'''

gdrive (Google Drive ) for KODI / XBMC Plugin

Copyright (C) 2013-2016 ddurdle

This program is free software: you can redistribute it and/or modify

it under the terms of the GNU General Public License as published by

the Free Software Foundation, either version 3 of the License, or

(at your option) any later version.

This program is distributed in the hope that it will be useful,

but WITHOUT ANY WARRANTY; without even the implied warranty of

MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the

GNU General Public License for more details.

You should have received a copy of the GNU General Public License

along with this program. If not, see <http://www.gnu.org/licenses/>.

'''

# cloudservice - required python modules

import os

import re

import urllib, urllib2

import cookielib

import unicodedata

# cloudservice - standard modules

from cloudservice import cloudservice

from resources.lib import authorization

from resources.lib import folder

from resources.lib import file

from resources.lib import package

from resources.lib import mediaurl

from resources.lib import crashreport

from resources.lib import cache

# cloudservice - standard XBMC modules

import xbmc, xbmcaddon, xbmcgui, xbmcplugin

# global variables

#addon = xbmcaddon.Addon(id='plugin.video.gdrive-testing')

#addon\_dir = xbmc.translatePath( addon.getAddonInfo('path') )

PROTOCOL = 'https://'

SERVICE\_NAME = 'dmdgdrive'

#

# Google Docs API 3 implementation of Google Drive

#

class gdrive(cloudservice):

AUDIO = 1

VIDEO = 2

PICTURE = 3

PROTOCOL = 'https://'

# magic numbers

MEDIA\_TYPE\_MUSIC = 1

MEDIA\_TYPE\_VIDEO = 2

MEDIA\_TYPE\_PICTURE = 3

MEDIA\_TYPE\_FOLDER = 0

CACHE\_TYPE\_MEMORY = 0

CACHE\_TYPE\_DISK = 1

CACHE\_TYPE\_STREAM = 2

API\_VERSION = '3.0'

##

# initialize (save addon, instance name, user agent)

##

def \_\_init\_\_(self, PLUGIN\_URL, addon, instanceName, user\_agent, settings, authenticate=True, useWRITELY=False):

self.PLUGIN\_URL = PLUGIN\_URL

self.addon = addon

self.instanceName = instanceName

self.protocol = 1

self.integratedPlayer = False

self.settings = settings

# gdrive specific \*\*\*

self.decrypt = False

self.useWRITELY = useWRITELY

#\*\*\*

self.crashreport = crashreport.crashreport(self.addon)

# self.crashreport.sendError('test','test')

try:

username = self.addon.getSetting(self.instanceName+'\_username')

except:

username = ''

self.authorization = authorization.authorization(username)

self.cookiejar = cookielib.CookieJar()

self.user\_agent = user\_agent

# gdrive specific \*\*\*

if (authenticate == True and not self.authorization.loadToken(self.instanceName,addon, 'auth\_writely')) or (not self.authorization.loadToken(self.instanceName,addon, 'auth\_wise')):

self.login()

xbmcgui.Dialog().ok(addon.getLocalizedString(30000), addon.getLocalizedString(30153), addon.getLocalizedString(30154))

#\*\*\*

self.cache = cache.cache()

##

# perform login

##

def login(self):

services = ['writely', 'wise']

for service in services:

url = PROTOCOL + 'www.google.com/accounts/ClientLogin'

header = { 'User-Agent' : self.user\_agent }

values = {

'Email' : self.authorization.username,

'Passwd' : self.addon.getSetting(self.instanceName+'\_password'),

'accountType' : 'HOSTED\_OR\_GOOGLE',

'source' : SERVICE\_NAME,

'service' : service

}

req = urllib2.Request(url, urllib.urlencode(values), header)

# try login

try:

response = urllib2.urlopen(req)

except urllib2.URLError, e:

# if e.code == 403:

#login denied

# xbmcgui.Dialog().ok(self.addon.getLocalizedString(30000), self.addon.getLocalizedString(30017))

xbmc.log(self.addon.getAddonInfo('name') + ': ' + str(e), xbmc.LOGERROR)

return

response\_data = response.read()

response.close()

# retrieve authorization token

for r in re.finditer('SID=(.\*).+?' +

'LSID=(.\*).+?' +

'Auth=(.\*).+?' ,

response\_data, re.DOTALL):

sid,lsid,auth = r.groups()

self.authorization.setToken('auth\_'+service,auth)

return

##

# return the appropriate "headers" for Google Drive requests that include 1) user agent, 2) authorization token, 3) api version

# returns: list containing the header

##

def getHeadersList(self, forceWritely=True):

#effective 2014/02, video stream calls require a wise token instead of writely token

if forceWritely == True:

return { 'User-Agent' : self.user\_agent, 'Authorization' : 'GoogleLogin auth=%s' % self.authorization.getToken('auth\_writely'), 'GData-Version' : self.API\_VERSION }

else:

return { 'User-Agent' : self.user\_agent, 'Authorization' : 'GoogleLogin auth=%s' % self.authorization.getToken('auth\_wise'), 'GData-Version' : self.API\_VERSION }

def setDecrypt(self):

self.decrypt = True

##

# return the appropriate "headers" for Google Drive requests that include 1) user agent, 2) authorization token, 3) api version

# returns: URL-encoded header string

##

def getHeadersEncoded(self, forceWritely=True):

return urllib.urlencode(self.getHeadersList(forceWritely))

##

# retrieve a list of videos, using playback type stream

# parameters: prompt for video quality (optional), cache type (optional)

# returns: list of videos

##

def getMediaList(self, folderName=False, title=False, contentType=7):

# retrieve all items

url = PROTOCOL+'docs.google.com/feeds/default/private/full'

if folderName==False and title==False:

url = url + '?showfolders=false'

elif title != False:

params = urllib.urlencode({'title': title, 'title-exact': 'false'})

url = PROTOCOL+'docs.google.com/feeds/default/private/full?' + params

# retrieve root items

elif folderName == 'STARRED-FILES':

url = PROTOCOL+'docs.google.com/feeds/default/private/full/-/starred'

elif folderName == 'STARRED-FILESFOLDERS':

url = PROTOCOL+'docs.google.com/feeds/default/private/full/-/starred?showfolders=true'

elif folderName == 'STARRED-FOLDERS':

url = PROTOCOL+'docs.google.com/feeds/default/private/full/-/folder/starred'

elif folderName == 'SHARED':

params = urllib.urlencode({'q': 'sharedWithMe=true'})

url = PROTOCOL+'docs.google.com/feeds/default/private/full/-/folder/?showfolders=true&shardWithMe'

elif folderName == '':

url = url + '/folder%3Aroot/contents'

