'''

Copyright (C) 2014-2016 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/>.

'''

from BaseHTTPServer import BaseHTTPRequestHandler,HTTPServer

from SocketServer import ThreadingMixIn

import threading

import re

import urllib, urllib2

import xbmc, xbmcaddon, xbmcgui, xbmcplugin

class ThreadedHTTPServer(ThreadingMixIn, HTTPServer):

"""Handle requests in a separate thread."""

class MyHTTPServer(ThreadingMixIn,HTTPServer):

def \_\_init\_\_(self, \*args, \*\*kw):

HTTPServer.\_\_init\_\_(self, \*args, \*\*kw)

self.ready = True

def setFile(self, playbackURL, chunksize, playbackFile, response, fileSize, url, service):

self.playbackURL = playbackURL

self.chunksize = chunksize

self.playbackFile = playbackFile

self.response = response

self.fileSize = fileSize

self.url = url

self.service = service

self.ready = True

self.state = 0

self.lock = 0

def setURL(self, playbackURL):

self.playbackURL = playbackURL

def setAccount(self, service, domain):

self.service = service

self.domain = domain

self.playbackURL = ''

self.crypto = False

self.ready = True

class myStreamer(BaseHTTPRequestHandler):

#Handler for the GET requests

def do\_POST(self):

# debug - print headers in log

headers = str(self.headers)

print(headers)

# passed a kill signal?

if self.path == '/kill':

self.server.ready = False

return

# redirect url to output

elif self.path == '/playurl':

content\_length = int(self.headers['Content-Length']) # <--- Gets the size of data

post\_data = self.rfile.read(content\_length) # <--- Gets the data itself

#print post\_data

for r in re.finditer('url\=([^\|]+)\|Cookie\=DRIVE\_STREAM\%3D([^\&]+)' ,

post\_data, re.DOTALL):

url = r.group(1)

drive\_stream = r.group(2)

print "drive\_stream = " + drive\_stream + "\n"

print "url = " + url + "\n"

self.server.playbackURL = url

self.server.drive\_stream = drive\_stream

self.server.service.authorization.setToken('DRIVE\_STREAM',drive\_stream)

self.send\_response(200)

self.end\_headers()

elif self.path == '/crypto\_playurl':

content\_length = int(self.headers['Content-Length']) # <--- Gets the size of data

post\_data = self.rfile.read(content\_length) # <--- Gets the data itself

#print post\_data

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

post\_data, re.DOTALL):

url = r.group(1)

drive\_stream = ''

print "drive\_stream = " + drive\_stream + "\n"

print "url = " + url + "\n"

self.server.crypto = True

self.server.playbackURL = url

self.server.drive\_stream = drive\_stream

self.send\_response(200)

self.end\_headers()

def do\_HEAD(self):

# debug - print headers in log

headers = str(self.headers)

print(headers)

# passed a kill signal?

if self.path == '/kill':

self.server.ready = False

return

# redirect url to output

elif self.path == '/play':

url = self.server.playbackURL

print 'HEAD ' + url + "\n"

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

req.get\_method = lambda : 'HEAD'

try:

response = urllib2.urlopen(req)

except urllib2.URLError, e:

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

print "ERROR\n" + self.server.service.getHeadersEncoded()

self.server.service.refreshToken()

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

req.get\_method = lambda : 'HEAD'

try:

response = urllib2.urlopen(req)

except:

print "STILL ERROR\n" + self.server.service.getHeadersEncoded()

return

else:

return

self.send\_response(200)

#print str(response.info()) + "\n"

self.send\_header('Content-Type',response.info().getheader('Content-Type'))

self.send\_header('Content-Length',response.info().getheader('Content-Length'))

self.send\_header('Cache-Control',response.info().getheader('Cache-Control'))

self.send\_header('Date',response.info().getheader('Date'))

self.send\_header('Content-type','video/mp4')

self.send\_header('Accept-Ranges','bytes')

#self.send\_header('ETag',response.info().getheader('ETag'))

#self.send\_header('Server',response.info().getheader('Server'))

self.end\_headers()

## may want to add more granular control over chunk fetches

#self.wfile.write(response.read())

response.close()

print "DONE"

self.server.length = response.info().getheader('Content-Length')

# redirect url to output

else:

url = str(self.server.domain) + str(self.path)

print 'GET ' + url + "\n"

#Handler for the GET requests

def do\_GET(self):

