'''

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 sys

import urllib, urllib2

import socket

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

from resources.lib import gSpreadsheets

KODI = True

if KODI:

# cloudservice - standard XBMC modules

import xbmc, xbmcaddon, xbmcgui, xbmcplugin, xbmcvfs

SERVICE\_NAME = 'dmdgdrive'

#

# Google Drive API 2 implementation of Google Drive

#

class gdrive(cloudservice):

AUDIO = 1

VIDEO = 2

PICTURE = 3

# magic numbers

MEDIA\_TYPE\_MUSIC = 1

MEDIA\_TYPE\_VIDEO = 2

MEDIA\_TYPE\_PICTURE = 3

MEDIA\_TYPE\_UNKNOWN = 4

MEDIA\_TYPE\_FOLDER = 0

API\_VERSION = '3.0'

PROTOCOL = 'https://'

API\_URL = PROTOCOL+'www.googleapis.com/drive/v2/'

##

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

##

def \_\_init\_\_(self, PLUGIN\_URL, addon, instanceName, user\_agent, settings, authenticate=True, gSpreadsheet=None, DBM=None):

self.integratedPlayer = False

self.PLUGIN\_URL = PLUGIN\_URL

self.addon = addon

self.instanceName = instanceName

self.protocol = 2

self.settings = settings

self.gSpreadsheet = gSpreadsheet

self.worksheetID = self.addon.getSetting(self.instanceName+'\_spreadsheet')

self.DBM = None

if KODI:

if authenticate == True:

self.type = int(addon.getSetting(instanceName+'\_type'))

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

try:

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

except:

username = ''

self.authorization = authorization.authorization(username)

else:

self.type = 2

self.crashreport = None

self.authorization = authorization.authorization(username)

import anydbm

self.DBM = anydbm.open(DBM,'c')

self.DBM.close()

self.cookiejar = cookielib.CookieJar()

self.user\_agent = user\_agent

# load the OAUTH2 tokens or force fetch if not set

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

if self.type ==4 or self.addon.getSetting(self.instanceName+'\_code'):

self.getToken(self.addon.getSetting(self.instanceName+'\_code'))

else:

xbmcgui.Dialog().ok(self.addon.getLocalizedString(30000), self.addon.getLocalizedString(30017), self.addon.getLocalizedString(30018))

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

#\*\*\*

self.cache = cache.cache()

if KODI:

self.cloudResume = self.addon.getSetting(self.instanceName+'\_resumepoint')

self.cloudSpreadsheet = self.addon.getSetting(self.instanceName+'\_spreadsheetname')

if self.cloudResume == '2':

if self.worksheetID == '':

try:

self.gSpreadsheet = gSpreadsheets.gSpreadsheets(self,addon, user\_agent)

spreadsheets = self.gSpreadsheet.getSpreadsheetList()

except:

pass

for title in spreadsheets.iterkeys():

if title == self.cloudSpreadsheet:#'CLOUD\_DB':

worksheets = self.gSpreadsheet.getSpreadsheetWorksheets(spreadsheets[title])

for worksheet in worksheets.iterkeys():

if worksheet == 'db':

self.worksheetID = worksheets[worksheet]

addon.setSetting(instanceName + '\_spreadsheet', self.worksheetID)

break

break

if self.gSpreadsheet is None:

self.gSpreadsheet = gSpreadsheets.gSpreadsheets(self,addon, user\_agent)

##

# get OAUTH2 access and refresh token for provided code

# parameters: OAUTH2 code

# returns: none

##

def getToken(self,code):

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

if (self.type == 2):

url = addon.getSetting(self.instanceName+'\_url')

values = {

'code' : code

}

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

elif (self.type == 3):

url = 'https://accounts.google.com/o/oauth2/token'

clientID =self.addon.getSetting(self.instanceName+'\_client\_id')

clientSecret = self.addon.getSetting(self.instanceName+'\_client\_secret')

header = { 'User-Agent' : self.user\_agent , 'Content-Type': 'application/x-www-form-urlencoded'}

req = urllib2.Request(url, 'code='+str(code)+'&client\_id='+str(clientID)+'&client\_secret='+str(clientSecret)+'&redirect\_uri=urn:ietf:wg:oauth:2.0:oob&grant\_type=authorization\_code', header)

elif (self.type ==1):

url = 'http://dmdsoftware.net/api/gdrive.php'

values = {

'code' : code

}

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

else:

url = 'https://script.google.com/macros/s/AKfycbw8fdhaq-WRVJXfOSMK5TZdVnzHvY4u41O1BfW9C8uAghMzNhM/exec'

values = {

'username' : self.authorization.username,

'passcode' : self.addon.getSetting(self.instanceName+'\_passcode')

}

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

xbmcgui.Dialog().ok(self.addon.getLocalizedString(30000), self.addon.getLocalizedString(30140), self.addon.getLocalizedString(30141))

# try login

try:

response = urllib2.urlopen(req)

except urllib2.URLError, e:

if e.code == 403:

#login issue

xbmcgui.Dialog().ok(self.addon.getLocalizedString(30000), self.addon.getLocalizedString(30017), self.addon.getLocalizedString(30118))

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

else:

xbmcgui.Dialog().ok(self.addon.getLocalizedString(30000), self.addon.getLocalizedString(30017), self.addon.getLocalizedString(30118))