# retrieve folder items

else:

url = url + '/folder%3A'+folderName+'/contents'

mediaFiles = []

while True:

req = urllib2.Request(url, None, self.getHeadersList())

# if action fails, validate login

try:

response = urllib2.urlopen(req)

except urllib2.URLError, e:

if e.code == 403 or e.code == 401:

self.login()

req = urllib2.Request(url, None, self.getHeadersList())

try:

response = urllib2.urlopen(req)

except urllib2.URLError, e:

xbmc.log(self.addon.getAddonInfo('name') + ': ' + str(e), xbmc.LOGERROR)

self.crashreport.sendError('getMediaList',str(e))

return

else:

xbmc.log(self.addon.getAddonInfo('name') + ': ' + str(e), xbmc.LOGERROR)

self.crashreport.sendError('getMediaList',str(e))

return

response\_data = response.read()

response.close()

# parsing page for videos

# video-entry

for r in re.finditer('\<entry[^\>]+\>(.\*?)\<\/entry\>' ,response\_data, re.DOTALL):

entry = r.group(1)

# fetch folder

for r in re.finditer('\<gd\:resourceId\>([^\:]\*)\:?([^\<]\*)\</gd:resourceId\>' ,

entry, re.DOTALL):

resourceType,resourceID = r.groups()

# entry is a folder

if (resourceType == 'folder'):

for q in re.finditer('<(title)>([^<]+)</title>' ,

entry, re.DOTALL):

titleType, title = q.groups()

# gdrive decryption specific \*\*\*

if 0:

import base64

try:

title = base64.b64decode(title)

except:

pass

#\*\*\*

media = package.package(None,folder.folder(resourceID,title))

mediaFiles.append(media)

# entry is NOT a folder

else:

processed =0

# fetch video title, download URL and docid for stream link

# Google Drive API format

for r in re.finditer('<title>([^<]+)</title><content type=\'(video)\/[^\']+\' src=\'([^\']+)\'.\*?rel=\'http://schemas.google.com/docs/2007/thumbnail\' type=\'image/[^\']+\' href=\'([^\']+)\'.\*?\;docid=([^\&]+)\&.\*?<gd:quotaBytesUsed>(\d+)</gd:quotaBytesUsed>' ,

entry, re.DOTALL):

title,mediaType,url,thumbnail,docid,fileSize = r.groups()

mediaFile = file.file(docid, title, title, self.MEDIA\_TYPE\_VIDEO, '', thumbnail, size=fileSize)

media = package.package(mediaFile,folder.folder('',''))

media.setMediaURL(mediaurl.mediaurl(url, '','',''))

mediaFiles.append(media)

#\*\*\*

processed = 1

if processed == 0:

#video that can't be processed (no thumbnail)

for r in re.finditer('<title>([^<]+)</title><content type=\'(video)\/[^\']+\' src=\'([^\']+)\'.+?\;docid=([^\&]+)\&.\*?\<gd\:quotaBytesUsed\>(\d+)\</gd\:quotaBytesUsed\>' ,

entry, re.DOTALL):

title,mediaType,url,docid,fileSize = r.groups()

mediaFile = file.file(docid, title, title, self.MEDIA\_TYPE\_VIDEO, '', '', size=fileSize)

media = package.package(mediaFile,folder.folder('',''))

media.setMediaURL(mediaurl.mediaurl(url, '','',''))

mediaFiles.append(media)

processed = 1

if processed == 0:

#for playing video.google.com videos linked to your google drive account

# Google Docs & Google Video API format

for r in re.finditer('<title>([^<]+)</title><link rel=\'alternate\' type=\'text/html\' href=\'([^\']+).+?rel=\'http://schemas.google.com/docs/2007/thumbnail\' type=\'image/[^\']+\' href=\'([^\']+)\'' ,

entry, re.DOTALL):

title,url,thumbnail = r.groups()

mediaFile = file.file(title, title, title, self.MEDIA\_TYPE\_VIDEO, '', thumbnail)

media = package.package(mediaFile,folder.folder('',''))

media.setMediaURL(mediaurl.mediaurl(url, '','',''))

mediaFiles.append(media)

processed = 1

if processed == 0:

# audio

for r in re.finditer('<title>([^<]+)</title><content type=\'audio\/[^\']+\' src=\'([^\']+)\'' ,

entry, re.DOTALL):

title,url = r.groups()

mediaFile = file.file(title, title, title, self.MEDIA\_TYPE\_MUSIC, '', '')

media = package.package(mediaFile,folder.folder('',''))

media.setMediaURL(mediaurl.mediaurl(url, '','',''))

mediaFiles.append(media)

processed = 1

if processed == 0:

# audio

for r in re.finditer('<title>([^<]+)</title><content type=\'application\/x-flac\' src=\'([^\']+)\'' ,

entry, re.DOTALL):

title,url = r.groups()

# there is no steaming for audio (?), so "download to stream"

mediaFile = file.file(title, title, title, self.MEDIA\_TYPE\_VIDEO, '', '')

media = package.package(mediaFile,folder.folder('',''))

media.setMediaURL(mediaurl.mediaurl(url, '','',''))

mediaFiles.append(media)

processed = 1

#\*\*\*

if processed == 0:

# pictures

for r in re.finditer('<title>([^<]+)</title><content type=\'image\/[^\']+\' src=\'([^\']+)\'.+?rel=\'http://schemas.google.com/docs/2007/thumbnail\' type=\'image/[^\']+\' href=\'([^\']+)\'' ,

entry, re.DOTALL):

title,url,thumbnail = r.groups()

