'''

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/>.

'''

#import os

import re

import urllib, urllib2

import sys

import os

# cloudservice - standard XBMC modules

import xbmc, xbmcaddon, xbmcgui, xbmcplugin

import xbmcvfs

from resources.lib import mediaurl

#from resources.lib import kodi\_common

#from resources.lib import settings

from resources.lib import streamer

PLUGIN\_URL = sys.argv[0]

plugin\_handle = None

try:

#global variables

plugin\_handle = int(sys.argv[1])

except:pass

def decode(data):

return re.sub("&#(\d+)(;|(?=\s))", \_callback, data).strip()

def decode\_dict(data):

for k, v in data.items():

if type(v) is str or type(v) is unicode:

data[k] = decode(v)

return data

#http://stackoverflow.com/questions/1208916/decoding-html-entities-with-python/1208931#1208931

def \_callback(matches):

id = matches.group(1)

try:

return unichr(int(id))

except:

return id

#

#

#

class cloudservice(object):

# CloudService v0.2.3

PLAYBACK\_RESOLVED = 1

PLAYBACK\_PLAYER = 2

PLAYBACK\_NONE = 3

def \_\_init\_\_(self): pass

##

# perform login

##

def login(self): pass

##

# if we don't have an authorization token set for the plugin, set it with the recent login.

# auth\_token will permit "quicker" login in future executions by reusing the existing login session (less HTTPS calls = quicker video transitions between clips)

##

def updateAuthorization(self,addon):

if self.authorization.isUpdated :#and addon.getSetting(self.instanceName+'\_save\_auth\_token') == 'true':

self.authorization.saveTokens(self.instanceName,addon)

##

# return the appropriate "headers" for requests that include 1) user agent, 2) any authorization cookies/tokens

# returns: list containing the header

##

def getHeadersList(self):

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

##

# return the appropriate "headers" for requests that include 1) user agent, 2) any authorization cookies/tokens

# returns: URL-encoded header string

##

def getHeadersEncoded(self):

return urllib.urlencode(self.getHeadersList())

##

# build STRM files to a given path for a given folder ID

# parameters: path, folder id, content type, dialog object (optional)

##

def buildSTRM(self, path, folderID='', contentType=1, pDialog=None, epath='', dpath='', encfs=False, spreadsheetFile=None):

import xbmcvfs

xbmcvfs.mkdir(path)

#musicPath = path + '/music'

#moviePath = path + '/movies'

#tvPath = path + '/tv'

#videoPath = path + '/video-other'

musicPath = path

moviePath = path

tvPath = path

videoPath = path

#xbmcvfs.mkdir(musicPath)

#xbmcvfs.mkdir(tvPath)

#xbmcvfs.mkdir(videoPath)

#xbmcvfs.mkdir(moviePath)

mediaItems = self.getMediaList(folderID,contentType=contentType)

if mediaItems and not encfs:

for item in mediaItems:

url = 0

if item.file is None:

self.buildSTRM(path + '/'+str(item.folder.title), item.folder.id, pDialog=pDialog, spreadsheetFile=spreadsheetFile)

else:

#'content\_type': 'video',

values = { 'username': self.authorization.username, 'title': item.file.title, 'filename': item.file.id}

if item.file.type == 1:

url = self.PLUGIN\_URL+ '?mode=audio&' + urllib.urlencode(values)

else:

url = self.PLUGIN\_URL+ '?mode=video&' + urllib.urlencode(values)

#url = self.PLUGIN\_URL+'?mode=video&title='+str(item.file.title)+'&filename='+str(item.file.id)+ '&username='+str(self.authorization.username)

if url != 0:

title = item.file.title

if pDialog is not None:

pDialog.update(message=title)

if not xbmcvfs.exists(str(path) + '/' + str(title)+'.strm'):

filename = str(path) + '/' + str(title)+'.strm'

strmFile = xbmcvfs.File(filename, "w")

strmFile.write(url+'\n')

strmFile.close()

episode = ''

# nekwebdev contribution

pathLib = ''

tv = item.file.regtv1.match(title)

if not tv:

tv = item.file.regtv2.match(title)

if not tv:

tv = item.file.regtv3.match(title)

if 0 and tv:

show = tv.group(1).replace("\S{2,}\.\S{2,}", " ")

show = show.rstrip("\.")

season = tv.group(2)

episode = tv.group(3)

pathLib = tvPath + '/' + show

if not xbmcvfs.exists(xbmc.translatePath(pathLib)):

xbmcvfs.mkdir(xbmc.translatePath(pathLib))

pathLib = pathLib + '/Season ' + season

if not xbmcvfs.exists(xbmc.translatePath(pathLib)):

xbmcvfs.mkdir(xbmc.translatePath(pathLib))

else:

movie = item.file.regmovie.match(title)

if movie:

pathLib = moviePath

else:

pathLib = videoPath

if pathLib != '':

filename = str(pathLib) + '/' + str(title)+'.strm'

if item.file.deleted and xbmcvfs.exists(filename):

xbmcvfs.delete(filename)

elif not item.file.deleted and not xbmcvfs.exists(filename):

strmFile = xbmcvfs.File(filename, "w")

strmFile.write(url+'\n')

strmFile.close()

if spreadsheetFile is not None:

spreadsheetFile.write(str(item.folder.id) + '\t' + str(item.folder.title) + '\t'+str(item.file.id) + '\t'+str(item.file.title) + '\t'+str(episode)+'\t\t\t\t'+str(item.file.checksum) + '\t\t' + "\n")

elif mediaItems and encfs:

self.settings.setEncfsParameters()

encryptedPath = self.settings.getParameter('epath', '')

dencryptedPath = self.settings.getParameter('dpath', '')

encfs\_source = self.settings.encfsSource

encfs\_target = self.settings.encfsTarget

encfs\_inode = self.settings.encfsInode

dirListINodes = {}

fileListINodes = {}

for item in mediaItems:

if item.file is None:

xbmcvfs.mkdir(encfs\_source + str(encryptedPath))

xbmcvfs.mkdir(encfs\_source + str(encryptedPath) + str(item.folder.title) + '/' )

if encfs\_inode == 0:

dirListINodes[(str(xbmcvfs.Stat(encfs\_source + str(encryptedPath) + str(item.folder.title)).st\_ino()))] = item.folder

else:

dirListINodes[(str(xbmcvfs.Stat(encfs\_source + str(encryptedPath) + str(item.folder.title)).st\_ctime()))] = item.folder

#service.addDirectory(item.folder, contextType=contextType, encfs=True)

else:

xbmcvfs.mkdir(encfs\_source + str(encryptedPath))

xbmcvfs.mkdir(encfs\_source + str(encryptedPath) + str(item.file.title))

if encfs\_inode == 0:

fileListINodes[(str(xbmcvfs.Stat(encfs\_source + str(encryptedPath)+ str(item.file.title)).st\_ino()))] = item

else:

fileListINodes[(str(xbmcvfs.Stat(encfs\_source + str(encryptedPath) + str(item.file.title)).st\_ctime()))] = item