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

return

response\_data = response.read()

response.close()

# retrieve code

code = ''

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

response\_data, re.DOTALL):

code = r.group(1)

if code != '':

xbmcgui.Dialog().ok(self.addon.getLocalizedString(30000), self.addon.getLocalizedString(30143))

else:

xbmcgui.Dialog().ok(self.addon.getLocalizedString(30000), self.addon.getLocalizedString(30144))

return

url = 'https://script.google.com/macros/s/AKfycbxgFuUcvNlXLlB5GZLiEjEaZDqZLS2oMd-f4yL-4Y2K50shGoY/exec'

values = {

'code' : code

}

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

# try login

try:

response = urllib2.urlopen(req)

except urllib2.URLError, e:

if e.code == 403:

#login issue

xbmcgui.Dialog().ok(self.addon.getLocalizedString(30000), self.addon.getLocalizedString(30017), self.addon.getLocalizedString(30118))

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

else:

xbmcgui.Dialog().ok(self.addon.getLocalizedString(30000), self.addon.getLocalizedString(30017), self.addon.getLocalizedString(30118))

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('\"access\_token\"\s?\:\s?\"([^\"]+)\".+?' +

'\"refresh\_token\"\s?\:\s?\"([^\"]+)\".+?' ,

response\_data, re.DOTALL):

accessToken,refreshToken = r.groups()

self.authorization.setToken('auth\_access\_token',accessToken)

self.authorization.setToken('auth\_refresh\_token',refreshToken)

self.updateAuthorization(self.addon)

xbmcgui.Dialog().ok(self.addon.getLocalizedString(30000), self.addon.getLocalizedString(30142))

for r in re.finditer('\"error\_description\"\s?\:\s?\"([^\"]+)\"',

response\_data, re.DOTALL):

errorMessage = r.group(1)

xbmcgui.Dialog().ok(self.addon.getLocalizedString(30000), self.addon.getLocalizedString(30119), errorMessage)

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

return

##

# refresh OAUTH2 access given refresh token

# parameters: none

# returns: none

##

def refreshToken(self):

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

if (self.type ==2):

url = self.addon.getSetting(self.instanceName+'\_url')

values = {

'refresh\_token' : self.authorization.getToken('auth\_refresh\_token')

}

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

elif (self.type ==3):

url = 'https://accounts.google.com/o/oauth2/token'

clientID = self.addon.getSetting(self.instanceName+'\_client\_id')

clientSecret = self.addon.getSetting(self.instanceName+'\_client\_secret')

header = { 'User-Agent' : self.user\_agent , 'Content-Type': 'application/x-www-form-urlencoded'}

req = urllib2.Request(url, 'client\_id='+clientID+'&client\_secret='+clientSecret+'&refresh\_token='+self.authorization.getToken('auth\_refresh\_token')+'&grant\_type=refresh\_token', header)

elif (self.type ==1):

url = 'http://dmdsoftware.net/api/gdrive.php'

values = {

'refresh\_token' : self.authorization.getToken('auth\_refresh\_token')

}

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

else:

url = 'https://script.google.com/macros/s/AKfycbxgFuUcvNlXLlB5GZLiEjEaZDqZLS2oMd-f4yL-4Y2K50shGoY/exec'

values = {

'refresh\_token' : self.authorization.getToken('auth\_refresh\_token')

}

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

# try login

try:

response = urllib2.urlopen(req)

except urllib2.URLError, e:

if e.code == 403:

#login issue

xbmcgui.Dialog().ok(self.addon.getLocalizedString(30000), self.addon.getLocalizedString(30017), self.addon.getLocalizedString(30118))

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

else:

xbmcgui.Dialog().ok(self.addon.getLocalizedString(30000), self.addon.getLocalizedString(30017), self.addon.getLocalizedString(30118))

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('\"access\_token\"\s?\:\s?\"([^\"]+)\".+?' ,

response\_data, re.DOTALL):

accessToken = r.group(1)

self.authorization.setToken('auth\_access\_token',accessToken)

self.updateAuthorization(self.addon)

for r in re.finditer('\"error\_description\"\s?\:\s?\"([^\"]+)\"',

response\_data, re.DOTALL):

errorMessage = r.group(1)

xbmcgui.Dialog().ok(self.addon.getLocalizedString(30000), self.addon.getLocalizedString(30119), errorMessage)

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

return

##

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

# returns: list containing the header

##

def getHeadersList(self, isPOST=False, additionalHeader=None, additionalValue=None, isJSON=False):

if self.authorization.isToken(self.instanceName,self.addon, 'auth\_access\_token') and not isPOST:

# return { 'User-Agent' : self.user\_agent, 'Authorization' : 'Bearer ' + self.authorization.getToken('auth\_access\_token') }

if additionalHeader is not None:

return { 'Cookie' : 'DRIVE\_STREAM='+ self.authorization.getToken('DRIVE\_STREAM'), 'Authorization' : 'Bearer ' + self.authorization.getToken('auth\_access\_token'), additionalHeader : additionalValue }

else:

return { 'Cookie' : 'DRIVE\_STREAM='+ self.authorization.getToken('DRIVE\_STREAM'), 'Authorization' : 'Bearer ' + self.authorization.getToken('auth\_access\_token') }

elif isJSON and self.authorization.isToken(self.instanceName,self.addon, 'auth\_access\_token'):