# there is no steaming for audio (?), so "download to stream"

cleanURL = re.sub('---', '', url)

cleanURL = re.sub('&amp;', '---', cleanURL)

mediaFile = file.file(title, title, title, self.MEDIA\_TYPE\_PICTURE, '', thumbnail)

media = package.package(mediaFile,folder.folder('',''))

media.setMediaURL(mediaurl.mediaurl(cleanURL, '','',''))

mediaFiles.append(media)

processed = 1

if processed == 0:

# pictures

for r in re.finditer('<title>([^<]+)</title><content type=\'application\/octet-stream\' src=\'([^\']+)\'' ,

entry, re.DOTALL):

title,url = r.groups()

# there is no steaming for audio (?), so "download to stream"

cleanURL = re.sub('---', '', url)

cleanURL = re.sub('&amp;', '---', cleanURL)

#\*\*\*

mediaFile = file.file(title, title, title, self.MEDIA\_TYPE\_PICTURE, '', '')

media = package.package(mediaFile,folder.folder('',''))

media.setMediaURL(mediaurl.mediaurl(cleanURL, '','',''))

mediaFiles.append(media)

processed = 1

# look for more pages of videos

nextURL = ''

for r in re.finditer('<link rel=\'next\' type=\'[^\']+\' href=\'([^\']+)\'' ,

response\_data, re.DOTALL):

nextURL = r.groups()

# are there more pages to process?

if nextURL == '':

break

else:

url = nextURL[0]

return mediaFiles

##

# retrieve a list of videos, using playback type stream

# parameters: cache type (optional)

# returns: list of videos

##

def downloadFolder(self,path,folder, context):

# retrieve all items

url = PROTOCOL+'docs.google.com/feeds/default/private/full'

# retrieve root items

if folder == '':

url = url + '/folder%3Aroot/contents'

# retrieve folder items

else:

url = url + '/folder%3A'+folder+'/contents'

import xbmcvfs

xbmcvfs.mkdir(path + '/'+folder)

while True:

req = urllib2.Request(url, None, self.getHeadersList())

# if action fails, validate login

try:

response = urllib2.urlopen(req)

except urllib2.URLError, e:

if e.code == 403 or e.code == 401:

self.login()

req = urllib2.Request(url, None, self.getHeadersList())

try:

response = urllib2.urlopen(req)

except urllib2.URLError, e:

xbmc.log(self.addon.getAddonInfo('name') + ': ' + str(e), xbmc.LOGERROR)

self.crashreport.sendError('downloadFolder',str(e))

return

else:

xbmc.log(self.addon.getAddonInfo('name') + ': ' + str(e), xbmc.LOGERROR)

self.crashreport.sendError('downloadFolder',str(e))

return

response\_data = response.read()

response.close()

# video-entry

for r in re.finditer('\<entry[^\>]+\>(.\*?)\<\/entry\>' ,response\_data, re.DOTALL):

entry = r.group(1)

# fetch folder

for r in re.finditer('\<gd\:resourceId\>([^\:]\*)\:?([^\<]\*)\</gd:resourceId\>' ,

entry, re.DOTALL):

resourceType,resourceID = r.groups()

# entry is NOT a folder

if not (resourceType == 'folder'):

if context != self.MEDIA\_TYPE\_PICTURE:

# fetch video title, download URL and docid for stream link

# Google Drive API format

for r in re.finditer('<title>([^<]+)</title><content type=\'(video)\/[^\']+\' src=\'([^\']+)\'.+?rel=\'http://schemas.google.com/docs/2007/thumbnail\' type=\'image/[^\']+\' href=\'([^\']+)\'' ,

entry, re.DOTALL):

title,mediaType,url,thumbnail = r.groups()

#for playing video.google.com videos linked to your google drive account

# Google Docs & Google Video API format

for r in re.finditer('<title>([^<]+)</title><link rel=\'alternate\' type=\'text/html\' href=\'([^\']+).+?rel=\'http://schemas.google.com/docs/2007/thumbnail\' type=\'image/[^\']+\' href=\'([^\']+)\'' ,

entry, re.DOTALL):

title,url,thumbnail = r.groups()

# audio

for r in re.finditer('<title>([^<]+)</title><content type=\'audio\/[^\']+\' src=\'([^\']+)\'' ,

entry, re.DOTALL):

title,url = r.groups()

elif context == self.MEDIA\_TYPE\_PICTURE:

# pictures

for r in re.finditer('<title>([^<]+)</title><content type=\'image\/[^\']+\' src=\'([^\']+)\'' ,

entry, re.DOTALL):

title,url = r.groups()

url = re.sub('&amp;', '&', url)

if not os.path.exists(path + '/'+folder+'/'+title):

self.downloadPicture(url,path +'/' + folder + '/' + title)

# pictures

for r in re.finditer('<title>([^<]+)</title><content type=\'application\/[^\']+\' src=\'([^\']+)\'' ,

entry, re.DOTALL):

title,url = r.groups()

url = re.sub('&amp;', '&', url)

if not os.path.exists(path + '/'+folder+'/'+title):

self.downloadPicture(url,path +'/' + folder + '/' + title)

# look for more pages of videos

nextURL = ''

for r in re.finditer('<link rel=\'next\' type=\'[^\']+\' href=\'([^\']+)\'' ,

response\_data, re.DOTALL):

nextURL = r.groups()

# are there more pages to process?

if nextURL == '':

break

else:

url = nextURL[0]

##

# retrieve a list of videos, using playback type stream

# parameters: cache type (optional)

# returns: list of videos

##

def decryptFolder(self,key,path,folder):

# retrieve all items

url = PROTOCOL+'docs.google.com/feeds/default/private/full'

# retrieve root items

if folder == '':

url = url + '/folder%3Aroot/contents'

# retrieve folder items

else:

url = url + '/folder%3A'+folder+'/contents'

import xbmcvfs

xbmcvfs.mkdir(path + '/'+folder)

while True:

req = urllib2.Request(url, None, self.getHeadersList())