#service.addMediaFile(item, contextType=contextType)

if encfs\_inode > 0:

xbmc.sleep(1000)

if contentType == 9:

mediaList = ['.mp4', '.flv', '.mov', '.webm', '.avi', '.ogg', '.mkv', '.iso', '.rmvb']

elif contentType == 10:

mediaList = ['.mp3', '.flac']

else:# contentType == 11:

mediaList = ['.jpg', '.png']

media\_re = re.compile("|".join(mediaList), re.I)

dirs, files = xbmcvfs.listdir(encfs\_target + str(dencryptedPath) )

url = 0

for dir in dirs:

index = ''

if encfs\_inode == 0:

index = str(xbmcvfs.Stat(encfs\_target + str(dencryptedPath) + dir).st\_ino())

else:

index = str(xbmcvfs.Stat(encfs\_target + str(dencryptedPath) + dir).st\_ctime())

if index in dirListINodes.keys():

xbmcvfs.rmdir(encfs\_target + str(dencryptedPath) + dir)

# dirTitle = dir + ' [' +dirListINodes[index].title+ ']'

encryptedDir = dirListINodes[index].title

dirListINodes[index].displaytitle = dir + ' [' +dirListINodes[index].title+ ']'

#service.addDirectory(dirListINodes[index], contextType=contextType, encfs=True, dpath=str(dencryptedPath) + str(dir) + '/', epath=str(encryptedPath) + str(encryptedDir) + '/' )

self.buildSTRM(path + '/'+str(dir), dirListINodes[index].id, pDialog=pDialog, contentType=contentType, encfs=True, dpath=str(dencryptedPath) + str(dir) + '/', epath=str(encryptedPath) + str(encryptedDir) + '/' , spreadsheetFile=spreadsheetFile)

elif index in fileListINodes.keys():

xbmcvfs.rmdir(encfs\_target + str(dencryptedPath) + dir)

fileListINodes[index].file.decryptedTitle = dir

if contentType < 9 or media\_re.search(str(dir)):

#service.addMediaFile(fileListINodes[index], contextType=contextType, encfs=True, dpath=str(dencryptedPath) + str(dir), epath=str(encryptedPath) )

#'content\_type': 'video',

values = { 'username': self.authorization.username, 'encfs':'True', 'dpath': str(dencryptedPath) + str(dir), 'epath': str(encryptedPath), 'title': item.file.title, 'filename': item.file.id}

if item.file.type == 1:

url = self.PLUGIN\_URL+ '?mode=audio&' + urllib.urlencode(values)

else:

url = self.PLUGIN\_URL+ '?mode=video&' + urllib.urlencode(values)

#url = self.PLUGIN\_URL+'?mode=video&title='+str(item.file.title)+'&filename='+str(item.file.id)+ '&username='+str(self.authorization.username)

if url != 0:

title = str(dir)

if pDialog is not None:

pDialog.update(message=title)

if not xbmcvfs.exists(str(path) + '/' + str(title)+'.strm'):

filename = str(path) + '/' + str(title)+'.strm'

strmFile = xbmcvfs.File(filename, "w")

strmFile.write(url+'\n')

strmFile.close()

url=0

# file is already downloaded

for file in files:

index = ''

if encfs\_inode == 0:

index = str(xbmcvfs.Stat(encfs\_target + str(dencryptedPath) + file).st\_ino())

else:

index = str(xbmcvfs.Stat(encfs\_target + str(dencryptedPath) + file).st\_ctime())

if index in fileListINodes.keys():

fileListINodes[index].file.decryptedTitle = file

if contentType < 9 or media\_re.search(str(file)):

#service.addMediaFile(fileListINodes[index], contextType=contextType, encfs=True, dpath=str(dencryptedPath) + str(file), epath=str(encryptedPath) )

#'content\_type': 'video',

values = { 'username': self.authorization.username, 'encfs':'True', 'dpath': str(dencryptedPath) + str(dir), 'epath': str(encryptedPath), 'title': item.file.title, 'filename': item.file.id}

if item.file.type == 1:

url = self.PLUGIN\_URL+ '?mode=audio&' + urllib.urlencode(values)

else:

url = self.PLUGIN\_URL+ '?mode=video&' + urllib.urlencode(values)

#url = self.PLUGIN\_URL+'?mode=video&title='+str(item.file.title)+'&filename='+str(item.file.id)+ '&username='+str(self.authorization.username)

if url != 0:

title = str(dir)

if pDialog is not None:

pDialog.update(message=title)

if not xbmcvfs.exists(str(path) + '/' + str(title)+'.strm'):

filename = str(path) + '/' + str(title)+'.strm'

strmFile = xbmcvfs.File(filename, "w")

strmFile.write(url+'\n')

strmFile.close()

##

# build STRM files to a given path for a given folder ID

# parameters: path, folder id, content type, dialog object (optional)

##

def buildSTRM2(self, path, contentType=1, pDialog=None, spreadsheetFile=None):

import xbmcvfs

xbmcvfs.mkdir(path)

musicPath = path + '/music'

moviePath = path + '/movies'

tvPath = path + '/tv'

videoPath = path + '/video-other'

xbmcvfs.mkdir(musicPath)

xbmcvfs.mkdir(tvPath)

xbmcvfs.mkdir(videoPath)

xbmcvfs.mkdir(moviePath)

changeToken = self.addon.getSetting(self.instanceName+'\_changetoken')

count = 0

nextPageToken = ''

largestChangeId = ''

while True:

(mediaItems, nextPageToken, largestChangeId) = self.getChangeList(contentType=contentType, nextPageToken=nextPageToken, changeToken=changeToken)

if mediaItems:

for item in mediaItems:

url = ''

if item.file is not None:

count = count + 1

if pDialog is not None:

pDialog.update(message='STRMs created '+str(count))

#'content\_type': 'video',

values = { 'username': self.authorization.username, 'title': item.file.title, 'filename': item.file.id}

if item.file.type == 1:

url = self.PLUGIN\_URL+ '?mode=audio&' + urllib.urlencode(values)

filename = musicPath + '/' + str(title)+'.strm'

if item.file.deleted and xbmcvfs.exists(filename):

xbmcvfs.delete(filename)

elif not item.file.deleted and not xbmcvfs.exists(filename):

strmFile = xbmcvfs.File(filename, "w")

strmFile.write(url+'\n')

strmFile.close()

else:

url = self.PLUGIN\_URL+ '?mode=video&' + urllib.urlencode(values)

title = item.file.title

episode = ''

# nekwebdev contribution

pathLib = ''

tv = item.file.regtv1.match(title)

if not tv:

tv = item.file.regtv2.match(title)

if not tv:

tv = item.file.regtv3.match(title)

if tv:

show = tv.group(1).replace("\S{2,}\.\S{2,}", " ")

show = show.rstrip("\.")