# return { 'User-Agent' : self.user\_agent, 'Authorization' : 'Bearer ' + self.authorization.getToken('auth\_access\_token') }

return { 'Content-Type': 'application/json', 'Cookie' : 'DRIVE\_STREAM='+ self.authorization.getToken('DRIVE\_STREAM'), 'Authorization' : 'Bearer ' + self.authorization.getToken('auth\_access\_token') }

elif self.authorization.isToken(self.instanceName,self.addon, 'auth\_access\_token'):

# return { 'User-Agent' : self.user\_agent, 'Authorization' : 'Bearer ' + self.authorization.getToken('auth\_access\_token') }

return { "If-Match" : '\*', 'Content-Type': 'application/atom+xml', 'Cookie' : 'DRIVE\_STREAM='+ self.authorization.getToken('DRIVE\_STREAM'), 'Authorization' : 'Bearer ' + self.authorization.getToken('auth\_access\_token') }

#return { 'Content-Type': 'application/atom+xml', 'Authorization' : 'Bearer ' + self.authorization.getToken('auth\_access\_token') }

elif self.authorization.isToken(self.instanceName,self.addon, 'DRIVE\_STREAM') and not isPOST:

if additionalHeader is not None:

return { 'Cookie' : 'DRIVE\_STREAM='+ self.authorization.getToken('DRIVE\_STREAM'), additionalHeader : additionalValue }

else:

return { 'Cookie' : 'DRIVE\_STREAM='+ self.authorization.getToken('DRIVE\_STREAM') }

else:

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

##

# 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):

return urllib.urlencode(self.getHeadersList())

##

# 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 = self.API\_URL +'files/'

# show all videos

if folderName=='VIDEO':

url = url + "?q=mimeType+contains+'video'"

# show all music

elif folderName=='MUSIC':

url = url + "?q=mimeType+contains+'audio'"

# show all music and video

elif folderName=='VIDEOMUSIC':

url = url + "?q=mimeType+contains+'audio'+or+mimeType+contains+'video'"

# show all photos and music

elif folderName=='PHOTOMUSIC':

url = url + "?q=mimeType+contains+'image'+or+mimeType+contains+'music'"

# show all photos

elif folderName=='PHOTO':

url = url + "?q=mimeType+contains+'image'"

# show all music, photos and video

elif folderName=='ALL':

url = url + "?q=mimeType+contains+'audio'+or+mimeType+contains+'video'+or+mimeType+contains+'image'"

# search for title

elif title != False or folderName == 'SAVED SEARCH':

encodedTitle = re.sub(' ', '+', title)

url = url + "?q=title+contains+'" + str(encodedTitle) + "'" + "+and+not+title+contains+'SAVED+SEARCH'"

# show all starred items

elif folderName == 'STARRED-FILES' or folderName == 'STARRED-FILESFOLDERS' or folderName == 'STARRED-FOLDERS':

url = url + "?q=starred%3dtrue"

# show all shared items

elif folderName == 'SHARED':

url = url + "?q=sharedWithMe%3dtrue"

# default / show root folder

elif folderName == '' or folderName == 'me' or folderName == 'root':

folderName = self.getRootID()

url = url + "?q='"+str(folderName)+"'+in+parents"

# retrieve folder items

else:

url = url + "?q='"+str(folderName)+"'+in+parents"

# contribution by dabinn

# filter out trashed items

url = url + "+and+trashed%3dfalse"

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.refreshToken()

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 r2 in re.finditer('\"items\"\:\s+\[[^\{]+(\{.\*?)\}\s+\]\s+\}' ,response\_data, re.DOTALL):

entryS = r2.group(1)

folderFanart = ''

folderIcon = ''

for r1 in re.finditer('\{(.\*?)\"appDataContents\"\:' , entryS, re.DOTALL):

entry = r1.group(1)

if 'fanart' in entry:

fanart = self.getMediaInfo(entry, folderName=folderName)

if fanart != '':

# fanart = re.sub('\&gd\=true', '', fanart)

#need to cache

folderFanart = fanart + '|' + self.getHeadersEncoded()

elif 'folder' in entry:

foldericon = self.getMediaInfo(entry, folderName=folderName)

if foldericon != '':

# foldericon = re.sub('\&gd\=true', '', foldericon)

#need to cache

folderIcon = foldericon + '|' + self.getHeadersEncoded()

for r1 in re.finditer('\{(.\*?)\"spaces\"\:' , entryS, re.DOTALL):

entry = r1.group(1)

media = self.getMediaPackage(entry, folderName=folderName, contentType=contentType, fanart=folderFanart, icon=folderIcon)

if media is not None:

mediaFiles.append(media)

# look for more pages of videos

nextURL = ''

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

response\_data, re.DOTALL):

nextURL = r.group(1)

# are there more pages to process?

if nextURL == '':

break

else:

url = nextURL

return mediaFiles

##

# retrieve a list of changes

# parameters: prompt for video quality (optional)

# returns: list of packages (file, folder)

##

def getChangeList(self, contentType=7, nextPageToken='', changeToken=''):

url = ''

maxChangeID = 0;

