoID: String; ACreate: Boolean = False):
TFlickrPhoto; overload;
```
Adds a photo with a specific Photo ID to a set, if the photo does not exist in the collection, the ACreate parameter can be used to create a photo object.

```
function AddPhoto(APhoto: TFlickrPhoto): TFlickrPhoto; overload;
```
Adds a photo object to the set, the Photo ID property is used to add the photo to the set.

```
procedure DownloadToFolder(const AFolder: string);
```
Download all the photos from a set to a folder.

```
function GetComments: Boolean;
```
Retrieves the comments and fills the comment collection of a set.

```
function GetPhotos: Boolean;
```
Retrieves the photos and fills the photo collection of a set.

```
function Remove: Boolean;
```
Remove a set.

Photo Methods:
```
function DownloadLargest(ATargetFile: String = ''): Boolean; overload;
```
Download the largest photo of the list of photo sizes.

```
function DownloadSmallest(ATargetFile: String = ''): Boolean; overload;
```
Download the smallest photo of the list of photo sizes.

```
function Download(ASize: String; ATargetFile: String = ''): Boolean;
overload;
```
Download the photo with a specific size of the list of photo sizes.

```
function FindSizeByLabel(ASizeLabel: String): TFlickrSize;
```
Return the photo size with a specific size label.

```
function GetGeoLocation: Boolean;
```
Gets the geo location and sets the Latitude and Longitude properties.

```
function GetInfo: Boolean;
```
Gets the information and fills the properties of a photo.

```
function GetSizes: Boolean;
```
Gets the downloadable sizes of the photo and fills the sizes collection.

```
function GetTags: Boolean;
```
Gets the tags and fills the tags collection of the photo.

```
function GetComments: Boolean;
```
Gets the comments and fills the comments collection of a photo.

```
function Remove: Boolean; overload;
```
Remove a photo.

```
function SetTags: Boolean;
```
Updates / sets the tags on a photo, found in the Tags collection.

```
function SetGeoLocation: Boolean;
```
Updates / sets the geo-location on a photo, found in the Latitude and Longitude properties.

```
function SetMeta: Boolean;
```
Updates / sets the title and description on a photo, found in the Title and Description properties.

TAdvFourSquare

Usage

TAdvFourSquare 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: TFourSquareCategories; Contains a list of FourSquare categories and sub-categories.

ID: string; The category ID.
Summary: string; The name of the category.
PluralName: string; The plural name of the category.
ShortName: string; The short name of the category.
IconURL: 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: TFourSquareCategories; Contains a list of sub-categories.

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

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

UserProfile: TFourSquareUserProfile; 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.
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.
HomeCity: string; The home city of the user.
Gender: TFourSquareGender; The gender of the user.
Email: The email address of the user.
PhoneNumber: The phone number of the user.
FacebookID: The Facebook ID of the user.
TwitterID: The Twitter ID of the user.
Bio: The description of the user.

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

ID: string; The CheckIn ID.
Created: TDateTime; The time when this CheckIn was created.
Comment: string; The comment for this CheckIn.
Venue: TFourSquareVenue; The venue related to this CheckIn.

Venues: TFourSquareVenues; Collection that contains a list of FourSquare venues.

ID: string; The venue ID.
Summary: string; The name 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.
Phone: string; The phone number 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.
Tips: TFourSquareTips; A list of tips that has been posted for the venue.

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

Photos: TFourSquarePhotos; A list of photos related to the venue.

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

Hours: TFourSquareHours; 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: TFourSquareIsOpen; Indicates if the venue is currently open, closed or if opening hours are unknown.

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

ID: string; The special ID.
Title: string; The title of the special.
SpecialType: TFourSquareSpecialType; The type of the special.
Summary: The description of the special.
Description: The description of how to unlock the special.
FinePrint: The specific rules of the special.

Methods:

GetUserProfile(Profile: TFourSquareUserProfile = nil): string;
Gets the user profile information for the Profile.ID or for the authenticated user if no Profile is provided. *

GetCheckIns: string;
Gets a list of CheckIns for the currently authenticated user. Fills up the UserProfile.CheckIns collection. *

GetCategories: string;
Gets a list of FourSquare Venue Categories and Sub-Categories. Fills up the Categories collection. *

GetNearbyVenues(Latitude, Longitude: double; Location: string = ''; Keyword: string = ''; CategoryID: string = ''; ResultCount: integer = 10): string;
Gets a list of nearby venues based on Latitude and Longitude coordinates or Location. *

GetVenue(Venue: TFourSquareVenue): string;
Gets extra information about a venue using the Venue.ID value. *

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

GetTips(Venue: TFourSquareVenue; Sort: TFourSquareSort = ftRecent; ResultCount: integer = 100; ResultOffset: integer = 0; Width: TFourSquareSize = fs100px; Height: TFourSquareSize = fs100px): string; *
Gets the Tips that have been posted for a venue. Fills up the Venues[].Tips collection. The size of the image linked in Venues[].Tips[].ImageURL is based on the Width & Height parameters.

GetPhotos(Venue: TFourSquareVenue; Width: TFourSquareSize = fs100px; Height: TFourSquareSize = fs100px) : string; *

Gets the Photos that have been posted for a venue. Fills up the Venues[].Photos collection. The size of the image linked in Venues[].Photos[].ImageURL is based on the Width & Height parameters.

CheckIn(VenueID: string; Comment: string = '');
Performs a CheckIn (with optional comment text) for the currently authenticated user a the venue specified through the VenueID.

* If the action is not executed successfully, the error type and error description are returned as a string value.

TAdvGCalendar

Usage

TAdvGCalendar is a component that provides access to the Google calendar service. It allows to read, create, update & delete Google Calendars and Google Calendar Events.

Organisation

Properties:

Calendars: TGCalendars; Class property 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.
Color: TGCalendarColor; The default event background color of the calendar. The index of the color type corresponds to the ID of the CalendarColors item.
BackgroundColor: TColor; The custom event background color of the calendar. If BackgroundColor is different from clNone the Color value may be ignored.
ForeGroundColor: TColor; The custom event text of the calendar.
DefaultReminders: TGDefaultReminders; List of default reminders for the calendar.

Minutes: integer; The number of minutes before the start of the event when the reminder is initiated.
Method: TReminderMethod; Indicates which method is used to send the reminder (rmPopup, rmEmail, rmSMS).

Items: TGCalendarItems; Class property 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: TDateTime; The start time of the event.
EndTime: TDateTime; The end time of the event.
IsAllDay: Boolean; Indicates if this is an all-day event.
Location: string; The location of the event.
Visibility: TVisbility; 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)
RecurringID: 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.
Color: TGItemColor; The background event color. The index of the color type corresponds to the ID of the ItemColors item. If Color is icDefault, the Color/BackgroundColor value of the Calendar this event belongs to is used.
Attendees: TGAttendees; List of attendees for the event.

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

SendNotifications: Boolean;
When set to true, attendees for the event will automatically receive notifications by email.

Reminders: TGReminders; List of reminders for the event.

Minutes: integer; The number of minutes before the start of the event when the reminder is initiated.
Method: TReminderMethod; Indicates which method is used to send the reminder (rmPopup, rmEmail, rmSMS).

ItemColors / CalendarColors: TGColors; Class property that contains a list of pre-defined Google calendar/event colors.
ID: integer; The color ID.
BackgroundColor: TColor; The background color definition.
ForegroundColor: TColor; The foreground color definition.

Methods:

GetCalendars;
Fill the list of Calendars.

AddCalendar(Calendar: TGCalendar);
Add a new Calendar entry to the list of Google Calendars. The Summary, Description, Location and TimeZone properties are used to initialize the Calendar.

DeleteCalendar(Calendar: TGCalendar);
Delete an existing Google Calendar.

UpdateCalendar(Calendar: TGCalendar);
Update an existing Google Calendar. The Summary, Description, Location and TimeZone properties are can be updated.

GetCalendar(ID: string; FromDate, ToDate: TDate); overload;
GetCalendar(ID: string; ChangedSince: TDateTime); overload;

GetCalendar(FromDate, ToDate: TDate); overload;
GetCalendar(ID: string); overload;
Fill the Items list with calendar events from a Google calendar for a certain timespan. If no ID 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.

GetColors;
Fill the list of ItemColors and CalendarColors with the predefined values from the Google Calendar service.

GetItemByID(CalendarID, ItemID: string): TGCalendarItem;
Retrieve a single event based on the ID of the Google calendar and the event ID.

Add(Item: TGCalendarItem);
Add a new event to the calendar as specified by Item.CalendarID.

Update(Item: TGCalendarItem);
Update an event.

Example:

```
ci: TGCalendarItem;

ci.Summary := 'Item Name';
ci.Description := 'Item Description';
ci.Location := 'Item Location';
AdvGCalendar1.Update(ci);
```

Delete(Item: TGCalendarItem);
Delete an item.

TAdvGContacts

Usage

TAdvGContacts 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: TGContacts; Contains a list of Google Contact items.

ID: string; The ID of the contact.
Birthday: TDateTime; The birthday of the contact.
Company: string; The company name related to the contact.
CustomItems: TGCustomItems; Contains a list of custom item data.

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

Emails: TGEmails; 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.
EmailType: TGFieldType; The type of email address.
Primary: boolean; Indicates if this is the contact's primary email address.

FirstName: string; The first name of the contact.
MiddleName: string; The middle name of the contact.
LastName: string; The last name of the contact.
FullName: string; The full name of the contact.
NickName: string; The nickname of the contact.
OwnerName: string; The ownername of the contact.
Groups: TGContactGroups; 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.
InstantMessengers: TGInstantMessengers; 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.
InstantMessengerType: TGIMType; The type of instant messenger.

JobTitle: string; The job title of the contact.

NamePrefix: string; The name prefix of the contact.
NameSuffix: string; The name suffix of the contact.
Notes: string; The notes for the contact.
PhoneNumbers: TGPhoneNumbers; 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.
PhoneType: TGPhoneType; The type of phone number.
Primary: boolean; Indicates if this is the contact's primary phone number.

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

Address: string; The full postal address.
AddressType: TFieldType; 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.
Relation: TGRelationType; The type of relation.
Value: string; The value text (Name) of the relation.

Title: string; The title of the contact.
Websites: TGWebSites: 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.
WebsiteType: TGWebsiteType; The type of website.

Updated: TDateTime; The timestamp when the contact item was last updated.

Groups: TGGroups; Contains a list of Google Group items.

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

Methods:

GetContactGroups;
Fill the list of groups.

GetContacts;
Fill the list of contacts.

Add(Item: TGContact);
Add a new Google contact item.

Example:

```
gc: TGContact;

gc.FirstName := edFirstName.Text;
gc.LastName := edLastName.Text;
AdvGContacts1.Add(gc);
```

Update(Item: TGContact);
Update a Google contact item.

Delete(Item: TGContact);
Delete a Google contact item.

AddGroup(Item: TGGroup);
Add a new Google group item.

UpdateGroup(Item: TGGroup);
Update a Google group item.

DeleteGroup(Item: TGGroup);
Delete a Google group item.

UpdateImage(Item: TGContact; Filename: string);
Add or update the image assigned to a Google contact item.

DeleteImage(Item: TGContact);
Delete the image assigned to a Google contact item.

TAdvGPlaces

Usage

TAdvGPlaces is a component that provides access to the Google places service. It enables to read places, navigate to a certain location and view all nearby places with their information.

Properties:

Places: TGPlacesItems : contains a list of all retrieved places
Place: TGPlacesItem: Contains all the info of a place

PlaceId: string : The unique place identifier
Title: string : The place title
Lat: double : The latitude of a place
Long: double : The longitude of a place
Icon: string : The accompanying icon
Open: Boolean : If the place is currently open
Vicinity: string : The vicinity of the place
Rating: string : The current rating
Phone: string : A universal phone number
Website: string : The places website
Photos: TPhotoItems : All photos
Types: TStringList : All describing types
Address: TGPlacesAddress : The full address

Photos: TPhotoItems : Contains a list of all photos of a place
Photo: TGPhotoItem : Contains all info of a photo

Reference: string : The photo identifier

Address: TGPlacesAddress : Contains all parts of the full address

FormattedAddress: string : A long string of full address info
Parts: TStringList : The address of a place in its separate parts

LastError: string : returns the last received error message if a search fails

Methods:

SearchNearby(ALong, ALat: double; AType: string = ''): boolean;
- Searches all nearby places (optionally from a type) by coordinates
Result is true when the request succeeds

SearchByText(ADiscription: string; AType: string): boolean;
- Searches all nearby places by a text string
Result is true when the request succeeds

HasNextPage; Boolean;
- Informs whether there are more matching places found

GetNextPlacesPage;
- Fetches the next page of places

AutoComplete(Search: string): string;
- Will try to autocomplete your search query

GetDetails(PlaceItem: TGPlacesItem);
- Gets the full Place information of the Place item passed as parameter

TAdvGTasks

Usage

TAdvGTasks is a component that provides access to the Google tasks service. It enables to read, update, create and delete taskslists and tasks. It also allows to read, update, create and delete task children.

Organisation

Properties:

TaskLists: TGTaskListItems : contains a list of all tasklists
TaskList: TGTaskListItem : Contains all info of a tasklist

Id: string : The tasklist identifier
Title: string : The tasklists title
Updated: TDateTime : When the tasklist was last updated
TaskItems: TGTaskItems: all tasks under this tasklist

Tasks: TGTaskItems : Contains a list of all tasks under a tasklist
Task: TGTaskItem : Contains all info of a (child)task

Id: string : The task identifier
Title: string : The tasks title
Updated: TDateTime : When the task was last updated
Position: integer : The tasks position inside the list
Status: string : The current status of the task : “Completed” | “needsaction”
Due: TDateTime : The tasks due date
Completed: TDateTime : The date and time when a task was completed

Methods:

GetTaskListItems(MaxResults: integer);
- Fill the list of tasklists

AddTaskList(TaskListName: string);
- Add a new task list item

UpdateTaskList(TaskListItem: TGTaskListItem);
- Updates a task list item

DeleteTaskList(TaskListItem: TGTaskListItem);
- Removes a tasklist

HasNextPage: boolean
- Informs whether there another page of tasklists

GetNextTaskListItems;
- Gets the next page of tasklists

GetTaskItems(MaxResults: integer; overloads..);
- Fetches all tasks for a specific tasklist

HasNextPage: Boolean;
- Informs whether there is another page of tasks

GetNextTaskItems;
- Fetches the next page of tasks

AddTaskToList(TaskItem: TGTaskItem);
- Add a task to the list

UpdateTask (TaskItem: TGTaskItem);
- Updates the specific task

DeleteTask(TaskItem: TGTaskItem);
- Removes a task from the tasklist

TAdvPayPal

Usage

TAdvPayPal is a component that facilitates access to the PayPal payment service. It allows users to make payments using an existing PayPal account and for an easy way to integrate a PayPal based payments service in Windows applications.
The component supports using the Sandbox (testing) environment as well as the Live environment.

Organisation

Properties:

APIEnvironment: TPayPalEnvironment; Switch between the Sandbox (testing) and Live API environment.

Payment: TAdvPayPalPayment; Class property that contains payment information.

ID: string; The payment id.
Currency: string; The currency of the payment.
Payer: TAdvPayPalPayer; The Source of the funds for this payment represented by a PayPal account.

ID: string; PayPal assigned payer ID.
Email: string; Email address of the payer.
FirstName: string; First name of the payer.
LastName: string; Last name of the payer.
Street: string; The street part of the shipping address.
City: string; The city part of the shipping address.
State: string; The state part of the shipping address.
PostalCode: string; The postal code part of the shipping address.
CountryCode: string; The country code part of the shipping address.
Recipient: string; Name of the recipient at the shipping address.

State: string; The state of the sale. If the payment transaction completed successfully the State returns ‘completed’.
Total: double; The total amount of the payment.
TransactionID: string; The transaction id.

PaymentError: TAdvPayPalError; Class property that contains payment error information.

ErrorMessage: string; Message describing the error.
ErrorName: string; Unique name of the error.
ErrorDescription: string; Reason for the error.
ErrorDetails: string; Name of the field that caused the error.

Transaction: TAdvPayPalTransaction; Class property that contains transaction information.

Currency: TPayPalCurrency; The currency of the transaction.
Description: string; The description of the transaction. (Optional)
Insurance: double; Amount charged for insurance. (Optional)
Items: TPayPalItems; The collection of items related to the transaction. It is required that the transaction contains at least one item before executing the payment.

Description: string; The description of the item. (Optional)
Name: string; Item name.
Price: double; Item cost.
Quantity: Number of a particular item.
SKU: string; Number of units per item. (Optional)

Shipping: double; Amount charged for shipping. (Optional)
Tax: double; Amount charged for tax. (Optional)

Methods:

DoPayment;
Execute a PayPal payment based on the Transaction configuration. During payment execution the user will be asked to authorize the payment using PayPal credentials. If the user cancels the authorization the OnPaymentCancelled event is triggered. If the payment succeeds the Payment class properties are assigned with the payment results and the OnPaymentAccepted event is triggered. If an error occurred during the payment execution, the PaymentError class properties are assigned and the OnPaymentFailed is triggered.

Events:

OnPaymentAccepted: TNotifyEvent;
Event fired when a Payment was successfully authorized and executed. Payment details are available in the Payment class properties.

OnPaymentCancelled: TNotifyEvent;
Event fired when a Payment was cancelled by the user.

OnPaymentFailed: TNotifyEvent;
Event fired when an error occurred during execution of the Payment. Error details are available in the PaymentError class properties.

Example:
Configure a transaction and execute a PayPal payment.

```
var
```

```
  pi: TPayPalItem;

AdvPayPal1.App.Key := 'xxx';
AdvPayPal1.App.Secret := 'yyy';

AdvPayPal1.Transaction.Currency := pcUSD;
AdvPayPal1.Transaction.Shipping := 5;

AdvPayPal1.Transaction.Items.Clear;

pi := AdvPayPal1.Transaction.Items.Add;
pi.Price := 1;
pi.Name := 'Item Name';
pi.Description := 'Item Description';
pi.Quantity := 1;

AdvPayPal1.DoPayment
```

TAdvPicasa

Note: This API has been deprecated by Google. Please use TAdvGooglePhotos instead.

Usage

TAdvPicasa 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: TPicasaAlbums; Collection that contains a list of Picasa albums.

> ID: string; The album ID.
> Title: string; The title of the album.
> ImageURL: string; The link to the image connected to this album.
> MaxPhotoSize: TPicasaPhotoSize; The maximum size of the photo when downloaded using TPicasaPhoto.ImageURL. Set to psOriginal to retrieve the photo in its original size (default).
> Photos: TPicasaPhotos; Contains a list of photos that the album contains.
> Summary: string; The summary of the album.
> Updated: TDateTime; The time when the album was last updated.

Photos: TPicasaPhotos; 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: TPicasaUser; The author of the photo.
Comments: TPicasaComments; Contains a list of comments for this photo.
FileName: string; The filename of the photo.
ImageURL: string; The link to the image file of the photo. (The size of the image is defined in TPicasaAlbum.MaxPhotoSize.
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.
Tags: TStringList; The tags that have been set for the photo.
ThumbnailURL: string The link to the thumbnail image file of the photo.
Updated: TDateTime; The time when the photo was last updated.

Comments: TPicasaComments; Collection that contains a list of Picasa comments.

ID: string; The comment ID.
Author: TPicasaUser; The author of the comment.
PublishDate: TDateTime: The time when the comment was published.
Text: string; The text of the comment.
Title: string; The title of the comment.

Author: TPicasaUser; Class property that contains user information.

ID: string; The user id.
FullName: string; The full name of the user.
NickName: string; The nickname of the user.

Methods:

GetAlbums: Boolean;
Fill the list of albums, accessible via AdvPicasa.Albums

Album[].GetPhotos: TPicasaPhotos;
Fill the list of Picasa photos assigned to the album, accessible via AdvPicasa.Albums[Index].Photos

CreateAlbum(Album: TPicasaAlbum): Boolean;
Create a new Picasa album with properties set via Album parameter on the Picasa web service.

Example:

```
var
  Album: TPicasaAlbum;

  Album := AdvPicasa1.Albums.Add;
  Album.Title := edAlbumTitle.Text;
  Album.Summary := edAlbumDescription.Text;
  AdvPicasa1.CreateAlbum(Album);
```

DeleteAlbum(Album: TPicasaAlbum): Boolean;
Delete an existing Picasa album and all photos assigned to the album.

DeletePhoto(Photo: TPicasaPhoto): Boolean;
Delete an existing Picasa photo.

UploadPhoto(Album: TPicasaAlbum; FileName: string; Summary: string; string = ''; Tags: string = '';
Latitude: double = -1; Longitude: double = -1): TPicasaPhoto;
Upload a new photo to an existing Picasa album.

Example:

