import os

import sys

import shutil

import logging

import tempfile

from optparse import OptionParser

from flvlib import \_\_versionstr\_\_

from flvlib.constants import TAG\_TYPE\_AUDIO, TAG\_TYPE\_VIDEO, TAG\_TYPE\_SCRIPT

from flvlib.constants import FRAME\_TYPE\_KEYFRAME

from flvlib.astypes import MalformedFLV, FLVObject

from flvlib.tags import FLV, EndOfFile, AudioTag, VideoTag, ScriptTag

from flvlib.tags import create\_script\_tag, create\_flv\_header

from flvlib.helpers import force\_remove

log = logging.getLogger('flvlib.index-flv')

class IndexingAudioTag(AudioTag):

SEEKPOINT\_DENSITY = 10

def \_\_init\_\_(self, parent\_flv, f):

AudioTag.\_\_init\_\_(self, parent\_flv, f)

def parse(self):

parent = self.parent\_flv

AudioTag.parse(self)

if not parent.first\_media\_tag\_offset:

parent.first\_media\_tag\_offset = self.offset

# If the FLV has video, we're done. No need to store audio seekpoint

# information anymore.

if not parent.no\_video:

return

# We haven't seen any video tag yet. Store every SEEKPOINT\_DENSITY tag

# offset and timestamp.

parent.audio\_tag\_number += 1

if (parent.audio\_tag\_number % self.SEEKPOINT\_DENSITY == 0):

parent.audio\_seekpoints.filepositions.append(self.offset)

parent.audio\_seekpoints.times.append(self.timestamp / 1000.0)

class IndexingVideoTag(VideoTag):

def parse(self):

parent = self.parent\_flv

VideoTag.parse(self)

parent.no\_video = False

if not parent.first\_media\_tag\_offset:

parent.first\_media\_tag\_offset = self.offset

if self.frame\_type == FRAME\_TYPE\_KEYFRAME:

parent.keyframes.filepositions.append(self.offset)

parent.keyframes.times.append(self.timestamp / 1000.0)

class IndexingScriptTag(ScriptTag):

def parse(self):

parent = self.parent\_flv

ScriptTag.parse(self)

if self.name == 'onMetaData':

parent.metadata = self.variable

parent.metadata\_tag\_start = self.offset

parent.metadata\_tag\_end = self.f.tell()

tag\_to\_class = {

TAG\_TYPE\_AUDIO: IndexingAudioTag,

TAG\_TYPE\_VIDEO: IndexingVideoTag,

TAG\_TYPE\_SCRIPT: IndexingScriptTag

}

class IndexingFLV(FLV):

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

FLV.\_\_init\_\_(self, f)

self.metadata = None

self.keyframes = FLVObject()

self.keyframes.filepositions = []

self.keyframes.times = []

self.no\_video = True

# If the FLV file has no video, there are no keyframes. We want to put

# some info in the metadata anyway -- Flash players use keyframe

# information as a seek table. In audio-only FLV files you can usually

# seek to the beginning of any tag (this is not entirely true for AAC).

# Most players still work if you just provide "keyframe" info that's

# really a table of every Nth audio tag, even with AAC.

# Because of that, until we see a video tag we make every Nth

# IndexingAudioTag store its offset and timestamp.

self.audio\_tag\_number = 0

self.audio\_seekpoints = FLVObject()

self.audio\_seekpoints.filepositions = []

self.audio\_seekpoints.times = []

self.metadata\_tag\_start = None

self.metadata\_tag\_end = None

self.first\_media\_tag\_offset = None

def tag\_type\_to\_class(self, tag\_type):

try:

return tag\_to\_class[tag\_type]

except KeyError:

raise MalformedFLV("Invalid tag type: %d", tag\_type)

def filepositions\_difference(metadata, original\_metadata\_size):

test\_payload = create\_script\_tag('onMetaData', metadata)

payload\_size = len(test\_payload)

difference = payload\_size - original\_metadata\_size

return test\_payload, difference

def retimestamp\_and\_index\_file(inpath, outpath=None, retimestamp=None):