# debug - print headers in log

headers = str(self.headers)

print(headers)

start = ''

end = ''

startOffset = 0

for r in re.finditer('Range\:\s+bytes\=(\d+)\-' ,

headers, re.DOTALL):

start = int(r.group(1))

break

for r in re.finditer('Range\:\s+bytes\=\d+\-(\d+)' ,

headers, re.DOTALL):

end = int(r.group(1))

break

# passed a kill signal?

if self.path == '/kill':

self.server.ready = False

return

# redirect url to output

elif self.path == '/play':

if (self.server.crypto and start != '' and start > 16 and end == ''):

#start = start - (16 - (end % 16))

print "START = " + str(start)

startOffset = 16-(( int(self.server.length) - start) % 16)+8

# if (self.server.crypto and start == 23474184 ):

#start = start - (16 - (end % 16))

# start = 23474184 - 8

url = self.server.playbackURL

print 'GET ' + url + "\n" + self.server.service.getHeadersEncoded() + "\n"

if start == '':

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

else:

req = urllib2.Request(url, None, self.server.service.getHeadersList(additionalHeader='Range', additionalValue='bytes='+str(start- startOffset)+'-' + str(end)))

try:

response = urllib2.urlopen(req)

except urllib2.URLError, e:

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

print "ERROR\n" + self.server.service.getHeadersEncoded()

self.server.service.refreshToken()

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

try:

response = urllib2.urlopen(req)

except:

print "STILL ERROR\n" + self.server.service.getHeadersEncoded()

return

else:

return

if start == '':

self.send\_response(200)

self.send\_header('Content-Length',response.info().getheader('Content-Length'))

else:

self.send\_response(206)

self.send\_header('Content-Length', str(int(response.info().getheader('Content-Length'))-startOffset))

#self.send\_header('Content-Range','bytes ' + str(start) + '-' +str(end))

if end == '':

self.send\_header('Content-Range','bytes ' + str(start) + '-' +str(int(self.server.length)-1) + '/' +str(int(self.server.length)))

else:

self.send\_header('Content-Range','bytes ' + str(start) + '-' + str(end) + '/' +str(int(self.server.length)))

#self.send\_header('Content-Range',response.info().getheader('Content-Range'))

print 'Content-Range!!!' + str(start) + '-' + str(int(self.server.length)-1) + '/' +str(int(self.server.length)) + "\n"

print str(response.info()) + "\n"

self.send\_header('Content-Type',response.info().getheader('Content-Type'))

# self.send\_header('Content-Length',response.info().getheader('Content-Length'))

self.send\_header('Cache-Control',response.info().getheader('Cache-Control'))

self.send\_header('Date',response.info().getheader('Date'))

self.send\_header('Content-type','video/mp4')

self.send\_header('Accept-Ranges','bytes')

#self.send\_header('ETag',response.info().getheader('ETag'))

#self.send\_header('Server',response.info().getheader('Server'))

self.end\_headers()

## may want to add more granular control over chunk fetches

#self.wfile.write(response.read())

if (self.server.crypto):

self.server.service.settings.setCryptoParameters()

from resources.lib import encryption

decrypt = encryption.encryption(self.server.service.settings.cryptoSalt,self.server.service.settings.cryptoPassword)

CHUNK = 16 \* 1024

decrypt.decryptStreamChunk(response,self.wfile, startOffset=startOffset)

else:

CHUNK = 16 \* 1024

while True:

chunk = response.read(CHUNK)

if not chunk:

break

self.wfile.write(chunk)

#response\_data = response.read()

response.close()

print "DONE"

# redirect url to output

elif self.path == '/playx':

# if (self.server.crypto and start != '' and end == ''):

# #start = start - (16 - (end % 16))

# print "START = " + str(start)

# start = start - (16-(( int(self.server.length) - start) % 16) )

# if (self.server.crypto and start == 23474184 ):

#start = start - (16 - (end % 16))

# start = 23474184 - 8

url = self.server.playbackURL

print 'GET ' + url + "\n" + self.server.service.getHeadersEncoded() + "\n"

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

try:

response = urllib2.urlopen(req)

except urllib2.URLError, e:

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

print "ERROR\n" + self.server.service.getHeadersEncoded()

self.server.service.refreshToken()

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

try:

response = urllib2.urlopen(req)

except:

print "STILL ERROR\n" + self.server.service.getHeadersEncoded()

return

else:

return

self.send\_response(200)

self.send\_header('Content-Length',response.info().getheader('Content-Length'))

print str(response.info()) + "\n"

self.send\_header('Content-Type',response.info().getheader('Content-Type'))