# if action fails, validate login

try:

response = urllib2.urlopen(req)

except urllib2.URLError, e:

if e.code == 403 or e.code == 401:

self.login()

req = urllib2.Request(url, None, self.getHeadersList())

try:

response = urllib2.urlopen(req)

except urllib2.URLError, e:

xbmc.log(self.addon.getAddonInfo('name') + ': ' + str(e), xbmc.LOGERROR)

self.crashreport.sendError('decryptFolder',str(e))

return

else:

xbmc.log(self.addon.getAddonInfo('name') + ': ' + str(e), xbmc.LOGERROR)

self.crashreport.sendError('decryptFolder',str(e))

return

response\_data = response.read()

response.close()

downloadList = []

# video-entry

for r in re.finditer('\<entry[^\>]+\>(.\*?)\<\/entry\>' ,response\_data, re.DOTALL):

entry = r.group(1)

# fetch folder

for r in re.finditer('\<gd\:resourceId\>([^\:]\*)\:?([^\<]\*)\</gd:resourceId\>' ,

entry, re.DOTALL):

resourceType,resourceID = r.groups()

# entry is NOT a folder

if not (resourceType == 'folder'):

# fetch video title, download URL and docid for stream link

# Google Drive API format

for r in re.finditer('<title>([^<]+)</title><content type=\'(video)\/[^\']+\' src=\'([^\']+)\'.+?rel=\'http://schemas.google.com/docs/2007/thumbnail\' type=\'image/[^\']+\' href=\'([^\']+)\'' ,

entry, re.DOTALL):

title,mediaType,url,thumbnail = r.groups()

#for playing video.google.com videos linked to your google drive account

# Google Docs & Google Video API format

for r in re.finditer('<title>([^<]+)</title><link rel=\'alternate\' type=\'text/html\' href=\'([^\']+).+?rel=\'http://schemas.google.com/docs/2007/thumbnail\' type=\'image/[^\']+\' href=\'([^\']+)\'' ,

entry, re.DOTALL):

title,url,thumbnail = r.groups()

# audio

for r in re.finditer('<title>([^<]+)</title><content type=\'audio\/[^\']+\' src=\'([^\']+)\'' ,

entry, re.DOTALL):

title,url = r.groups()

# pictures

for r in re.finditer('<title>([^<]+)</title><content type=\'image\/[^\']+\' src=\'([^\']+)\'' ,

entry, re.DOTALL):

title,url = r.groups()

url = re.sub('&amp;', '&', url)

filename = path + '/'+folder+'/'+ encryption.decrypt(title)

if not os.path.exists(filename) or os.path.getsize(filename) == 0:

open(filename, 'a').close()

downloadList.append(downloadfile.downloadfile(url,filename))

#self.downloadDecryptPicture(key, url,filename)

# pictures

for r in re.finditer('<title>([^<]+)</title><content type=\'application\/[^\']+\' src=\'([^\']+)\'' ,

entry, re.DOTALL):

title,url = r.groups()

url = re.sub('&amp;', '&', url)

filename = path + '/'+folder+'/'+ encryption.decrypt(title)

if not os.path.exists(filename) or os.path.getsize(filename) == 0:

open(filename, 'a').close()

downloadList.append(downloadfile.downloadfile(url,filename))

#self.downloadDecryptPicture(key, url,filename)

if downloadList:

for file in sorted(downloadList, key=lambda item: item.name):

self.downloadDecryptPicture(key, file.url,file.name)

# look for more pages of videos

nextURL = ''

for r in re.finditer('<link rel=\'next\' type=\'[^\']+\' href=\'([^\']+)\'' ,

response\_data, re.DOTALL):

nextURL = r.groups()

# are there more pages to process?

if nextURL == '':

break

else:

url = nextURL[0]

##

# retrieve a playback url

# returns: url

##

def getPlaybackCall(self, package=None, title='', isExact=True):

try:

pquality = int(self.addon.getSetting('preferred\_quality'))

pformat = int(self.addon.getSetting('preferred\_format'))

acodec = int(self.addon.getSetting('avoid\_codec'))

except :

pquality=-1

pformat=-1

acodec=-1

mediaURLs = []

docid = ''

if package is None and title != '':

# search by video title

if isExact == True:

params = urllib.urlencode({'title': title, 'title-exact': 'true'})

else:

params = urllib.urlencode({'title': title, 'title-exact': 'false'})

url = PROTOCOL+'docs.google.com/feeds/default/private/full?' + params

req = urllib2.Request(url, None, self.getHeadersList())

# if action fails, validate login

try:

response = urllib2.urlopen(req)

except urllib2.URLError, e:

if e.code == 403 or e.code == 401:

self.login()

req = urllib2.Request(url, None, self.getHeadersList())

try:

response = urllib2.urlopen(req)

except urllib2.URLError, e:

xbmc.log(self.addon.getAddonInfo('name') + ': ' + str(e), xbmc.LOGERROR)

self.crashreport.sendError('getPlaybackCall',str(e))

return

else:

xbmc.log(self.addon.getAddonInfo('name') + ': ' + str(e), xbmc.LOGERROR)

self.crashreport.sendError('getPlaybackCall',str(e))

return

response\_data = response.read()

response.close()

# fetch video title, download URL and docid for stream link

for r in re.finditer('\<entry[^\>]+\>(.\*?)\<\/entry\>' ,response\_data, re.DOTALL):

entry = r.group(1)

for q in re.finditer('<title>([^<]+)</title><content type=\'([^\/]+)\/[^\']+\' src=\'([^\']+)\'.\*\;docid=([^\&]+)\&' ,

entry, re.DOTALL):

title,mediaType,url,docid = q.groups()

#mediaURLs.append(url, 'original', 0, 3)

mediaURLs.append(mediaurl.mediaurl(url, '9999 - original', 0, 9999))

else:

docid = package.file.id

url = PROTOCOL+'docs.google.com/feeds/default/private/full/file:' + docid

req = urllib2.Request(url, None, self.getHeadersList())

# if action fails, validate login

try:

response = urllib2.urlopen(req)

except urllib2.URLError, e:

if e.code == 403 or e.code == 401:

self.login()

req = urllib2.Request(url, None, self.getHeadersList())

try:

response = urllib2.urlopen(req)

except urllib2.URLError, e:

xbmc.log(self.addon.getAddonInfo('name') + ': ' + str(e), xbmc.LOGERROR)

self.crashreport.sendError('getPlaybackCall',str(e))

return

else:

xbmc.log(self.addon.getAddonInfo('name') + ': ' + str(e), xbmc.LOGERROR)

self.crashreport.sendError('getPlaybackCall',str(e))

return

response\_data = response.read()

response.close()