season = tv.group(2)

episode = tv.group(3)

pathLib = tvPath + '/' + show

if not xbmcvfs.exists(xbmc.translatePath(pathLib)):

xbmcvfs.mkdir(xbmc.translatePath(pathLib))

pathLib = pathLib + '/Season ' + season

if not xbmcvfs.exists(xbmc.translatePath(pathLib)):

xbmcvfs.mkdir(xbmc.translatePath(pathLib))

else:

movie = item.file.regmovie.match(title)

if movie:

pathLib = moviePath

else:

pathLib = videoPath

if pathLib != '':

filename = str(pathLib) + '/' + str(title)+'.strm'

if item.file.deleted and xbmcvfs.exists(filename):

xbmcvfs.delete(filename)

elif not item.file.deleted and not xbmcvfs.exists(filename):

strmFile = xbmcvfs.File(filename, "w")

strmFile.write(url+'\n')

strmFile.close()

if spreadsheetFile is not None:

spreadsheetFile.write(str(item.folder.id) + '\t' + str(item.folder.title) + '\t'+str(item.file.id) + '\t'+str(item.file.title) + '\t'+str(episode)+'\t\t\t\t'+str(item.file.checksum) + '\t\t' + "\n")

self.addon.setSetting(self.instanceName + '\_changetoken', str(largestChangeId))

if nextPageToken == '' or nextPageToken is None:

break

##

# retrieve a directory url

# parameters: folder id, context type, whether the directory is encfs, encfs:decryption path, encfs:encryption path

# returns: fully qualified url

##

def getDirectoryCall(self, folder, contextType='video', encfs=False, dpath='', epath=''):

if encfs:

values = {'instance': self.instanceName, 'encfs': 'true', 'folder': folder.id, 'content\_type': contextType, 'dpath': dpath, 'epath':epath}

elif folder.id != '':

values = {'instance': self.instanceName, 'folder': folder.id, 'content\_type': contextType, 'epath':epath}

elif folder.title != '':

values = {'instance': self.instanceName, 'foldername': folder.title, 'content\_type': contextType, 'epath':epath}

return self.PLUGIN\_URL+'?mode=index&' + urllib.urlencode(values)

##

# download/retrieve a media file

# parameters: whether to playback file, media url object, package object, whether to force download (overwrite), whether the file is encfs, folder name (option)

##

def downloadMediaFile(self, mediaURL, item, package, force=False, folderName='', playback=1, player=None):

progress = ''

cachePercent = int(self.settings.cachePercent)

if cachePercent < 1:

cachePercent = 1

elif cachePercent > 100:

cachePercent = 100

fileSize = (int)(package.file.size)

if fileSize == '' or fileSize < 1000:

fileSize = 5000000

sizeDownload = fileSize \* (cachePercent\*0.01)

if sizeDownload < 1000000:

sizeDownload = 1000000

CHUNK = int(self.settings.cacheChunkSize)

if CHUNK < 1024:

CHUNK = 16 \* 1024

count = 0

try:

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

except:

pass

if not xbmcvfs.exists(path) and not os.path.exists(path):

path = ''

while path == '':

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

if not xbmcvfs.exists(path) and not os.path.exists(path):

path = ''

else:

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

if self.settings.cacheSingle:

playbackFile = str(path) + '/cache.mp4'

force= True

else:

try:

xbmcvfs.mkdir(str(path) + '/'+ str(package.file.id))

except: pass

playbackFile = str(path) + '/' + str(package.file.id) + '/' + str(mediaURL.order) + '.stream.mp4'

if not xbmcvfs.exists(str(path) + '/' + str(package.file.id) + '/' + str(package.file.id) + '.name') or force:

nameFile = xbmcvfs.File(str(path) + '/' + str(package.file.id) + '/' + str(package.file.id)+'.name' , "w")

nameFile.write(package.file.title +'\n')

nameFile.close()

if not xbmcvfs.exists(str(path) + '/' + str(package.file.id) + '/' + str(mediaURL.order) + '.stream.resolution') or force:

resolutionFile = xbmcvfs.File(str(path) + '/' + str(package.file.id) + '/' + str(mediaURL.order) + '.stream.resolution' , "w")

resolutionFile.write(mediaURL.qualityDesc +'\n')

resolutionFile.close()

if (not xbmcvfs.exists(playbackFile) or xbmcvfs.File(playbackFile).size() == 0 or xbmcvfs.File(playbackFile).size() < package.file.size) or force:

#seek to end of file for append

# - must use python for append (xbmcvfs not supported)

# - if path is not local or KODI-specific user must restart complete download

if os.path.exists(playbackFile) and xbmcvfs.File(playbackFile).size() < package.file.size and xbmcvfs.File(playbackFile).size() != 0 and not force:

req = urllib2.Request(mediaURL.url, None, self.getHeadersList(additionalHeader='Range', additionalValue='bytes='+str(xbmcvfs.File(playbackFile).size())+'-'+str(package.file.size)))

f = open(playbackFile, 'a')

else:

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

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

#print "DEBUG url = " + mediaURL.url + ", sizeDownload = " + str(sizeDownload) + ", playback = " + str(playback) + ", playbackFile = " + str(playbackFile)

# if playbackURL != '':

# progress = xbmcgui.DialogProgress()

# progressBar = sizeDownload

# progress.create(self.addon.getLocalizedString(30000), self.addon.getLocalizedString(30035), package.file.title)

# else:

progress = xbmcgui.DialogProgressBG()

progressBar = fileSize

progress.create(self.addon.getLocalizedString(30035), package.file.title)

# if action fails, validate login

try:

response = urllib2.urlopen(req)

except urllib2.URLError, e:

self.refreshToken()

req = urllib2.Request(mediaURL.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('downloadMediaFile',str(e))

return

downloadedBytes = 0

while sizeDownload > downloadedBytes:

progress.update((int)(float(downloadedBytes)/progressBar\*100),self.addon.getLocalizedString(30035))

chunk = response.read(CHUNK)

if not chunk: break

f.write(chunk)

downloadedBytes = downloadedBytes + CHUNK

if playback != self.PLAYBACK\_NONE:

item.setPath(playbackFile)

if playback == self.PLAYBACK\_RESOLVED:

xbmcplugin.setResolvedUrl(int(sys.argv[1]), True, item)

else:

#xbmc.executebuiltin("XBMC.PlayMedia("+playbackFile+")")

player.PlayStream(playbackFile, item, package.file.resume, startPlayback=True, package=package)

while not (player.isPlaying()) and not player.isExit:

xbmc.sleep(1000)

#print str(player.playStatus)

try:

count =1

while True:

if not self.settings.cacheContinue and player is not None and count % 12 == 0:

if not player.playStatus:

progress.close()

f.close()

return

count = count + 1

downloadedBytes = downloadedBytes + CHUNK

progress.update((int)(float(downloadedBytes)/progressBar\*100),self.addon.getLocalizedString(30092))

chunk = response.read(CHUNK)

if not chunk: break

f.write(chunk)

xbmc.sleep(1)

f.close()

progress.close()

except: pass

##

# download/retrieve a media file

# parameters: whether to playback file, media url object, package object, whether to force download (overwrite), whether the file is encfs, folder name (option)

##

def downloadEncfsFile(self, mediaURL, package, playbackURL='', force=False, folderName='', playback=1,item='', player=None, srt=None):

progress = ''

cachePercent = int(self.settings.encfsCachePercent)

if cachePercent < 1:

cachePercent = 1

elif cachePercent > 100:

cachePercent = 100

fileSize = (int)(package.file.size)

if fileSize == '' or fileSize < 1000:

fileSize = 5000000

sizeDownload = fileSize \* (cachePercent\*0.01)

if sizeDownload < 3000000:

sizeDownload = 3000000

CHUNK = int(self.settings.encfsCacheChunkSize)

if CHUNK < 1024:

CHUNK = 131072

count = 0

path = re.sub(r'\/[^\/]+$', r'', folderName)

if folderName == path:

path = re.sub(r'\\[^\\]+$', r'', folderName) #needed for windows?