# retrieve all items

url = self.API\_URL +'changes'

# url = url + "?includeDeleted=false&includeSubscribed=false&maxResults=1000"

url = url + "?includeDeleted=true&includeSubscribed=false&maxResults=300"

if (changeToken != ''):

url = url + '&startChangeId=' + str(changeToken)

if (nextPageToken != ''):

url = url + '&pageToken=' + str(nextPageToken)

nextURL = ''

nextPageToken = ''

while True:

mediaFiles = []

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

# if action fails, validate login

try:

response = urllib2.urlopen(req)

xbmc.sleep(5000)

except urllib2.URLError, e:

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

self.refreshToken()

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

try:

response = urllib2.urlopen(req)

except socket.timeout, e:

return ([],nextPageToken,changeToken)

except urllib2.URLError, e:

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

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

return

else:

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

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

return

except socket.timeout, e:

return ([],nextPageToken,changeToken)

response\_data = response.read()

response.close()

# parsing page for videos

# video-entry

for r2 in re.finditer('\"items\"\:\s+\[[^\{]+(\{.\*?)\}\s+\]\s+\}' ,response\_data, re.DOTALL):

entryS = r2.group(1)

for r1 in re.finditer('\{(.\*?)\"spaces\"\:' , entryS, re.DOTALL):

entry = r1.group(1)

for r in re.finditer('\"id\"\:\s+\"(\d+)\"' ,

entry, re.DOTALL):

id = r.group(1)

if id > maxChangeID:

maxChangeID = id

media = self.getMediaPackage(entry, contentType=contentType)

if media is not None:

mediaFiles.append(media)

# look for more pages of videos

for r in re.finditer('\"largestChangeId\"\:\s+\"(\d)\"' ,

response\_data, re.DOTALL):

largestChangeId = r.group(1)

# look for more pages of videos

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

response\_data, re.DOTALL):

nextURL = r.group(1)

## look for more pages of videos

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

response\_data, re.DOTALL):

nextPageToken = r.group(1)

print 'nextPageToken = '+ str(nextPageToken) + ' nextPageToken = ' + str(maxChangeID) + "\n"

return (mediaFiles, nextPageToken, maxChangeID)

# are there more pages to process?

if nextURL == '':

break

else:

url = nextURL

return (mediaFiles, nextPageToken, maxChangeID)

##

# retrieve a list of videos, using playback type stream

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

# returns: list of videos

### \*\*\* looks like this does nothing?

##

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

mediaFiles = []

for r1 in re.finditer('\{(.\*?)\"spaces\"\:' , entryS, re.DOTALL):

entry = r1.group(1)

media = self.getMediaPackage(entry, folderName=folderName, contentType=contentType, fanart=folderFanart, icon=folderIcon)

if media is not None:

mediaFiles.append(media)

return mediaFiles

##

# retrieve a list of videos, using playback type stream

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

# returns: list of videos

##

def scanMediaList(self, folderName):

# retrieve all items

url = self.API\_URL +'files/'

url = url + "?q='"+str(folderName)+"'+in+parents"

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.refreshToken()

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 r2 in re.finditer('\"items\"\:\s+\[[^\{]+(\{.\*?)\}\s+\]\s+\}' ,response\_data, re.DOTALL):

entryS = r2.group(1)

folderFanart = ''

folderIcon = ''

for r1 in re.finditer('\{(.\*?)\"appDataContents\"\:' , entryS, re.DOTALL):

entry = r1.group(1)

if 'fanart' in entry:

fanart = self.getMediaInfo(entry, folderName=folderName)

if fanart != '':

# fanart = re.sub('\&gd\=true', '', fanart)

#need to cache

folderFanart = fanart + '|' + self.getHeadersEncoded()

elif 'folder' in entry:

foldericon = self.getMediaInfo(entry, folderName=folderName)

if foldericon != '':

# foldericon = re.sub('\&gd\=true', '', foldericon)

#need to cache

folderIcon = foldericon + '|' + self.getHeadersEncoded()

# file property - gdrive

if self.cloudResume == 1:

self.setProperty(folderName,'icon', 1)

# look for more pages of videos

nextURL = ''

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

response\_data, re.DOTALL):

nextURL = r.group(1)

# are there more pages to process?

if nextURL == '':

break

else:

url = nextURL

return mediaFiles

##

# retrieve a media package

# parameters: given an entry

# returns: package (folder,file)

##

def getMediaPackage(self, entry, folderName='',contentType=2, fanart='', icon=''):

resourceID = 0

resourceType = ''

title = ''

fileSize = 0

thumbnail = ''

fileExtension = ''

md5 = ''

deleted = False

url = ''