# fetch video title, download URL and docid for stream link

for r in re.finditer('\<entry[^\>]+\>(.\*?)\<\/entry\>' ,response\_data, re.DOTALL):

entry = r.group(1)

for q in re.finditer('<title>([^<]+)</title><content type=\'([^\/]+)\/[^\']+\' src=\'([^\']+)\'.\*\;docid=([^\&]+)\&' ,

entry, re.DOTALL):

title,mediaType,url,docid = q.groups()

#mediaURLs.append(url, 'original', 0, 3)

mediaURLs.append(mediaurl.mediaurl(url, '9999 - original', 0, 9999))

if docid != '':

# player using docid

params = urllib.urlencode({'docid': docid})

url = PROTOCOL+'docs.google.com/get\_video\_info?docid=' + str(docid)

req = urllib2.Request(url, None, self.getHeadersList(self.useWRITELY))

# else:

#try to use no authorization token (for pubic URLs)

# header = { 'User-Agent' : self.user\_agent, 'GData-Version' : self.API\_VERSION }

# req = urllib2.Request(url, None, header)

# if action fails, validate login

try:

response = urllib2.urlopen(req)

except urllib2.URLError, e:

if e.code == 403 or e.code == 401:

self.login()

req = urllib2.Request(url, None, self.getHeadersList(self.useWRITELY))

try:

response = urllib2.urlopen(req)

except urllib2.URLError, e:

xbmc.log(self.addon.getAddonInfo('name') + ': ' + str(e), xbmc.LOGERROR)

self.crashreport.sendError('getPlaybackCall',str(e))

return

else:

xbmc.log(self.addon.getAddonInfo('name') + ': ' + str(e), xbmc.LOGERROR)

self.crashreport.sendError('getPlaybackCall',str(e))

return

response\_data = response.read()

response.close()

# decode resulting player URL (URL is composed of many sub-URLs)

urls = response\_data

urls = urllib.unquote(urllib.unquote(urllib.unquote(urllib.unquote(urllib.unquote(urls)))))

urls = re.sub('\\\\u003d', '=', urls)

urls = re.sub('\\\\u0026', '&', urls)

serviceRequired = ''

for r in re.finditer('ServiceLogin\?(service)=([^\&]+)\&' ,

urls, re.DOTALL):

(service, serviceRequired) = r.groups()

#effective 2014/02, video stream calls require a wise token instead of writely token

#backward support for account not migrated to the 2014/02 change

if serviceRequired == 'writely':

self.useWRITELY = True

req = urllib2.Request(url, None, self.getHeadersList(self.useWRITELY))

try:

response = urllib2.urlopen(req)

except urllib2.URLError, e:

if e.code == 403 or e.code == 401:

self.login()

req = urllib2.Request(url, None, self.getHeadersList(self.useWRITELY))

try:

response = urllib2.urlopen(req)

except urllib2.URLError, e:

xbmc.log(self.addon.getAddonInfo('name') + ': ' + str(e), xbmc.LOGERROR)

self.crashreport.sendError('getPlaybackCall',str(e))

return

else:

xbmc.log(self.addon.getAddonInfo('name') + ': ' + str(e), xbmc.LOGERROR)

self.crashreport.sendError('getPlaybackCall',str(e))

return

response\_data = response.read()

response.close()

# decode resulting player URL (URL is composed of many sub-URLs)

urls = response\_data

urls = urllib.unquote(urllib.unquote(urllib.unquote(urllib.unquote(urllib.unquote(urls)))))

urls = re.sub('\\\\u003d', '=', urls)

urls = re.sub('\\\\u0026', '&', urls)

serviceRequired = ''

for r in re.finditer('ServiceLogin\?(service)=([^\&]+)\&' ,

urls, re.DOTALL):

(service, serviceRequired) = r.groups()

if serviceRequired != '':

log('an unexpected service token is required: %s' % (serviceRequired), True)

elif serviceRequired == 'wise':

self.useWRITELY = False

req = urllib2.Request(url, None, self.getHeadersList(self.useWRITELY))

try:

response = urllib2.urlopen(req)

except urllib2.URLError, e:

if e.code == 403 or e.code == 401:

self.login()

req = urllib2.Request(url, None, self.getHeadersList(self.useWRITELY))

try:

response = urllib2.urlopen(req)

except urllib2.URLError, e:

xbmc.log(self.addon.getAddonInfo('name') + ': ' + str(e), xbmc.LOGERROR)

self.crashreport.sendError('getPlaybackCall',str(e))

return

else:

xbmc.log(self.addon.getAddonInfo('name') + ': ' + str(e), xbmc.LOGERROR)

self.crashreport.sendError('getPlaybackCall',str(e))

return

response\_data = response.read()

response.close()

# decode resulting player URL (URL is composed of many sub-URLs)

urls = response\_data

urls = urllib.unquote(urllib.unquote(urllib.unquote(urllib.unquote(urllib.unquote(urls)))))

urls = re.sub('\\\\u003d', '=', urls)

urls = re.sub('\\\\u0026', '&', urls)

serviceRequired = ''

for r in re.finditer('ServiceLogin\?(service)=([^\&]+)\&' ,

urls, re.DOTALL):

(service, serviceRequired) = r.groups()

if serviceRequired != '':

log('an unexpected service token is required: %s' % (serviceRequired), True)

elif serviceRequired != '':

log('an unexpected service token is required: %s' % (serviceRequired), True)

# do some substitutions to make anchoring the URL easier

urls = re.sub('\&url\='+PROTOCOL, '\@', urls)

# itag code reference http://en.wikipedia.org/wiki/YouTube#Quality\_and\_codecs

#itag\_dict = {1080: ['137', '37', '46'], 720: ['22', '136', '45'],

# 480: ['135', '59', '44', '35'], 360: ['43', '134', '34', '18', '6'],

# 240: ['133', '5', '36'], 144: ['160', '17']}

# <setting id="preferred\_quality" type="enum" label="30011" values="perfer best (1080,720,<720)|prefer 720 (720,<720,>720)|prefer SD (480,<480)" default="0" />