```
  if (OpenDialog1.Execute) then
  begin
    AdvPicasa1.UploadPhoto(AdvPicasa1.Albums[i],
      OpenDialog1.FileName, Description, Tags,
      Latitude, Longitude;
  end;
```

UpdatePhoto(Photo: TPicasaPhoto): Boolean;
Update an existing Picasa photo.

SearchPhotos(Keywords: string;var MoreResults: Boolean;const Page: integer = 0;const PageSize:
integer = 10): integer;
Search in the available public Picasa albums for photos that match the provide keyword(s).
MoreResults returns true if more photos are available, false otherwise. Returns the number of
photos retrieved and retrieved photos are accessible via the collection property
AdvPicasa.SearchResults: TPicasaPhotos

GetComments(Photo: TPicasaPhoto): TPicasaComments;
Fill the list of Picasa comments that have been made for the photo.

DownloadPhoto(TargetFile: string;Photo: TPicasaPhoto);
Download a photo to a local folder.

DownloadToFolder(const Folder: string;Album: TPicasaAlbum); overload;
Download all photos assigned to the album to a local folder.

DownloadToFolder(const Folder: string;Photos: TPicasaPhotos); overload;
Download all photos within the photos collection to a local folder.

AddFolderToAlbum(Folder, Title, Summary: string): TPicasaAlbum;
Upload all files from a local folder to a new or existing Picasa album.
If an album exists with a Title equal to the Title parameter value, the photos are added to the
existing album, if not a new album is automatically created.

FindAlbumByTitle(AlbumTitle: string): TPicasaAlbum;
Returns a Picasa album for which the Title matches the AlbumTitle parameter value.

Events:

OnBeforeAddPhoto: TOnBeforeAddPhotoEvent;
Event fired before a photo is added with the UploadPhoto/AddFolderToAlbum method.

OnAfterAddPhoto: TOnAfterAddPhotoEvent;
Event fired after a photo has been added with the UploadPhoto/AddFolderToAlbum method.

OnBeforeDownloadPhoto: TOnBeforeDownloadPhotoEvent;
Event fired before a photo is downloaded with the DownloadPhoto/DownloadToFolder method.

OnAfterDownloadPhoto: TOnAfterDownloadPhotoEvent;
Event fired after a photo has been downloaded with the DownloadPhoto/DownloadToFolder method.

TAdvGooglePhotos

Usage

TAdvGooglePhotos is a component that facilitates access to the Google Photos service. It allows retrieving your albums and photos as well as creating albums, uploading / downloading photos to albums.

Organisation

Properties:

Albums: TGooglePhotosAlbums; Collection that contains a list of GooglePhotos albums.

ID: string; The album ID.
Title: string; The title of the album.
ImageURL: string; The link to the image connected to this album.
Photos: TGooglePhotosPhotos; Contains a list of photos associated with the Album after GetPhotos was called.
ItemsCount: Indicates the number of photos associated with the Album.
URL: The URL to the Album in the GooglePhotos web interface.

SearchResults / Photos: TGooglePhotosPhotos; Collection that contains a list of GooglePhotos photos.

ID: string; The photo ID.
AlbumID: string; The ID of the album the photo belongs to.
FileName: string; The filename of the photo.
ImageURL: string; The link to the image file of the photo in it's original size. To access a custom size of the photo, use CustomSizeImageURL(Width, Height).
Summary: string; The description of the photo.
Tags: TStringList; The tags that have been set for the photo.
Created: TDateTime; The timestamp when the photo was created.
Width: Int64; The original width of the photo.
Height: Int64; The original height of the photo.
MimeType: string; The mimetype of the photo.

Methods:

GetAlbums: Boolean;
Fill the list of albums, accessible via AdvGooglePhotos.Albums

Album[].GetPhotos: TGooglePhotosPhotos;
Fill the list of GooglePhotos photos assigned to the album, accessible via AdvGooglePhotos.Albums[Index].Photos

Albums[].Photos[].CustomSizeImageURL(Width, Height): string;
The link to the image file of the photo with a custom size based on the provided Width and Height values.

CreateAlbum(Album: TGooglePhotosAlbum): Boolean;
Create a new GooglePhotos album with properties set via Album parameter on the GooglePhotos web service.

Example:

```
var
  Album: TGooglePhotosAlbum;

  Album := AdvGooglePhotos1.Albums.Add;
  Album.Title := edAlbumTitle.Text;
  Album.Summary := edAlbumDescription.Text;
  AdvGooglePhotos1.CreateAlbum(Album);
```

UploadPhoto(Album: TGooglePhotosAlbum; FileName: string; Summary: string): TGooglePhotosPhoto;
Upload a new photo to an existing GooglePhotos album.

Example:

```
  if (OpenDialog1.Execute) then
  begin
    AdvGooglePhotos1.UploadPhoto(AdvGooglePhotos1.Albums[i],
      OpenDialog1.FileName, Description);
  end;
```

SearchPhotos(AlbumID): integer;
Search for photos associated with specific Album ID. If no ID is specified all available photos are returned. Retrieved photos are accessible via the collection property AdvGooglePhotos.SearchResults: TGooglePhotosPhotos

DownloadPhoto(TargetFile: string;Photo: TGooglePhotosPhoto);
Download a photo to a local folder.

DownloadToFolder(const Folder: string;Album: TGooglePhotosAlbum); overload;
Download all photos assigned to the album to a local folder.

DownloadToFolder(const Folder: string;Photos: TGooglePhotosPhotos); overload;
Download all photos within the photos collection to a local folder.

AddFolderToAlbum(Folder, Title, Summary: string): TGooglePhotosAlbum;
Upload all files from a local folder to a new or existing GooglePhotos album.
If an album exists with a Title equal to the Title parameter value, the photos are added to the existing album, if not a new album is automatically created.

FindAlbumByTitle(AlbumTitle: string): TGooglePhotosAlbum;
Returns a GooglePhotos album for which the Title matches the AlbumTitle parameter value.

Events:

OnBeforeAddPhoto: TOnBeforeAddPhotoEvent;
Event fired before a photo is added with the UploadPhoto/AddFolderToAlbum method.

OnAfterAddPhoto: TOnAfterAddPhotoEvent;
Event fired after a photo has been added with the UploadPhoto/AddFolderToAlbum method.

OnBeforeDownloadPhoto: TOnBeforeDownloadPhotoEvent;
Event fired before a photo is downloaded with the DownloadPhoto/DownloadToFolder method.

OnAfterDownloadPhoto: TOnAfterDownloadPhotoEvent;
Event fired after a photo has been downloaded with the DownloadPhoto/DownloadToFolder method.

TAdvYouTube

Usage

TAdvYouTube is a component that facilitates access to the Google YouTube service. It allows retrieving your videos as well as uploading new videos. Photos can be uploaded with a title and description. The component also enables liking / unliking a video.

Organisation

Properties:

Videos: TYouTubeVideos; Collection that contains a list of YouTube videos.

ID: string; The video ID.
CategoryID: string; The id of the category the video belongs to.
ChannelID: string; The id of the YouTube channel the video belongs to.
ChannelTitle: string; The title of the YouTube channel the video belongs to.
Description: string; The description of the video.
ImageURL: string; The link to the thumbnail preview image of the video.
Link: string; The link to the video on YouTube.
PublishDate: TDateTime; The time when the video was published.
Rating: TYouTubeRating; The rating of the video. (Requires calling GetRating first)
Title: string; The title of the video.

Methods:

GetAllVideos: integer;
Fill the list of videos with all video items that belong to the authenticated user, accessible via AdvYouTube.Videos. MaxResults defines the maximum number of items that are returned.

GetFirstVideos(MaxResults: integer): integer;
Fill the list of videos with the first page of video items, accessible via AdvYouTube.Videos. MaxResults defines the maximum number of items that are returned. Returns the total number of videos.

GetNextVideos(MaxResults: integer): integer;
Adds the next page of video items to the list of videos, accessible via AdvYouTube.Videos. MaxResults defines the maximum number of items that are returned. Returns the total number of videos.

GetDetails(AVideo: TYouTubeVideo);
Get detailed information for a specific video. This fills the CategoryID and ChannelTitle properties for a specific video.

GetRating(AVideo: TYouTubeVideo): TYouTubeRating;
Get the authenticated user’s rating for a specific YouTube video. This also assigns the Rating property of the video.

SetRating(AVideo: TYouTubeVideo; ARating: TYouTubeRating);
Set the authenticated user's rating for a specific YouTube video. This also assigns the Rating property of the video.

DeleteVideo(AVideo: TYouTubeVideo);
DeleteVideo(AVideoID: string);
Delete an existing YouTube video.

UploadVideo(AFileName, ATitle, ADescription: string): TYouTubePhoto;
Upload a new video to a YouTube account.

Example:

```
  if OpenDialog1.Execute then
  begin
     AdvYouTube1.UploadVideo(OpenDialog1.FileName, edTitle.Text,
edDescription.Text);
  end;
```

UpdateVideo(AVideo: TYouTubeVideo);
Update an existing YouTube video. Properties than can be updated are Title, Description and CategoryID.

Events:

OnBeforeAddVideo: TOnBeforeAddVideoEvent;
Event fired before a photo is added with the UploadPhoto/AddFolderToAlbum method.

OnAfterAddPhoto: TOnAfterAddVideoEvent;
Event fired after a photo has been added with the UploadPhoto/AddFolderToAlbum method.

TAdvInstagram

Note: Unfortunately a large part of the Instagram API has been deprecated and is no longer functional.
More information can be found here:
https://developers.facebook.com/blog/post/2018/01/30/instagram-graph-api-updates/

Usage

TAdvInstagram is a component that facilitates access to the Instagram service. It allows retrieving your photos, your followers (and their photos) and the profiles you are following (and their photos) as well as retrieving the tags, likes and comments for a photo. The component also enables searching photos by tag, username or location.

Organisation

Properties:

Users: TInstagramUsers; Collection that contains a list of Instagram users.

ID: string; The user ID.
Bio: string; The description of the user.
FullName: string; The full name of the user.
UserName: string; The name of the user.
Following: integer; The number of users the user is following.
Followers: integer; The number of users that is following the user.
Photos: integer; The number of photos the user has posted.

Photos: TInstagramPhotos; Collection that contains a list of Instagram photos.

ID: string; The photo ID.
Caption: string; The caption of the photo.
Location: TInstagramLocation; The location of the photo.

ID: string; The location ID.
Summary: string; The description of the location.
Latitude: string; The latitude of the location.
Longitude: string; The longitude of the location.

Filter: string; The filter used on the photo.
Tags: TStringList; The tags associated to the photo.
Comments: TInstagramComments; The comments for the photo.
CommentsCount: integer; The number of comments the photo has received.
Likes: TInstagramUsers; A list of users that has liked the photo.
LikesCount: integer; The number of likes the photo has received.

UserLikes: Boolean; Indicates if the currently authenticated user has liked the photo.
Link: string; A link to the photo on Instagram.
CreatedTime: TDateTime; The time the photo was created.
Images: TInstagramImages; The Images associated with the photo.
From: TInstagramUser; The user that posted the photo.

Tags: TInstagramTags; Collection that contains a list of Instagram tags.

Summary: string; The name of the tag.
MediaCount: integer; The number of Instagram photos that have received this tag.

Comments: TInstagramComments; Collection that contains a list of Instagram comments.

ID: string; The comment ID.
From: TInstagramUser; The user who posted the comment.
Text: string; The text content of the comment.
CreatedTime: TDateTime; The time the comment was posted.

Images: TInstagramImages; Collection that contains a list of Instagram images.

URL: string; The URL of the image.
Width: integer; The width of the image.
Height: integer; The height of the image.

Methods:

Follow(ID: string): string;
Follow the Instagram user associated with the ID.

UnFollow(ID: string): string;
Don't follow the Instagram user associated with the ID.

PostComment(ID, Text: string): string;
Post a comment to the Instagram photo associated with the ID.

DeleteComment(MediaID, CommentID: string): string;
Delete the comment (using CommentID) associated with the MediaID.

Like(ID: string): string;
Like the Instagram photo associated with the ID.

UnLike(ID: string): string;
UnLike the Instagram photo associated with the ID.

DownloadPhoto(ATargetFile, APhotoURL: String): Boolean;
Download a photo to a local folder.

GetProfile(UserID: string; User: TInstagramUser);
Retrieve the full profile information for an Instagram user.

GetFeed(Count: integer = 0; MaxID: string = ''; MinID: string = ''): Boolean;
Retrieve the Instagram feed for the currently authenticated user. Fills up the Photos collection.

GetMediaByUser(UserID: string = ''; Count: integer = 0; MaxID: string = ''; MinID: string = ''): Boolean;
Retrieve photos posted by the Instagram user associated with the UserID. Fills up the Photos collection.

GetMediaByTag(HashTag: string; MaxTagID: string = ''; MinTagID: string = ''): string;
Retrieve photos tagged with “HashtTag”. Fills up the Photos collection.

GetMediaByLocation(Latitude, Longitude: Double): Boolean;
Retrieve photos for a certain Location.

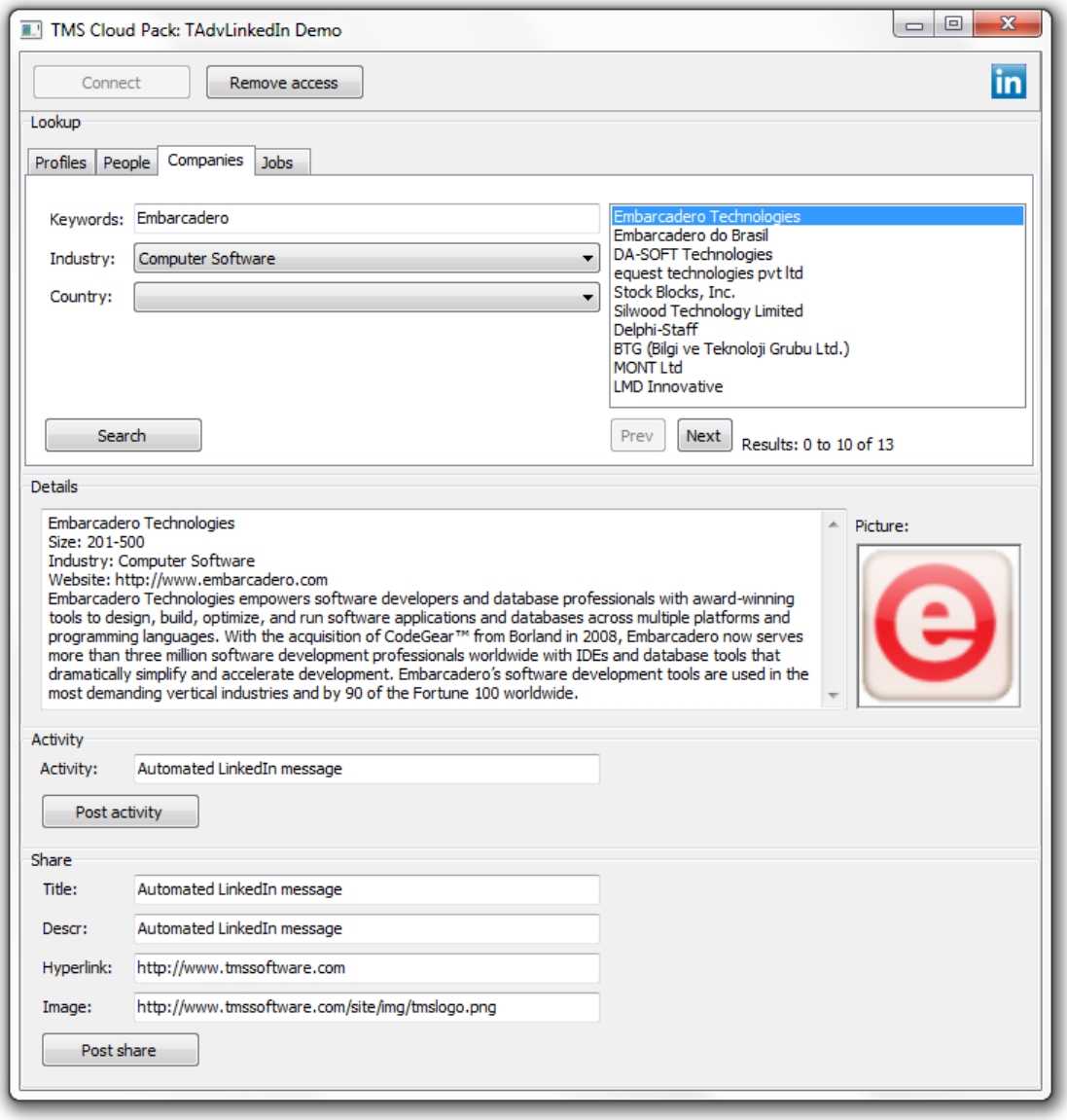
GetFollowing(UserID: string = '');
Retrieve the users that a user is following. Fills up the Users collection.

GetFollowers(UserID: string = '');
Retrieve the users that are following a user. Fills up the Users collection.

SearchUsers(Keyword: string);
Search Instagram users by keyword. Fills up the Users collection.

SearchTags(Keyword: string);
Search Instagram tags by keyword. Fills up the Tags collection.

TAdvLinkedIn



Usage

TAdvLinkedIn is a component that provides access to the LinkedIn API service. It allows to post an activity and share a message (with optional URL and image), retrieve a list of connections. It also enables searching for People, Companies and Jobs that are listed on LinkedIn.

Organisation

Properties:

DefaultProfile: TLinkedInProperty; Class property that contains the LinkedIn 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.
ConnectionsCount: integer; The number of connections of the user.
ConnectionsCapped: boolean; Indicates if the ConnectionsCount value is capped. (LinkedIn won't communicate if a user has 500+ connections)
Positions: TLinkedInPositions; List of the user's job positions.

ID: string; The ID for this position.
Company: TLinkedInCompany; The company related to this position.
EndMonth: integer; The month indicating when the position ended.
EndYear: integer; The year 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.
StartMonth: integer; The month indicating when the position began.
StartYear: integer; The year indicating when the position began.

Location: string; The location of the user.
CountryCode: string; The country code of the user.
CurrentShare: TLinkedInShare; The current share of the user.

ID: string; The ID for this share.
Author: TLinkedInAuthor; The author for this share.

ID: string; The ID for the author.
FirstName: string; The first name of the author.
LastName: string; The last name of the author.

Comment: string; The comment for this share.
Description: string; The description for this share.
TimeStamp: TDateTime; The time this share was posted.
Title: string; The title of this share.
URL: string; The URL for this share.

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.
LastModified: TDateTime; The time when the user's profile was last modified.

Associations: string; The associations of the user.
Honors: string; The honors of the user.
ProposalComments: string; The description of how the user approaches proposals.
Publications: TLinkedInPublications; A list of the user's publications.

ID: string; The ID for this publication.
PublicationDate: TDateTime; The time this publication was published.
PublicationTitle: string; The title of this publication.

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

ID: string; The ID for this patent.
PatentDate: TDateTime; The time related to this patent.
PatentTitle: string; The title of this patent.

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

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

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

ID: string; The ID for this skill.
SkillName: The name of this skill.

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

ID: string; The ID for this certification.
CertificationName: The name of this certification.

Educations: TLinkedInEducations; 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: TLinkedInCourses; A list of the user's courses.

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

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

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

Recommendations: TLinkedInRecommendations; 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: TLinkedInProfile; The profile of the person who made the recommendation.

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

ID: string; The ID for this object.
ObjectName: string; The name of this object.
ObjectType: TLinkedInObjecType; The type of this object.

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

ID: string; The ID for this job bookmark.
Company: TLinkedInCompany; 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.

Groups: TLinkedInGroups; A list of LinkedIn groups the user is a member of.

ID: string; The ID for this group.
GroupName: string; The name of this group.
MembershipState: string; The state of the user's membership to the specified group.

BirthDate: TDateTime: The user's birth date.
Resources: TLinkedInResources; A list of URLs the user has chosen to share on their LinkedIn profile.

ID: string; The ID for this resource.
ResourceName: string; The name of the resource.
URL: string; The URL of the resource.

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

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

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

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

Connections: TLinkedInConnections; A list of user profiles the currently authenticated user is connected with.

Classes:

TLinkedInCompany:

ID: string; The ID for the company.
BlogURL: string The URL for the company blog.
CompanyName: string; The name of the company.
CompanyType: string; The type of the company.
Description: string; The description of the company.
ImageURL: string; The URL of the company logo image.
Industries: TLinkedInObjects; A list containing the names of the company's industries.
Locations: TLinkedInLocations; 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: TStringList; A list of the company's specialties.
StockExchange: string; 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.
Website: string; The company website address.

TLinkedInJob:

ID: string; The ID for this job.
Company: TLinkedInCompany; The hiring company for this job.
CountryCode: string; The country code for this job.
Description: string; The description for the position.
ExperienceLevel: string; The experience level for the position.
Industries: TLinkedInObjects; A list of industries related to this position.
IsActive: boolean; Indicates of the job listing is active.
JobURL: string; The URL for the job listing.
JobPoster: TLinkedInProfile; The user who posted the job listing.
Location: string; The location for this job.
Position: string; The description of the job position.
PostingDate: TDateTime; 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.

TCompanyResult:

ID: string; The ID for this company.
CompanyName: string; The name of the company.

TJobResult:

ID: string; The ID for this job.
Company: TLinkedInCompany; The company for this job.
Description: string; The description for this position.
Location: string; The location for this job.
Position: string; The description of the job position.

TPeopleResult:

ID: string; The ID for this person.
FirstName: string; The first name of this person.
LastName: string; The last name of this person.

Methods:

Activity(const: Msg: string): boolean;
Post an activity message on the stream of the user.

GetDefaultProfile;
Fill the DefaultProfile class property with the profile information of the user.

GetConnections;
Fill the Connections list with the connections of the user.

GetProfile(const ID: string): TLinkedInProfile;
Get the profile information of a user based on the user ID.

GetCompanyInfo(const ID: string): TLinkedInCompany;
Get the company information based on the ID of the company.

GetJobInfo(const ID: string): TLinkedInJob;
Get the job information based on the ID of the job.

SearchCompany(SP: TCompanySearchParam; var ResultCount; const Page: integer): TCompanyResults;
Search for companies based on the values in TCompanySearchParam.
Returns a list of companies.

SearchJob(SP: TJobSearchParam; var ResultCount; const Page: integer): TJobResults;
Search for job listings based on the values in TJobSearchParam.
Returns a list of jobs.

SearchPeople(SP: TPeopleSearchParam; var ResultCount; const Page: integer): TPeopleResults;
Search for people based on the values in TPeopleSearchParam.
Returns a list of people.

Example:

```
var
  sp: TPeopleSearchParam;
  pr: TPeopleResults;
  resultcount, maxresult: integer;

  sp.Keywords := edKeywords.Text;
  sp.FirstName := edFirstName.Text;
  sp.LastName := edLastName.Text;
```

```
  sp.CompanyName := edCompany.Text;
  sp.CountryCode := icUnitedStates;

  pr := AdvLinkedIn1.SearchPeople(sp, resultcount, 0);
```

Share(const Title, Msg, HyperLink, ImageLink: string): boolean;
Share a message with optional title, hyperlink and imagelink on the stream of the user.

Example:

```
  AdvLinkedIn1.Share(edTitle.Text, edDescription.Text, edHyperlink.Text,
edImageLink.Text);
```

TAdvLiveCalendar

Note: This API is deprecated. Please use TAdvOutlookCalendar instead.

Usage

TAdvLiveCalendar 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: TLiveCalendars; Contains a list of Live calendars.

ID: string; The calendar ID.
Summary: 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.

Items: TLiveCalendarItems; Contains a list Live calendar events.

ID: string; The event ID.
CalendarID: string; The ID of the calendar this event belongs to.
Summary: string; The name of the event.
Description: string; The description of the event.
StartTime: TDateTime; The start time of the event.
EndTime: TDateTime; The end time of the event.
IsAllDay: Boolean; Indicates if this is an all-day event.
Location: string; The location of the event.
Visibility: TVisbility; 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:

GetCalendars;
Fill the list of Calendars.

GetCalendar(ID: string; FromDate, ToDate: TDate); overload;
GetCalendar(FromDate, ToDate: TDate); overload;
GetCalendar(ID: string); overload;
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.

GetItemByID(CalendarID, ItemID: string): TLiveCalendarItem;
Retrieve a single event based on the ID of the Live Calendar and the event ID.

Add(Item: TLiveCalendarItem);
Add a new event to the Calendar as specified by Item.CalendarID.

Update(Item: TLiveCalendarItem);
Update an event.

Example:

```
ci: TLiveCalendarItem;

ci.Summary := 'Item Name';
ci.Description := 'Item Description';
ci.Location := 'Item Location';
AdvLiveCalendar1.Update(ci);
```

Delete(Item: TLiveCalendarItem);
Delete an event.

TAdvLiveContacts

Note: This API is deprecated. Please use TAdvOutlookContacts instead.

Usage

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

Organisation

Properties:

Items: TLiveContactItems; 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: TGender; 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:

GetContacts;
Fill the list of Items.

Add(Item: TLiveContactItem);
Add a new Live contact item.

Example:

```
ci: TLiveContactItem;

ci.FirstName := edFirstName.Text;
ci.LastName := edLastName.Text;
AdvLiveContacts1.Add(ci);
```

TAdvOutlookCalendar

Usage

TAdvOutlookCalendar is a component that provides access to the Outlook Calendar service. It allows to retrieve a list of Outlook Calendars and read, create, update and delete Outlook Calendar events.

Organisation

Properties:

Calendars: TOutlookCalendars; Contains a list of Outlook calendars.

ID: string; The calendar ID.
Summary: string; The name of the calendar.

Items: TOutlookCalendarItems; Contains a list Outlook calendar events.

ID: string; The event ID.
CalendarID: string; The ID of the calendar this event belongs to.
Created: TDateTime; The timestamp when the event was created.
Update: TDateTime; The timestamp when the event was last modified.
Summary: string; The name of the event.
Description: string; The description of the event.
StrippedDescription: string; The description of the event without HTML tags.
StartTime: TDateTime; The start time of the event.
StartTimeZone: TDateTime; The timezone of the start time.
EndTime: TDateTime; The end time of the event.
EndTimeZone: The timezone of the end time.
IsAllDay: Boolean; Indicates if this is an all-day event.
IsRecurrent: Boolean; Indicates if this is a recurring event.
Location: string; The location of the event.
Sensitivity: TSensitivity; Indicates the level of privacy for the even (normal, personal, private or confidential).
ShowAs: TShowAs; Indicates the status of the event (free, tentative, busy, out of office, working elsewhere, unknown).

TimeZone: Sets the TimeZone to use when retrieving Calendar events with the GetCalendar method. When left empty the default TimeZone (UTC) is used. A full list of available TimeZone values can be found here: https://docs.microsoft.com/en-us/previous-versions/office/office-365-api/api/version-2.0/complex-types-for-mail-contacts-calendar#DateTimeTimeZoneV2

Methods:

AddCalendar(Name: string): Boolean;
Create a new Calendar with the specified Name. Returns True if the action was successful, False otherwise.

DeleteCalendar(Calendar: TOutlookCalendar); Boolean;
Delete an existing Calendar. Returns True if the action was successful, False otherwise.

UpdateCalendar(Calendar: TOutlookCalendar; Name: string): Boolean;
Updates the name of an existing Calendar. Returns True if the action was successful, False if otherwise.

GetCalendars;
Fill the list of Calendars.

GetCalendar(ID: string; FromDate, ToDate: TDate; PageSize: integer = 100;PageIndex: integer = 0); overload;
GetCalendar(FromDate, ToDate: TDate; PageSize: integer = 100;PageIndex: integer = 0); overload;
GetCalendar(ID: string; PageSize: integer = 100;PageIndex: integer = 0); overload;
Fill the Items list with calendar events from an Outlook Calendar for a certain timespan. If no ID is provided, the events are retrieved from the default Outlook Calendar. If no FromDate/ToDate is specified the events are retrieved for the default timespan.
Optionally a PageSize and PageIndex can be provided, if not, the first 100 items are returned.
By default the StartTime and EndTime of events are returned in the UTC timezone. To change the timezone, use the TimeZone property.

GetItemByID(ItemID: string): TOutlookCalendarItem;
Retrieve a single event based on the Outlook Calendar Event ID.

Add(Item: TOutlookCalendarItem);
Add a new event to the Calendar as specified by Item.CalendarID.

Update(Item: TOutlookCalendarItem);
Update an event.

Example:

```
ci: TOutlookCalendarItem;

ci.Summary := 'Item Name';
ci.Description := 'Item Description';
ci.Location := 'Item Location';
AdvOutlookCalendar1.Update(ci);
```

Delete(Item: TOutlookCalendarItem);
Delete an event.

TAdvOutlookContacts

Usage

TAdvOutlookContacts is a component that provides access to the Outlook Contacts service. It enables to create, read, update and delete Outlook Contacts.

Organisation

Properties:

Items: TOutlookContactItems; Contains a list of Outlook 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.
DisplayName: string; The displayname of the contact.
BirthDay: integer; The birthday of the contact.
EmailAddresses: TOutlookContactEmails; List of email addresses for the contact.
MobilePhone: string; The contact's mobile phone number.
HomeAddress: TOutlookContactAddress;The contact's home address.
BusinessAddress: TOutlookContactAddress; The contact's business address.
JobTitle: string; The contact's job title.
CompanyName: string; The contact's company name.
Image: string; Downloads the contact's profile image and returns the filename.

Methods:

GetContacts (PageSize: integer = 100; PageIndex: integer = 0);
Fill the list of Items.
Optionally a PageSize and PageIndex can be provided, if not, the first 100 items are returned.

AddCalendar(Name: string): Boolean;
Add a new Outlook Calendar.

DeleteCalendar(Calendar: TOutlookCalendar): Boolean;
Delete an existing Outlook Calendar.

UpdateCalendar(Calendar: TOutlookCalender; Name: string): Boolean;
Update an existing Outlook Calendar and change it's name.

Add(AItem: TOutlookContactItem);
Add a new Outlook contact item.

Example:

```
ci: TOutlookContactItem;
```

```
ci.FirstName := edFirstName.Text;
ci.LastName := edLastName.Text;
AdvOutlookContacts1.Add(ci);
```

Delete(AItem: TOutlookContactItem);
Delete(AContactID: string);
Delete an Outlook contact item.

Update(AItem: TOutlookContactItem);
Update an existing Outlook contact.

DownloadContactImage(AContactID: string): string;
Download the profile image of an Outlook contact.
Returns the filename of the downloaded image.

UploadContactImage(AContactID: string; FileName: string): boolean;
Upload an image file for an Outlook contact profile picture.
Returns true if the upload succeeded, false if not.

TAdvOutlookMail

Usage

TAdvOutlookMail is a component that provides access to the Outlook Mail service. It enables to retrieve and send email messages. File attachments can be included when sending emails.

Organisation

Properties:

Folders: TOutlookMailFolders; Contains a list of Outlook Folders.

ID: string; The ID of the folder.
DisplayName: string; The name of the folder.
ItemCount: integer; The number of items in the folder.

Items: TOutlookMailItems; Contains a list of Outlook Mail items.

ID: string; The ID of the mail. *
Body: string; The text content of the mail.
MailType: TOutlookMailType; Indicates if the type of the Body content is plain text (mtPlainText) or HTML (mtHTML).
ReceivedDateTime: TDateTime; The date and time the message was received. *
Subject: string; The subject text of the message.
FromName: string; The mailbox owner and sender name of the message. *
FromEmail: string; The mailbox owner and sender email of the message. *
SenderName: string; The account name that is actually used to generate the message. *
SenderEmail: string; The account email that is actually used to generate the message. *
RecipientNames: TStringList; The list of recipient names for the message.
RecipientEmails: TStringList; The list of recipient email addresses for the message.
CcRecipientNames: TStringList; The list of cc recipient names for the message.
CcRecipientEmails: TStringList; The list of cc recipient email addresses for the message.
BccRecipientNames: TStringList; The list of bcc recipient names for the message.
BccRecipientEmails: TStringList; The list of bcc recipient email addresses for the message.
Attachments: TStringList; The list of filenames for the files attached to the message. **
HasAttachments: boolean; Indicates if the message has attachments. *
IsRead: boolean; Indicates if the message has been read *
IsDraft: boolean; Indicates if the message is a draft or not *

* Only used when retrieving email messages with GetEmails.
** Only used when creating a new email message as a parameter value for SendMessage.

Methods:

GetFolders: Boolean;
Fill the list of Folders. Returns true if the operation succeeded.

GetMails(FolderID: string; PageSize: integer; PageIndex: integer): Boolean;
Fill the list of Items. Optionally a FolderID can be specified to only retrieve items from a specific folder. By default only the items from the 'Inbox' folder are retrieved. If no FolderID is specified the items for all folders are returned. For specific system folders the name of the folder can be used as FolderID, this includes 'Inbox', 'Drafts', 'SentItems' and 'DeletedItems'.
Optionally a PageSize and PageIndex can be provided, if not, the first 100 items are returned.
Returns true if the operation succeeded.

SendMessage(AOutlookMailItem: TOutlookMailItem): Boolean;
SendMessage(Subject, Body: string; Recipients; TStringList, CcRecipients: TStringList; BccRecipients: TStringList; MailType: TOutlookMailType; Attachments: TStringList): Boolean;
Send an email message. At least one Recipient email address is required, the other parameters are optional. Returns true if the operation succeeded.

Example A:

```
var
  mi: TOutlookMailItem;
begin
  mi := TOutlookMailItem.Create(nil);

  mi.Subject := edSubject.Text;
  mi.Body := meBody.Text;
  mi.RecipientEmails.Add('name@domain.com');
  mi.CcRecipientEmails.Add('name2@domain.com');
  mi.BccRecipientEmails.Add('name3@domain.com');
  mi.Attachments.Add('path\filename.ext');
  mi.MailType := mtHTML;

  AdvOutlookMail1.SendMessage(mi);

  mi.Free;

end;
```

TAdvURLShortener

Usage

TAdvURLShortener is a component that enables to make a short version of a long URL by using the Google URL Shortener API service.

Organisation

Properties:

APIKey: string;

A valid Google API Key is required to use the Google URL Shortener API service.

Methods:

ShortenURL(URL: string): string;

Returns a short version of the provided URL.

TAdvWeather

Note: This API is deprecated.

Usage

TAdvWeather is a component that uses the Wunderground.com weather service to get information on the current weather status for a location, a 4 day weather forecast or a 10 day weather forecast.

Organisation

Properties:

property Condition: TWeatherCondition;

Class property that holds various properties returning the current weather condition for a location. This class property is filled with information when GetConditions() is called.

property Location: TWeatherLocation;
Exact location information, including longitude, latitude, altitude.

property TempC: double;
Temperature in degrees Celcius

property TempF: double;
Temperature in Fahrenheit

property Weather: string;
Textual weather description

property WindDir: string;
Wind direction in text format

property WindDegrees: integer;
Wind direction in degrees

property WindMph: double;
Wind speed in miles per hour

property WindKph: double;
Wind speed in kilometer per hour

property FeelsLikeC: double;
Corrected temperature feeling in degrees Celcius

property FeelsLikeF: double;
Corrected temperature feeling in Fahrenheit

property PressureMb: double;
Air pressure in millibar

property PressureIn: double;
Air pressure in inch of Mercury

property DewPointF: double;
Dew point in Fahrenheit

property DewPointC: double;
Dew point in degrees Celcius

property VisibilityMi: double;
Visibility in Miles

property VisibilityKm: double;
Visibility in kilometer

property UV: string;
UV index

property Icon: string;
Name of the icon as text

property IconURL: string;
URL of icon representing the weather condition

property Precip1hrMetric: string;
Precipitation

property TextForeCast: TTextWeatherForeCast;

Collection of TTextWeatherForeCastItem items that contain the forecast text for the next 4 or 10 days, depending on whether GetForecast() or GetForeCast10Day() was called.

The TTextWeatherForeCastItem exposes following properties:

property Icon: string;
Name of the icon as text

property IconURL: string;
URL of the image that represents the weather condititon

property Title: string;
Title of the weather forecast.

property Text: string;
Textual weather description with numbers in non metric format.

property TextMetric: string;
Textual weather description with numbers in metric format.

property ForeCast: TWeatherForeCast;

Collection of TWeatherForeCastItem items that contain the weather conditions for the next 10 days for a specific location.
The TWeatherForeCastItem is a class that exposes following properties:

property Date: TDateTime;
Date of the forecast

property TempHighF: double;
Max temperature of the day in Fahrenheit

property TempHighC: double;
Max temperature of the day in Celcius

property TempLowF: double;
Min temperature of the day in Fahrenheit

property TempLowC: double;
Min temperature of the day in Celcius

property Conditions: string;
Textual weather condition description

property IconURL: string;
URL of icon representing the weather

property MaxWindMph: double;
Max wind speed in miles per hour

property MaxWindKmh: double;
Max wind speed in kilometer per hour

property MaxWindDir: string;
Direction where wind is coming most from

property MaxWindDegrees: integer;
Direction in degrees where wind is coming most from

property AveWindMph: double;
Average wind speed in miles per hour

property AveWindKmh: double;
Average wind speed in kilometer per hour

property AveWindDir: string;
Average wind direction

property AveWindDegrees: integer;
Average wind direction in degrees

property AveHumidity: integer;
Average humidity during the day

property MaxHumidity: integer;
Max humidity during the day

property MinHumidity: integer;
Min humidity during the day

property SnowAllDay: TWeatherSnow;
Snow expectations for all day

property SnowDay: TWeatherSnow;
Snow expectations for day

property SnowNight: TWeatherSnow;
Snow expectations for night

property QPFAllDay: TWeatherQPF;
Rain forecast for all day

property QPFDay: TWeatherQPF;
Rain forecast for day

property QPFNight: TWeatherQPF;
Rain forecast for night

Methods:

procedure GetConditions(ACountry, ACity: string);

Retrieves the weather conditions for the location and fills the TAdvWeather.Conditions property with the info retrieved.

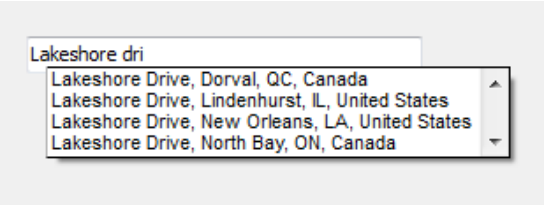
procedure GetForecast(ACountry, ACity: string);

Retrieves the weather 4 day forecast for the location and fills the TAdvWeather.Forecast collection and TAdvWeather.TextForecast collection property with the info retrieved.

procedure GetForecast10Day(ACountry, ACity: string);

Retrieves the weather 10 day forecast for the location and fills the TAdvWeather.Forecast collection and TAdvWeather.TextForecast collection property with the info retrieved.

TAdvCloudLookupEdit



Usage

Edit control with lookup dropdown showing matches while typing. Settings for the lookup are under TAdvCloudLookupEdit.Lookup. Lookup items can be statically preset or can come from a lookup provider class such as TAdvGoogleLocationLookupProvider, TAdvWeatherLookupProvider… or a custom lookup provider.

Organisation

Properties

Lookup: TLookupSettings:

property AcceptOnTab: boolean;
When true, pressing tab accepts the currently selected value in the lookup list

property Color: TColor;
Sets the background color of the lookup dropdown

property DisplayCount: Integer;
Sets the nr. of items visible in the lookup dropdown. More items are reached via scrollbar

property Enabled: Boolean;
When true, the lookup functionality is enabled

property NumChars: Integer;
Sets the number of characters that need to be typed before a first lookup is performed

property Provider: TCloudLookupProvider;
Set to an instance of a lookup provider component

TAdvWeatherLocationLookupProvider

Note: This API is deprecated.

Usage

TAdvWeatherLocationLookupProvider is a component that can perform lookup based on the available locations the Wunderground weather service provides weather data for.

Organisation

The TAdvWeatherLocationLookupProvider is either directly connected to TAdvCloudLookupEdit.Lookup.Provider or otherwise the lookup method can also be directly called:

Example:

```
TAdvWeatherLocationLookupProvider.Lookup('tms software', Listbox1.Items);
```

This fills the listbox with possible matches.

TAdvGoogleLookupProvider

Usage

TAdvGoogleLookupProvider is a component that can perform lookup on a partial string for the most frequently used Google search terms.

Organisation

The TAdvGoogleLookupProvider is either directly connected to TAdvCloudLookupEdit.Lookup.Provider or otherwise the lookup method can also be directly called:

Example:

```
TAdvGoogleLookupProvider.Lookup('tms software', Listbox1.Items);
```

This fills the listbox with possible matches.

TAdvGoogleLocationLookupProvider

Usage

TAdvGoogleLookupProvider is a component that can perform address lookup based on Google Maps address data.

Organisation

The TAdvGoogleLocationLookupProvider is either directly connected to TAdvCloudLookupEdit.Lookup.Provider or otherwise the lookup method can also be directly called:

Example:

```
TAdvGoogleLocationLookupProvider.Lookup('tms software', Listbox1.Items);
```

This fills the listbox with possible matches.

TAdvCloudImage

Usage

Image component that can automatically retrieve an image from an URL and display it on the form. It supports BMP, PNG, GIF, ICO, JPEG formats. The image can also be directly assigned from a local file.

Organisation

Properties

PicturePosition: TPicturePosition;
Sets the position of the image within the control:

bpTopLeft: position at top left in the control
bpTopRight: position at bottom right in the control
bpBottomLeft: position at bottom left in the control
bpBottomRight: position at bottom right in the control
bpCenter: position centered in the control
bpTiled: tile the image to fill the control
bpStretched: stretch the image to fill the control
bpStretchedWithAspectRatio: take aspect ratio in account for stretch to control size
bpShrink: only perform a shrink when image size is bigger than control size
bpShrinkWithAspectRatio: only perform a shrink taking aspect ratio in account when image size is bigger than control size

WebPicture: TCloudPicture;
Get or set the actual image in the control

URL: string
Sets the URL where to retrieve the image

Events:
OnDownloadCancel: TDownLoadCancelEvent;
Event triggered when download is for some reason cancelled

OnDownloadComplete: TDownLoadCompleteEvent;
Event triggered when download is complete

OnDownloadError: TDownLoadErrorEvent;
Event triggered when an error happens during the download

OnDownloadProgress: TDownLoadProgressEvent;
Event triggered during the download to track progress

TAdvCloudExifImage

Usage

Component that can update and retrieve gps latitude and longitude information from a JPEG image file through exif data.

Properties

AutoLoad: Automatically load the data when the FileName property is assigned.
AutoUpdate: Automatically update the data on the loaded file when the properties Latitude / Longitude or the Degrees Minutes and Seconds for each coordinate are set.
FileName: The filename of the jpeg file.
Latitude: The latitude of the jpeg file in decimal format, when the file does not contain gps data, the Latitude is 0.
LatitudeDegrees: The degrees of the latitude.
LatitudeMinutes: The minutes of the latitude.
LatitudeSeconds: The seconds of the latitude.
Longitude: The longitude of the jpeg file in decimal format, when the file does not contain gps data, the longitude is 0.
LongitudeDegrees: The degrees of the longitude.
LongitudeMinutes: The minutes of the longitude.
LongitudeSeconds: The seconds of the longitude.

Methods

LoadData: LoadData can be used to load the data from the file when the FileName propery is set, and the AutoLoad property is False.
UpdateData: UpdateData can be used to update the data to the file when the FileName property is set, one of the Latitude or Longitude properties are set and the AutoUpdate property is False.

There are also some built-in class methods that can be used to easily update / retrieve the gps data from a jpeg file in one call without the need for a component creation. Just add the CloudExifImage unit in the uses list and call the methods as demonstrated in a sample below:

```
var
  lat, lon: Double;
begin
  TAdvCloudExifImage.GetFileGSPInfo('C:\temp\image1.jpg', lat, lon); //retrieve
  ShowMessage(floattostr(lat) + ' : ' + floattostr(lon));
  TAdvCloudExifImage.SetFileGSPInfo('C:\temp\image1.jpg', 50.234, 3.541); //update
end;
```

TAdvPushOver

Usage

TAdvPushOver 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

TAdvPushOver descends from TCloudBase and exposes following properties and methods:

Properties

property PushOverMessage: TPushOverMessage

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

property User: string;
PushOver ID of the user.

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

property Device: string;
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.

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

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

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

property Priority: TMessagePriority;
Can be any of following values: (mpNone,mpQuiet,mpHighPriority,mpConfirmation);
mpNone is the default property 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

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

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

```
TMessageSound = (msDefault,msBike,msBugle,msCashRegister,msClassical,msCosmic,
  msFalling,msGamelan,msIncoming,msIntermission,msMagic,msMechanical,msPianoBar,
  msSiren,msSpaceAlarm,msTugBoat,msAlien,msClimb,msPersistent,msEcho,msUpDown,msNone);
```

Methods

function PushMessage(AUser, AMessage: string): boolean;

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

function PushMessage(AUser, ATitle, AMessage: string): boolean;

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

function PushMessage(AUser, ADevice, ATitle, AMessage: string): boolean;

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

function PushMessage(AMessage: TPushOverMessage): boolean;

Sends a message with all properties as defined by AMessage.

Sample code

```
begin
  // set the PushOver application ID
  AdvPushOver1.App.Key := APPID;
  // fill in the message details
  AdvPushOver1.PushOverMessage.User := edUser.Text;
  AdvPushOver1.PushOverMessage.Title := edTitle.Text;
  AdvPushOver1.PushOverMessage.Message := edMemo.Lines.Text;
  AdvPushOver1.PushOverMessage.URL := edURL.Text;
  AdvPushOver1.PushOverMessage.Device := edDevice.Text;
  AdvPushOver1.PushOverMessage.Sound :=
TMessageSound(integer(edSound.Items.Objects[edSound.ItemIndex]));
  // Send the message
  AdvPushOver1.PushMessage(AdvPushOver1.PushOverMessage);
end;
```

TAdvTwilio

Usage

TAdvTwilio is a component that sends SMS messages to a cell phone via the Twilio service.

Sample code

```
var
  smsg: TAdvTwilio;
begin
  smsg := TAdvTwilio.Create(Self);
  smsg.App.Key := twilio_accountSID;
  smsg.App.Secret := twilio_authtoken;
  smsg.App.Name := twilio_accountphonenumber;
  smsg.SendSMS('+00112345678','Send SMS to cell phone via TAdvTwilio');
  smsg.Free;
end;
```

TAdvEsendex

Usage

TAdvEsendex is a component that sends SMS messages to a cell phone via the Esendex service.

Sample code

```
var
  smsg: TAdvEsendex;
begin
  smsg := TAdvEsendex.Create(Self);
  smsg.App.Key := esendex_username;
  smsg.App.Secret := esendex_password;
  smsg.App.Name := esendex_accountID;
  smsg.SendSMS('+00112345678','Send SMS to cell phone via TAdvEsendex');
  smsg.Free;
end;
```

TAdvBulkSMS

Usage

TAdvBulkSMS is a component that sends SMS messages to a cell phone via the BulkSMS service.

Sample code

```
var
  smsg: TAdvBulkSMS;
begin
  smsg := TAdvBulkSMS.Create(Self);
  smsg.App.Key := bulksms_username;
  smsg.App.Secret := bulksms_password;
  smsg.SendSMS('+00112345678','Send SMS to cell phone via TAdvBulkSMS');
  smsg.Free;
end;
```

TAdvTelAPI

Usage

TAdvTelAPI is a component that sends SMS messages to a cell phone via the TelAPI service.

Sample code

```
var
  smsg: TAdvTelAPI;
begin
  smsg := TAdvTelAPI.Create(Self);
  smsg.App.Key := telapi_accountSID;
  smsg.App.Secret := telapi_authtoken;
  smsg.SendSMS('+00112345678','Send SMS to cell phone via TAdvTelAPI');
  smsg.Free;
end;
```

TAdvCloudConvert

Usage

TAdvCloudConvert is a component that can convert one file format to another file format. The input file is uploaded to the API service and after the conversion the output file is downloaded. The available conversion types can be consulted at: https://cloudconvert.com/formats

Methods

function ConvertFile(FileName: string; InputFormat: string; OutputFormat: string; Converter: string = ''; Options: TStringList = nil): string;

Method to convert the file with FileName from InputFormat to OutputFormat. Optionally the Converter value can be added to select which converter to use if multiple converters are available for the desired conversion. Optionally Options can be defined for the conversion. The result is the the URL of the converted file. Details of the conversion are returned in the ConvertResults property.

function ConvertAndDownload(InputFileName: string = ''; OutputFileName: string = ''; InputFormat: string = ''; OutputFormat: string = ''; Converter: string = ''; Options: TStringList = nil): boolean;

Method to convert the InputFileName to OutputFileName. If no InputFormat and/or OutputFormat is provided, the file format is automatically determined based on the filename extension(s). Optionally the Converter value can be added to select which converter to use if multiple converters are available for the desired conversion. Optionally Options can be defined for the conversion. Returns true if the conversion succeeded, false otherwise. Details of the conversion are returned in the ConvertResults property.

procedure Download(Url, TargetFile: String);

Method to download a converted file. The result of the ConvertFile method or the ConvertResults.OutputFile.Url can be used as the Url parameter.

Sample code

1) **Convert a DOC file to a PDF file**

```
AdvCloudConvert1.ConvertAndDownload('test.doc', 'test.pdf');
```

2) **Convert the first 5 pages of a PDF file to PNG files**
   Note: Converting a multi-page file (such as a PDF file) to an image format will result in a separate image file for each converted page. The images are returned in a single ZIP file.

```
  options := TStringList.Create;
  options.Add('page_range=1-5');
  AdvCloudConvert1.ConvertAndDownload('test.pdf', 'test.zip', '', 'png',
'', options);
  options.Free;
```

Conversion details returned by TAdvCloudConvert are:

AdvCloudConvert.ConvertResults.ID: The conversion ID
AdvCloudConvert.ConvertResults.StarTime: The timestamp when the conversion started
AdvCloudConvert.ConvertResults.EndTime: The timestamp when the conversion was finished
AdvCloudConvert.ConvertResults.ExpiryTime: The timestamp when the converted file download expires
AdvCloudConvert.ConvertResults.InputFile: The input file details
AdvCloudConvert.ConvertResults.OutputFile: The output file details
AdvCloudConvert.ConvertResults.OutputFile.FileName: The output file name
AdvCloudConvert.ConvertResults.OutputFile.FileSize: The output file size
AdvCloudConvert.ConvertResults.OutputFile.Url: The output file url
AdvCloudConvert.ConvertResults.OutputFile.Files: The list of files contained in the output file (only if the output file is a ZIP file)

TAdvBarcode

Usage

TAdvBarcode is a component that can generate barcodes and QR codes. The barcode or QR code is provided to the service and an URL to an image with the barcode or QR code is returned or saved as an image file.

Properties

BarcodeOptions: Configure the barcode settings.

Height: integer; The height of the barcode
ReverseColor: Indicates if the barcode is displayed as white on black instead of black on white
ShowBorder: Boolean; Indicates if a border is displayed around the barcode
ShowText: Indicates if the text value of the barcode is displayed beneath the barcode
Width: Boolean; The width of the barcode

ImageType: TBarcodeImageType; Set the image format that is used when a barcode is generated. Supported image formats are GIF, JPG and PNG.

QRCodeOptions: Configure the QR code options
ECCLevel: TQRCodeECC; Set the Error Correction Capibility Level of the QR code
Size: TQRCodeSize; Set the size of the QR code

Methods

function GetBarcode(AValue: string; AFileName: string; AType: TBarcodeType = btC39): string;

Method to generate a barcode based on AValue. Optionally the TBarcodeType can be set with AType. The supported barcode types are: 'Code 39', 'Code 128a', 'Code 128b', 'Code128c' and '2 of 5 interleaved'. The result is saved as an image file with AFileName as filename. The image format can be set with the ImageType property and the available options can be configured in BarcodeOptions.

function GetQRcode(AValue: string; AFileName: string): string;

Method to generate a QR code based on AValue. The result is saved as an image file with AFileName as filename. The image format can be set with the ImageType property and the available options can be configured in QRcodeOptions.

function GetBarcodeURL(AValue: string; AType: TBarcodeType = btC39): string;

Method to generate a barcode based on AValue. Optionally the TBarcodeType can be set with AType. The supported barcode types are: 'Code 39', 'Code 128a', 'Code 128b', 'Code128c' and '2 of 5 interleaved'. The result is an URL to an image file with the barcode. The image format can be set with the ImageType property and the available options can be configured in BarcodeOptions.

function GetQRcodeURL(AValue: string): string;

Method to generate a QR code based on AValue. The result is an URL to an image file with the barcode. The image format can be set with the ImageType property and the available options can be configured in QRcodeOptions.

Sample code

1) **Display a barcode with a TAdvCloudImage that contains the value '12345'**

```
AdvCloudImage1.URL := AdvBarcode1.GetBarcodeURL('12345');
```

2) **Generate an image file with a QR code with a width and height of 210 pixels that contains the value 'http://www.tmssoftware.com'**

```
AdvBarcode1.QRcodeOptions.Size := qs210;
AdvBarcode1.GetQRCode('http://www.tmssoftware.com', 'QRcode.png');
```

TAdvIPLocation

Usage

TAdvIPLocation is a component that can determine the geolocation of an IP address. It can determine the geolocation of the machine where it is executed, the geolocation of any IP address or the geolocation of a server based on its domain name.

Note: A valid API Key from www.ipstack.com is required for this function.
The API Key must be assigned to the APIKey property

Sample code

```
  if AdvIPLocation1.GetIPLocation then
    Showmessage('I am located in country : ' +
AdvIPLocation1.IPInfo.CountryName);

  if AdvIPLocation1.GetIPLocation('107.128.206.99') then
    Showmessage('IP address is located in country : ' +
AdvIPLocation1.IPInfo.CountryName);

  if AdvIPLocation1.GetIPLocationFromServer('www.google.com') then
    Showmessage('Google.com is located in country : ' +
AdvIPLocation1.IPInfo.CountryName);
```

Details returned by TAdvIPLocation are:

AdvIPLocation.IPInfo.ContinentCode: code of the continent. Example: Europe = EU
AdvIPLocation.IPInfo.ContinentName: name of the continent
AdvIPLocation.IPInfo.CountryCode: code of the country. Example: Brasil = BR
AdvIPLocation.IPInfo.CountryName: name of the country
AdvIPLocation.IPInfo.RegionCode: code of the region
AdvIPLocation.IPInfo.RegionName: name of the region
AdvIPLocation.IPInfo.ZIPCode: ZIP code
AdvIPLocation.IPInfo.City: name of the city
AdvIPLocation.IPInfo.Metrocode: code of the nearby metro
AdvIPLocation.IPInfo.Areacode: code of the nearby area

AdvIPLocation.IPInfo.Longitude: longitude coordinate of the geolocation
AdvIPLocation.IPInfo.Latitude: latitude coordinate of the geolocation

TCloudDataSet

Usage

TCloudDataSet is a TDataSet descending component that offers seamless TDataSet based data binding for cloud based data storage. It can be used with the myCloudData, Google DataStore or Apple's CloudKit.

Just like any other VCL TDataset, it can be connected via a TDataSource to DB-aware controls for seamless data operations.

Organisation

To start using a TCloudDataSet, it needs to be connected to a TCloudDataStore instance via a TCloudDataStoreAdapter. For access to myCloudData the TAdvmyCloudData is provided, for access to the Google DataStore, the TAdvGDataStore component is provided and to connect to the CloudKit data service, the TAdvCloudKit component is provided.

Setup

Drop a TAdvmyCloudData and/or TAdvGDataStore and/or TAdvCloudKit component on the form. Drop a TCloudDataStoreAdapter on the form as well as a TCloudDataSet. Connect the TCloudDataStoreAdapter to TCloudDataSet.Adapter and connect the TAdvmyCloudData, TAdvGDataStore or TAdvCloudKit to the TCloudDataStoreAdapter.DataStore property.
To connect the TCloudDataSet to a DB-aware control, drop a TDataSource on the form and connect the TCloudDataSet to the TDataSource via TDataSource.DataSet property.

Configure the TAdvmyCloudData.App settings with the myCloudData settings or TAdvGDataStore.App settings with the Google application settings or TAdvCloudKit.App settings with the Apple CloudKit application settings.

Start to connect by performing an authentication and authorization for myCloudData, Google DataStore or Apple CloudKit by calling TAdvmyCloudData.DoAuth, TAdvGDataStore.DoAuth or TAdvCloudKit.DoAuth. In case a previously obtained access token was persisted, it can be loaded with TAdvmyCloudData.LoadTokens, TAdvGDataStore.LoadTokens or TAdvCloudKit.LoadTokens and the service can be accessed right-away without going through an authentication and authorization. With this setup and after having successfully obtained an access token, setting the TCloudDataSet.Active = true is sufficient to make the dataset connect live to the cloud datastore service. As long as the TCloudDataSet.Active = true, CRUD operations on the dataset will be executed live on the cloud datastore service.

Database schema

The TAdvmyCloudData and TAdvGDataStorec components are able to automatically retrieve the schema of a table from the myCloudData service or Google data store service respectively. The schema can be read back after connection via the MetaData collection.

The Apple CloudKit service doesn't support this automatic retrieval at this time and as such, the schema must be programmatically setup before activating the dataset. The schema or metadata is setup at TAdvCloudKit level (and is then automatically retrieved via the TAdvCloudDataStoreAdapter in the connected TCloudDataSet). To setup this metadata, add items to the TAdvCloudKit.MetaData collection. This example source code demonstrates a simple setup (that should match the setup of an Apple CloudKit table):

```
var
  mi: TDataStoreMetaDataItem;

begin
  AdvCloudKit1.MetaData.Clear;

  mi := AdvCloudKit1.MetaData.Add;
  mi.PropertyName := 'Name';
  mi.DataType := ftWideString;

  mi := AdvCloudKit1.MetaData.Add;
  mi.PropertyName := 'Children';
  mi.DataType := ftInteger;

  mi := AdvCloudKit1.MetaData.Add;
  mi.PropertyName := 'Salary';
  mi.DataType := ftFloat;

  mi := AdvCloudKit1.MetaData.Add;
  mi.PropertyName := 'BirthDate';
  mi.DataType := ftDateTime;
end;
```

Supported Field Types

TAdvmyCloudData:

- ftWideString
- ftInteger
- ftFloat
- ftBoolean
- ftDateTime
- ftDate
- ftTime
- ftBlob (Requires a subscription)

TAdvGDataStore:

- ftWideString
- ftInteger
- ftFloat
- ftBoolean
- ftDateTime (This field type is not automatically discovered via the Google DataStore service, it is required that the MeataDataItem.DataType property is set via the OnMetaDataRetrieved event)

TAdvCloudKit

- ftWideString
- ftInteger
- ftFloat
- ftDateTime

Properties of TCloudDataSet

property Active: Boolean
Set Active to true to open the dataset

property Adapter: TCloudDataStoreAdapter
Sets the datastore adapter to connect to the datastore component for the dataset

property ReadOnly: Boolean
Set the dataset read-only or not

Properties of TCloudDataStoreAdapter

property CloudDataStore: TCloudDataStore
Sets the datastore component to connect a TCloudDataSet with

Events of TCloudDataStoreAdapter

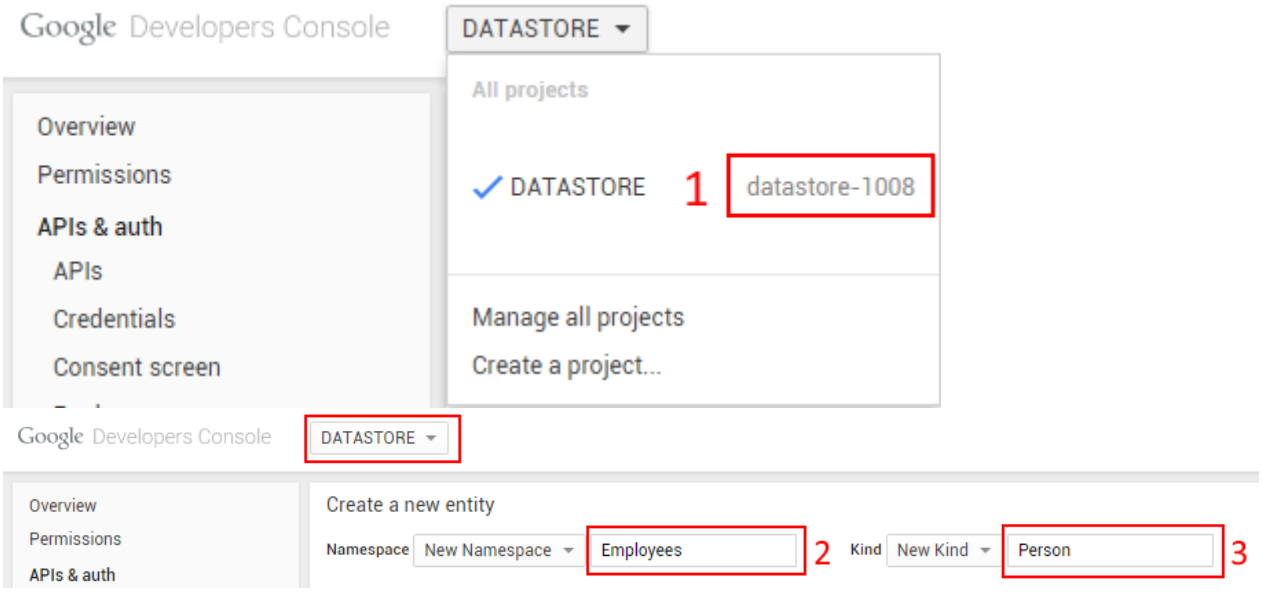
property OnMetaDataRetrieved(Sender: TObject; MetaDataItem: TDataStoreMetaDataItem; var Allow: Boolean);
Event triggered for each TDataStoreMetaDataItem in the TAdvmyCloudData/TAdvGDataStore/TAdvCloudKit assigned to CloudDataStore. Set Allow to false if a datafield should not be added to the dataset.

Note: (TAdvGDataStore only) If the table contains a DateTime field, it is required to set the DataField property of the MetaDataItem to ftDateTime.

Example:

```
procedure TForm1.CloudDataStoreAdapter1MetaDataRetrieved(Sender: TObject;
MetaDataItem: TDataStoreMetaDataItem; var Allow: Boolean);
begin
  if MetaDataItem.PropertyName = 'BirthDate' then
    MetaDataItem.DataType := ftDateTime;
end;
```

Properties of TAdvGDataStore



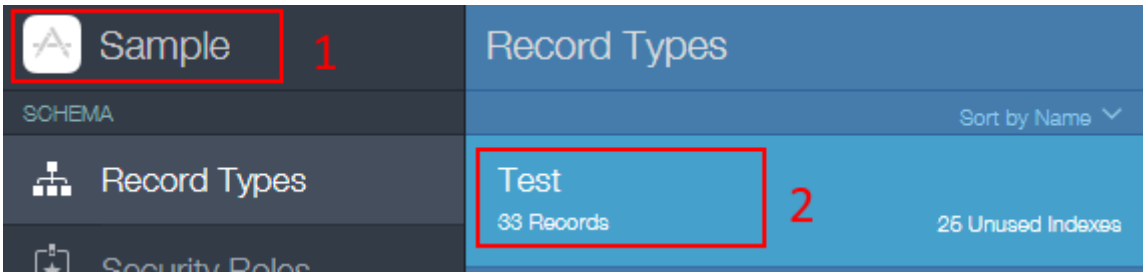
property Database: string
Sets the database name to be used from the Google DataStore service. See (2) in screenshot above

property MetaData: TDataStoreMetaData
Collection of metadata for the connected table

property Project: string
Sets the Project name as defined at the Google DataStore service. See (1) in screenshot above

property Table: string
Sets the table name to be used from the Google DataStore service. See (3) in screenshot above

Properties of TAdvCloudKit



property Access: TCloudKitAccess (caPublic/caPrivate)
Determines the access type for the CloudKit service

property Database: string
Sets the database name to be used from the CloudKit service. See (2) in screenshot above

property Environment: TCloudKitEnvironment (ceDevelopment/ceProduction)
Determines the environment type for the CloudKit service

property MetaData: TDataStoreMetaData
Collection of metadata for the connected table

property Project: string
Sets the Project name as defined at the CloudKit service. See (1) in screenshot above

Properties of TAdvmyCloudData

See the TAdvmyCloudData topic below for detailed information.

TAdvmyCloudData

Usage

TAdvmyCloudData is a component that provides seamless access to the myCloudData.net service that allows to create tables with meta data of choice to store data in the cloud. A user can have access to one or more tables. After login, the collection of tables available to the user is returned with TAdvmyCloudData.GetTables and accessible via TAdvmyCloudData.Tables. A table on myCloudData.net is represented by the class TMyCloudDataTable. A table has metadata, entities, filters, sort order and shares. The table entities are represented by the class TMyCloudDataEntity. The shares are represented by the class TMyCloudDataShare. The meta data for a table is retrieved via Table.GetMetaData. The entities are retrieved via the Table.Query function and the list of shares is retrieved with Table.Shares. The filter is accessible via the collection Table.Filters and the sort ordering can be setup via the collection Table.SortOrder.

Organisation

TMyCloudDataEntity

The TMyCloudDataEntity class is the class that wraps an entity (in database terminology also often referred to as record). The TMyCloudDataEntity class has following methods & properties:

Properties

property Value[AName: string]: Variant;
property Blob[AName: string]: TMyCloudDataBlob;

Methods

procedure Update;
procedure Insert;
procedure Delete;

To get or set a value for a field within the entity, you can use Value[AName: string]: Variant.

Example:

```
entity.Value['NAME'] := 'Bill Gates;
entity.Value['NETWORTH'] := 46546114646;
entity.Value['BIRTHDATE'] := DatePicker.Date;
```

All supported field types: int/int64/double/string/date/datetime/time/boolean can be get or set this way. The binary blob fields can be accessed as:

```
blob := entity.Blob['BIN'];
```

With blob an instance of the class TMyCloudDataBlob. This explained further in the paragraph covering this class.

TMyCloudDataEntities

This is the collection of entities retrieved via Table.Query.
Typical operations on entities are as such:

1. Create a new entity in the cloud storage:

```
var
  ent: TMyCloudDataEntity;
begin
  ent := Table.Entities.Add;
  ent.Value['NAME'] := 'Elon Musk';
  ent.Value['STATE'] := 'California';
  ent.Value['COMPANY'] := 'Tesla';
  ent.Insert;
end;
```

2. Update an existing entity in the cloud storage:

```
var
  ent: TMyCloudDataEntity;
begin
  ent := Table.Entities[x];
  ent.Value['COMPANY'] := 'SpaceX';
  ent.Update;
end;
```

3. Delete an entity permanently from the cloud storage:

```
var
  ent: TMyCloudDataEntity;
begin
  ent := Table.Entities[y];
  ent.Delete;
end;
```

TMyCloudDataBlob

The TMyCloudDataBlob class is a wrapper class for binary data stored in a blob in the cloud service. Note that the blob storage capability is not available for a free account in myCloudData.net but requires a subscription.
The TMyCloudDataBlob class has following properties and methods:

Properties

property Table: TMyCloudDataTable
Table to which the blob belongs

property Entity: TMyCloudDataEntity
Entity to which the blob belongs

property Field: string
Name of the field holding the blob

Methods

function LoadFromFile(AFileName: string): boolean;
Load data from AFileName into the blob field. Returns true if the upload was successful, false otherwise.

function SaveToFile(AFileName: string): boolean;
Get data from the blob field and save it to a file. Returns true if the download was successful, false otherwise.

function LoadFromStream(AStream: TStream): boolean;
Load data from the stream into the blob field. Returns true if the upload was successful, false otherwise.

function SaveToStream(AStream: TStream): boolean;
Get data from the blob field and save it to a stream. Returns true if the download was successful, false otherwise.

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

Example:

```
var
  ent: TMyCloudDataEntity;
begin
  blob: TMyCloudDataBlob;

  ent := Table.Entities.Add;
  blob := ent.Blob['BIN'];
```

```
  blob.LoadFromFile('mybinfile.bin');
end;
```

Note that for performance reasons, blobs are returned via the entity only and retrieved from the cloud storage at the time SaveToStream() or SaveToFile() is executed.

TMyCloudDataLookupFieldValue

The TMyCloudDataLookupFieldValue class contains the key and lookup values associated with a LookupField.

Properties

property KeyValue: Variant;
The key value of the lookupdata

property LookupValue: Variant;
The lookup value of the lookupdata

property Tag: integer;
Defines a custom related value

TMyCloudDataLookupFieldValues

This is the collection of lookup field values from a LookupField.

Properties

property LookupValues[AKey: Variant]: Variant;

TMyCloudDataLookupField

The TMyCloudDataLookupField class contains the lookupdata for a LookupField.

Properties

property LookupField: string;
The name of the field in the parent table that the lookup data is associated with

property TableID: Int64;
The ID of the table that contains the lookup data

property LookupFieldValues: TMyCloudDataLookupFieldValues
The list of key and lookup values

TMyCloudDataLookupFieldList

This is the collection of lookup fields retrieved via Table.GetLookupData.

Properties

property Field[AFieldName: string]: TMyCloudDataLookupFieldValues;
property List[AFieldName: string]: TStringList;

TMyCloudDataTable

This class represents the table in the cloud storage and is part of the set of tables in the collection TAdvMyCloudData.Tables retrieved with TAdvMyCloudData.GetTables.

A TAdvMyCloudData class has following properties and methods:

Properties

property ID: int64;
Read-only property returning the unique identifier of the table

property OwnerID: int64;
Read-only property returning the unique owner identifier of the table

property IsOwner: boolean;
Read-only property returns true when the logged in user owns the table

property Name: string;
Gets or sets the name of the table

property Permissions: TMyCloudDataPermissions read FPermissions;
Permissions the user has on the table. Permissions are: CRUD, i.e. create/read/update/delete

property MetaData: TMyCloudDataMetaData;
Access to the metadata of the table via a collection after calling GetMetaData

property Entities: TMyCloudDataEntities;
Access to the entities of the table via a collection after calling Query

property Filters: TMyCloudDataFilters;
Access to the filter conditions for a query via a collection

property SortOrder: TMyCloudDataSortOrderList;
Access to the sort order settings for a query via a collection

property LookupFieldList: TMyCloudDataLookupFieldList;
Access to the list of lookup fields via a collection after calling GetLookupData.

Methods

function GetMetaData: boolean;
Retrieves the metadata from a table, stored in MetaData

function SetMetaData: boolean;
Updates the metadata of the table on the cloud storage with Table.MetaData

function Query: boolean; overload;
Simply query for all entities of the table, stored in Entities

function Query(Fields: TStringArray): boolean; overload;
Query with specifier of selection of fields to return, stored in Entities

function Query(AFields: TStringList): boolean; overload;
Query with specifier of selection of fields to return, stored in Entities

procedure Share(Email: string; Permissions: TMyCloudDataPermissions);
Share a table with another user defined by email

procedure RemoveShare(Email: string);
Remove an existing share with another user via email

procedure Delete;
Delete the table from the cloud storage

function GetShares: TMyCloudDataShares;
Retrieve the list of email addresses & permissions with who the table was shared

function GetLookupData: Boolean;
Retrieves the lookup data from a table, stored in LookupFieldList. It is required that the Table metadata contains valid values for the LookupTable, LookupField and LookupKeyField properties in order to be able to retrieve lookup data.

To perform a simple query, use:

```
Table.Query;
```

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

Table.Query(['NAME','COMPANY','BIRTHDATE'];

This will fill the Entities collection but the entities will only hold the fields NAME, COMPANY, BIRTHDATE

To filter data, following code can be used:

```
var
  filter: TMyCloudDataFilter;
begin
  Table.Filters.Clear; // removes all filter conditions
  filter := Table.Filters.Add('NAME', coLike, 'Musk', loNone);
  Table.Query;
end;
```

Or alternatively:

```
var
  filter: TMyCloudDataFilter;
begin
  Table.Filters.Clear; // removes all filter conditions
  filter := Table.Filters.Add;
  filter.FieldName := 'NAME';
  filter.ComparisonOperator := coLike;
  filter.Value := 'Musk';
  filter.LogicalOperator := loNone;
  Table.Query;
end;
```

Note that the ComparisonOperator can be any of the following values:
coEqual, coNotEqual, coLike, coGreater, coGreaterOrEqual, coLess, coLessOrEqual, coStartsWith, coEndsWith, coNull, coNotNull;

The LogicalOperator that sets the logical operation between two sequential filter conditions can be: loAND, loOR, loNone

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

```
  Table.SortOrder.Clear;
  Table.SortOrder.Add('NAME', soAscending);
  Table.SortOrder.Add('COMPANY', soDescending);
  Table.Query;
```

Or alternatively:

```
var
  sortorder: TMyCloudDataSortOrderItem;
begin
  Table.SortOrder.Clear;
  sortorder := Table.SortOrder.Add;
  sortorder.FieldName := 'NAME';
  sortorder.SortOrder := soAscending;
  sortorder := Table.SortOrder.Add;
  sortorder.FieldName := 'COMPANY';
  sortorder.SortOrder := soDescending;
```

```
  Table.Query;
end;
```

TMyCloudDataTables

This is the collection of all the tables a user has access to, either because the user owns the table or the table was shared with the user.

TMyCloudDataMetaDataItem

This class holds the information about a single meta data item in the meta data collection of a table. The meta data item class has following properties:

Properties

General Properties
These properties define the behavior of the MetaDataItem on the myCloudData server as well as in a client application.

property PropertyName: string;
Gets or sets the field name

property DataType: TFieldType
Gets or sets the field type. The field type can be any of following value: ftString, ftWideString, ftInt, ftBigInt, ftFloat, ftBlob, ftSmallInt, ftWord, ftBoolean, ftDate, ftDateTime, ftTime

property Data: Boolean
Returns true when the meta data item pertains actual data

property Size: integer
Optionally gets or sets the size of a field (only available for fields of type ftString, ftWideString)

Client Properties
These properties can define the appearance of the MetaDataItem and how it behaves in a client application. These properties have no influence on the behavior of the MetaDataItem on the myCloudData server.

property LabelText: string;
Gets or sets the label text associated with the field

property DefaultValue: string;
Gets or sets the default value associated with the field

property Width: integer;
Gets or sets the width associated with the field

property Order: integer;
Gets or sets the order index associated with the field (relative to the other fields in the table)

property Mask: string;
Gets or sets the field content mask

property Minimum: double;
Gets or sets the minimum allowed value associated with the field

property Maximum: double;
Gets or sets the maximum allowed value associated with the field

property MinimumDate: tdatetime;
Gets or sets the minimum allowed date and/or time associated with the field

property MaximumDate: tdatetime;
Gets or sets the maximum allowed date and/or time associated with the field

property Visible: Boolean;
Gets or sets the visibility associated with the field. Default is true

property Description: string;
Gets or sets the description value associated with the field

property Enabled: Boolean;
Gets or sets the enabled status associated with the field. Default is true

property Required: Boolean;
Gets or sets the required status associated with the field. Default is false

property LookupTable: int64;
Gets or sets the ID of the lookup table associated with the field

property LookupField: string;
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.

property LookupKeyField: string;
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.

The meta data item is part of the meta data collection TMyCloudDataMetaData accessible via Table.MetaData.

Typical operations on the meta data are:

1. Retrieval of meta data & list all fields in a listbox

```
var
   i: integer;
begin
   Table.GetMetaData;

   for i := 0 to Table.MetaData.Count - 1 do
   begin
     listbox.Items.Add(Table.MetaData[i].PropertyName);
   end;
end;
```

2. Creating meta data for a new table

```
   Table.MetaData.Clear;
   Table.MetaData.Add('NAME', ftWideString, 50);
   Table.MetaData.Add('COMPANY', ftWideString, 50);
   Table.MetaData.Add('BIN', ftBlob);
   Table.SetMetaData;
```

Or alternatively

```
var
  metadata: TMyCloudDataMetaDataItem;
begin
   Table.MetaData.Clear;
   metadata := Table.MetaData.Add;
   metadata.PropertyName := 'NAME';
   metadata.DataType := ftWideString;
   metadata.Size := 50;
   metadata := Table.MetaData.Add;
   metadata.PropertyName := 'COMPANY';
   metadata.DataType := ftWideString;
   metadata.Size := 50;
   metadata := Table.MetaData.Add;
   metadata.PropertyName := 'BIN';
   metadata.DataType := ftBlob;
   Table.SetMetaData;
end;
```

TMyCloudDataShares

To share a table with another myCloudData.net user, call:

```
Table.Share('myfriend@company.com', [pCreate, pRead, pUpdate]);
```

This adds a share with user myfriend@company.com. Note that it is required that myfriend@company.com is recognized as a valid myCloudData.net user. The myCloudData.net will not send a notification of the share itself. It is the responsibility of the user to do so. Here a share is created with all permissions except the permission to delete entities in the table.

To remove the share at a later time, call:

```
Table.RemoveShare('myfriend@company.com');
```

When the share existed for the user it will be removed.

To see with who a table is shared, use:

```
var
  shares: TMyCloudDataShares;
begin
  shares := Table.GetShares;

  for i := 0 to shares.Count - 1 do
  begin
    listbox.Items.Add(shares[i].Email);
  end;
end;
```

TAdvMyCloudData

TAdvMyCloudData is the class that wraps the entire access to the myCloudData.net service. Its interface is compatible with the Apple CloudKit and Google DataStore components, so usage of myCloudData.net, CloudKit or Google DataStore can be fully abstracted. Note that in addition to the common interface, myCloudData.net introduces many unique capabilities only available in myCloudData.net.

Methods & properties available in TAdvMyCloudData:

Properties

property TableId: int64
Gets or sets the unique ID of the table TAdvMyCloudData can work on

Methods

Table related methods:

function GetTables: boolean;
Retrieves the list of tables and makes these accessible via TAdvMyCloudData.Tables

function TableByName(AName: string): TMyCloudDataTable;
Retrieves on table based on its name

function TableList: TStrings;
Retrieves the list of available tables as stringlist

function AddTable(ATable: TMyCloudDataTable): boolean;
Creates a new table from an existing TMyCloudDataTable class

function CreateTable(ATableName: string): TMyCloudDataTable;
Creates a new table with name ATableName and returns an instance to the table

function DeleteTable(AID: int64): boolean; overload;
Deletes a table based on its unique ID

function DeleteTable(ATable: TMyCloudDataTable): boolean; overload;
Deletes a table based on an existing TMyCloudDataTable class

function UpdateTable(ATable: TMyCloudDataTable): boolean;
Updates table info, such as name, permissions based on an existing TMyCloudDataTable class

function ShareTable(ATable: TMyCloudDataTable; AEmail: string; APermissions: string): boolean;
Share a table with specific permissions with another myCloudData.net user

function GetTableShares(ATable: TMyCloudDataTable): boolean;
Fills the Table.Shares collection with shares found

Metadata related methods:

function GetMetaData: boolean;
Retrieves the metadata for the table specified by TAdvMyCloudData.TableId

function AddMetaData(AMetaData: TMyCloudDataMetaDataItem): boolean;
Sets the metadata for table specified by TAdvMyCloudData.TableId

function UpdateMetaData(AOldFieldName, ANewFieldName: string; ADataType: TFieldType = ftUnknown; ASize: integer = -1): boolean;
Modifies the meta data of a single field (specified by ANewFieldName and limited to to the field name, field datatype and field size) for a table specified by TAdvMyCloudData.TableId

function UpdateMetaData(AMetaData: TMyCloudDataMetaDataitem): boolean;
Modifies all the meta data of a single field (specified by AMetaData) for a table specified by TAdvMyCloudData.TableId.

function DeleteMetaData(APropertyName: string): boolean;
Delete a field from the meta data for a table specified by TAdvMyCloudData.TableId

Entity related methods:

function Insert(AValues: TStringList): TDatastoreEntity; override;
Inserts entity values via a stringlist

function Query: boolean; overload; override;
Retrieves entities for a table specified by TAdvMyCloudData.TableId

function Query(AFields: TStringList): boolean;
Retrieves entities with fields limitied to the specified list for a table specified by TAdvMyCloudData.TableId

function Query(AFields: TStringList; AFilters: TMyCloudDataFilters): boolean;
Retrieves entities with filter conditions

function Query(AFields: TStringList; ASortOrder: TMyCloudDataSortOrderList): boolean;
Retrieves entities with sort order specified

function Query(AFields: TStringList; AFilters: TMyCloudDataFilters; ASortOrder: TMyCloudDataSortOrderList): boolean; overload;
Retrieves entities with filter conditions and sort order specified

function Delete(AID: string): boolean; override;
Delete an entity with ID from a table specified by TAdvMyCloudData.TableId

function Delete(AIDList: TStringList): boolean; override;
Delete multiple entities from a table specified by TAdvMyCloudData.TableId. Only entities with an ID specified in AIDList will be deleted.

function DeleteAll: boolean;
Delete all entities from a table specified by TAdvMyCloudData.TableId

function Update(AEntity: TDataStoreEntity): boolean; override;
Update the entity in a table specified by TAdvMyCloudData.TableId

function Update (AIDList, AFieldValues: TStringList): boolean; override;
Update multiple entities in a table specified by TAdvMyCloudData.TableId. Only entities with an ID specified in AIDList will be updated. Only fields specified in AFieldValues will be updated.

Example:

```
var
  slIDs, slFields: TStringList;
begin
  slIDs := TStringList.Create;
  slIDs.Add('1');
  slIDs.Add('2');

  slFields := TStringList.Create;
  slFields.CommaText := 'FieldName=FieldValue';

  AdvMyCloudData1.Update(slIDs, slFields);
```

```
  slIDs.Free;
  slFields.Free;
```

function UpdateAll(AFieldValues: TStringList): boolean;
Update all entities in a table specified by TAdvMyCloudData.TableId. Only fields specified in AFieldValues will be updated.

function Download(ATableID, AEntityId: Int64; AFieldName: string; const TargetFile: string): boolean;
Download a blob to a file value from a specific entity in a specific table and fieldname

function Download(ATableID, AEntityId: Int64; AFieldName: string; AStream: TStream): boolean;
Download a blob value in a stream from a specific entity in a specific table and fieldname

function Upload(ATableID, AEntityId: Int64; AFieldName: string; FileName: string): boolean;
Upload a file to a blob field from a specific entity in a specific table and fieldname

function Upload(ATableID, AEntityId: Int64; AFieldName: string; AStream: TStream): boolean;
Upload a stream to a blob field from a specific entity in a specific table and fieldname

Users related methods:

function GetUser: boolean;
Gets information about the currently authenticated user. Fills the User properties.

function GetUsers: boolean;
Get list of users, for non-admin users, this retrieves the logged in user. Fills the Users collection property

function AddUser(AUser: TMyCloudDataUser): boolean;
Available for admin level users only

function DeleteUser(AID: int64): boolean;
Available for admin level users only

Additional properties:

property User: TMyCloudDataUser;
Contains information about the currently authenticated user.

property Users: TMyCloudDataUsers;
Collection of users, filled by the GetUsers method

property Tables: TMyCloudDataTables;
Collection of users, filled by the GetTables method

property PageIndex: integer;
When > 0, this specifies the page of entities to retrieve for the Query() methods

property PageSize: integer;
When > 0, specifies the maximum number of entities to return for the Query() methods.
The last page is retrieved when number of entities is smaller than PageSize.

TAdvmyCloudDataConnection

TAdvmyCloudDataConnection is a non-visual component that is basically the component that can create a connection (authentication & authorization) with the cloud data myCloudData and keeps the access token for this connection and can share this access token with TAdvmyCloudDataDataSet datasets. The application key and secret of your myCloudData account can be set via TAdvMyCloudDataConnection.App and when settings Connected = true, TAdvmyCloudDataConnection will test its current access token (if there is one) or start a new authentication/authorization sequence. Note that TAdvmyCloudDataConnection can do this both at design-time and run-time.
The obtained access token by TAdvmyCloudDataConnection is then shared with instances of TAdvmyCloudDataDataSet via setting TAdvmyCloudDataDataSet.Connection to TAdvmyCloudDataConnection.

Typical setup scenario:

```
// have the connection persist its token in a .INI file

  AdvMyCloudDataConnection1.PersistTokens.Location := plIniFile;
  AdvMyCloudDataConnection1.PersistTokens.Key :=
'.\mycloudddatadataset.ini';
  AdvMyCloudDataConnection1.PersistTokens.Section := 'tokens';

// set myCloudData key & secret

  AdvMyCloudDataConnection1.App.Key := MYCLOUDDATAKEY;
  AdvMyCloudDataConnection1.App.Secret := MYCLOUDDATASECRET;
  AdvMyCloudDataConnection1.App.CallBackPort := 8888;
  AdvMyCloudDataConnection1.App.CallBackURL := 'http://127.0.0.1:8888';

// have the AdvMyCloudDataSet use the access token of the connection

  AdvMyCloudDataSet1.Connection := AdvMyCloudDataConnection1;

// test existing access token and when invalid or not existent, do a new
// authentication/authorization to obtain a new one

