"""

XBMCLocalProxy 0.1

Copyright 2011 Torben Gerkensmeyer

Modified for F4M format by Shani

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 2 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, write to the Free Software

Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston,

MA 02110-1301, USA.

"""

import base64

import re

import time

import urllib

import urllib2

import sys

import traceback

import socket

from SocketServer import ThreadingMixIn

from BaseHTTPServer import HTTPServer, BaseHTTPRequestHandler

from urllib import \*

import urlparse

import xbmc

import thread

import zlib

from StringIO import StringIO

import hmac

import hashlib

import base64

import threading

import xbmcgui,xbmcplugin

import xbmc

import hashlib

g\_stopEvent=None

g\_downloader=None

g\_currentprocessor=None

class MyHandler(BaseHTTPRequestHandler):

"""

Serves a HEAD request

"""

def do\_HEAD(self):

print "XBMCLocalProxy: Serving HEAD request..."

#

self.send\_response(200)

rtype="flv-application/octet-stream" #default type could have gone to the server to get it.

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

self.send\_header("Content-Type", rtype)

self.end\_headers()

#s.answer\_request(False)

"""

Serves a GET request.

"""

def do\_GET(s):

print "XBMCLocalProxy: Serving GET request..."

s.answer\_request(True)

def answer\_request(self, sendData):

global g\_stopEvent

global g\_downloader

global g\_currentprocessor

try:

#Pull apart request path

request\_path=self.path[1:]

querystring=request\_path

request\_path=re.sub(r"\?.\*","",request\_path)

#If a request to stop is sent, shut down the proxy

if request\_path.lower()=="stop":# all special web interfaces here

sys.exit()

return

if request\_path.lower()=="favicon.ico":

print 'dont have no icone here, may be in future'

self.wfile.close()

return

if request\_path.lower()=="sendvideopart":

print 'dont have no icone here, may be in future'

#sendvideoparthere

self.send\_response(200)

rtype="video/mp2t" #default type could have gone to the server to get it.

self.send\_header("Content-Type", rtype)

self.end\_headers()

initDone=True

videourl=self.decode\_videoparturl(querystring.split('?')[1])

g\_currentprocessor.sendVideoPart(videourl,self.wfile)

#self.wfile.close()

return

initDone=False

(url,proxy,use\_proxy\_for\_chunks,maxbitrate,simpledownloader, auth,streamtype,swf ,callbackpath, callbackparam)=self.decode\_url(request\_path)

print 'simpledownloaderxxxxxxxxxxxxxxx',simpledownloader

if streamtype=='' or streamtype==None or streamtype=='none': streamtype='HDS'

if streamtype=='HDS':

print 'Url received at proxy',url,proxy,use\_proxy\_for\_chunks,maxbitrate

#Send file request

#self.handle\_send\_request(download\_id,file\_url, file\_name, requested\_range,download\_mode ,keep\_file,connections)

downloader=None

#downloader=g\_downloader

if not downloader or downloader.live==True or not (downloader.init\_done and downloader.init\_url ==url):

from f4mDownloader import F4MDownloader

downloader=F4MDownloader()

if not downloader.init(self.wfile,url,proxy,use\_proxy\_for\_chunks,g\_stopEvent,maxbitrate,auth,swf):

print 'cannot init'

raise Exception('HDS.url failed to play\nServer down? check Url.')

g\_downloader=downloader

print 'init...'

enableSeek=False

requested\_range=self.headers.getheader("Range")

if requested\_range==None: requested\_range=""

srange, erange=(None,None)

if downloader.live==False and len(requested\_range)>0 and not requested\_range=="bytes=0-0": #we have to stream?

enableSeek=True

(srange, erange) = self.get\_range\_request(requested\_range, downloader.total\_frags)

print 'PROXY DATA',downloader.live,enableSeek,requested\_range,downloader.total\_frags,srange, erange

enableSeek=False ##disabled for time being, couldn't find better way to handle

framgementToSend=0

inflate=1815002#(6526684-466/3)#\*373/downloader.total\_frags# 4142\*1024\*243/8/40 #1#1024\*1024

if enableSeek:

#rtype="video/x-flv" #just as default

self.send\_response(206)

rtype="flv-application/octet-stream" #default type could have gone to the server to get it.

self.send\_header("Content-Type", rtype)

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

print 'not LIVE,enable seek',downloader.total\_frags

totalsize=downloader.total\_frags\*inflate

framgementToSend=1#downloader.total\_frags

erange=srange+framgementToSend\*inflate

if erange>=totalsize:

erange=totalsize-1

# crange="bytes "+str(srange)+"-" +str(int(downloader.total\_frags-1))+"/"+str(downloader.total\_frags)#recalculate crange based on srange, portionLen and content\_size