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

entry, re.DOTALL):

resourceID = r.group(1)

break

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

entry, re.DOTALL):

resourceID = r.group(1)

break

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

entry, re.DOTALL):

resourceType = r.group(1)

break

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

entry, re.DOTALL):

title = r.group(1)

break

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

entry, re.DOTALL):

fileSize = r.group(1)

break

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

entry, re.DOTALL):

thumbnail = r.group(1)

break

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

entry, re.DOTALL):

if self.settings.encfsDownloadType == 0:

url = r.group(1)

else:

url = self.API\_URL +'files/' + str(resourceID) + '?alt=media'

break

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

entry, re.DOTALL):

fileExtension = r.group(1)

break

for r in re.finditer('\"md5Checksum\"\:\s+\"([^\"]+)\"' ,

entry, re.DOTALL):

md5 = r.group(1)

break

height =0

width = 0

for r in re.finditer('\"height\"\:\s+(\d+)' ,

entry, re.DOTALL):

height = r.group(1)

break

for r in re.finditer('\"width\"\:\s+(\d+)' ,

entry, re.DOTALL):

width = r.group(1)

break

for r in re.finditer('\"deleted\"\:\s+true,' ,

entry, re.DOTALL):

deleted = True

break

# file property - gdrive

resume = 0

playcount = 0

if self.cloudResume == 1:

for r in re.finditer('\"key\"\:\s+\"resume\"[^\"]+\"visibility\"\:\s+\"[^\"]+\"[^\"]+\"value\"\:\s+\"([^\"]+)\"' ,

entry, re.DOTALL):

resume = r.group(1)

break

# file property - gdrive

for r in re.finditer('\"key\"\:\s+\"playcount\"[^\"]+\"visibility\"\:\s+\"[^\"]+\"[^\"]+\"value\"\:\s+\"([^\"]+)\"' ,

entry, re.DOTALL):

playcount = r.group(1)

break

duration = 0

for r in re.finditer('\"durationMillis\"\:\s+\"(\d+)\"' ,

entry, re.DOTALL):

duration = r.group(1)

duration = int(int(duration) / 1000)

break

# entry is a folder

if (resourceType == 'application/vnd.google-apps.folder'):

for r in re.finditer('SAVED SEARCH\|([^\|]+)' ,

title, re.DOTALL):

newtitle = r.group(1)

title = '\*' + newtitle

resourceID = 'SAVED SEARCH'

media = package.package(None,folder.folder(resourceID,title, fanart, thumb=icon))

return media

# entry is a video

elif ((fileExtension.lower() == '' or fileExtension.lower() not in ('','sub')) and (resourceType == 'application/vnd.google-apps.video' or 'video' in resourceType or fileExtension.lower() in ('iso','ts', 'mkv', 'rmvb')) and contentType in (0,1,2,4,7)):

mediaFile = file.file(resourceID, title, title, self.MEDIA\_TYPE\_VIDEO, fanart, thumbnail, size=fileSize, resolution=[height,width], playcount=int(playcount), duration=duration, checksum=md5)

if self.settings.parseTV:

tv = mediaFile.regtv1.match(title)

if not tv:

tv = mediaFile.regtv2.match(title)

if not tv:

tv = mediaFile.regtv3.match(title)

if tv:

show = tv.group(1).replace(".", " ")

show = show.replace('-',"")

season = tv.group(2)

episode = tv.group(3)

showtitle = tv.group(4).replace(".", " ")

showtitle = showtitle.replace('-',"")

mediaFile.setTVMeta(show,season,episode,showtitle)

if deleted:

mediaFile.deleted = True

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

media.setMediaURL(mediaurl.mediaurl(url, 'original', 0, 9999))

try:

if float(resume) > 0:

if duration > 0 and float(resume)/duration < (float(100 - int(self.settings.skipResume))/100):

mediaFile.resume = float(resume)

else:

mediaFile.playcount = mediaFile.playcount + 1

except: pass

return media

# entry is a music file

elif ((resourceType == 'application/vnd.google-apps.audio' or (fileExtension.lower() == '' or fileExtension.lower() in ('flac', 'mp3')) or 'audio' in resourceType) and contentType in (1,2,3,4,6,7)):

mediaFile = file.file(resourceID, title, title, self.MEDIA\_TYPE\_MUSIC, fanart, '', size=fileSize, checksum=md5)

if self.settings.parseMusic:

for r in re.finditer('([^\-]+) \- ([^\-]+) \- (\d+) \- ([^\.]+)\.' ,

title, re.DOTALL):

artist,album,track,trackTitle = r.groups()

mediaFile.setAlbumMeta(album,artist,'',track,'', trackTitle)

break

if deleted:

mediaFile.deleted = True

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

media.setMediaURL(mediaurl.mediaurl(url, 'original', 0, 9999))

return media

# entry is a photo

elif ((resourceType == 'application/vnd.google-apps.photo' or 'image' in resourceType) and contentType in (2,4,5,6,7)):

mediaFile = file.file(resourceID, title, title, self.MEDIA\_TYPE\_PICTURE, fanart, thumbnail, size=fileSize, download=url, checksum=md5)

photoURL = url

#\*\*\* downsize photo

if width > self.settings.photoResolution:

photoURL = re.sub('=s\d\d\d', '=s'+ str(self.settings.photoResolution), thumbnail)

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

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

return media

# entry is a photo, but we are not in a photo display

elif (resourceType == 'application/vnd.google-apps.photo' or 'image' in resourceType):

return

# entry is unknown

elif (resourceType == 'application/vnd.google-apps.unknown'):

mediaFile = file.file(resourceID, title, title, self.MEDIA\_TYPE\_UNKNOWN, fanart, thumbnail, size=fileSize, checksum=md5)