# <setting id="preferred\_format" type="enum" label="30012" values="MP4,WebM,flv|MP4,flv,WebM|flv,WebM,MP4|flv,MP4,WebM|WebM,MP4,flv|WebM,flv,MP4" default="0" />

# <setting id="avoid\_codec" type="enum" label="30013" values="none|VP8/vorbis" default="0"/>

itagDB={}

containerDB = {'x-flv':'flv', 'webm': 'WebM', 'mp4;+codecs="avc1.42001E,+mp4a.40.2"': 'MP4'}

for r in re.finditer('(\d+)/(\d+)x(\d+)/(\d+/\d+/\d+)\&?\,?' ,

urls, re.DOTALL):

(itag,resolution1,resolution2,codec) = r.groups()

if codec == '9/0/115':

itagDB[itag] = {'resolution': resolution2, 'codec': 'h.264/aac'}

elif codec == '99/0/0':

itagDB[itag] = {'resolution': resolution2, 'codec': 'VP8/vorbis'}

else:

itagDB[itag] = {'resolution': resolution2}

# fetch format type and quality for each stream

count=0

for r in re.finditer('\@([^\@]+)' ,urls):

videoURL = r.group(1)

for q in re.finditer('itag\=(\d+).\*?type\=video\/([^\&]+)\&quality\=(\w+)' ,

videoURL, re.DOTALL):

(itag,container,quality) = q.groups()

count = count + 1

order=0

if pquality > -1 or pformat > -1 or acodec > -1:

if int(itagDB[itag]['resolution']) == 1080:

if pquality == 0:

order = order + 1000

elif pquality == 1:

order = order + 3000

elif pquality == 3:

order = order + 9000

elif int(itagDB[itag]['resolution']) == 720:

if pquality == 0:

order = order + 2000

elif pquality == 1:

order = order + 1000

elif pquality == 3:

order = order + 9000

elif int(itagDB[itag]['resolution']) == 480:

if pquality == 0:

order = order + 3000

elif pquality == 1:

order = order + 2000

elif pquality == 3:

order = order + 1000

elif int(itagDB[itag]['resolution']) < 480:

if pquality == 0:

order = order + 4000

elif pquality == 1:

order = order + 3000

elif pquality == 3:

order = order + 2000

try:

if itagDB[itag]['codec'] == 'VP8/vorbis':

if acodec == 1:

order = order + 90000

else:

order = order + 10000

except :

order = order + 30000

try:

if containerDB[container] == 'MP4':

if pformat == 0 or pformat == 1:

order = order + 100

elif pformat == 3 or pformat == 4:

order = order + 200

else:

order = order + 300

elif containerDB[container] == 'flv':

if pformat == 2 or pformat == 3:

order = order + 100

elif pformat == 1 or pformat == 5:

order = order + 200

else:

order = order + 300

elif containerDB[container] == 'WebM':

if pformat == 4 or pformat == 5:

order = order + 100

elif pformat == 0 or pformat == 1:

order = order + 200

else:

order = order + 300

else:

order = order + 100

except :

pass

try:

mediaURLs.append(mediaurl.mediaurl(PROTOCOL + videoURL, str(order+count) + ' - ' + itagDB[itag]['resolution'] + ' - ' + containerDB[container] + ' - ' + itagDB[itag]['codec'], 0, order+count))

# videos[str(order+count) + ' - ' + itagDB[itag]['resolution'] + ' - ' + containerDB[container] + ' - ' + itagDB[itag]['codec']] = PROTOCOL + videoURL

except KeyError:

mediaURLs.append(mediaurl.mediaurl(PROTOCOL + videoURL, str(order+count) + ' - ' + itagDB[itag]['resolution'] + ' - ' + container, 0, order+count))

# videos[str(order+count) + ' - ' + itagDB[itag]['resolution'] + ' - ' + container] = PROTOCOL + videoURL

return (mediaURLs,package)

def downloadPicture(self,url, file):

req = urllib2.Request(url, None, self.getHeadersList())

# import xbmcvfs

# f = xbmcvfs.File(file, 'w')

# f = open(file,'wb')

# if action fails, validate login

try:

# f.write(urllib2.urlopen(req).read())

# f.write(urllib2.urlopen(req).read())

# f.close()

open(file,'wb').write(urllib2.urlopen(req).read())

except urllib2.URLError, e:

self.login()

req = urllib2.Request(url, None, self.getHeadersList())

try:

open(file,'wb').write(urllib2.urlopen(req).read())

# f.write(urllib2.urlopen(req).read())

# f.close()

except urllib2.URLError, e:

xbmc.log(self.addon.getAddonInfo('name') + ': ' + str(e), xbmc.LOGERROR)

self.crashreport.sendError('downloadPicture',str(e))

return

def downloadDecryptPicture(self,key,url, file):

req = urllib2.Request(url, None, self.getHeadersList())

# if action fails, validate login

try:

# open('/tmp/tmp','wb').write(urllib2.urlopen(req).read())

# encryption.decrypt\_file(key,'/tmp/tmp',file)

encryption.decrypt\_stream(key,urllib2.urlopen(req),file)

except urllib2.URLError, e:

if e.code == 403 or e.code == 401:

self.login()

req = urllib2.Request(url, None, self.getHeadersList())

try:

encryption.decrypt\_stream(key,urllib2.urlopen(req),file)

# open('/tmp/tmp','wb').write(urllib2.urlopen(req).read())

# encryption.decrypt\_file(key,'/tmp/tmp',file)

except urllib2.URLError, e:

xbmc.log(self.addon.getAddonInfo('name') + ': ' + str(e), xbmc.LOGERROR)

self.crashreport.sendError('downloadDecryptPicture',str(e))

return

else:

xbmc.log(self.addon.getAddonInfo('name') + ': ' + str(e), xbmc.LOGERROR)

self.crashreport.sendError('downloadDecryptPicture',str(e))

return

# for playing public URLs

def getPublicStream(self,url):

try:

pquality = int(self.addon.getSetting('preferred\_quality'))

pformat = int(self.addon.getSetting('preferred\_format'))

acodec = int(self.addon.getSetting('avoid\_codec'))

except :

pquality=-1

pformat=-1

acodec=-1

mediaURLs = []