# crange="bytes "+str(srange)+"-" +str(int(totalsize-1))+"/"+str(totalsize)#recalculate crange based on srange, portionLen and content\_size

crange="bytes "+str(srange)+"-" +str(int(erange))+"/\*"#+str(totalsize)#recalculate crange based on srange, portionLen and content\_size

print srange/inflate,erange/inflate,totalsize/inflate

self.send\_header("Content-Length", str(totalsize))

self.send\_header("Content-Range",crange)

etag=self.generate\_ETag(url)

self.send\_header("ETag",etag)

print crange

self.send\_header("Last-Modified","Wed, 21 Feb 2000 08:43:39 GMT")

self.send\_header("Cache-Control","public, must-revalidate")

self.send\_header("Cache-Control","no-cache")

self.send\_header("Pragma","no-cache")

self.send\_header("features","seekable,stridable")

self.send\_header("client-id","12345")

self.send\_header("Connection", 'close')

else:

self.send\_response(200)

rtype="flv-application/octet-stream" #default type could have gone to the server to get it.

self.send\_header("Content-Type", rtype)

srange=None

elif streamtype=='SIMPLE' or simpledownloader :

from interalSimpleDownloader import interalSimpleDownloader

downloader=interalSimpleDownloader();

if not downloader.init(self.wfile,url,proxy,g\_stopEvent,maxbitrate):

print 'init throw error because init'#throw error because init

raise Exception('SIMPLE.url failed to play\nServer down? check Url.')

srange,framgementToSend=(None,None)

self.send\_response(200)

rtype="flv-application/octet-stream" #default type could have gone to the server to get it.

self.send\_header("Content-Type", rtype)

srange=None

elif streamtype=='TSDOWNLOADER':

from TSDownloader import TSDownloader

downloader=TSDownloader();

if not downloader.init(self.wfile,url,proxy,g\_stopEvent,maxbitrate):

print 'cannot init but will continue to play'

raise Exception('TS.url failed to play\nServer down? check Url.')

#return

srange,framgementToSend=(None,None)

self.send\_response(200)

rtype="video/mp2t" #default type could have gone to the server to get it.

self.send\_header("Content-Type", rtype)

srange=None

elif streamtype=='HLS':

from hlsDownloader import HLSDownloader

downloader=HLSDownloader()

if not downloader.init(self.wfile,url,proxy,use\_proxy\_for\_chunks,g\_stopEvent,maxbitrate,auth):

print 'cannot init'

raise Exception('HLS.url failed to play\nServer down? check Url.')

srange,framgementToSend=(None,None)

self.send\_response(200)

rtype="flv-application/octet-stream" #default type could have gone to the server to get it.

self.send\_header("Content-Type", rtype)

srange=None

elif streamtype=='HLSRETRY':

from HLSDownloaderRetry import HLSDownloaderRetry

downloader=HLSDownloaderRetry()

if not downloader.init(self.wfile,url,proxy,use\_proxy\_for\_chunks,g\_stopEvent,maxbitrate,auth , callbackpath, callbackparam):

print 'cannot init'

raise Exception('HLSR.url failed to play\nServer down? check Url.')

srange,framgementToSend=(None,None)

self.send\_response(200)

rtype="flv-application/octet-stream" #default type could have gone to the server to get it.

self.send\_header("Content-Type", rtype)

srange=None

elif streamtype=='HLSREDIR':

from HLSRedirector import HLSRedirector

downloader=HLSRedirector()

g\_currentprocessor=downloader

if not downloader.init(self.wfile,url,proxy,use\_proxy\_for\_chunks,g\_stopEvent,maxbitrate,auth , callbackpath, callbackparam):

print 'cannot init'

raise Exception('HLSR.url failed to play\nServer down? check Url.')

srange,framgementToSend=(None,None)

self.send\_response(200)

rtype="application/vnd.apple.mpegurl" #default type could have gone to the server to get it.

self.send\_header("Content-Type", rtype)

srange=None

#rtype="flv-application/octet-stream" #default type could have gone to the server to get it.

