'''

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/>.

'''

# cloudservice - standard modules

import os

import re

# cloudservice - standard XBMC modules

import xbmcgui, xbmcvfs

#

# This class handles fetching files from local when cached, rather then making calls to the web service

#

class cache:

# CloudService v0.2.6

##

##

def \_\_init\_\_(self, package=None):

self.package = package

self.cachePath = ''

self.files = []

self.srt = []

##

# set the media package

##

def setPackage(self, package):

self.package = package

##

# set the SRT for the video file

##

def setSRT(self, service):

#load cachePath if not already loaded

if not service.settings.cacheSRT and self.cachePath == '':

self.cachePath = service.settings.cachePath

#only makes sense to cache SRT if the cachePath exists

if service.settings.cacheSRT and self.cachePath != '':

cachePath = str(self.cachePath) + '/' + str(self.package.file.id)+'/'

if not xbmcvfs.exists(cachePath):

xbmcvfs.mkdirs(cachePath)

srt = service.getSRT(self.package)

if srt:

for file in srt:

if not xbmcvfs.exists(str(cachePath) + str(file[0])):

service.downloadGeneralFile(file[1], str(cachePath) + str(file[0]))

self.srt.append(str(cachePath) + str(file[0]))

#fetch SRT URLs but we won't cache the files

else:

srt = service.getSRT(self.package)

if srt:

for file in srt:

self.srt.append(str(file[1]) + '|' + service.getHeadersEncoded())

##

# set the CC for the video file

##

def setCC(self, service):

#load cachePath if not already loaded

if self.cachePath == '':

self.cachePath = service.settings.cachePath

# there is no cache path setting or the setting is unset -- we should assume user does not want to use caching

# CC files need to be cached (so they can be converted to SRT) -- don't do anything if we don't have the cachePath

if self.cachePath == '':

return

else:

cachePath = str(self.cachePath) + '/' + str(self.package.file.id)+'/'

if not xbmcvfs.exists(cachePath):

xbmcvfs.mkdirs(cachePath)

cachePath = str(cachePath) + str(self.package.file.id)

cc = service.getTTS(self.package.file.srtURL)

if cc:

for file in cc:

if not xbmcvfs.exists(cachePath + str(file[0])):

service.downloadTTS(file[1], str(cachePath) + str(file[0]))

self.srt.append(str(cachePath) + str(file[0]))

##

# fetch the SRT

##

def getSRT(self, service):

#load cachePath if not already loaded

# if self.cachePath == '':

# self.cachePath = service.settings.cachePath

#

# if self.cachePath != '':

#

# dirs, files = xbmcvfs.listdir(str(self.cachePath) + '/'+ str(self.package.file.id) + '/')

# for file in files:

# if str(os.path.splitext(file)[1]).lower() in ('.srt', '.sub', '.ass', '.ssa') or str(os.path.splitext(file)[1]).lower() in ('srt', 'sub', 'ass', 'ssa'):

# self.srt.append(str(self.cachePath) + '/'+ str(self.package.file.id) + '/' + file)

return self.srt

##

# set the thumbnail

##

def setThumbnail(self, service, url=''):

#load cachePath if not already loaded

if self.cachePath == '':

self.cachePath = service.settings.cachePath

# there is no cache path setting or the setting is unset -- we should assume user does not want to use caching

if not service.settings.cacheThumbnails or self.cachePath == '':

if url == '':

return self.package.file.thumbnail

else:

return url

if url == '':

url = self.package.file.thumbnail

#simply no thumbnail

if url == '':

return ""

#user doesn't want to cache thumbnails

if not service.settings.cacheThumbnails:

return url

cachePath = str(self.cachePath) + str(self.package.file.id) + '/'

cacheFile = str(self.cachePath) + str(self.package.file.id) + '.jpg'

if not xbmcvfs.exists(cachePath):

xbmcvfs.mkdirs(cachePath)

if not xbmcvfs.exists(cacheFile):

cacheFile = service.downloadGeneralFile(url, cacheFile)

if cacheFile is None:

return url

return cacheFile

##

# get the thumbnail

##

def getThumbnail(self,service, url='', fileID=''):

# user isn't caching thumbnails

if not service.settings.cacheThumbnails or self.cachePath == '':

if url != '':

return url + '|' + service.getHeadersEncoded()

elif self.package != None and self.package.file != None:

return self.package.file.thumbnail + '|' + service.getHeadersEncoded()

else:

return ''

if fileID == '':

if xbmcvfs.exists(str(self.cachePath) + str(self.package.file.id) + '/' + str(self.package.file.id) + '.jpg'):

return str(self.cachePath) + str(self.package.file.id) + '/' + str(self.package.file.id) + '.jpg'

else:

return self.package.file.thumbnail + '|' + service.getHeadersEncoded()

else:

if xbmcvfs.exists(str(self.cachePath) + str(fileID) + '/' + str(fileID) + '.jpg'):

return str(self.cachePath) + str(fileID) + '/' + str(fileID) + '.jpg'

else:

return url + '|' + service.getHeadersEncoded()

##

# get a list of offline files for this file

##

def getFiles(self,service):

#load cachePath if not already loaded

if self.cachePath == '':

self.cachePath = service.settings.cachePath

localResolutions = []

localFiles = []

# no local cache, no local files to look for

if self.cachePath == '':

return (localResolutions,localFiles)

cachePath = str(self.cachePath) + '/' + str(self.package.file.id) + '/'

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

if xbmcvfs.exists(cachePath) or os.path.exists(cachePath):

dirs,files = xbmcvfs.listdir(cachePath)

for file in files:

if '.stream.mp4' in file:

try:

resolutionFile = xbmcvfs.File(cachePath + str(os.path.splitext(file)[0]) + '.resolution')

resolution = resolutionFile.read()

resolutionFile.close()

except:

resolution = file

localResolutions.append('offline - ' + str(resolution))

localFiles.append(str(cachePath) + str(file))

return (localResolutions,localFiles)