#ensure the folder and path exists

try:

xbmcvfs.mkdirs(path)

except: pass

playbackFile = folderName

if (not xbmcvfs.exists(playbackFile) or long(xbmcvfs.File(playbackFile).size()) == 0 or long(xbmcvfs.File(playbackFile).size()) < long(package.file.size)) or force:

if not self.settings.encfsStream and not self.settings.encfsCacheSingle:

progress = xbmcgui.DialogProgressBG()

progressBar = fileSize

progress.create(self.addon.getLocalizedString(30035), playbackURL)

downloadedBytes = 0

#seek to end of file for append

# - must use python for append (xbmcvfs not supported)

# - if path is not local or KODI-specific user must restart complete download

if os.path.exists(playbackFile) and long(xbmcvfs.File(playbackFile).size()) < long(package.file.size) and long(xbmcvfs.File(playbackFile).size()) != 0 and not force:

req = urllib2.Request(mediaURL.url, None, self.getHeadersList(additionalHeader='Range', additionalValue='bytes='+str(xbmcvfs.File(playbackFile).size())+'-'+str(package.file.size)))

f = open(playbackFile, 'a')

# if action fails, validate login

try:

response = urllib2.urlopen(req)

except urllib2.URLError, e:

self.refreshToken()

req = urllib2.Request(mediaURL.url, None, self.getHeadersList(additionalHeader='Range', additionalValue='bytes='+str(xbmcvfs.File(playbackFile).size())+'-'+str(package.file.size)))

try:

response = urllib2.urlopen(req)

except urllib2.URLError, e:

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

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

return

else:

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

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

# if action fails, validate login

try:

response = urllib2.urlopen(req)

except urllib2.URLError, e:

self.refreshToken()

req = urllib2.Request(mediaURL.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('downloadMediaFile',str(e))

return

while sizeDownload > downloadedBytes:

if not self.settings.encfsStream and not self.settings.encfsCacheSingle:

progress.update((int)(float(downloadedBytes)/progressBar\*100),self.addon.getLocalizedString(30035))

chunk = response.read(CHUNK)

if not chunk: break

f.write(chunk)

downloadedBytes = downloadedBytes + CHUNK

if playbackURL != '':

if playback == True:#self.PLAYBACK\_NONE:

item.setPath(playbackURL)

#if playback == self.PLAYBACK\_RESOLVED:

xbmcplugin.setResolvedUrl(int(sys.argv[1]), True, item)

#else:

#xbmc.executebuiltin("XBMC.PlayMedia("+playbackFile+")")

#player.PlayStream(playbackURL, item, package.file.resume, startPlayback=True, package=package)

# while not (player.isPlaying()) and not player.isExit:

# xbmc.sleep(1000)

#print str(player.playStatus)

# load captions

if (self.settings.srt or self.settings.cc):

while not (player.isPlaying()):

xbmc.sleep(1000)

for file in srt:

if file != '':

try:

file = file.decode('unicode-escape')

file = file.encode('utf-8')

except:

pass

player.setSubtitles(file)

# need to seek?

#if seek > 0:

#player.PlayStream('', item, 99, startPlayback=False, package=package)

try:

count =1

while True:

if (self.settings.encfsStream or not self.settings.encfsContinue) and player is not None and count % 12 == 0:

if not player.playStatus:

if not self.settings.encfsStream and not self.settings.encfsCacheSingle:

progress.close()

f.close()

return

count = count + 1

downloadedBytes = downloadedBytes + CHUNK

if not self.settings.encfsStream and not self.settings.encfsCacheSingle:

progress.update((int)(float(downloadedBytes)/progressBar\*100),self.addon.getLocalizedString(30092))

chunk = response.read(CHUNK)

if not chunk: break

f.write(chunk)

xbmc.sleep(1)

f.close()

if not self.settings.encfsStream and not self.settings.encfsCacheSingle:

progress.close()

except: pass

##

# download/retrieve a media file

# parameters: whether to playback file, media url object, package object, whether to force download (overwrite), whether the file is encfs, folder name (option)

##

def downloadEncfsFile2(self, mediaURL, package, playbackURL='', force=False, folderName='', playback=1,item='', player=None, srt=None):

cachePercent = int(self.settings.encfsCachePercent)

if cachePercent < 1:

cachePercent = 1

elif cachePercent > 100:

cachePercent = 100

fileSize = (int)(package.file.size)

if fileSize == '' or fileSize < 1000:

fileSize = 5000000

sizeDownload = fileSize \* (cachePercent\*0.01)

if sizeDownload < 3000000:

sizeDownload = 3000000

CHUNK = int(self.settings.encfsCacheChunkSize)

if CHUNK < 1024:

CHUNK = 131072

count = 0

path = re.sub(r'\/[^\/]+$', r'', folderName)

if folderName == path:

path = re.sub(r'\\[^\\]+$', r'', folderName) #needed for windows?

#ensure the folder and path exists

try:

xbmcvfs.mkdirs(path)

except: pass

playbackFile = folderName

downloadedBytes = 0

#seek to end of file for append

# - must use python for append (xbmcvfs not supported)

# - if path is not local or KODI-specific user must restart complete download

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

if (1):

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

# if action fails, validate login

try:

response = urllib2.urlopen(req)

except urllib2.URLError, e:

self.refreshToken()

req = urllib2.Request(mediaURL.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('downloadMediaFile',str(e))

return

CHUNK = 4096\*100

header = response.read(CHUNK)

#f.write(header)

if (1):

f.write(header)

f.close()

if playbackURL != '':

if playback == True:#self.PLAYBACK\_NONE:

if (0):

pid = os.fork()

if (pid == 0): #child

from BaseHTTPServer import BaseHTTPRequestHandler,HTTPServer

server = streamer.MyHTTPServer(('', 8003), streamer.myStreamer)

server.setFile(playbackURL,CHUNK, playbackFile, response, fileSize, mediaURL.url, self)

while server.ready:

try:

server.handle\_request()

except: pass

server.socket.close()

# os.\_exit()

else:

item.setPath('http://localhost:8005')

xbmcplugin.setResolvedUrl(int(sys.argv[1]), True, item)

from BaseHTTPServer import BaseHTTPRequestHandler,HTTPServer

try:

server = streamer.MyHTTPServer(('', 8006), streamer.myStreamer)

except:

req = urllib2.Request('http://localhost:8005/kill', None, None)

try:

response = urllib2.urlopen(req)

except: pass

server = streamer.MyHTTPServer(('', 8006), streamer.myStreamer)

server.setFile(playbackURL,CHUNK, playbackFile, response, fileSize, mediaURL.url, self)

item.setPath('http://localhost:8006')

xbmcplugin.setResolvedUrl(int(sys.argv[1]), True, item)

setCC = True

while server.ready:

try:

server.handle\_request()

except:

break

if (setCC and (self.settings.srt or self.settings.cc)):

setCC = False

while not (player.isPlaying()):

xbmc.sleep(1000)

for file in srt:

if file != '':

try:

file = file.decode('unicode-escape')

file = file.encode('utf-8')

except:

pass

player.setSubtitles(file)

server.socket.close()

#else:

#xbmc.executebuiltin("XBMC.PlayMedia("+playbackFile+")")

#player.PlayStream(playbackURL, item, package.file.resume, startPlayback=True, package=package)