#try to use no authorization token (for pubic URLs)

header = { 'User-Agent' : self.user\_agent, 'GData-Version' : self.API\_VERSION }

req = urllib2.Request(url, None, header)

try:

response = urllib2.urlopen(req)

except urllib2.URLError, e:

if e.code == 403 or e.code == 401:

self.login()

req = urllib2.Request(url, None, self.getHeadersList(self.useWRITELY))

try:

response = urllib2.urlopen(req)

except urllib2.URLError, e:

xbmc.log(self.addon.getAddonInfo('name') + ': ' + str(e), xbmc.LOGERROR)

self.crashreport.sendError('getPublicStream',str(e))

return

else:

xbmc.log(self.addon.getAddonInfo('name') + ': ' + str(e), xbmc.LOGERROR)

self.crashreport.sendError('getPublicStream',str(e))

return

response\_data = response.read()

response.close()

serviceRequired = ''

for r in re.finditer('ServiceLogin\?(service)=([^\&]+)\&' ,

response\_data, re.DOTALL):

(service, serviceRequired) = r.groups()

for r in re.finditer('AccountChooser.+?(service)=([^\']+)\'' ,

response\_data, re.DOTALL):

(service, serviceRequired) = r.groups()

#effective 2014/02, video stream calls require a wise token instead of writely token

#backward support for account not migrated to the 2014/02 change

if serviceRequired == 'writely':

self.useWRITELY = True

if (self.authorization.getToken('auth\_writely') == ''):

self.login();

req = urllib2.Request(url, None, self.getHeadersList(self.useWRITELY))

try:

response = urllib2.urlopen(req)

except urllib2.URLError, e:

if e.code == 403 or e.code == 401:

self.login()

req = urllib2.Request(url, None, self.getHeadersList(self.useWRITELY))

try:

response = urllib2.urlopen(req)

except urllib2.URLError, e:

xbmc.log(self.addon.getAddonInfo('name') + ': ' + str(e), xbmc.LOGERROR)

self.crashreport.sendError('getPublicStream',str(e))

return

else:

xbmc.log(self.addon.getAddonInfo('name') + ': ' + str(e), xbmc.LOGERROR)

self.crashreport.sendError('getPublicStream',str(e))

return

response\_data = response.read()

response.close()

serviceRequired = ''

for r in re.finditer('ServiceLogin\?(service)=([^\&]+)\&' ,

urls, re.DOTALL):

(service, serviceRequired) = r.groups()

if serviceRequired != '':

log('an unexpected service token is required: %s' % (serviceRequired), True)

elif serviceRequired == 'wise':

self.useWRITELY = False

if (self.authorization.getToken('auth\_wise') == ''):

self.login();

req = urllib2.Request(url, None, self.getHeadersList(self.useWRITELY))

try:

response = urllib2.urlopen(req)

except urllib2.URLError, e:

if e.code == 403 or e.code == 401:

self.login()

req = urllib2.Request(url, None, self.getHeadersList(self.useWRITELY))

try:

response = urllib2.urlopen(req)

except urllib2.URLError, e:

xbmc.log(self.addon.getAddonInfo('name') + ': ' + str(e), xbmc.LOGERROR)

self.crashreport.sendError('getPublicStream',str(e))

return

else:

xbmc.log(self.addon.getAddonInfo('name') + ': ' + str(e), xbmc.LOGERROR)

self.crashreport.sendError('getPublicStream',str(e))

return

response\_data = response.read()

response.close()

serviceRequired = ''

for r in re.finditer('ServiceLogin\?(service)=([^\&]+)\&' ,

urls, re.DOTALL):

(service, serviceRequired) = r.groups()

if serviceRequired != '':

log('an unexpected service token is required: %s' % (serviceRequired), True)

elif serviceRequired != '':

log('an unexpected service token is required: %s' % (serviceRequired), True)

for r in re.finditer('\"fmt\_list\"\,\"([^\"]+)\"' ,

response\_data, re.DOTALL):

fmtlist = r.group(1)

title = ''

for r in re.finditer('\"title\"\,\"([^\"]+)\"' ,

response\_data, re.DOTALL):

title = r.group(1)

#thumbnail

# downloadURL = ''

# for r in re.finditer('\,\[\,\"[^\"]+\"\,\"([^\"]+)\"' ,

# response\_data, re.DOTALL):

# downloadURL = r.group(1)

# downloadURL = re.sub('\\\\u003d', '=', downloadURL)

itagDB={}

containerDB = {'x-flv':'flv', 'webm': 'WebM', 'mp4;+codecs="avc1.42001E,+mp4a.40.2"': 'MP4'}

for r in re.finditer('(\d+)/(\d+)x(\d+)/(\d+/\d+/\d+)\&?\,?' ,

fmtlist, re.DOTALL):

(itag,resolution1,resolution2,codec) = r.groups()

if codec == '9/0/115':

itagDB[itag] = {'resolution': resolution2, 'codec': 'h.264/aac'}

elif codec == '99/0/0':

itagDB[itag] = {'resolution': resolution2, 'codec': 'VP8/vorbis'}

else:

itagDB[itag] = {'resolution': resolution2}

for r in re.finditer('\"url\_encoded\_fmt\_stream\_map\"\,\"([^\"]+)\"' ,

response\_data, re.DOTALL):

urls = r.group(1)

urls = urllib.unquote(urllib.unquote(urllib.unquote(urllib.unquote(urllib.unquote(urls)))))

urls = re.sub('\\\\u003d', '=', urls)

urls = re.sub('\\\\u0026', '&', urls)

# urls = re.sub('\d+\&url\='+self.PROTOCOL, '\@', urls)

urls = re.sub('\&url\='+self.PROTOCOL, '\@', urls)

# for r in re.finditer('\@([^\@]+)' ,urls):

# videoURL = r.group(0)

# videoURL1 = self.PROTOCOL + videoURL

# fetch format type and quality for each stream

count=0

for r in re.finditer('\@([^\@]+)' ,urls):

videoURL = r.group(1)

for q in re.finditer('itag\=(\d+).\*?type\=video\/([^\&]+)\&quality\=(\w+)' ,

videoURL, re.DOTALL):