# self.send\_header('Content-Length',response.info().getheader('Content-Length'))

self.send\_header('Cache-Control',response.info().getheader('Cache-Control'))

self.send\_header('Date',response.info().getheader('Date'))

self.send\_header('Content-type','video/mp4')

# self.send\_header('Accept-Ranges','bytes')

#self.send\_header('ETag',response.info().getheader('ETag'))

#self.send\_header('Server',response.info().getheader('Server'))

self.end\_headers()

## may want to add more granular control over chunk fetches

#self.wfile.write(response.read())

if (self.server.crypto):

self.server.service.settings.setCryptoParameters()

from resources.lib import encryption

decrypt = encryption.encryption(self.server.service.settings.cryptoSalt,self.server.service.settings.cryptoPassword)

CHUNK = 16 \* 1024

decrypt.decryptStreamChunk(response,self.wfile, 16)

else:

CHUNK = 16 \* 1024

while True:

chunk = response.read(CHUNK)

if not chunk:

break

self.wfile.write(chunk)

#response\_data = response.read()

response.close()

print "DONE"

# redirect url to output

else:

url = str(self.server.domain) + str(self.path)

print 'GET ' + url + "\n"

#TO DELETE

def do\_GET2(self):

# passed a kill signal?

if self.path == '/kill':

self.server.ready = False

return

# debug - print headers in log

headers = str(self.headers)

print(headers)

# client passed a range of bytes to fetch

start = ''

end = ''

count = 0

for r in re.finditer('Range\:\s+bytes\=(\d+)\-' ,

headers, re.DOTALL):

start = int(r.group(1))

break

for r in re.finditer('Range\:\s+bytes\=\d+\-(\d+)' ,

headers, re.DOTALL):

end = int(r.group(1))

if end == 0:

end = ''

break

# pass back the appropriate headers

if start == '':

self.send\_response(200)

self.send\_header('Content-Length',self.server.fileSize)

else:

self.send\_response(206)

if start > 0:

count = int(start/int(self.server.chunksize))

self.send\_header('Content-Length',str(self.server.fileSize-(count\*int(self.server.chunksize))))

self.send\_header('Content-Range','bytes ' + str(start) + '-' + str(self.server.fileSize-1)+'/'+str(self.server.fileSize))

req = urllib2.Request(self.server.url, None, self.server.service.getHeadersList(additionalHeader='Range', additionalValue='bytes '+ str(start) + '-' + str(end)))

try:

response = urllib2.urlopen(req)

except urllib2.URLError, e:

print "error " + str(e.code) + ' header Range' + str(start) + '-' + str(end)

self.server.service.refreshToken()

req = urllib2.Request(self.server.url, None, self.server.service.getHeadersList(additionalHeader='Range', additionalValue='bytes '+ str(start) + '-' + str(end)))

try:

response = urllib2.urlopen(req)

except urllib2.URLError, e:

print "error " + str(e.code)

return

self.send\_header('Content-type','video/mp4')

self.send\_header('Accept-Ranges','bytes')

self.end\_headers()

#while self.server.state == 2:

# self.server.state = 3

#while self.server.state == 3:

# xbmc.sleep(10)

# is streamer ready to serve packets?

if self.server.state == 0:

## fetch the entire stream?

#self.server.state = 2

#try:

if count == 0:

with open(self.server.playbackURL, "rb") as f:

while True:

chunk = f.read(self.server.chunksize)

if chunk:

self.wfile.write(chunk)

count = count + 1

else:

break

f.close()

#fi = open(self.server.playbackFile, 'ab')

#self.server.state = 1

if self.server.lock != 0:

self.server.lock = 2

xbmc.sleep(1000)

self.server.lock = 1

while self.server.lock ==1:#self.server.state == 2:

chunk = self.server.response.read(self.server.chunksize)

if not chunk: break

fi = open(self.server.playbackFile, 'wb')

fi.seek(self.server.chunksize\*count,0)

fi.write(chunk)

fi.close()

with open(self.server.playbackURL, "rb") as f:

f.seek(self.server.chunksize\*count,0)

chunk = f.read(self.server.chunksize)

self.wfile.write(chunk)

f.close()

count = count + 1

self.server.lock = 0

#fi.close()

#except: pass

#if self.server.state == 2:

# self.server.ready = False

#self.server.state = 0

self.server.ready = False

# else:

# self.server.state = 1

# self.server.ready = False

return