  AdvMyCloudDataConnection1.Connected := true;
```

When a connection is successful, the AdvmyCloudDataConnection.OnConnect event is triggered and from there the dataset can be set active for example.

TAdvmyCloudDataDataSet

TAdvmyCloudDataDataSet is a dataset wrapper for the myCloudData service data. It can either directly connect to the myCloudData service or it can use an existing connection of a TAdvmyCloudDataConnection instance. To do so, assign a TAdvmyCloudDataConnection to TAdvmyCloudDataDataSet.Connection.

To have TAdvmyCloudDataDataSet connect to the cloud data, set TAdvmyCloudDataDataSet.Active = true.

The table to use is either defined by TAdvmyCloudDataDataSet.Table (i.e. the name of the table) or TAdvmyCloudDataDataSet.TableID (i.e. the unique ID of the table).

An additional feature of TAdvmyCloudDataDataSet is the capability to define a filter. This is done via the TAdvmyCloudDataDataSet.Filter property. This is a collection of filter conditions. It is easy to add one or multiple filters via this filter both at run-time and design-time. The filter condition exists of:

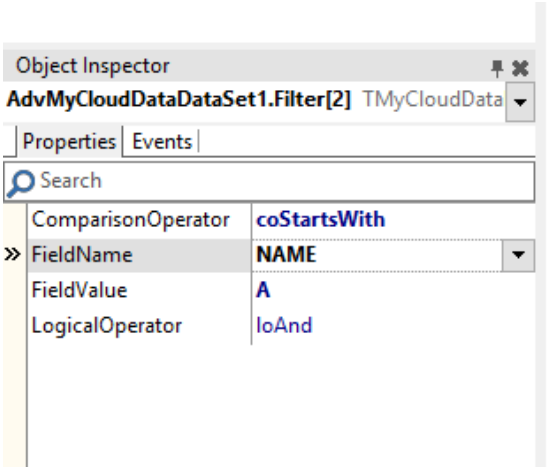
TMyCloudDataFilter

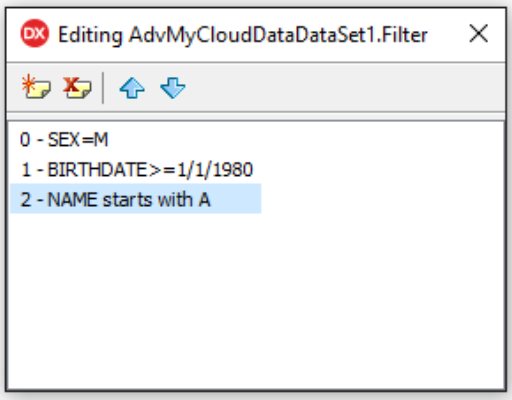
ComparisonOperator: TComparisonOperator = (coEqual, coNotEqual, coLike, coGreater, coGreaterOrEqual, coLess, coLessOrEqual, coStartsWith, coEndsWith, coNull, coNotNull);

FieldName: string
FieldValue: TValue
LogicalOperator: TLogicalOperator = (loAnd, loOr, loNone);

Note that the filter field value to set is of the type TValue. This allows to directly use different types to assign filter values for different field types.

Visually, at design time this is presented in the IDE as:





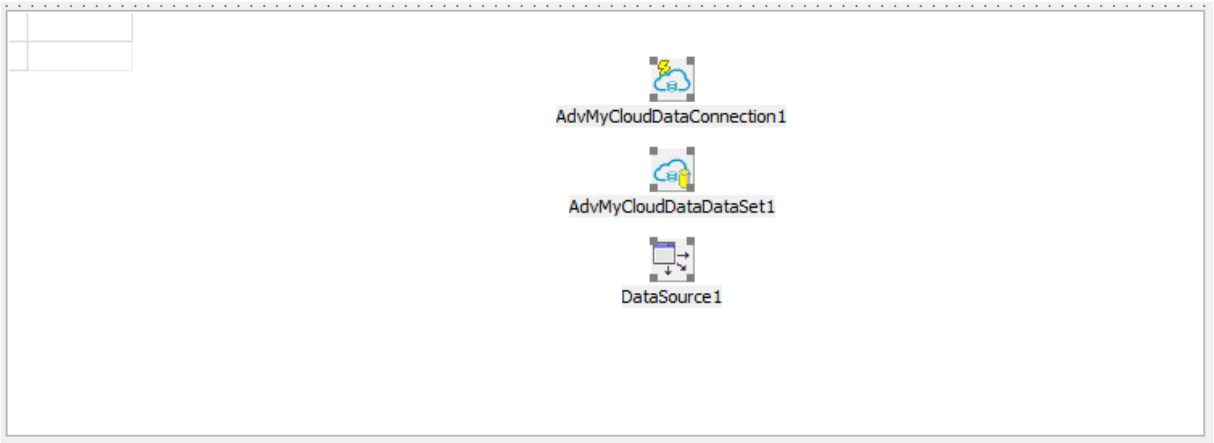
Working with TAdvmyCloudDataDataSet is the same as with any other dataset in Delphi or C++Builder. Connect it via a datasourece to a DB-aware control and you can view and edit the DB record field values.

Example programmatic setup:

```
DataSource1.DataSet := AdvMyCloudDataDataSet1;
DBGrid1.DataSource := DataSource1;

AdvMyCloudDataDataSet1.Connection := AdvMyCloudDataConnection1;
AdvMyCloudDataDataSet1.Table := 'CONTACTS';
AdvMyCloudDataDataSet1.Active := true;
```

and this results in (design-time)



and run-time

ID	NAME	EMAIL	SEX	PICTURE	BIRTHDATE	FRIEND
7	Angelina Jolie	angelina@jolie.com	F	(Blob)	02/02/1977	False
6	Bill Gates	bill@microsoft.com	M	(Blob)	22/02/1958	False
8	Elon Musk	elon.musk@tesla.com	M	(Blob)	13/11/1972	False
9	Megan Fox	megan@fox.com	F	(Blob)	05/12/1979	False
12	Anders Hejlsberg	anders@microsoft.com	M	(Blob)	06/02/1959	False
11	Dieter Zetsche	dieter@daimler.com	M	(Blob)	10/10/1950	False
13	Michael Jackson	michael@jackson.com	M	(Blob)	12/10/1950	False
14	Tobias Moers	tobias.moers@mercedes-a	M	(Blob)	10/12/1965	False
16	Mark Zuckerberg	mark@facebook.com	M	(Blob)	07/12/1977	False
17	Gisele Bündchen	gisele@buendchen.de	F	(Blob)	12/07/1984	False
18	Cindy Crawford	cindy@crawford.com	F	(Blob)	15/01/1978	False
19	Heidi Klum	heidi@klum.de	F	(Blob)	23/09/1981	False

TAdvCardDAV

Usage

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

Properties

property Active: Boolean.
Set Active to true or call TAdvCardDAV.Open to connect to the server

property Filter: TAdvCardDAVFilter;
Connect a TAdvCardDAVFilter component if data returned by the component needs to be filtered.

property ForceSynchronize: Boolean;
When true, the component will automatically try to perform a synchronization after opening the connection (when the server supports sync).

property IgnoreCertificateError: Boolean;
When true, ignores errors raised when the server certificate is not valid or recognized.

property Items: TCardDavItemList;
List of TCardDavItem items. A TCardDAVItem contains the information of a contact and the information is available via TCardDAVItem.vCard.

property Password: string;
Account password

property Synchronizer: TAdvWebDAVSync;
Connect a synchronizer component to synchronize locally stored data with data on the server

property URL: string;
Sets the URL of the CardDAV server

property Username;
Account username

Methods

procedure TAdvCardDAV.Open;
Establishes a connection to the server and retrieves the contacts data. This is equivalent to setting Active = true.

procedure TAdvCardDAV.Close;
Closes the connection to the server.

procedure TAdvCardDAV.Sync;
Performs a synchronization of local data with data on the server.

procedure TAdvCardDAV.ApplyFilter;
Reapply the filter with its filter conditions that is connected to the TAdvCardDAV component.

Events

AfterExecuteCommand: TWDRunEvent;
Event triggered after an underlying WebDAV command was executed.

BeforeExecuteCommand: TWDRunEvent;
Event triggered before an underlying WebDAV command was executed.

The event is declared as:

```
TWDRunEvent = procedure(Method: TWebDavMethod; AUrl: String;
  ASource, AResponseContent: TStream; var Result) of object;
```

Method: Is the WebDAV command being executed
AURL: the WebDAV server URL called
ASource: stream with content sent to the server
AResponseContent: stream with content returned from the server

Example

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

```
var
  i: integer;
  cdi: TCardDAVItem;
  fn,ln: string;

begin
  AdvCardDav1.Url := URL_OF_SERVER;
  AdvCardDav1.Username := ACCOUNT_USERNAME;
  AdvCardDav1.Password := ACCOUNT_PASSWORD;

  AdvCardDav1.Active := true;

  for i := 0 to AdvCardDav1.Items.Count - 1 do
  begin
    cdi := AdvCardDav1.Items[i];

    fn := cdi.vCard.vContacts[0].FirstName;
    ln := cdi.vCard.vContacts[0].LastName;
    ListBox1.Items.Add(fn + ' ' + ln);
  end;
end;
```

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

```
  cdi := AdvCardDav1.Items[IndexOfItemToDelete];
  cdi.Delete;
```

to update an item, use:

```
  cdi := AdvCardDav1.Items[IndexOfItemToUpdate];
  // change the contact data via:
  // cdi.vCard.vContacts[0].Company := NewCompany;
  cdi.Update;
```

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

```
var
  cdi: TCardDavItem;
begin
  cdi := AdvCardDav1.Items.Insert;
  //set the contact data via:
  cdi.vCard.vContacts.Add;
  cdi.vCard.vContacts[0].FirstName := 'Mike';
  cdi.vCard.vContacts[0].LastName := 'Strongwood';
  cdi.Post;
end;
```

TAdvCalDAV

Usage

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

Properties

property Active: Boolean.
Set Active to true or call TAdvCalDAV.Open to connect to the server

property Filter: TAdvCalDAVFilter;
Connect a TAdvCalDAVFilter component if data returned by the component needs to be filtered.

property ForceSynchronize: Boolean;
When true, the component will automatically try to perform a synchronization after opening the connection (when the server supports sync).

property IgnoreCertificateError: Boolean;
When true, ignores errors raised when the server certificate is not valid or recognized.

property Items: TCalDAVItemList;
List of TCalDAVItem items. A TCalDAVItem contains the information of a contact and the information is available via TCalDAVItem.vCalendar.

property Password: string;
Account password

property Synchronizer: TAdvWebDAVSync;
Connect a synchronizer component to synchronize locally stored data with data on the server

property URL: string;
Sets the URL of the CalDAV server

property Username;
Account username

Methods

procedure TAdvCalDAV.Open;
Establishes a connection to the server and retrieves the contacts data. This is equivalent to setting Active = true.

procedure TAdvCalDAV.Close;
Closes the connection to the server.

procedure TAdvCalDAV.Sync;
Performs a synchronization of local data with data on the server.

procedure TAdvCalDAV.ApplyFilter;
Reapply the filter with its filter conditions that is connected to the TAdvCalDAV component.

Events

AfterExecuteCommand: TWDRunEvent;
Event triggered after an underlying WebDAV command was executed.

BeforeExecuteCommand: TWDRunEvent;
Event triggered before an underlying WebDAV command was executed.

The event is declared as:

```
TWDRunEvent = procedure(Method: TWebDavMethod; AUrl: String;
  ASource, AResponseContent: TStream; var Result) of object;
```

Method: Is the WebDAV command being executed
AURL: the WebDAV server URL called
ASource: stream with content sent to the server
AResponseContent: stream with content returned from the server

Example

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

```
var
  i: integer;
  cdi: TCalDAVItem;
  dts,dte: TDateTime;
  sum: string;

begin
  AdvCalDav1.Url := URL_OF_SERVER;
  AdvCalDav1.Username := ACCOUNT_USERNAME;
  AdvCalDav1.Password := ACCOUNT_PASSWORD;

  AdvCalDav1.Active := true;
  for i := 0 to AdvCalDav1.Items.Count - 1 do
  begin
    cdi := AdvCalDav1.Items[i];
    dts := cdi.vCalendar.vEvents[0].DTStart;
    dte := cdi.vCalendar.vEvents[0].DTEnd;
    sum := cdi.vCalendar.vEvents[0].Summary;
    ListBox1.Items.Add(sum + ' ' + DateToStr(dts)+' - ' + DateToStr(dte));
```

```
  end;
end;
```

Note that the CalDAV server typically supports managing multiple calendars. TAdvCalDAV retrieves the calendars available on the server via the TCalDAVCalendarCollection, i.e.

TAdvCalDAV.Calendars: TCalDAVCalendarCollection;

To list all calendars on the server, following code can be used:

```
var
  i: integer;

begin
  for i := 0 to AdvCalDav1.Calendars.Count - 1 do
  begin
    ListBox1.Items.AddAdvCalDav1.Calendars[i]);
  end;
end;
```

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

```
  cdi := AdvCalDav1.Items[IndexOfItemToDelete];
  cdi.Delete;
```

to update an item, use:

```
  cdi := AdvCalDav1.Items[IndexOfItemToUpdate];
  // change the contact data via:
  // cdi.vCalendar.vEvents[0].Summary := NewSummary;
  cdi.Update;
```

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

```
var
  cdi: TCalDavItem;
  CalendarName: string;
begin
  CalendarName := 'Private';
  cdi := AdvCalDav1.Items.Insert(CalendarName);
  //set the event data via:
  cdi.vCalendar.vEvents.Add;
  cdi.vCalendar.vEvents[0].DTStart := Now;
```

```
  cdi.vCalendar.vEvents[0].DTEnd := Now + 1;
  cdi.vCalendar.vEvents[0].Summary := 'Summary of item';
  cdi.Post;
end;
```

TAdvCardDAVFilter

Usage

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

To use the TAdvCardDAVFilter, drop it on the form and connect it to TAdvCardDAV.Filter. Add one or more filter conditions via TAdvCardDAVFilter.Items.

Properties

The TCardDAVFilterItem has following properties:

FieldKind: selects what vCard field to set a condition for
Required: when true, this field should be available in the vCard
SearchKind: sets the filter action to perform with the Value
Value: filter condition value.

Methods

procedure Apply;
Applies the filtering for the currently set filter conditions

function HasFilteredItems: Boolean;
Returns true when valid filter conditions are present

Example

When we want to retrieve only contact persons working for a specific company, following code could be used:

```
var
  cdfi: TCardDAVFilterItem;
begin
  cdfi := AdvCardDAVFilter1.Items.Add;
  cdfi.FieldKind := cdfCompany;
  cdfi.SearchKind := EqualTo;
  cdfi.Value := 'Embarcadero';
  AdvCardDAVFilter1.Filtered := true;

  AdvCardDAV1.Filter := AdvCardDAVFilter1;
end;
```

TAdvCalDAVFilter

Usage

TAdvCalDAVFilter is a component that retrieves vCalendar information from a CalDAV server.It allows to retrieve the events from a calendar for a specific period only.

To use the TAdvCalDAVFilter, drop it on the form and connect it to TAdvCalDAV.Filter. Set the filter condition, i.e. the StartDate and EndDate of the period for which to retrieve calendar events.

Properties

The TCalDAVFilter has following properties:

EndDate: sets the end date of the period to retain events for
Filtered: when true, filtering is active
StartDate: sets the start date of the period to retain events for

Methods

procedure Apply;
Applies the filtering for the currently set StartDate & EndDate properties

procedure FilterByDate(AStartDate, AEndDate: TDateTime);
Applies filtering between given AStartDate and AEndDate

procedure NextSevenDays;
Increments the filter for a week and applies the new filter

procedure NextThirtyDays;
Increments the filter for a next period of 30 days and applies the new filter

procedure DaysBetween(CountOfPriorDays, CountOfNextDays: integer);
Sets the filter to CountOfPriorDays before current date to CountOfNextDays after current date and applies the new filter

procedure ThisWeek;
Sets the filter to start and end date of the current week and applies the new filter

procedure ThisMonth;
Sets the filter to start and end date of the current month and applies the new filter

procedure ThisYear;
Sets the filter to start and end date of the current year and applies the new filter

Example

When we want to retrieve only events in the month of June 2013, following code could be used:

```
begin
  AdvCalDAVFilter1.StartDate := EncodeDate(2013,6,1);
  AdvCalDAVFilter1.EndDate := EncodeDate(2013,6,30);
  AdvCalDAVFilter1.Filtered := true;
  AdvCalDAV1.Filter := AdvCalDAVFilter1;
end;
```

TiCloudContacts

Usage

TiCloudContacts is a component that retrieves contacts information from iCloud. TiCloudContacts is a component with a specific implementation for iCloud descending from TAdvCardDAV so its interface and use is similar.

TiCloudContacts can be used in the same way as TAdvCardDAV. Filtering on TiCloudContacts can be done also with the TAdvCardDAVFilter.

TiCloudCalendar

Usage

TiCloudCalendar is a component that retrieves calendar information from iCloud. TiCloudCalendar is a component with a specific implementation for iCloud descending from TAdvCalDAV so its interface and use is similar.

TiCloudCalendar can be used in the same way as TAdvCalDAV. Filtering on TiCloudCalendar can be done also with the TAdvCalDAVFilter.

TAdvWebDAVStorage

Usage

TAdvWebDAVStorage is a component that can store WebDAV/CalDAV/CardDAV to multiple storage targets including memory, XML file, dataset and custom storage.

Properties

property StorageTargets: TWebDAVSyncStorageTargets;
Collection of local storage targets. Multiple storage targets can be added. Data is persisted in the storage targets during synchronizations. The storage targets can be: memory, XML, dataset or custom storage.

The TWebDAVSyncStorageTarget object has following properties:

property DBFieldMapping;
Sets the relation between CardDAV/CalDAV fields and fields already available in the dataset used for storage. This relation consists of the mapping of category information on dataset fields and/or mapping of item data on dataset fields.

property IsDefault: Boolean;
Only one storage target can be the default target. This is the storage that will be used to initially load the local data from.

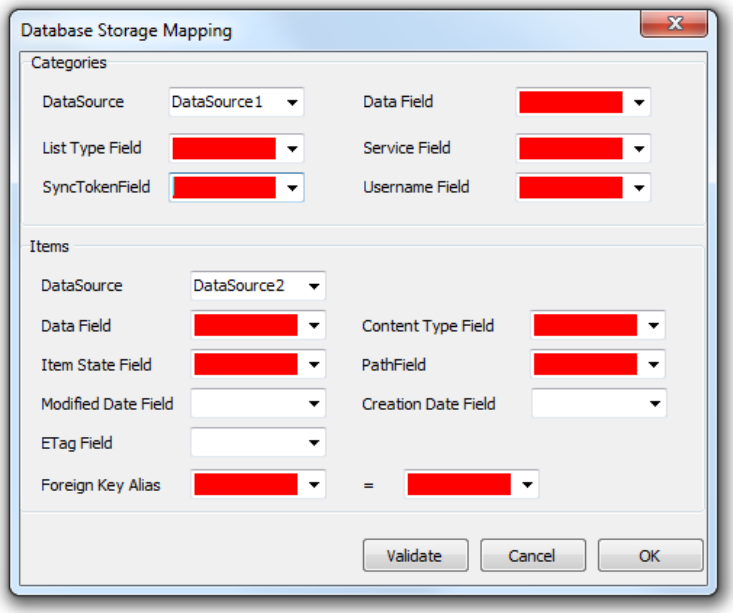
property Location: sets the XML filename when XML file storage target is chosen.

property StorageObject: TWebDavStorageObject;
Selects on of the 4 storage targets:
wdssInMemory: storage is in memory
wdssXmlFile: storage is in an XML file
wdssDB: storage is in a dataset
wdssCustom: custom storage, storage read/write operations are done via events

property SupportedComponents: TWDComponentTypes;
Selects what server types are supported for the storage. This can be WebDAV server, CalDAV server and CardDAV server. To connect the storage to a WebDAV, CalDAV or CardDAV client, make sure to set the appropriate flags to true to support the specific type of data the client needs.

Database mapping

When an existing database should be used as storage, the needed fields for the server categories and items need to be mapped and this is done via WebDAVSyncStorageTarget.DBFieldMapping. This property has a design-time editor that makes it easier to perform this mapping. To perform this mapping, drop 2 datasets connected to 2 datasources on the form, invoke the DBFieldMapping design-time editor and link the first datasource to Categories and the second datasource to items.



When this is linked, select the fields from the dataset that should be used. The requirements for these fields are:

Categories

Data Field: memo field
ListType field: string field (min. 32 chars)
Service field: string field (min. 255 chars)
SyncToken field: string field (min. 255 chars)
UserName field: string field (min. 128 chars)

Items

Data Field: memo field
Item state field: string field (min. 32 chars)
Content Type field: string field
ETag field: string field (min. 64 chars)
Path field: string field (min. 255 chars)
ModifiedDateTime: datetime field
CreationDateFIeld: datetime field

Foreign key alias: key field
Parent key alias: key field

Configuring the storage

To create a storage that can be used for a CardDAV client and that will allow full synchronization and that will keep local data in an XML file, follow these steps:

- Drop a TAdvWebDAVStorage component on the form
- Add a StorageTarget under TAdvWebDAVStorage.StorageTargets
- Set WebDavSyncStorageTarget.SupportedComponents.wdctCardDav = true
- Set WebDavSyncStorageTarget.StorageObject = wdssInMemory
- Add another StorageTarget under TAdvWebDAVStorage.StorageTargets
- Set WebDavSyncStorageTarget.SupportedComponents.wdctCardDav = true
- Set WebDavSyncStorageTarget.StorageObject = wdssXMLFile
- Set WebDavSyncStorageTarget.Location = 'c:\my documents\contacts.xml';
- Set WebDavSyncStorageTarget.IsDefault = true

By setting WebDavSyncStorageTarget.IsDefault = true, the XML file is the default storage and that means that the TAdvWebDAVStorage will use it to automatically save and load the data from the XML file when it is created.

TAdvWebDAVSync

Usage

TAdvWebDAVSync is a component that performs synchronization between data on WebDAV/CardDAV/CalDAV server and data in storage. It synchronizes by reading updated data from server and pushing locally updated data to server.

The TAdvWebDAVSync sits between the TAdvCardDAV or TAdvCalDAV component and the TAdvWebDAVStorage component.

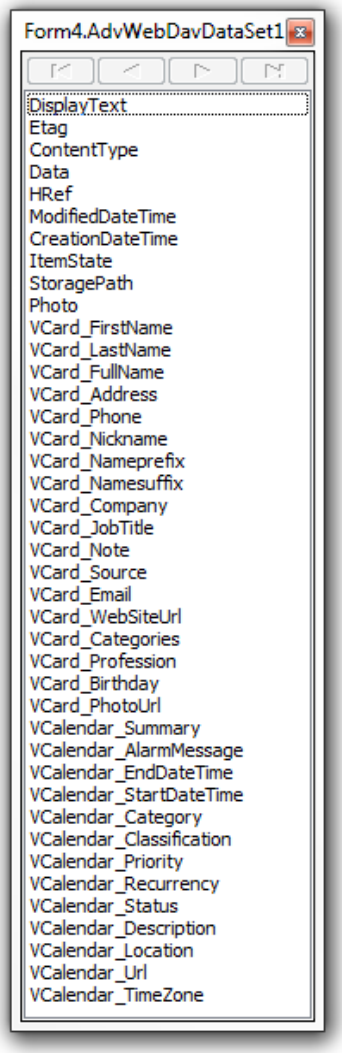
To use it, drop TAdvWebDAVSync on the form and connect it to TAdvCardDAV.Synchronizer or TAdvCalDAVSynchronizer. Drop a TAdvWebDAVStorage component on the form, add minimum one storage target and connect it to TAdvCalDAVSynchronizer.Storage.
When calling AdvCardDAV.Sync or AdvCalDAV.Sync, a synchronization will now take place between the local data and the data on the server. This means that updated information from the server will be retrieved and local changed data will be pushed on the server.

TAdvWebDAVDataSet

Usage

TAdvWebDAVDataSet is a component that exposes WebDAV/CardDAV/CalDAV data as TDataSet so direct binding with DB-aware controls is possible.

Using the TAdvWebDAVDataSet is simple. Drop it on the form and assign a TAdvWebDAVStorage to TAdvWebDAVDataSet.Storage. The TAdvWebDAVDataSet is now ready to expose the data in the local storage as a regular VCL dataset to DB-aware components. When it is connected to a storage that is in turn connected to a TAdvCardDAV client, it will have usable dataset fields VCard_* and when connected to a storage connected to a TAdvCalDAV client, it will have usable dataset fields VCalendar_*. Other fields in this dataset are strictly internally used for synchronization.



Fields available in the TAdvWebDAVDataSet. Use VCard_* fields when connected to a AdvCardDAV client, use VCalendar_* fields when connected to a AdvCalDAV client. When connected to a generic WebDAV server, the data will be available in the Data field. Note that for a AdvCardDAV or AdvCalDAV connected dataset, a full copy of all data will also be in the Data field but in vCalendar file or vCard file format. The TAdvvCard or TAdvvCalendar component could also be used to parse or generate the data in this format.

TAdvWebDAVCollectionFieldDataSet

Usage

TAdvWebDAVCollectionFieldDataSet is a component exposing collection type WebDAV data fields as TDataSet. It exposes the vCard phone, email, timezone & address list for a contact as dataset

As the vCard Phone, Email, Address field is actually a field that can have multiple entries, a special dataset component was created to allow easy binding of DB-aware components to edit these lists of data in the vCard.

To use TAdvWebDAVCollectionFieldDataSet, drop it on the form and connect it to TAdvWebDAVDataSet and select via the property CollectionFieldd what specific field this dataset will expose, i.e.:

property TAdvWebDAVCollectionFieldDataSet.CollectionField: TWDDataSetCollectionField;
wdcfPhone: field used for list of phone numbers is used
wdcfAddress: field used for list of addresses is used
wdcfEmailAddress: field used for list of email-addresses is used
wdcfTimeZone: field used for list of timezones is used

For the wdcfPhone collection field, the dataset contains:

phonenumber: string field
istext: Boolean field
isvoice: Boolean field
isfax: Boolean field
iscell: Boolean field
isvideo: Boolean field
ispager: Boolean field
istextphone: Boolean field
fieldtype: string field

For the wdcfEmailAddress collection field, the dataset contains:

emailaddress: string field
emailtype: string field
preferred: Boolean field

for the wdcfAddress collection field, the dataset contains:

street: string field
country: string field
number: string field
postcode: string field
city: string field

region: string field
addresstype: string field
mailinglabel: memo field

TAdvDropBoxDataStore

Note: This API is deprecated. DropBox does not offer a replacement API at this time.

Usage

TAdvDropBoxDataStore is a component that provides access to the DropBox DataStore service. The component supports creating and deleting datastores as well as reading, inserting, updating and deleting records. Note that the DropBox datastore is a loosely typed not relational recordset. This means that you can insert in a record any set of name/value pairs.

Properties

The **TAdvDropBoxDataStore** has following public properties:

DataStoreList: TDataStoreList; The collection of datastores
LastError: string; The description of the last error that occurred

The **TDataStore** has following properties:

ID: string; The datastore id
DataStoreName: string; The name of the datastore
DateTime: TDateTime; The timestamp of the datastore
Revision: integer; The revision number of the datastore
Rows: TDataStoreRows; The collection of rows that belong to the datastore
Title: string; The title of the datastore

The **TDataStoreRow** has following properties:

ID: string; The row id
TableName: string; The name of the table this row belongs to
Fields: TDataStoreFields; The collection of fields that belong to the row

The **TDataStoreField** has following properties:

FieldName: string; The name of the field
Value: TValue; The value of the field
Tag: TTagInt; The tag value of the field

Methods

CreateDataStore(DataStoreName: string): TDataStore;
Creates a new datastore with the name defined by the DataStoreName parameter.

DeleteDataStore(DataStore: TDataStore);
Deletes an existing datastore.

GetDataStores
Fills the DataStoreList collection with the existing datastores of the currently authenticated user.

GetDataStoreByName(DataStoreName: string): TDataStore;
Returns the datastore with the matching DataStoreName value.

DataStoreList[].SetMetaData(Title: string; DateTime: TDateTime);
Set the meta data for the datastore.

DatatStoreList[].GetRecords: integer;
Fills the Rows collection with records that belong to the datastore. Returns the number of rows in the collection.

DataStoreList[].GetNewID: string;
Returns a unique ID for creating a new TDataStoreRow.

DataStoreList[].InsertRecord(Row: TDataStoreRow): Boolean;
Inserts a new row in a datastore. Returns true if the operation succeeded, false otherwise. When an error occurred, it can be retrieved via public read-only property LastError.

DataStoreList[].DeleteRecord(Row: TDataStoreRow): Boolean;
Deletes an existing row from a datastore. Returns true if the operation succeeded, false otherwise. When an error occurred, it can be retrieved via public read-only property LastError.

DataStoreList[].UpdateRecord(Row: TDataStoreRow): Boolean;
Updates an existing row in a datastore. Returns true if the operation succeeded, false otherwise. When an error occurred, it can be retrieved via public read-only property LastError.

DataStoreList[].DeleteFields(Row: TDataStoreRow; FieldNames: TStringList): Boolean;
Deletes one or more fields (defined in the FieldNames parameter) from a row. Returns true if the operation succeeded, false otherwise.

DataStoreList[].Rows.Find(ID: string): TDataStoreRow;
Returns the row with a matching ID value.

DataStoreList[].Rows[].AddData(AName: string; AValue: string);
Adds a new field to a datastore row.

DataStoreList[].Rows[].Fields.AddPair(AFieldName: string; AValue: string): TDataStoreFIeld;
DataStoreList[].Rows[].Fields.AddPair(AFieldName: string; AValue: integer): TDataStoreFIeld;
DataStoreList[].Rows[].Fields.AddPair(AFieldName: string; AValue: double): TDataStoreFIeld;

DataStoreList[].Rows[].Fields.AddPair(AFieldName: string; AValue: boolean): TDataStoreFIeld;
Adds a new field to a datastore row based on the AFieldName and AValue parameter values. The AValue can be a string, integer, double or boolean.

Sample

This code snippet shows how a datastore with a 2 row recordset can be programmatically created on the DropBox datastore:

```
var
  ds: TDataStore;
  dr: TDataStoreRow;


begin
  ds := AdvDropBoxDataStore1.CreateDataStore('demo');

  dr := ds.Rows.Add;
  dr.ID := ds.GetNewID;
  dr.TableName := 'Capitals';

  dr.Fields.AddPair('Name','Paris');
  dr.Fields.AddPair('Country','France');
  dr.Fields.AddPair('Citizens',10000000);
  dr.Fields.AddPair('Europe',true);
  ds.InsertRecord(dr);

  dr := ds.Rows.Add;
  dr.ID := ds.GetNewID;
  dr.TableName := 'Capitals';

  dr.Fields.AddPair('Name','Brussels');
  dr.Fields.AddPair('Country','Belgium');
  dr.Fields.AddPair('Citizens',3000000);
  dr.Fields.AddPair('Europe',true);

  ds.InsertRecord(dr);
end;
```

TAdvPryv

Usage

TAdvPryv is a component that provides access to the Pryv service.
The component supports retrieving, inserting, updating and deleting of streams and events.
Different types of events are supported: Text, Picture, File, Position (Geolocation) …

Properties

The **TAdvPryv** has following public properties:

Events: TList; The collection of events
Streams: TPryvStreams; The collection of streams and substreams

The Events collection can contain a number of different class objects, all derived from **TPryvObject**.

The **TPryvObject** has following properties:

ID: string; The ID of the event
StreamID: string; The ID of the stream this event belongs to
DateTime: TDateTime; The datetime associated with the event
Description: string; The description of the event
Tags: TStringList; The list of tags associated with the event
Created: TDateTime; The datetime when the event was created
Updated: TDateTime; The datetime when the event was last updated

The **TPryvText** has following properties (in addition to all properties from TPryvObject):

Content: string; The content text of the event

The **TPryvValue** has following properties (in addition to all properties from TPryvObject):

Content: string; The measurement value of the event
UnitValue: string; The measurement unit of the event

This class can be used to hold any event type that is defined by a measurement unit (i.e: “length/cm”) and a measurement value (i.e.: ‘100’).
A list of currently available numerical event types can be found at: http://pryv.github.io/event-types/#directory-numerical-types.
Besides the documented event types, it’s also possible to define custom event types.

Sample:

This code snippet demonstrates how to add a new Event with a custom type ('mymeasurement/myvalue') to an existing Stream:

```
var
  val: TPryvValue;
begin

   val := TPryvValue.Create;
   AdvPryv1.Events.Add(val);
   val.Content := '10';
   val.StreamID := AdvPryv1.Streams[0].ID;
   val.UnitValue := 'mymeasurement/myunit';
   val.DateTime := Now;
   val.Tags.CommaText := 'custom, value';
   val.Description := 'my custom value';
   AdvPryv1.AddEvent(val);
end;
```

The **TPryvPosition** has following properties (in addition to all properties from TPryvObject):

Latitude: double; The latitude coordinate of the event
Longitude: double; The longitude coordinate of the event

The **TPryvPicture** has following properties (in addition to all properties from TPryvObject):

FileName: string; The FileName of the image file associated with the event
ImageURL: string; The url of the image file associated with the event

The **TPryvFile** has following properties (in addition to all properties from TPryvObject):

FileName: string; The FileName of the file associated with the event
FileURL: string; The url of the file associated with the event

The **TPryvStreamItem** has following properties:

ID: string; The stream id
ParentID: string; The id of the parent stream (only available for substreams)
Summary: string; The name of the stream
Created: TDateTime; The datetime when the stream was created
Updated: TDateTime; The datetime when the stream was last updated
SubStreams: TPryvStreams; The collection of substreams for this stream

Methods

AddStream(Stream: TPryvStreamItem);
Creates a new stream with the values defined in the Stream parameter. This method can be used to add streams as well as substreams.

DeleteStream(Stream: TPryvStreamItem);
Deletes an existing stream.

GetStreams;
Fills the Streams collection with the existing streams and substreams of the currently authenticated user.

UpdateStream(Stream: TPryvStreamItem);
Updates an existing stream with the values defined in the Stream parameter.

AddEvent(Event: TPryvObject);
Creates a new event with the values defined in the Event parameter. This method can be used to add Events of types TPryvText, TPryvValue and TPryvPosition.

AddEvent(Event: TPryvFile; FileName: string);
Creates a new event with the values defined in the Event parameter. This method can only used to add Events of type TPryvFile. The local file referenced in the FileName parameter will be uploaded to the Pryv service and associated with the Event.

AddEvent(Event: TPryvPicture; FileName: string);
Creates a new event with the values defined in the Event parameter. This method can only used to add Events of type TPryvPicture. The local image file referenced in the FileName parameter will be uploaded to the Pryv service and associated with the Event.

DeleteEvent(Event: TPryvObject);
Deletes an existing event.

GetEvents(Streams: TStringArray);
GetEvents(Streams: TStringArray; Tags: TStringArray; MaxResults: integer; Page: integer; SortAscending: Boolean);
GetEvents(FromTime: TDateTime; ToTime: TDateTime; MaxResults: integer; Page: integer; SortAscending: Boolean);
Fills the Events collection with the existing events of the currently authenticated user.
Parameters:
Streams: If assigned, only events that belong to the specified stream ids and their sub-streams will be returned.
Tags: If assigned, only events that have any of the provided tags assigned will be returned.
MaxResults, Page: can be used to retrieve paged results
SortAscending: If true, events will be sorted from oldest to newest
FromTime: The start time of the timeframe events are retrieved for
ToTime: The end time of the timeframe events are retrieved for

UpdateEvent(Event: TPryvObject);
Updates an existing event with the values defined in the Event parameter.

Note that for TPryvPicture the FileName and ImageURL properties can not be updated, for TPryvFile the FileName and FileURL properties can not be updated.

Download(Event: TPryvFile; TargetFile: string);
Download(Event: TPryvPicture; TargetFile: string);
The (image) file associated with the Event will be downloaded to the TargetFile.

Sample

This code snippet shows how to add a new Event of type Position to a new Stream:

```
Var
  it: TPryvStreamItem;
  pos: TPryvPosition;

Begin
  it := AdvPryv1.Streams.Add;
  it.Summary := edStreamName.Text;
  AdvPryv1.AddStream(it);

  pos := TPryvPosition.Create;
  AdvPryv1.Events.Add(pos);
  pos.Latitude := StrToFloat(edLatitude.Text);
  pos.Longitude := StrToFloat(edLongitude.Text);
  pos.StreamID := it.ID;
  pos.DateTime := Now;
  pos.Tags.CommaText := 'location';
  pos.Description := 'position description';
  AdvPryv1.AddEvent(pos);
```

TAdvTrello

Usage

TAdvTrello is a component that provides access to the Trello service. Trello is the free, flexible and visual way to organize anything with anyone.
The component supports retrieving, inserting, updating and deleting of boards, lists, cards, attachments and checklists.

TAdvCustomTrello

Methods
GetBoardLabels(ABoard: TAdvTrelloBoard);
Retrieves the list of all available labels of the board. The labels can be read after retrieval via Aboard.LabelNames.

GetBoardMemberships(ABoard: TAdvTrelloBoard);
Retrieves a list of all members (invited and confirmed) of the board. Aboard.Memberships.

GetMember(AUsername: string = 'me');
Gets the member details by the provided username. The member details are returned via AdvTrello.Member.

GetLists(ABoard: TAdvTrelloBoard);
Retrieves a list of all available board lists (open and closed). The lists can be read after retrieval via Aboard.Lists.

GetCards(AList: TAdvTrelloList);
Retrieves a list of all available card inside the provided list (open and closed).
The members, checklists, labels and actions are included in this call. The cards can be read via AList.Items[index]: TAdvTrelloCard.

SetBoardClosed(ABoard: TAdvTrelloBoard ; IsClosed: Boolean = True);
Toggles the closed property of the board.

SetListClosed(AList: TAdvTrelloList; IsClosed: Boolean = True);
Toggles the closed property of the list.

SetCardClosed(ACard: TAdvTrelloCard; IsClosed: Boolean = True);
Toggles the closed property of the card.

DeleteCard(ACard: TAdvTrelloCard);
Permanently removes the card from the list.

AddBoardMember(ABoard: TAdvTrelloBoard; AMemberEmail: string);
Adds or invites a member to the board by emailaddress.

AddCardComment(ACard: TAdvTrelloCard; AComment: string);
Adds a comment to the card.

AddCardChecklist(ACard: TAdvTrelloCard; AChecklistName: string);
Adds a new checklist to the card with empty checklist items. The checklist is accessible via the Card.CheckLists list.

AddCardChecklistItem(ACard: TAdvTrelloCard; AChecklist: TAdvTrelloCardCheckList; ACheckItemName: string);
Adds a new checklist item to the provided checklist inside a card.

AddCardLabel(ACard: TAdvTrelloCard; ALabel: TAdvTrelloBoardLabelNames);
Adds a label to the card. The label is accessible via the Card.Labels list.

AddCardAttachment(ACard: TAdvTrelloCard; AFileName: string);
Adds a file attachment to the card.

AddCardAttachmentUrl(ACard: TAdvTrelloCard; AUrl: string);
Adds an URL attachment to the card.

UpdateCardComment(AAction: TAdvTrelloCardAction; AComment: string);
Updates an existing comment of the provided card. Note that comments are named Actions in Trello and can be retrieved via the Card.Actions list.

UpdateCardDescription(ACard: TAdvTrelloCard; ADescription: string);
Updates the cards description.

UpdateCardDueDate(ACard: TAdvTrelloCard; ADateTime: TDateTime);
Updates the cards due date.

UpdateCardCheckListName(ACheckList: TAdvTrelloCardCheckList; AChecklistName: string);
Updates the provided checklists name.

UpdateCardCheckListItemName(ACard: TAdvTrelloCard; ACheckList: TAdvTrelloCardCheckList; AChecklistItem: TAdvTrelloCardCheckListItem; AChecklistItemName: string);
Updates the provided check items name.

UpdateCardLabel(ABoard: TAdvTrelloBoard; ACard: TAdvTrelloCard; ALabel: TAdvTrelloCardLabel; ALabelName: string);
Updates the name of the specified cards label.

DeleteCardComment(AAction: TAdvTrelloCardAction);
Deletes the specified cards comment.

DeleteCardLabel(ACard: TAdvTrelloCard; ALabelID: string);
Deletes the specified label.

DeleteCardAttachment(ACard: TAdvTrelloCard; AAttachment: TAdvTrelloCardAttachment);
Permanently deletes the specified attachment.

AddBoard(AName: string): string;
Adds a board by name. The board is accessible via the AdvTrello.Member.Boards list.

AddCardMember(ABoard: TAdvTrelloBoard; ACard: TAdvTrelloCard; AUsername: string): Boolean;
Adds a member to the specified card.

AddList(ABoard: TAdvTrelloBoard; AName: string): string;
Adds a list by name to the specified board.

AddCard(AList: TAdvTrelloList; AName: string): string;
Adds a card by name to the specified list.

Properties

Member: TAdvTrelloMember
The full member with all its boards.

TAdvTrelloMember

Methods

GetBoardByID(ABoardId: string): TAdvTrelloBoard;
Returns the full board by provided board id.

Properties

Username: string
The username of the member.

ID: string
The ID of the member.

Initials: string
The initials of the member. Ex: John Doe = JD

Membertype: string
The account type (normal, admin, disabled)

AvatarSource: string
The source of the avatar.

AvatarHash: string
The hash of the avatar.

Bio: string
The description on the members account.

FullName: string
The full written name of the member.

URL: string
The url to the members page.

Email: string
The emailaddress of the member.

GravatarHash: string
The hash of the gravatar.

IDBoards: TList<string>
A list of all members board ID's .

IDOrganizations: TList<string>
A list of all members Organization ID's.

Boards: TAdvTrelloBoardList
All the members boards

TAdvTrelloBoard

Methods

GetListByName(AName: string): TAdvTrelloList;
Gets the full board list by its name.

GetListById(AID: string): TAdvTrelloList;
Gets the full board list by its ID.

Properties

ID: string
The ID of the board.

Desc: string
The boards description.

Name: string
The name of the board.

Closed: Boolean
True if the board has been closed.

IDOrganization: string
The corresponding organization ID.

ShortLink: string
The shortened URL of the board

DateLastActivity: TDateTime
The date and time of the last activity

Starred: Boolean
If the board has been starred

URL: string
The URL of the board.

Memberships: TList<TAdvTrelloBoardMembership>
All members who are connected to the board.

LabelNames: TList<TAdvTrelloBoardLabelNames>
List of available labels.

DateLastView: TDateTime
The date and time of the last view

ShortURL: string
The shortened URL to the board.

Lists: TAdvTrelloListList
A list of all lists inside the board.

TAdvTrelloList

Methods

GetCardById(AID: string): TAdvTrelloCard;
Gets the full Card by its ID.

Properties

ID: string
The ID of the list.

Name: string
The name of the list.

Closed: Boolean
Indicates if the list has been closed.

Position: Single
The position of the list inside the board.

Subscribed: Boolean
If you are Subscribed.

Cards: TAdvTrelloCardList
All the Cards inside the current list.

TAdvTrelloCard

Methods

GetActionById(AID: string): TAdvTrelloCardAction;
Gets a cards action by its ID. This can be a comment, activity, member add notification, …

GetAttachmentById(AID: string): TAdvTrelloCardAttachment;
Gets the specific attachment by its ID.

getLabelById(AID: string): TAdvTrelloCardLabel;
Gets the specific Card label by its ID.

GetChecklistById(AID: string): TAdvTrelloCardCheckList;
Gets the specific Card Checklist by its ID.

Properties

ID: string
The ID of the card.

Closed: Boolean
Indicates if the Card has been closed.

DateLastActivity: TDateTime
The date and time of the last activity.

Desc: string
The description of the card.

DescData: string
The data of the card.

IDShort: Integer;
A shortened ID of the card.

Name: string
The name of the card.

Position: Single
The position of the card inside the list.

ShortLink: string
The shortened URL of the card.

Due: TDateTime
The due date of the card in date and time format.

Email: string
The emailaddress of the Card.

ShortURL: string
The shortened URL of the card.

Subscribed: Boolean
Indicates if you are subscribed to the card.

URL: string
The full URL of the card.

Attachments: TList<TAdvTrelloCardAttachment>
A list of all the attachments.

Members: TList<TAdvTrelloCardMember>
A list of all linked members.

Actions: TAdvTrelloCardActionList
A list of all actions of the card.

CheckLists: TAdvTrelloCardCheckListList
A list of all the checklists of the card.

Labels: TList<TAdvTrelloCardLabel>
A list of all the labels that are linked to this card.

TAdvTrelloCardChecklist

Methods

GetChecklistItemById(AID: string): TAdvTrelloCardCheckListItem;
Gets the specific checklist item by its ID.

Properties

ID: string
The ID of the checklist.

Name: string
The display name of the checklist.

Position: Single
The position of the checklist inside the card.

CheckItems: TAdvTrelloCardCheckListItemList
A list of all the items of this checklist.

TAdvTrelloCardChecklistItem

Properties
ID: string
The ID of the checklist item.

Name: string
The name of the checklist item.

NameData: string
The data of the checklist item.

Position: Single
The position of the checklist item inside the current checklist.

State: string
The current state of the checklist item.

TAdvTrelloCardAction

Properties
ID: string
The ID of the action

IDMemberCreator: string
The ID of the member that caused the action.

Text: string
The corresponding text of the action.

ActionType: string
The type of the action. Ex: CommentCard

Date: TDateTime
The date and time of the action.

TAdvGSheets

Usage

TAdvGSheets is a component that provides access to the Google Sheets service. It enables to read, create and edit sheets.

TAdvCustomGSheets

Methods

CreateWorkbook(ATitle: string);
Creates a workbook with the given title.

GetAvailableWorkSheets: TList<TAdvGSheetsWorksheet>;
Gets a list <name and id> of all available worksheets on your account.

GetWorkSheet;
Gets the full worksheet for the worksheet ID set by TAdvGSheets.WorksheetID.

GetWorkSheetID(AWorkSheetTitle: string): string;
Gets the worksheet ID by the specified title. In case there are multiple sheets with the same name, the most recent sheet ID will be returned.

Properties

WorkSheetID: The ID of the worksheet

WorkSheetTitle: The title of the worksheet

TAdvGSheetsSheet

Methods

GetCells;
Gets all the cells of the current sheet.

AddRow(AValues: TStringList): Boolean;
Adds a row to the current sheet with the specified column based values.

GetCell(ACol, ARow: Integer): TAdvGSheetsCell;
Gets the cell by its column/row coordinates.

UpdateCell(var ACell: TAdvGSheetsCell): Boolean; overload;
Updates the cell via a cell object.

UpdateCell(ACol, ARow: Integer; AValue: string): Boolean; overload;
Updates the cell value as string via the cell coordinates.

RemoveRow(ARowIndex: Integer): Boolean;
Removes a row by its row coordinate.

SetWorkSheetSize(AColCount, ARowCount: Integer);
Increases / decreases the size of the current sheet by the specified size.

Properties

Cells: TAdvGSheetsCellList
A list of all the cells inside the sheet.

ColCount: Integer
The column count.

Content: string
The sheets' content.

ID: string
The (unique Google) ID of the current sheet.

RowCount: Integer
The row count.

Title: string
The title of the sheet.

Updated: string
string based date of last update time.

TAdvGSheetsCell

Properties

Col: Integer
The column coordinate of the cell.

Content: string
The content of the cell.

ID: Integer
The cells ID.

InputValue: string
The actual input value.

Position: string
The string based cell coordinate. Ex: A1R1

Row: Integer
The row coordinate of the cell.

Updated: string
string based date of last update time.

TAdvGMail

Usage

TAdvGMail is a component that provides access to the Google Mail service. It enables to retrieve and send email messages. File attachments can be included when sending emails.

TAdvGMail

Methods

procedure GetMails(Labels: string = 'INBOX'; AMaxResults: Integer = 50);
Retrieves all the mails from the specified mail folder. The emails can be read via the TAdvGMail.Mails list.

GetLabels;
Retrieves all the labels. After retrieval, the labels can be read via the AdvGMail.Label list.

SendMessage(AMessage: TAdvGMailMessage; ALabels: string = 'SENT'): Boolean;
Sends a message, with the option to assign one or more labels to the message. When multiple labels are used, the list needs to be a comma separated string.

UpdateMessageLabels(AMessageId, ALabels: string): Boolean;
Updates the messages labels. When multiple labels are used, the list needs to be a comma separated string.

Properties

Mails: TAdvGMailMessageList
All the mails from the folder specified when calling GetMails.

TGMailMessage

Properties

ID: string
The (unique Google) ID of the email.

BCCRecipients: TStringList
The list of all BCC recipients.

Body: string
The body text of the message.

CCRecipients: TStringList
The list of all CC recipients.

FileName: string
The name of the associated file.

Headers: TList<TAdvGMailHeader>
All messages headers.

MessageType: TAdvGMailMessageType
The type of the Message. <plain text or HTML>

Snippet: string
A short substring of the actual body text.

Subject: string
The subject of the message.

ToRecipients: TStringList
The list of all recipients.

TGMailLabel

Properties

ID: string
The (unique Google) ID of the label.

Name: string
The name of the label.

LabelType: string
The type of the label.

TAdvExceptionless

Usage

TAdvExceptionless is a component that provides access to the Exceptionless API which is capable of managing exceptions and log messages.

Organisation

TExceptionlessProject

Methods

procedure LogMessage(AMessage: string; ASource: string = ''; ALevel: TExceptionlessLogLevel = ellInfo);
Sends a log message with an optional source and level parameter.

procedure LogException(AMessage, AType, AStackTrace: string);
Sends an exception with a message, an exception type and an optional stack trace.

TAdvExceptionless

Methods

function GetProjectByID(AID: string): TExceptionlessProject;
Returns an existing project with a specific ID.

function GetProjectByName(AName: string): TExceptionlessProject;
Returns an existing project with a specific Name.

procedure GetProjects;
Retrieves all projects from Exceptionless and stores them in the Projects collection.

procedure LogMessage(AMessage, ASource, AProjectID: string; ALevel: TExceptionlessLogLevel = ellInfo);
Sends a log message with an optional source and level parameter to a specific project.

procedure LogMessage(AMessage, ASource: String; AProject: TExceptionlessProject; ALevel: TExceptionlessLogLevel = ellInfo);
Sends a log message with an optional source and level parameter to a specific project.

procedure LogException(AMessage, AType, AStackTrace, AProjectID: string);

Sends an exception with a message, an exception type and an optional stack trace to a specific project.

procedure LogException(AMessage, AType, AStackTrace: string; AProject: TExceptionlessProject);
Sends an exception with a message, an exception type and an optional stack trace to a specific project.

Properties

Projects: TExceptionlessProjects;
Contains the projects with a name and ID after calling GetProjects.

Authentication

Exceptionless authentication is no more than entering your UserName and Password (obtains after signing up at the Exceptionless website) and calling DoAuth. Below is a sample that demonstrates this.