# while not (player.isPlaying()) and not player.isExit:

# xbmc.sleep(1000)

#print str(player.playStatus)

# load captions

if (self.settings.srt or self.settings.cc):

while not (player.isPlaying()):

xbmc.sleep(1000)

for file in srt:

if file != '':

try:

file = file.decode('unicode-escape')

file = file.encode('utf-8')

except:

pass

player.setSubtitles(file)

##

# download remote picture

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

##

def downloadGeneralFile(self, url, file, force=False):

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

# already downloaded

if not force and xbmcvfs.exists(file) and xbmcvfs.File(file).size() > 0:

return

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

# if action fails, validate login

try:

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

f.close()

except urllib2.URLError, e:

self.refreshToken()

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

try:

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('downloadGeneralFle',str(e))

return None

#can't write to cache for some reason

except IOError:

return None

return file

##

# retrieve/download a general file

# parameters: title of video, whether to prompt for quality/format (optional), medial url object, package object, whether to force download (overwrite), whether folder is encrypted, folder name

##

def downloadGeneralFileOLD(self, playback, mediaURL, package, force=False, encfs=False, folderName=''):

cachePercent = int(self.settings.cachePercent)

if cachePercent < 1:

cachePercent = 1

elif cachePercent > 100:

cachePercent = 100

fileSize = (int)(package.file.size)

if fileSize == '' or fileSize < 1000:

fileSize = 5000000

sizeDownload = fileSize \* (cachePercent\*0.01)

if sizeDownload < 1000000:

sizeDownload = 1000000

CHUNK = int(self.settings.cacheChunkSize)

if CHUNK < 1024:

CHUNK = 131072

count = 0

if encfs:

try:

path = self.addon.getSetting('encfs\_source')

except:

pass

else:

try:

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

except:

pass

#workaround for this issue: https://github.com/xbmc/xbmc/pull/8531

if not xbmcvfs.exists(path) and not os.path.exists(path):

path = ''

while path == '':

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

if not xbmcvfs.exists(path) and not os.path.exists(path):

path = ''

else:

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

if encfs:

try:

xbmcvfs.mkdir(str(path) + '/'+str(folderName))

except: pass

playbackFile = str(path) + '/' + str(folderName) + '/' + str(package.file.title)

else:

try:

xbmcvfs.mkdir(str(path) + '/'+ str(package.file.id))

except: pass

playbackFile = str(path) + '/' + str(package.file.id) + '/' + str(mediaURL.order) + '.stream.mp4'

if (not xbmcvfs.exists(playbackFile) or xbmcvfs.File(playbackFile).size() == 0) or force:

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

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

if playback != '':

progress = xbmcgui.DialogProgress()

progressBar = sizeDownload

progress.create(self.addon.getLocalizedString(30000), self.addon.getLocalizedString(30035), package.file.title)

else:

progress = xbmcgui.DialogProgressBG()

progressBar = fileSize

progress.create(self.addon.getLocalizedString(30035), package.file.title)

# if action fails, validate login

try:

response = urllib2.urlopen(req)

except urllib2.URLError, e:

self.refreshToken()

req = urllib2.Request(mediaURL.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('downloadMediaFile',str(e))

return

downloadedBytes = 0

while sizeDownload > downloadedBytes:

progress.update((int)(float(downloadedBytes)/progressBar\*100),self.addon.getLocalizedString(30035))

chunk = response.read(CHUNK)

if not chunk: break

f.write(chunk)

downloadedBytes = downloadedBytes + CHUNK

if playback != '':

try:

progress.close()

except:

pass

#item = xbmcgui.ListItem(path=playbackFile)

item = xbmcgui.ListItem(package.file.displayTitle(), iconImage=package.file.thumbnail,

thumbnailImage=package.file.thumbnail)#, path=playbackPath+'|' + service.getHeadersEncoded())

item.setInfo( type="Video", infoLabels={ "Title": package.file.title , "Plot" : package.file.title } )

xbmcplugin.setResolvedUrl(playback, True, item)

xbmc.executebuiltin("XBMC.PlayMedia("+playbackFile+")")

try:

while True:

downloadedBytes = downloadedBytes + CHUNK

progress.update((int)(float(downloadedBytes)/progressBar\*100),self.addon.getLocalizedString(30092))

chunk = response.read(CHUNK)

if not chunk: break

f.write(chunk)

f.close()

progress.close()

except: pass

##

# Add a directory to a directory listing screen

# parameters: folder object, context type, local path (optional), whether folder is encfs, encfs:decryption path, encfs:encryption path

##

def addDirectory(self, folder, contextType='video', localPath='', encfs=False, dpath='', epath=''):

fanart = self.addon.getAddonInfo('path') + '/fanart.jpg'

if folder is None:

listitem = xbmcgui.ListItem('[Decrypted Folder]')

# listitem.addContextMenuItems(cm, False)

listitem.setProperty('fanart\_image', fanart)

xbmcplugin.addDirectoryItem(plugin\_handle, localPath, listitem,

isFolder=True, totalItems=0)

else:

if folder.id == 'SAVED SEARCH':

listitem = xbmcgui.ListItem(decode(folder.displayTitle()), iconImage=decode(folder.thumb), thumbnailImage=decode(folder.thumb))

values = {'instance': self.instanceName, 'title': folder.title}

url = self.PLUGIN\_URL+'?mode=search&content\_type='+contextType + '&' + urllib.urlencode(values)

xbmcplugin.addDirectoryItem(plugin\_handle, url, listitem,

isFolder=True, totalItems=0)

elif folder.id == 'CLOUD\_DB\_GENRE':

listitem = xbmcgui.ListItem(decode(folder.displayTitle()), iconImage=decode(folder.thumb), thumbnailImage=decode(folder.thumb))

values = {'instance': self.instanceName, 'title': folder.title}

url = self.PLUGIN\_URL+'?mode=cloud\_dbtest&action=genre&content\_type='+contextType + '&' + urllib.urlencode(values)

xbmcplugin.addDirectoryItem(plugin\_handle, url, listitem,

isFolder=True, totalItems=0)

elif folder.id == 'CLOUD\_DB\_TITLE':

listitem = xbmcgui.ListItem(decode(folder.displayTitle()), iconImage=decode(folder.thumb), thumbnailImage=decode(folder.thumb))

values = {'instance': self.instanceName, 'title': folder.title}

url = self.PLUGIN\_URL+'?mode=cloud\_dbtest&action=title&content\_type='+contextType + '&' + urllib.urlencode(values)

xbmcplugin.addDirectoryItem(plugin\_handle, url, listitem,

isFolder=True, totalItems=0)

elif folder.id == 'CLOUD\_DB\_RESOLUTION':

listitem = xbmcgui.ListItem(decode(folder.displayTitle()), iconImage=decode(folder.thumb), thumbnailImage=decode(folder.thumb))

values = {'instance': self.instanceName, 'title': folder.title}

url = self.PLUGIN\_URL+'?mode=cloud\_dbtest&action=resolution&content\_type='+contextType + '&' + urllib.urlencode(values)

xbmcplugin.addDirectoryItem(plugin\_handle, url, listitem,

isFolder=True, totalItems=0)

elif folder.id == 'CLOUD\_DB\_YEAR':

listitem = xbmcgui.ListItem(decode(folder.displayTitle()), iconImage=decode(folder.thumb), thumbnailImage=decode(folder.thumb))