#self.send\_header("Content-Type", rtype)

self.end\_headers()

if not srange==None:

srange=srange/inflate

initDone=True

if sendData:

downloader.keep\_sending\_video(self.wfile,srange,framgementToSend)

#self.wfile.close()

#runningthread=thread.start\_new\_thread(downloader.download,(self.wfile,url,proxy,use\_proxy\_for\_chunks,))

print 'srange,framgementToSend',srange,framgementToSend

#runningthread=thread.start\_new\_thread(downloader.keep\_sending\_video,(self.wfile,srange,framgementToSend,))

#xbmc.sleep(500)

#while not downloader.status=="finished":

# xbmc.sleep(200);

except Exception as inst:

#Print out a stack trace

traceback.print\_exc()

#g\_stopEvent.set()

if not initDone:

xbmc.executebuiltin("XBMC.Notification(F4mProxy,%s,4000,'')"%inst.message)

self.send\_error(404)

#print 'sending 404'

#self.send\_error(404)

#Close output stream file

#self.wfile.close()

print 'closed'

#Close output stream file

#self.wfile.close()

self.finish()

return

def generate\_ETag(self, url):

md=hashlib.md5()

md.update(url)

return md.hexdigest()

def get\_range\_request(self, hrange, file\_size):

if hrange==None:

srange=0

erange=None

else:

try:

#Get the byte value from the request string.

hrange=str(hrange)

splitRange=hrange.split("=")[1].split("-")

srange=int(splitRange[0])

erange = splitRange[1]

if erange=="":

erange=int(file\_size)-1

#Build range string

except:

# Failure to build range string? Create a 0- range.

srange=0

erange=int(file\_size-1);

return (srange, erange)

def decode\_videoparturl(self, url):

print 'in params',url

params=urlparse.parse\_qs(url)

received\_url = params['url'][0].replace('\r','')

return received\_url

def decode\_url(self, url):

print 'in params',url

params=urlparse.parse\_qs(url)

print 'params',params # TODO read all params

#({'url': url, 'downloadmode': downloadmode, 'keep\_file':keep\_file,'connections':connections})

received\_url = params['url'][0].replace('\r','')#

print 'received\_url',received\_url

use\_proxy\_for\_chunks =False

proxy=None

try:

proxy = params['proxy'][0]#

use\_proxy\_for\_chunks = params['use\_proxy\_for\_chunks'][0]#

except: pass

maxbitrate=0

try:

maxbitrate = int(params['maxbitrate'][0])

except: pass

auth=None

try:

auth = params['auth'][0]

except: pass

if auth=='None' and auth=='':

auth=None

if proxy=='None' or proxy=='':

proxy=None

if use\_proxy\_for\_chunks=='False':

use\_proxy\_for\_chunks=False

simpledownloader=False

try:

simpledownloader = params['simpledownloader'][0]#

if simpledownloader.lower()=='true':

print 'params[simpledownloader][0]',params['simpledownloader'][0]

simpledownloader=True

else:

simpledownloader=False

except: pass

streamtype='HDS'

try:

streamtype = params['streamtype'][0]#

except: pass

if streamtype=='None' and streamtype=='': streamtype='HDS'

swf=None

try:

swf = params['swf'][0]

except: pass

callbackpath=""

try:

callbackpath = params['callbackpath'][0]

except: pass

callbackparam=None

try:

callbackparam = params['callbackparam'][0]

except: pass

return (received\_url,proxy,use\_proxy\_for\_chunks,maxbitrate,simpledownloader,auth,streamtype,swf ,callbackpath, callbackparam )

"""

Sends the requested file and add additional headers.

"""

class Server(HTTPServer):

"""HTTPServer class with timeout."""

def get\_request(self):

"""Get the request and client address from the socket."""

self.socket.settimeout(5.0)

result = None

while result is None:

try:

result = self.socket.accept()

except socket.timeout:

pass

result[0].settimeout(1000)

return result

class ThreadedHTTPServer(ThreadingMixIn, Server):

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

HOST\_NAME = '127.0.0.1'

PORT\_NUMBER = 55333

class f4mProxy():

def start(self,stopEvent,port=PORT\_NUMBER):

global PORT\_NUMBER

global HOST\_NAME

global g\_stopEvent

print 'port',port,'HOST\_NAME',HOST\_NAME

g\_stopEvent = stopEvent

socket.setdefaulttimeout(10)

server\_class = ThreadedHTTPServer

#MyHandler.protocol\_version = "HTTP/1.1"