```
AdvExceptionless1.UserName := 'myUserName@mail.com';
AdvExceptionless1.Password := 'myPassword';
If not AdvExceptionless1.TestTokens then
 AdvExceptionless1.DoAuth;

AdvExceptionless1.GetProjects;
```

TAdvGAnalytics

Usage

TAdvGAnalytics is a component that provides access to the Google Analytics service. It enables to get insights into how visitors find and use your site, and how to keep them coming back.

TAdvCustomGAnalytics

Methods

GetData(AStartDate: string = 'today'; AEndDate: string = 'today'; AMaxResults: Integer = 1000): string;
Gets all the data that was requested by adding dimensions & metrics to the *RequestData* property.

Properties

Data: TGData
The returned data.

RequestData: TGRequestData
The place to distinguish the needed dimensions and metrics.

ErrorMessages: TGErrorMessages
Can set the error message for metrics and dimensions, when that error occurs.

For more information about metrics and dimensions, please see the analytics reporting api and analytics realtime reporting api

TGData

Property

Data: TList<TArray<string>>
A list of all requested data (in the same order as requested)

TAdvMSComputerVision

Usage

TAdvMSComputerVision is a component that provides access to the Microsoft Computer Vision API. It provides information about visual content found in an image. The component can also perform Optical Character Recognition (OCR) which detects text in an image and extracts the recognized words into machine-readable character string.

Organisation

Properties:

Analysis: TMSComputerVisionAnalysis; Contains image analysis data.

Colors: TMSComputerVisionColor: Information about the detected colors.
Tags: TStringList: A detailed list of words related to the image.
Categories: TStringList; Categorizes image content according to a taxonomy defined in documentation.
CategoryScores: TMSComputerVisionScoreList; Indicates the amount of confidence in the Categories values.
Descriptions: TStringList; Describes the image content with a complete English sentence.
Faces: TComputerVisionFaces; List of faces detected in the image content.
IsAdultContent: Boolean; Indicates if the image contains adult content.
AdultScore: double; Indicates the amount of confidence in the IsAdultContent value.
IsRacyContent: Boolean; Indicates if the image contains racy content.
RacyScore: double; Indicates the amount of confidence in the IsRacyContent value.
Width: integer; The width of the image.
Height: integer; The height of the image.
Format: string; The image format.
ClipArtType; TMSComputerVisionClipArtType; The type of clipart detected in the image.
DrawingType: TMSComputerVisionDrawingType: The type of drawing detected in the image.

TMSComputerVisionColor:

AccentColor: string; The accent color detected in the image.
DominantForeGround: string; The dominant foreground color detected in the image.
DominantBackground: string; The dominant background color detected in the image.
DominantColors: TStringList; The list of dominant colors detected in the image.
IsBlackAndWhite: Boolean; Indicates if the images is a black & white image.

TMSComputerVisionFace:

Age: integer; The age of the detected face.
Gender: TMSComputerVisionGendeder; The gender of the detected face.
Bounds: TMSComputerVisionBounds: The region on the image where the face was detected.

OCR: TMSComputerVisionOCR; Contains image OCR data.

Language: TMSComputerVisionLanguage; The language of text detected in the image.
Regions: TMSComputerVisionRegions: The regions in the image where text was detected.
Text: TStringList; All text detected in the image.

TMSComputerVisionRegion:

Bounds: TMSComputerVisionBounds; The region on the image where the region of text was detected.
Lines: TMSComputerVisionLines; The lines of text contained in the region.

TMSComputerVisionLine:

Bounds: TMSComputerVisionBounds; The region on the image where the line of text was detected.
Words: TMSComputerVisionWords; The words detected in the line.

TMSComputerVisionWord:

Bounds: TMSComputerVisionBounds; The region on the image where the word of text was detected.
Text: string; The word detected in the image.

Methods:

ProcessURL(AURL: string; ProcessType: TMSComputerVisionType): Boolean;
Process the image referenced in the AURL parameter.
If ProcessType is set to ctAnalysis, fills the Analysis class properties.
If ProcessType is set to ctOCR, fills the OCR class properties.
Returns true if the request succeeded.

ProcessFile(AFileName: string; ProcessType: TMSComputerVisionType): Boolean;
Upload and process the image referenced in the AFileName parameter.
If ProcessType is set to ctAnalysis, fills the Analysis class properties.
If ProcessType is set to ctOCR, fills the OCR class properties.
Returns true if the request succeeded.

TAdvMSEmotion

Usage

TAdvMSEmotion is a component that provides access to the Microsoft Emotion API. It provides information about the emotions detected in facial expressions found in an image. Each supported emotion is assigned a score. In interpreting results from the Emotion API, the emotion detected should be interpreted as the emotion with the highest score, as scores are normalized to sum to one.

Organisation

Properties:

Emotions: TMSEmotionEmotions; Contains image facial expressions data.

> Bounds: TMSEmotionBounds; The region on the image that the emotion scores refer to.
> AngerScore: double; Indicates the amount this emotion detected.
> ContemptScore: double; Indicates the amount this emotion detected.
> DisgustScore: double; Indicates the amount this emotion detected.
> FearScore: double; Indicates the amount this emotion detected.
> HappinessScore: double; Indicates the amount this emotion detected.
> NeutralScore: double; Indicates the amount this emotion detected.
> SadnessScore: double; Indicates the amount this emotion detected.
> SurpriseScore: double; Indicates the amount this emotion detected.

Methods:

ProcessURL(AURL: string): Boolean;
Process the image referenced in the AURL parameter.
Fills the list Emotions detected in the image.
Returns true if the request succeeded.

ProcessFile(AFileName: string): Boolean;
Upload and process the image referenced in the AFileName parameter.
Fills the list Emotions detected in the image.
Returns true if the request succeeded.

TAdvMSBingSpeech

Usage

TAdvMSBingSpeech is a component that provides access to the Microsoft Bing Speech API. It provides support to synthesize text to spoken audio. Multiple locales and related voice fonts are supported.

Organisation

Methods:

Synthesize(AText: string; TargetFile: string; VoiceFont: TMSBingSpeechVoice): Boolean;
Convert AText value to an audio file and download the file.
The format of the file file is: “riff-16khz-16bit-mono-pcm” (Wave).
Optionally set one of the pre-defined locales/voice fonts with the VoiceFont parameter.
Returns true if the request succeeded.

Synthesize(AText: string; TargetStream: string; VoiceFont: TMSBingSpeechVoice): Boolean;
Convert AText value to an audio stream.
Optionally set one of the pre-defined locales/voice fonts with the VoiceFont parameter.
Returns true if the request succeeded.

Note: Bing Speech API Keys requested before November 15th 2016 are no longer valid due to changes in the authentication process of the Bing Speech API.

A new API Key can be requested after cancelling the existing API Key.

TAdvImgur

Usage

TAdvImgur is a component that provides access to the Imgur API. It provides support to upload an image file to a publically accessible URL.

Organisation

Methods:

UploadImage(FileName: string): string;
Upload an image file. Returns the URL for the uploaded file.

TAdvCloudinary

Usage

TAdvCloudinary is a component that provides access to the Cloudinary API. It provides support to upload an image file to a publically accessible URL.

Organisation

Properties:

App.CloudName: string;
Extra property for authentication. Required to be assigned with the CloudName value of the Cloudinary app.
Detailed information is available at: http://www.tmssoftware.com/site/cloudkey.asp

App.PresetUpload: string;
Extra property for authentication. Required to be assigned with the preset value configured in the Cloudinary app.
Detailed information is available at: http://www.tmssoftware.com/site/cloudkey.asp

Methods:

UploadImage(FileName: string): string;
Upload an image file. Returns the URL for the uploaded file.

TAdvFirebaseObjectDatabase

Usage

TAdvFirebaseObjectDatabase is a component that provides access to the Google Firebase Database API. It provides support to retrieve, add, update and remove objects.

Organisation

Properties:

DatabaseName: string;
Required to be assigned with the name of the Firebase database related to the authenticated user. Detailed information is available at: http://www.tmssoftware.com/site/cloudkey.asp

TableName: string;
Assign the name of the table that is used to retrieve, add, update and remove with the methods indicated below.

GeneralRules: TStringList;
Contains the general rules specified for the Firebase Database. Use the GetRules method to fill the list with existing rules.

IndexRules: TList<TIndexRule>;
Contains the index rules specified for the Firebase Database. These are required when using one of the Query* methods. Use the GetRules method to fill the list with existing rules. Use the AddIndexRule method to add a rule.

Methods:

InsertObject(AObject: TFirebaseObject): Boolean;
Insert a new object in the Table specified in the TableName property.

WriteObject(AObject: TFirebaseObject): Boolean;
Update an existing object in the Table specified in the TableName property.

ReadObject(ID: string): TFirebaseObject;
ReadObject(ID: string; AObject: TFirebaseObject): Boolean;
Retrieve an existing object based on it's ID from the Table specified in the TableName property.

DeleteObject(ID: string): Boolean;
DeleteObject(AObject: TFirebaseObject); Boolean;
Delete an existing object from the Table specified in the TableName property.
Returns true if the action was successful, false otherwise.

InsertList(AList: TFirebaseObjectList): Boolean;
Insert a list of TFirebaseObject objects in the Table specified in the TableName property.
Returns true if the action was successful, false otherwise.

WriteList(AList: TFirebaseObjectList): Boolean;
Updates a list of TFirebaseObject objects in the Table specified in the TableName property. Returns true if the action was successful, false otherwise.

ReadList: TFirebaseObjectList;
ReadList(AList: TFirebaseObjectList): Boolean;
Retrieves a list of TFirebaseObject objects from the Table specified in the TableName property.

DeleteList(AList: TFirebaseObjectList): Boolean;
Deletes a list of TFirebaseObject objects from the Table specified in the TableName property. Returns true if the action was successful, false otherwise.

QueryList(PropertyName, KeyWord: string): TFirebaseObjectList;
QueryList(PropertyName, KeyWord: string; AList: TFirebaseObjectList);
QueryList(PropertyName, KeyWord: boolean): TFirebaseObjectList;
QueryList(PropertyName, KeyWord: boolean; AList: TFirebaseObjectList);
QueryList(PropertyName, KeyWord: double): TFirebaseObjectList;
QueryList(PropertyName, KeyWord: double; AList: TFirebaseObjectList);
Retrieves a filtered list of TFirebaseObject objects from the Table specified in the TableName property. The list is filtered on the field specified in the PropertyName value based on the KeyWord value. The KeyWord can be a string, Boolean or integer/double value depending on the content of the field.
Note it is required that the PropertyName is present in the list of IndexRules. Use the AddIndexRule to add a new PropertyName.

QueryList(PropertyName, StartAt, EndAt: string): TFirebaseObjectList;
QueryList(PropertyName, StartAt, EndAt: string; AList: TFirebaseObjectList);
Retrieves a filtered list of TFirebaseObject objects from the Table specified in the TableName property. The list is filtered on the field specified in the PropertyName value based on the StartAt and EndAt values.
For example if the StartAt is “A” and the EndAt is “A” the result will contain a list of all objects for which the PropertyName field text starts with “A”.
Note it is required that the PropertyName is present in the list of IndexRules. Use the AddIndexRule to add a new PropertyName.

QueryList(PropertyName, MinValue, MaxValue: string): TFirebaseObjectList;
QueryList(PropertyName, MinValue, MaxValue: string; AList: TFirebaseObjectList);
Retrieves a filtered list of TFirebaseObject objects from the Table specified in the TableName property. The list is filtered on the field specified in the PropertyName value based on the MinValue and MaxValue values.
For example if the MinValue is “0” and the EndAt value is “100” the result will contain a list of all objects for which the PropertyName field data is higher or equal to “0” and lower or equal to “100”.
Note it is required that the PropertyName is present in the list of IndexRules. Use the AddIndexRule to add a new PropertyName.

AddIndexRule(ATableName: string; PropertyName: string): Boolean;
Add a new rule to the list of IndexRules and submit the new data to the Firebase Database rules.
If the rule based on the TableName and PropertyName already exists, no data is changed.
When using one of the Query* methods it is required that the PropertyName (= Fieldname) has been added to the list of index rules.
Returns true if the action was successful, false otherwise.

DeleteTable(ATableName: string);
Delete all objects and table object itself specified in the ATableName parameter value.
Returns true if the action was successful, false otherwise.

IDExists(ID: string);
Returns if the an object with the specified ID parameter value exists in the Table specified with the TableName property value.

SetRules: Boolean;
Submits the data from the GeneralRules and IndexRules to the Firebase Database.
Returns true if the action was successful, false otherwise.

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 an INI file, registry key or database
- Load the tokens from an INI file or registry key

The component has methods:

LoadTokens: load & decrypt access and refresh token from INI file or registry

SaveTokens: encrypt and save access and refresh tokens to INI file or registry

TestTokens: Boolean: performs a test if the access token is still accepted

RefreshAccess: Boolean: tries to get a new access token with the refresh token

and the properties:

- PersistTokens: TPersistTokens
- TokensAsString: string; encrypted string holding all token values

PersistTokens has following properties:

DataSource: TDataSource : datasource referring to a dataset where tokens can be persisted
Location: TPersistLocation : can be INI file, registry or database
Key: string : When the location is INI file this holds the INI filename otherwise the registry key name
Section: string : When the location is INI file this holds the INI file section name, otherwise the registry value name

When database is selected, the component will try to persist following values in the dataset with following field names:

ACCESS_TOKEN : string field : access token
AUTH_TOKEN: string field : authentication token
ACCESS_TOKEN_SECRET: string field : access token secret
AUTH_TOKEN_SECRET : string field : authentication token secret
REFRESH_TOKEN : string field : refresh token
EXTRA_DATA: string field : any extra data to persist when a specific cloud service needs this

Note that if the field is not found, the value won't be persisted, so it is not required to persist all values, the minimum value will be the access token.

A typical flow is:

```
var
  acc: boolean;

if Storage.App.Key <> '' then
begin
  // load tokens from registry
  Storage.LoadTokens;
  // test if the access token is still accepted
  acc := Storage.TestTokens;
  // when not, try to get a new access token with the refresh token
  if not acc then
    acc := Storage.RefreshAccess;
  // when no new access token can be obtained, do new authentication
  if not acc then
    Storage.DoAuth
end
```

and the access token, refresh token is saved from the OnReceivedAccessToken event:

```
procedure TForm1.CloudAccess1ReceivedAccessToken(Sender: TObject);
var
  cs: TCloudStorage;
begin
  Authenticated := true;

  if (Sender is TCloudStorage) then
  begin
    cs := Sender as TCloudStorage;
    cs.SaveTokens;
  end;
end;
```

The property TokensAsString can be used to save and load the token values in a DB field for example.

This can also be simplified by using the CloudService.Connect function that will perform the LoadTokens / TestTokens / Refresh all at once and when needed, start a new authentication / authorization. It is as such important that before calling CloudService.Connect, the application key and secret is set as well as the location of the token persistence.

When this simplified method is chosen, the code becomes:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  AdvOneDrive1.Connect;
end;

procedure TForm1.AdvOneDrive1Connected(Sender: TObject);
begin
  // Call a form method (Init) that initializes controls on the form
  // when a connection is made
  Init;
end;
```

Tips and FAQ

Logging

For debugging purposes, it can be interesting to see the log of all HTTPS access. Set CloudComponent.Logging = true and by default a log file will be generated under \My Documents. To override the default log file, set the filename with the public property CloudComponent.LogFileName: string;

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 specifiers wl.skydrive_update from the TAdvSkyDrive scopes or remove from https://www.googleapis.com/auth/drive.file from the TAdvGDrive scopes.