(itag,container,quality) = q.groups()

count = count + 1

order=0

if pquality > -1 or pformat > -1 or acodec > -1:

if int(itagDB[itag]['resolution']) == 1080:

if pquality == 0:

order = order + 1000

elif pquality == 1:

order = order + 3000

elif pquality == 3:

order = order + 9000

elif int(itagDB[itag]['resolution']) == 720:

if pquality == 0:

order = order + 2000

elif pquality == 1:

order = order + 1000

elif pquality == 3:

order = order + 9000

elif int(itagDB[itag]['resolution']) == 480:

if pquality == 0:

order = order + 3000

elif pquality == 1:

order = order + 2000

elif pquality == 3:

order = order + 1000

elif int(itagDB[itag]['resolution']) < 480:

if pquality == 0:

order = order + 4000

elif pquality == 1:

order = order + 3000

elif pquality == 3:

order = order + 2000

try:

if itagDB[itag]['codec'] == 'VP8/vorbis':

if acodec == 1:

order = order + 90000

else:

order = order + 10000

except :

order = order + 30000

try:

if containerDB[container] == 'MP4':

if pformat == 0 or pformat == 1:

order = order + 100

elif pformat == 3 or pformat == 4:

order = order + 200

else:

order = order + 300

elif containerDB[container] == 'flv':

if pformat == 2 or pformat == 3:

order = order + 100

elif pformat == 1 or pformat == 5:

order = order + 200

else:

order = order + 300

elif containerDB[container] == 'WebM':

if pformat == 4 or pformat == 5:

order = order + 100

elif pformat == 0 or pformat == 1:

order = order + 200

else:

order = order + 300

else:

order = order + 100

except :

pass

try:

mediaURLs.append( mediaurl.mediaurl(self.PROTOCOL + videoURL, itagDB[itag]['resolution'] + ' - ' + containerDB[container] + ' - ' + itagDB[itag]['codec'], str(itagDB[itag]['resolution'])+ '\_' + str(order+count), order+count, title=title))

except KeyError:

mediaURLs.append(mediaurl.mediaurl(self.PROTOCOL + videoURL, itagDB[itag]['resolution'] + ' - ' + container, str(itagDB[itag]['resolution'])+ '\_' + str(order+count), order+count, title=title))

return mediaURLs

##

# retrieve a media file

# parameters: title of video, whether to prompt for quality/format (optional), cache type (optional)

##

def downloadMediaFile(self,url, title, fileSize):

opener = urllib2.build\_opener(urllib2.HTTPCookieProcessor(cookielib.CookieJar()))

opener.addheaders = [('User-Agent', self.user\_agent), ('Authorization' , 'GoogleLogin auth='+self.authorization.getToken('auth\_writely')), ('GData-Version' , self.API\_VERSION)]

request = urllib2.Request(url)

# if action fails, validate login

cachePercent = 0

try:

cachePercent = int(self.addon.getSetting('cache\_percent'))

except:

cachePercent = 10

if cachePercent < 1:

cachePercent = 1

elif cachePercent > 100:

cachePercent = 100

fileSize = (int)(fileSize)

if fileSize == '' or fileSize < 1000:

fileSize = 5000000

sizeDownload = fileSize \* (cachePercent\*0.01)

if sizeDownload < 1000000:

sizeDownload = 1000000

CHUNK = 0

try:

CHUNK = int(self.addon.getSetting('chunk\_size'))

except:

CHUNK = 32 \* 1024

if CHUNK < 1024:

CHUNK = 16 \* 1024

count = 0

path = ''

try:

path = self.addon.getSetting('cache\_folder')

except:

pass

import os.path

if not os.path.exists(path):

path = ''

while path == '':

path = xbmcgui.Dialog().browse(0,self.addon.getLocalizedString(30090), 'files','',False,False,'')

if not os.path.exists(path):

path = ''

else:

self.addon.setSetting('cache\_folder', path)

# if action fails, validate login

try:

response = opener.open(request)

except urllib2.URLError, e:

if e.code == 403 or e.code == 401:

self.login()

opener.addheaders = [('User-Agent', self.user\_agent), ('Authorization' , 'GoogleLogin auth='+self.authorization.getToken('auth\_writely')), ('GData-Version' , self.API\_VERSION)]

request = urllib2.Request(url)

try:

response = opener.open(request)

except urllib2.URLError, e:

xbmc.log(self.addon.getAddonInfo('name') + ': ' + str(e), xbmc.LOGERROR)

self.crashreport.sendError('downloadMediaFile',str(e))

return

else:

xbmc.log(self.addon.getAddonInfo('name') + ': ' + str(e), xbmc.LOGERROR)

self.crashreport.sendError('downloadMediaFile',str(e))

return

progress = xbmcgui.DialogProgress()

progress.create(self.addon.getLocalizedString(30000),self.addon.getLocalizedString(30035),title,'\n')

# with open(path + 'test.mp4', 'wb') as fp:

filename = 'cache.mp4'

fp = open(path + filename, 'wb')

downloadedBytes = 0

while sizeDownload > downloadedBytes:

progress.update((int)(float(downloadedBytes)/sizeDownload\*100),self.addon.getLocalizedString(30035),(str)(cachePercent) + ' ' +self.addon.getLocalizedString(30093),'\n')

chunk = response.read(CHUNK)

if not chunk: break

fp.write(chunk)

downloadedBytes = downloadedBytes + CHUNK

self.response = response

self.fp = fp

return path + filename

##

# retrieve a media file

# parameters: title of video, whether to prompt for quality/format (optional), cache type (optional)

##

def continuedownloadMediaFile(self, url):

CHUNK = 0

try:

CHUNK = int(self.addon.getSetting('chunk\_size'))

except:

CHUNK = 32 \* 1024

if CHUNK < 1024:

CHUNK = 16 \* 1024

# fp = open(url, 'a')

while True:

chunk = self.response.read(CHUNK)

if not chunk: break

self.fp.write(chunk)

self.fp.close()