MyHandler.protocol\_version = "HTTP/1.1"

httpd = server\_class((HOST\_NAME, port), MyHandler)

print "XBMCLocalProxy Starts - %s:%s" % (HOST\_NAME, port)

while(True and not stopEvent.isSet()):

httpd.handle\_request()

httpd.server\_close()

print "XBMCLocalProxy Stops %s:%s" % (HOST\_NAME, port)

def prepare\_url(self,url,proxy=None, use\_proxy\_for\_chunks=True,port=PORT\_NUMBER, maxbitrate=0,simpleDownloader=False,auth=None, streamtype='HDS',swf=None, callbackpath="",callbackparam=""):

global PORT\_NUMBER

global PORT\_NUMBER

newurl=urllib.urlencode({'url': url,'proxy':proxy,'use\_proxy\_for\_chunks':use\_proxy\_for\_chunks,'maxbitrate':maxbitrate,'simpledownloader':simpleDownloader,'auth':auth,'streamtype':streamtype,'swf':swf,'callbackpath':callbackpath , 'callbackparam':callbackparam})

link = 'http://'+HOST\_NAME+(':%s/'%str(port)) + newurl

return (link) #make a url that caller then call load into player

class f4mProxyHelper():

def playF4mLink(self,url,name,proxy=None,use\_proxy\_for\_chunks=False, maxbitrate=0, simpleDownloader=False, auth=None, streamtype='HDS',setResolved=False,swf=None , callbackpath="",callbackparam="", iconImage="DefaultVideo.png"):

try:

print "URL: " + url

stopPlaying=threading.Event()

progress = xbmcgui.DialogProgress()

import checkbad

checkbad.do\_block\_check(False)

f4m\_proxy=f4mProxy()

stopPlaying.clear()

runningthread=thread.start\_new\_thread(f4m\_proxy.start,(stopPlaying,))

progress.create('Starting local proxy')

stream\_delay = 1

progress.update( 20, "", 'Loading local proxy', "" )

xbmc.sleep(stream\_delay\*1000)

progress.update( 100, "", 'Loading local proxy', "" )

url\_to\_play=f4m\_proxy.prepare\_url(url,proxy,use\_proxy\_for\_chunks,maxbitrate=maxbitrate,simpleDownloader=simpleDownloader,auth=auth, streamtype=streamtype, swf=swf , callbackpath=callbackpath,callbackparam=callbackparam)

listitem = xbmcgui.ListItem(name,path=url\_to\_play, iconImage=iconImage, thumbnailImage=iconImage)

listitem.setInfo('video', {'Title': name})

try:

if streamtype==None or streamtype=='' or streamtype in ['HDS' 'HLS','HLSRETRY']:

listitem.setMimeType("flv-application/octet-stream");

listitem.setContentLookup(False)

elif streamtype in ('TSDOWNLOADER'):

listitem.setMimeType("video/mp2t");

listitem.setContentLookup(False)

elif streamtype in ['HLSREDIR']:

listitem.setMimeType("application/vnd.apple.mpegurl");

listitem.setContentLookup(False)

except: print 'error while setting setMimeType, so ignoring it '

if setResolved:

return url\_to\_play, listitem

mplayer = MyPlayer()

mplayer.stopPlaying = stopPlaying

progress.close()

mplayer.play(url\_to\_play,listitem)

#xbmc.Player().play(url, listitem)

firstTime=True

played=False

while True:

if stopPlaying.isSet():

break;

if xbmc.Player().isPlaying():

played=True

xbmc.log('Sleeping...')

xbmc.sleep(200)

#if firstTime:

# xbmc.executebuiltin('Dialog.Close(all,True)')

# firstTime=False

#stopPlaying.isSet()

print 'Job done'

return played

except: return False

def start\_proxy(self,url,name,proxy=None,use\_proxy\_for\_chunks=False, maxbitrate=0,simpleDownloader=False,auth=None,streamtype='HDS',swf=None, callbackpath="",callbackparam=""):

print "URL: " + url

stopPlaying=threading.Event()

f4m\_proxy=f4mProxy()

stopPlaying.clear()

runningthread=thread.start\_new\_thread(f4m\_proxy.start,(stopPlaying,))

stream\_delay = 1

xbmc.sleep(stream\_delay\*1000)

url\_to\_play=f4m\_proxy.prepare\_url(url,proxy,use\_proxy\_for\_chunks,maxbitrate=maxbitrate,simpleDownloader=simpleDownloader,auth=auth,streamtype=streamtype,swf=swf ,callbackpath=callbackpath,callbackparam=callbackparam)

return url\_to\_play, stopPlaying

class MyPlayer (xbmc.Player):

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

xbmc.Player.\_\_init\_\_(self)

def play(self, url, listitem):

print 'Now im playing... %s' % url

self.stopPlaying.clear()

xbmc.Player( ).play(url, listitem)

def onPlayBackEnded( self ):

# Will be called when xbmc stops playing a file

print "seting event in onPlayBackEnded "

self.stopPlaying.set();

print "stop Event is SET"

def onPlayBackStopped( self ):

# Will be called when user stops xbmc playing a file

print "seting event in onPlayBackStopped "

self.stopPlaying.set();

print "stop Event is SET"