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

media.setMediaURL(mediaurl.mediaurl(url, 'original', 0, 9999))

return media

# all files (for saving to encfs)

elif (contentType >= 8):

mediaFile = file.file(resourceID, title, title, self.MEDIA\_TYPE\_UNKNOWN, fanart, '', size=fileSize, checksum=md5)

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

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

return media

##

# retrieve a media package

# parameters: given an entry

# returns: package (folder,file)

##

def getMediaInfo(self, entry, folderName=''):

resourceID = 0

resourceType = ''

title = ''

fileSize = 0

thumbnail = ''

fileExtension = ''

url = ''

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

entry, re.DOTALL):

resourceID = r.group(1)

break

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

entry, re.DOTALL):

resourceType = r.group(1)

break

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

entry, re.DOTALL):

title = r.group(1)

break

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

entry, re.DOTALL):

thumbnail = r.group(1)

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

entry, re.DOTALL):

fileExtension = r.group(1)

break

# entry is a photo

if ('fanart' in title and (resourceType == 'application/vnd.google-apps.photo' or 'image' in resourceType)):

return self.API\_URL +'files/' + str(resourceID) + '?alt=media'

# entry is a photo

elif ('folder' in title and (resourceType == 'application/vnd.google-apps.photo' or 'image' in resourceType)):

return self.API\_URL +'files/' + str(resourceID) + '?alt=media'

return ''

##

# retrieve a srt file for playback

# parameters: package

# returns: download url for srt

##

def getSRT(self, package):

if package is None or (package.file is None and package.folder is None):

return

# retrieve all items

url = self.API\_URL +'files'

# merge contribution by dabinn

q = ''

# search in current folder for SRT files

if package.folder is not None and (package.folder.id != False and package.folder.id != ''):

q = "'"+str(package.folder.id)+"' in parents"

else:

# search for title in SRT file

if package.file is not None and (package.file.title != False and package.file.title != ''):

title = os.path.splitext(package.file.title)[0]

if q != '':

q = q + ' and '

q = q + "title contains '" + str(title) + "'"

url = url + "?" + urllib.urlencode({'q':q})

#generate two lists of SRT files

#1) list of files (multiple languages) from the same folder that exactly match the title of the video

#2) list of candidate files that are from the same folder but don't match the title of the video (if exceeds 4, ask user to select -- likely TV series or folder containing multiple movies)

srt = []

srtCandidates = []

srtCandidatesTitles = []

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.refreshToken()

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

try:

response = urllib2.urlopen(req)

except urllib2.URLError, e:

#skip SRT

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

#self.crashreport.sendError('getSRT',str(e))

return

else:

#skip SRT

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

#self.crashreport.sendError('getSRT',str(e))

return

response\_data = response.read()

response.close()

# parsing page for videos

# video-entry

for r2 in re.finditer('\"items\"\:\s+\[[^\{]+(\{.\*?)\}\s+\]\s+\}' ,response\_data, re.DOTALL):

entryS = r2.group(1)

for r1 in re.finditer('\{(.\*?)\"appDataContents\"\:' ,entryS, re.DOTALL):

entry = r1.group(1)

resourceID = 0

resourceType = ''

title = ''

url = ''

fileExtension = ''

isExactMatch = False

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

entry, re.DOTALL):

fileExtension = r.group(1)

break

if fileExtension.lower() in ('srt','idx','sub','ass','ssa'):

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

entry, re.DOTALL):

resourceID = r.group(1)

break

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

entry, re.DOTALL):

title = r.group(1)

#if os.path.splitext(package.file.title)[0] in title:

## contribution by dabinn

if title.startswith(os.path.splitext(package.file.title)[0]):

isExactMatch = True

break

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

entry, re.DOTALL):

url = r.group(1)

if isExactMatch:

srt.append([title,url])

else:

srtCandidates.append([title,url])

srtCandidatesTitles.append(title)

break

# look for more pages of videos

nextURL = ''

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

response\_data, re.DOTALL):

nextURL = r.group(1)

# are there more pages to process?

if nextURL == '':

break

else:

url = nextURL

if len(srt) > 0:

return srt

elif len(srtCandidates) > 4:

ret = xbmcgui.Dialog().select(self.addon.getLocalizedString(30183), srtCandidatesTitles)

if ret >= 0:

return [srtCandidates[ret]]

else:

return

elif len(srtCandidates) > 0:

return srtCandidates

##

# Google Drive specific

#

# retrieve tts file(s) for playback

# -- will download tts file(s) associated with the video to path

# parameters: baseURL - TTS Base URL

# returns: nothing

##

def getTTS(self, baseURL):

if baseURL == '':

return ''

# retrieve all items

url = baseURL +'&hl=en-US&type=list&tlangs=1&fmts=1&vssids=1'

cc = []

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.refreshToken()

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('getTTS',str(e))

return

else:

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

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

return

response\_data = response.read()

response.close()

# parsing page for videos

# video-entry

count=0

for r in re.finditer('\<track id\=\"\d+\" .\*? lang\_code\=\"([^\"]+)\" .\*? lang\_translated\=\"([^\"]+)\" [^\>]+\>' ,response\_data, re.DOTALL):

lang,language = r.groups()

cc.append([ '.'+str(count) + '.'+ str(lang) + '.srt',baseURL+'&type=track&lang='+str(lang)+'&name&kind&fmt=1'])