values = {'instance': self.instanceName, 'title': folder.title}

url = self.PLUGIN\_URL+'?mode=cloud\_dbtest&action=year&content\_type='+contextType + '&' + urllib.urlencode(values)

xbmcplugin.addDirectoryItem(plugin\_handle, url, listitem,

isFolder=True, totalItems=0)

elif folder.id == 'CLOUD\_DB\_COUNTRY':

listitem = xbmcgui.ListItem(decode(folder.displayTitle()), iconImage=decode(folder.thumb), thumbnailImage=decode(folder.thumb))

values = {'instance': self.instanceName, 'title': folder.title}

url = self.PLUGIN\_URL+'?mode=cloud\_dbtest&action=country&content\_type='+contextType + '&' + urllib.urlencode(values)

xbmcplugin.addDirectoryItem(plugin\_handle, url, listitem,

isFolder=True, totalItems=0)

elif folder.id == 'CLOUD\_DB\_DIRECTOR':

listitem = xbmcgui.ListItem(decode(folder.displayTitle()), iconImage=decode(folder.thumb), thumbnailImage=decode(folder.thumb))

values = {'instance': self.instanceName, 'title': folder.title}

url = self.PLUGIN\_URL+'?mode=cloud\_dbtest&action=director&content\_type='+contextType + '&' + urllib.urlencode(values)

xbmcplugin.addDirectoryItem(plugin\_handle, url, listitem,

isFolder=True, totalItems=0)

elif folder.id == 'CLOUD\_DB\_STUDIO':

listitem = xbmcgui.ListItem(decode(folder.displayTitle()), iconImage=decode(folder.thumb), thumbnailImage=decode(folder.thumb))

values = {'instance': self.instanceName, 'title': folder.title}

url = self.PLUGIN\_URL+'?mode=cloud\_dbtest&action=studio&content\_type='+contextType + '&' + urllib.urlencode(values)

xbmcplugin.addDirectoryItem(plugin\_handle, url, listitem,

isFolder=True, totalItems=0)

else:

listitem = xbmcgui.ListItem(decode(folder.displayTitle()), iconImage=decode(folder.thumb), thumbnailImage=decode(folder.thumb))

cm=[]

# is a real folder

if folder.id != '':

if contextType != 'image' and not encfs:

values = {'username': self.authorization.username, 'title': folder.title, 'folder': folder.id, 'content\_type': contextType }

cm.append(( self.addon.getLocalizedString(30042), 'XBMC.RunPlugin('+self.PLUGIN\_URL+'?mode=buildstrm&'+ urllib.urlencode(values)+')', ))

#encfs

elif contextType != 'image':

values = {'username': self.authorization.username, 'epath': epath, 'dpath': dpath, 'encfs':'true' ,'title': folder.title, 'folder': folder.id, 'content\_type': contextType }

cm.append(( self.addon.getLocalizedString(30042), 'XBMC.RunPlugin('+self.PLUGIN\_URL+'?mode=buildstrm&'+ urllib.urlencode(values)+')', ))

elif contextType == 'image':

# slideshow

if encfs:

values = {'encfs': 'true', 'username': self.authorization.username, 'title': folder.title, 'folder': folder.id}

else:

values = {'username': self.authorization.username, 'title': folder.title, 'folder': folder.id}

#cm.append(( self.addon.getLocalizedString(30126), 'XBMC.RunPlugin('+self.PLUGIN\_URL+'?mode=slideshow&'+urllib.urlencode(values)+')', ))

if (self.protocol == 2):

if contextType != 'image':

#download folder

values = {'instance': self.instanceName, 'title': folder.title, 'folder': folder.id}

cm.append(( self.addon.getLocalizedString(30113), 'XBMC.RunPlugin('+self.PLUGIN\_URL+'?mode=downloadfolder&'+urllib.urlencode(values)+')', ))

if contextType == 'audio' and not encfs:

#playback entire folder

values = {'instance': self.instanceName, 'folder': folder.id}

cm.append(( self.addon.getLocalizedString(30162), 'XBMC.RunPlugin('+self.PLUGIN\_URL+'?mode=audio&content\_type='+contextType+'&'+urllib.urlencode(values)+')', ))

elif contextType == 'video' and not encfs:

#playback entire folder

values = {'instance': self.instanceName, 'folder': folder.id}

cm.append(( self.addon.getLocalizedString(30162), 'XBMC.RunPlugin('+self.PLUGIN\_URL+'?mode=video&content\_type='+contextType+'&'+urllib.urlencode(values)+')', ))

#add encfs option unless viewing as encfs already

if not encfs:

cm.append(( self.addon.getLocalizedString(30192), 'XBMC.Container.Update('+self.PLUGIN\_URL+'?mode=index&content\_type='+contextType+'&encfs=true&'+urllib.urlencode(values)+')', ))

cm.append(( self.addon.getLocalizedString(30193), 'XBMC.Container.Update('+self.PLUGIN\_URL+'?mode=index&content\_type='+contextType+'&encfs=true&'+urllib.urlencode(values)+')', ))

#if within encfs and pictures, disable right-click default photo options; add download-folder

if encfs and contextType == 'image':

values = {'instance': self.instanceName, 'epath': epath, 'foldername': folder.title, 'folder': folder.id}

cm.append(( self.addon.getLocalizedString(30113), 'XBMC.RunPlugin('+self.PLUGIN\_URL+'?mode=downloadfolder&content\_type='+contextType+'&encfs=true&'+urllib.urlencode(values)+')', ))

listitem.addContextMenuItems(cm, True)

elif contextType == 'image':

cm.append(( self.addon.getLocalizedString(30113), 'XBMC.RunPlugin('+self.PLUGIN\_URL+'?mode=downloadfolder&content\_type='+contextType+'&'+urllib.urlencode(values)+')', ))

cm.append(( self.addon.getLocalizedString(30163), 'XBMC.RunPlugin('+self.PLUGIN\_URL+'?mode=scan&content\_type='+contextType+'&'+urllib.urlencode(values)+')', ))

else:

if contextType != 'image' and not encfs:

values = {'username': self.authorization.username, 'title': folder.title, 'content\_type': contextType }

cm.append(( self.addon.getLocalizedString(30042), 'XBMC.RunPlugin('+self.PLUGIN\_URL+'?mode=buildstrm&'+ urllib.urlencode(values)+')', ))

listitem.addContextMenuItems(cm, False)

listitem.setProperty('fanart\_image', folder.fanart)

xbmcplugin.addDirectoryItem(plugin\_handle, self.getDirectoryCall(folder, contextType, encfs=encfs, dpath=dpath, epath=epath), listitem,

isFolder=True, totalItems=0)

##

# Add a media file to a directory listing screen

# parameters: package, context type, whether file is encfs, encfs:decryption path, encfs:encryption path

##

def addMediaFile(self, package, contextType='video', encfs=False, dpath='', epath=''):

thumbnail = self.cache.getThumbnail(self, package.file.thumbnail,package.file.id)

listitem = xbmcgui.ListItem(package.file.displayTitle(), iconImage=package.file.thumbnail,

thumbnailImage=package.file.thumbnail)

# audio file, not in "pictures"

if package.file.type == package.file.AUDIO and contextType != 'image':

if package.file.hasMeta:

#infolabels = decode\_dict({ 'title' : package.file.displayTrackTitle(), 'tracknumber' : package.file.trackNumber, 'artist': package.file.artist, 'album': package.file.album,'genre': package.file.genre,'premiered': package.file.releaseDate, 'size' : package.file.size })

infolabels = decode\_dict({ 'title' : package.file.displayTitle(), 'size' : package.file.size })

else:

infolabels = decode\_dict({ 'title' : package.file.displayTitle(), 'size' : package.file.size })

listitem.setInfo('Music', infolabels)

playbackURL = '?mode=audio'

if self.integratedPlayer:

listitem.setProperty('IsPlayable', 'false')

else:

listitem.setProperty('IsPlayable', 'true')

# encrypted file, viewing in "pictures", assume image

elif package.file.type == package.file.UNKNOWN and contextType == 'image':

infolabels = decode\_dict({ 'title' : package.file.displayTitle() , 'plot' : package.file.plot })

listitem.setInfo('Pictures', infolabels)

playbackURL = '?mode=photo'

listitem.setProperty('IsPlayable', 'false')

# encrypted file, viewing in "video", assume video

elif package.file.type == package.file.UNKNOWN and contextType == 'video':

infolabels = decode\_dict({ 'title' : package.file.displayTitle() , 'plot' : package.file.plot, 'size' : package.file.size })

listitem.setInfo('Video', infolabels)

playbackURL = '?mode=video'

if self.integratedPlayer:

listitem.setProperty('IsPlayable', 'false')

else:

listitem.setProperty('IsPlayable', 'true')

if float(package.file.cloudResume) > 0:

listitem.setProperty('isResumable', 1)

# encrypted file, viewing in "music", assume audio

elif package.file.type == package.file.UNKNOWN and contextType == 'audio':

if package.file.hasMeta:

infolabels = decode\_dict({ 'title' : package.file.displayTrackTitle(), 'tracknumber' : package.file.trackNumber, 'artist': package.file.artist, 'album': package.file.album,'genre': package.file.genre,'premiered': package.file.releaseDate, 'size' : package.file.size })

else:

infolabels = decode\_dict({ 'title' : package.file.displayTitle(), 'size' : package.file.size })

listitem.setInfo('Music', infolabels)

playbackURL = '?mode=audio'

if self.integratedPlayer:

listitem.setProperty('IsPlayable', 'false')

else:

listitem.setProperty('IsPlayable', 'true')

# audio file, viewing in "pictures"

elif package.file.type == package.file.AUDIO and contextType == 'image':

if package.file.hasMeta:

infolabels = decode\_dict({ 'title' : package.file.displayTrackTitle(), 'tracknumber' : package.file.trackNumber, 'artist': package.file.artist, 'album': package.file.album,'genre': package.file.genre,'premiered': package.file.releaseDate, 'size' : package.file.size })

else:

infolabels = decode\_dict({ 'title' : package.file.displayTitle(), 'size' : package.file.size })

listitem.setInfo('Music', infolabels)

playbackURL = '?mode=audio'

listitem.setProperty('IsPlayable', 'false')

# video file

elif package.file.type == package.file.VIDEO:

if package.file.hasMeta:

infolabels = decode\_dict({ 'title' : package.file.displayShowTitle() , 'plot' : package.file.plot, 'TVShowTitle': package.file.show, 'EpisodeName': package.file.showtitle, 'season': package.file.season, 'episode': package.file.episode,'size' : package.file.size })

else:

if package.file.actors != None:

infolabels = decode\_dict({ 'title' : package.file.displayTitle() , 'cast': package.file.actors, 'plot' : package.file.plot, 'ratingandvotes' : package.file.rating, 'director': package.file.director, 'set': package.file.set, 'country': package.file.country, 'genre': package.file.genre, 'year': package.file.year, 'size' : package.file.size})

else:

infolabels = decode\_dict({ 'title' : package.file.displayTitle() , 'plot' : package.file.plot, 'ratingandvotes' : package.file.rating, 'director': package.file.director, 'set': package.file.set, 'country': package.file.country, 'genre': package.file.genre, 'year': package.file.year, 'size' : package.file.size})

listitem.setInfo('Video', infolabels)

playbackURL = '?mode=video'

if self.integratedPlayer:

listitem.setProperty('IsPlayable', 'false')

else:

listitem.setProperty('IsPlayable', 'true')

if float(package.file.cloudResume) > 0:

listitem.setProperty('isResumable', "1")

if int(package.file.playcount) > 0: #or (float(package.file.resume) > 0 and package.file.duration > 0 and package.file.resume/package.file.duration > (1-self.settskipResume)):

listitem.setInfo('video', {'playcount':int(package.file.playcount)})

if package.file.resolution is not None and int(package.file.resolution[0]) > 0:

listitem.addStreamInfo('video', {'width': package.file.resolution[1], 'height': package.file.resolution[0], 'duration':package.file.duration})

# image file

elif package.file.type == package.file.PICTURE:

infolabels = decode\_dict({ 'title' : package.file.displayTitle() , 'plot' : package.file.plot, 'size' : package.file.size })

listitem.setInfo('Pictures', infolabels)

listitem.setProperty('mimetype', 'image/jpeg')

playbackURL = '?mode=photo'

# listitem.setProperty('IsPlayable', 'false')

listitem.setProperty('IsPlayable', 'true')

if package.mediaurl.url != '':

url = package.mediaurl.url +'|' + self.getHeadersEncoded()

else:

url = package.file.download+'|' + self.getHeadersEncoded()

xbmcplugin.addDirectoryItem(plugin\_handle, url, listitem,

isFolder=False, totalItems=0)

return url

# otherwise, assume video

else:

infolabels = decode\_dict({ 'title' : package.file.displayTitle() , 'plot' : package.file.plot, 'size' : package.file.size })

listitem.setInfo('Video', infolabels)

playbackURL = '?mode=video'

if self.integratedPlayer:

listitem.setProperty('IsPlayable', 'false')

else:

listitem.setProperty('IsPlayable', 'true')

if float(package.file.cloudResume) > 0:

listitem.setProperty('isResumable', 1)

listitem.setProperty('fanart\_image', package.file.fanart)

if package.file.cloudResume > 0:

listitem.setProperty('ResumeTime', str(package.file.resume))

if package.file.duration > 0:

listitem.setProperty('TotalTime', str(package.file.duration))

cm=[]

try:

url = package.getMediaURL()

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

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

except:

cleanURL = ''

# url = PLUGIN\_URL+playbackURL+'&title='+package.file.title+'&filename='+package.file.id+'&instance='+str(self.instanceName)+'&folder='+str(package.folder.id)

if encfs:

values = {'instance': self.instanceName, 'dpath': dpath, 'epath': epath, 'encfs': 'true', 'title': package.file.title, 'filename': package.file.id, 'folder': package.folder.id}

elif package.file.id == '':

values = {'instance': self.instanceName, 'title': package.file.title, 'sheet': 'x', 'year': package.file.year, 'folder': package.folder.id}

else:

values = {'instance': self.instanceName, 'title': package.file.title, 'filename': package.file.id, 'folder': package.folder.id}

url = self.PLUGIN\_URL+ str(playbackURL)+ '&' + urllib.urlencode(values)

if (contextType != 'image' and package.file.type != package.file.PICTURE):