# look for more pages of videos

nextURL = ''

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

response\_data, re.DOTALL):

nextURL = r.group(1)

# are there more pages to process?

if nextURL == '':

break

else:

url = nextURL

return cc

##

# retrieve the resource ID for root folder

# parameters: none

# returns: resource ID for root

##

def getRootID(self):

# retrieve all items

url = self.API\_URL +'files/root'

resourceID = ''

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.refreshToken()

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('getRootID',str(e))

return

else:

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

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

return

response\_data = response.read()

response.close()

for r1 in re.finditer('\{(.\*?)\"appDataContents\"\:' ,response\_data, re.DOTALL):

entry = r1.group(1)

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

entry, re.DOTALL):

resourceID = r.group(1)

return resourceID

# look for more pages of videos

nextURL = ''

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

response\_data, re.DOTALL):

nextURL = r.group(1)

# are there more pages to process?

if nextURL == '':

break

else:

url = nextURL

return resourceID

##

# retrieve the download URL for given resorce ID

# parameters: resource ID

# returns: download URL

##

def getDownloadURL(self, docid):

url = self.API\_URL +'files/' + str(docid) + '?alt=media'

return url

url = self.API\_URL +'files/' + 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.refreshToken()

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('getDownloadURL',str(e))

return

else:

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

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

return

response\_data = response.read()

response.close()

for r1 in re.finditer('\{(.\*?)\"appDataContents\"\:' ,response\_data, re.DOTALL):

entry = r1.group(1)

url = ''

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

entry, re.DOTALL):

url = r.group(1)

return url

##

# retrieve the details for a file given resource ID

# parameters: resource ID

# returns: entry

##

def getMediaDetails(self, docid):

url = self.API\_URL +'files/' + 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.refreshToken()

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('getDownloadURL',str(e))

return

else:

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

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

return

response\_data = response.read()

response.close()

for r1 in re.finditer('\{(.\*?)\"appDataContents\"\:' ,response\_data, re.DOTALL):

entry = r1.group(1)

return self.getMediaPackage(entry)

##

# Google Drive specific

# retrieve a playback url

# parameters: package (optional), title of media file, isExact allowing for fuzzy searches

# returns: url for playback

##

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

try:

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

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

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

aformat = int(self.addon.getSetting('avoid\_format'))

except :

pquality=-1

pformat=-1

acodec=-1

aformat=-1

mediaURLs = []

docid = ''

# for playback from STRM with title of video provided (best match)

if package is None and title != '':

url = self.API\_URL +'files/'

# search by video title

encodedTitle = re.sub(' ', '+', title)

encodedTitle = re.sub('\'', '\\\'', encodedTitle)

encodedTitle = re.sub('&', '\\\&', encodedTitle)

#encodedTitle = re.sub('?', '\\\?', encodedTitle)

#encodedTitle = re.sub('#', '\\\#', encodedTitle)

#encodedTitle = re.sub('(', '\\\(', encodedTitle)

#encodedTitle = re.sub(')', '\\\)', encodedTitle)

#encodedTitle = re.sub('$', '\\\$', encodedTitle)

if isExact == True:

url = url + "?q=title%3d'" + str(encodedTitle) + "'"

else:

url = url + "?q=title+contains+'" + str(encodedTitle) + "'"

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.refreshToken()

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-0',str(e))

return

else:

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

self.crashreport.sendError('getPlaybackCall-0',str(e))

return

response\_data = response.read()

response.close()

for r1 in re.finditer('\{(.\*?)\"appDataContents\"\:' ,response\_data, re.DOTALL):

entry = r1.group(1)

package = self.getMediaPackage(entry)

docid = package.file.id

mediaURLs.append(package.mediaurl)

#given docid, fetch original playback

else:

docid = package.file.id

# new method of fetching original stream -- using alt=media

url = self.API\_URL +'files/' + str(docid) + '?alt=media'

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

# old method of fetching original stream -- using downloadURL

# fetch information if no thumbnail cache (we need thumbnail url) or we want to download (we need filesize)

if self.cache.getThumbnail(self, fileID=docid) == '' or self.settings.download or 1:

url = self.API\_URL +'files/' + str(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.refreshToken()

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-1',str(e))

return

else:

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

self.crashreport.sendError('getPlaybackCall-1',str(e))

return

response\_data = response.read()

response.close()

if package is not None and package.folder is not None:

package = self.getMediaPackage(response\_data, package.folder.id)

#docid = package.file.id

#mediaURLs.append(package.mediaurl)

# encryption?

if package is None:

# new method of fetching original stream -- using alt=media

url = self.API\_URL +'files/' + str(docid) + '?alt=media'

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

return (mediaURLs, package)

# there are no streams for music

if package.file.type == self.MEDIA\_TYPE\_MUSIC:

return (mediaURLs, package)

# fetch streams (video)

if docid != '':

# player using docid

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

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

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-2',str(e))

return

else:

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

self.crashreport.sendError('getPlaybackCall-2',str(e))

return

response\_data = response.read()

response.close()

for r in re.finditer('([^\=]+)\=([^\;]+)\;', str(response.headers['set-cookie']), re.DOTALL):

cookieType,cookieValue = r.groups()

if cookieType == 'DRIVE\_STREAM':

print cookieValue

self.authorization.setToken(cookieType,cookieValue)

# 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)

#set closed-caption

ttsURL=''

for r in re.finditer('&ttsurl\=(.\*?)\&reportabuseurl' ,

urls, re.DOTALL):

ttsURL = r.group(1)

ttsURL = ttsURL+'&v='+docid #+ '&type=track&lang=en&name&kind&fmt=1'

package.file.srtURL = ttsURL

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

try:

response = urllib2.urlopen(req)

except urllib2.URLError, e:

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

self.refreshToken()

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-3',str(e))

return

else:

#return what we have -- video file may not have streams (not processed yet)

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

#self.crashreport.sendError('getPlaybackCall-3',str(e))

return (mediaURLs, package)

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)

# do some substitutions to make anchoring the URL easier

urls = re.sub('\&url\='+self.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 == 2:

order = order + 9000

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

if pquality == 0:

order = order + 2000

elif pquality == 1:

order = order + 1000

elif pquality == 2:

order = order + 9000

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

if pquality == 0:

order = order + 3000

elif pquality == 1:

order = order + 2000

elif pquality == 2:

order = order + 1000

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

if pquality == 0:

order = order + 4000

elif pquality == 1:

order = order + 3000

elif pquality == 2:

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

if aformat == 1:

order = order + 90000

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))

except KeyError:

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

return (mediaURLs, package)

##

# Google Drive specific

# download a TTS and save as a SRT

# parameters: url of picture, file location with path on disk

# returns: nothing

##

def downloadTTS(self, url, file):

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

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

# if action fails, validate login

try:

response = urllib2.urlopen(req)

except urllib2.URLError, e:

self.refreshToken()

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('downloadTTS',str(e))

return

response\_data = response.read()

response.close()

count=0

#convert TTS (google drive) to SRT

for q in re.finditer('\<text start\=\"([^\"]+)\" dur\=\"([^\"]+)\"\>([^\<]+)\</text\>' ,

response\_data, re.DOTALL):

start,duration,text = q.groups()

count = count + 1

startTimeSec = float(start) % 60

startTimeMin = float(start) / 60 %60

startTimeHour = float(start) / (60\*60)

startTimeMSec = (float(start) - int(float(start))) \* 1000

endTimeSec = (float(start) + float(duration)) % 60

endTimeMin = (float(start) + float(duration)) / 60 %60

endTimeHour = (float(start) + float(duration)) / (60\*60)

endTimeMSec = ((float(start) - int(float(start))) + (float(duration) - int(float(duration)))) \* 1000

if endTimeMSec > 1000:

endTimeMSec = endTimeMSec % 1000

text = re.sub('&amp;#39;', "'", text)

text = re.sub('&amp;lt;', "<", text)

text = re.sub('&amp;gt;', ">", text)

text = re.sub('&amp;quot;', '"', text)

f.write("%d\n%02d:%02d:%02d,%03d --> %02d:%02d:%02d,%03d\n%s\n\n" % (count, startTimeHour, startTimeMin, startTimeSec, startTimeMSec, endTimeHour, endTimeMin, endTimeSec, endTimeMSec, text))

f.close()

##

# Google Drive specific

# get videos streams for a public URL

# parameters: public url

# returns: list of MediaURLs

##

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, self.getHeadersList())

try:

response = urllib2.urlopen(req)

except urllib2.URLError, e:

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

self.refreshToken()

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('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()

for r in re.finditer('([^\s]+)\=([^\;]+)\;', str(response.headers['set-cookie']), re.DOTALL):

cookieType,cookieValue = r.groups()

if cookieType == 'DRIVE\_STREAM':

print cookieValue

self.authorization.setToken(cookieType,cookieValue)

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)

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

# 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

##

# Google Drive API2 specific

# set a file property

# file property - gdrive

# parameters: doc id, key, value

##

def setProperty(self, docid, key, value):

url = self.API\_URL +'files/' + str(docid) + '/properties/' + str(key) + '?visibility=PUBLIC'

propertyValues = '{"value": "'+str(value)+'", "key": "'+str(key)+'", "visibility": "PUBLIC"}'

req = urllib2.Request(url, propertyValues, self.getHeadersList())

req.get\_method = lambda: 'PATCH'

req.add\_header('Content-Type', 'application/json')

try:

response = urllib2.urlopen(req)

# response = opener.open(url, None,urllib.urlencode(header))

except urllib2.URLError, e:

if e.code == 401:

self.refreshToken()

req = urllib2.Request(url, propertyValues, self.getHeadersList())

req.get\_method = lambda: 'PATCH'

req.add\_header('Content-Type', 'application/json')

try:

response = urllib2.urlopen(req)

except urllib2.URLError, e:

#doesn't have access

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

return

else:

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

#self.crashreport.sendError('setProperty',str(e))

return

#maybe doesn't exist - try to create

elif e.code != 403:

# else:

url = self.API\_URL +'files/' + str(docid) + '/properties'

req = urllib2.Request(url, propertyValues, self.getHeadersList())

req.add\_header('Content-Type', 'application/json')

try:

response = urllib2.urlopen(req)

except:

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

return

else:

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

#self.crashreport.sendError('setProperty',str(e))

return

# some other kind of error

else:

return

response\_data = response.read()

response.close()