#STRM

if encfs:

valuesBS = {'username': self.authorization.username, 'dpath': dpath, 'epath': epath, 'encfs': 'true', 'title': package.file.title, 'filename': package.file.id, 'content\_type': 'video'}

cm.append(( self.addon.getLocalizedString(30042), 'XBMC.RunPlugin('+self.PLUGIN\_URL+'?mode=buildstrm&type='+str(package.file.type)+'&'+urllib.urlencode(valuesBS)+')', ))

else:

valuesBS = {'username': self.authorization.username, 'title': package.file.title, 'filename': package.file.id, 'content\_type': 'video'}

cm.append(( self.addon.getLocalizedString(30042), 'XBMC.RunPlugin('+self.PLUGIN\_URL+'?mode=buildstrm&type='+str(package.file.type)+'&'+urllib.urlencode(valuesBS)+')', ))

if (self.protocol == 2):

# play-original for video only

if (contextType == 'video'):

if (package.file.type != package.file.AUDIO and self.settings.promptQuality) and not encfs:

cm.append(( self.addon.getLocalizedString(30123), 'XBMC.RunPlugin('+url + '&strm=false&original=true'+')', ))

else:

cm.append(( self.addon.getLocalizedString(30151), 'XBMC.RunPlugin('+url + '&strm=false&promptquality=true'+')', ))

# if the options are disabled in settings, display option to playback with feature

if not self.settings.srt:

cm.append(( self.addon.getLocalizedString(30138), 'XBMC.RunPlugin('+url + '&strm=false&srt=true'+')', ))

if not self.settings.cc:

cm.append(( self.addon.getLocalizedString(30146), 'XBMC.RunPlugin('+url + '&strm=false&cc=true'+')', ))

cm.append(( self.addon.getLocalizedString(30147), 'XBMC.RunPlugin('+url + '&strm=false&seek=true'+')', ))

# cm.append(( self.addon.getLocalizedString(30148), 'XBMC.RunPlugin('+url + '&resume=true'+')', ))

# values = {'instance': self.instanceName, 'folder': package.folder.id}

# folderurl = self.PLUGIN\_URL+ str(playbackURL)+ '&' + urllib.urlencode(values)

# cm.append(( 'folder', 'XBMC.RunPlugin('+folderurl+')', ))

if contextType != 'image':

# download

cm.append(( self.addon.getLocalizedString(30113), 'XBMC.RunPlugin('+url + '&download=true'+')', ))

# download + watch

if not encfs:

cm.append(( self.addon.getLocalizedString(30124), 'XBMC.RunPlugin('+url + '&play=true&download=true'+')', ))

elif package.file.type == package.file.PICTURE: #contextType == 'image':

cm.append(( self.addon.getLocalizedString(30126), 'XBMC.SlideShow('+self.PLUGIN\_URL+ '?mode=index&' + urllib.urlencode(values)+')', ))

#encfs

# if (self.protocol == 2):

# cm.append(( self.addon.getLocalizedString(30130), 'XBMC.RunPlugin('+self.PLUGIN\_URL+ '?mode=downloadfolder&encfs=true&' + urllib.urlencode(values)+'&content\_type='+contextType+')', ))

#CLOUD\_DB

if self.gSpreadsheet is not None:

cm.append(( self.addon.getLocalizedString(30177) + self.addon.getLocalizedString(30178), 'XBMC.RunPlugin('+self.PLUGIN\_URL+'?mode=cloud\_db&' + urllib.urlencode(values) + '&action=watch'+')', ))

cm.append(( self.addon.getLocalizedString(30177) + self.addon.getLocalizedString(30179), 'XBMC.RunPlugin('+self.PLUGIN\_URL+'?mode=cloud\_db&' + urllib.urlencode(values) + '&action=queue'+')', ))

url = url + '&content\_type='+contextType

# listitem.addContextMenuItems( commands )

# if cm:

if package.file.type == package.file.PICTURE: #contextType == 'image':

listitem.addContextMenuItems(cm, True)

else:

listitem.addContextMenuItems(cm, False)

xbmcplugin.addDirectoryItem(plugin\_handle, url, listitem,

isFolder=False, totalItems=0)

return url

##

# Return the user selected media source

# parameters: list of media url objects, folder id, file id

# returns: select media url object

##

def getMediaSelection(self, mediaURLs, folderID, filename):

options = []

newMediaURLs = []

mediaURLs = sorted(mediaURLs)

if self.settings.playOriginal:

for mediaURL in mediaURLs:

if mediaURL.qualityDesc == 'original':

options.append(mediaURL.qualityDesc)

newMediaURLs.append(mediaURL)

originalURL = mediaURL.url

else:

for mediaURL in mediaURLs:

options.append(mediaURL.qualityDesc)

newMediaURLs.append(mediaURL)

if mediaURL.qualityDesc == 'original':

originalURL = mediaURL.url

mediaURL = ''

# if self.settings.download or self.settings.cache:

# mediaURL = mediaurl.mediaurl(originalURL, 'original', 0, 9999)

# return mediaURL

#elif self.settings.playOriginal:

# mediaURL = mediaurl.mediaurl(originalURL +'|' + self.getHeadersEncoded(), 'original', 0, 9999)

# return mediaURL

#playbackPath = str(self.settings.cachePath) + '/' + str(filename) + '/'

localResolutions = []

localFiles = []

# if we are not downloading-only

if self.settings.play:

(localResolutions,localFiles) = self.cache.getFiles(self)

totalList = localFiles + newMediaURLs

mediaCount = len(localFiles)

if self.settings.promptQuality:

ret = xbmcgui.Dialog().select(self.addon.getLocalizedString(30033), localResolutions + options)

if ret >= mediaCount:

mediaURL = totalList[ret]

if self.settings.download or self.settings.cache:

mediaURL.url = totalList[ret].url

else:

mediaURL.url = totalList[ret].url +'|' + self.getHeadersEncoded()

else:

mediaURL = mediaurl.mediaurl(str(totalList[ret]), 'offline', 0, 0)

mediaURL.offline = True

else:

if len(localFiles) == 0:

mediaURL = totalList[0]

if self.settings.download or self.settings.cache:

mediaURL.url = totalList[0].url

else:

mediaURL.url = totalList[0].url +'|' + self.getHeadersEncoded()

else:

mediaURL = mediaurl.mediaurl(str(totalList[0]), 'offline', 0, 0)

mediaURL.offline = True

# elif self.settings.promptQuality and len(options) > 1 and not self.settings.cache:

# ret = xbmcgui.Dialog().select(self.addon.getLocalizedString(30033), options)

# mediaURL = mediaURLs[ret]

# if not self.settings.download:

# mediaURLs[ret].url = mediaURLs[ret].url +'|' + self.getHeadersEncoded()

# else:

# mediaURLs[0].url = mediaURLs[0].url +'|' + self.getHeadersEncoded()

# mediaURL = mediaURLs[0]

return mediaURL

##

# download remote picture

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

##

def downloadPicture(self, url, file):

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

# already downloaded

if xbmcvfs.exists(file) and xbmcvfs.File(file).size() > 0:

return

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

# if action fails, validate login

try:

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

f.close()

except urllib2.URLError, e:

self.refreshToken()

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

try:

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 None

#can't write to cache for some reason

except IOError:

return None

return file