# no retimestamping needed

if retimestamp is None:

return index\_file(inpath, outpath)

# retimestamp the input in place and index

elif retimestamp == 'inplace':

from flvlib.scripts.retimestamp\_flv import retimestamp\_file\_inplace

log.debug("Retimestamping file `%s' in place", inpath)

# retimestamp the file inplace

if not retimestamp\_file\_inplace(inpath):

log.error("Failed to retimestamp `%s' in place", inpath)

return False

return index\_file(inpath, outpath)

# retimestamp the input into a temporary file

elif retimestamp == 'atomic':

from flvlib.scripts.retimestamp\_flv import retimestamp\_file\_atomically

log.debug("Retimestamping file `%s' atomically", inpath)

try:

fd, temppath = tempfile.mkstemp()

os.close(fd)

# preserve the permission bits

shutil.copymode(inpath, temppath)

except EnvironmentError, (errno, strerror):

log.error("Failed to create temporary file: %s", strerror)

return False

if not retimestamp\_file\_atomically(inpath, temppath):

log.error("Failed to retimestamp `%s' atomically", inpath)

# remove the temporary files

force\_remove(temppath)

return False

# index the temporary file

if not index\_file(temppath, outpath):

force\_remove(temppath)

return False

if not outpath:

# If we were not writing directly to the output file

# we need to overwrite the original

try:

shutil.move(temppath, inpath)

except EnvironmentError, (errno, strerror):

log.error("Failed to overwrite the original file with the "

"retimestamped and indexed version: %s", strerror)

return False

else:

# if we were writing directly to the output file we need to remove

# the retimestamped temporary file

force\_remove(temppath)

return True

def index\_file(inpath, outpath=None):

out\_text = (outpath and ("into file `%s'" % outpath)) or "and overwriting"

log.debug("Indexing file `%s' %s", inpath, out\_text)

try:

f = open(inpath, 'rb')

except IOError, (errno, strerror):

log.error("Failed to open `%s': %s", inpath, strerror)

return False

flv = IndexingFLV(f)

tag\_iterator = flv.iter\_tags()

last\_tag = None

try:

while True:

tag = tag\_iterator.next()

# some buggy software, like gstreamer's flvmux, puts a metadata tag

# at the end of the file with timestamp 0, and we don't want to

# base our duration computation on that

if tag.timestamp != 0:

last\_tag = tag

except MalformedFLV, e:

message = e[0] % e[1:]

log.error("The file `%s' is not a valid FLV file: %s", inpath, message)

return False

except EndOfFile:

log.error("Unexpected end of file on file `%s'", inpath)

return False

except StopIteration:

pass

if not flv.first\_media\_tag\_offset:

log.error("The file `%s' does not have any media content", inpath)

return False

if not last\_tag:

log.error("The file `%s' does not have any content with a "

"non-zero timestamp", inpath)

return False

metadata = flv.metadata or {}

if flv.metadata\_tag\_start:

original\_metadata\_size = flv.metadata\_tag\_end - flv.metadata\_tag\_start

else:

log.debug("The file `%s' has no metadata", inpath)

original\_metadata\_size = 0

keyframes = flv.keyframes

if flv.no\_video:

log.info("The file `%s' has no video, using audio seekpoints info",

inpath)

keyframes = flv.audio\_seekpoints

duration = metadata.get('duration')

if not duration:

# A duration of 0 is nonsensical, yet some tools put it like that. In

# that case (or when there is no such field) update the duration value.

duration = last\_tag.timestamp / 1000.0

metadata['duration'] = duration

metadata['keyframes'] = keyframes

metadata['metadatacreator'] = 'flvlib %s' % \_\_versionstr\_\_

# we're going to write new metadata, so we need to shift the

# filepositions by the amount of bytes that we're going to add to

# the metadata tag

test\_payload, difference = filepositions\_difference(metadata,

original\_metadata\_size)

if difference:

new\_filepositions = [pos + difference

for pos in keyframes.filepositions]

metadata['keyframes'].filepositions = new\_filepositions

payload = create\_script\_tag('onMetaData', metadata)

else:

log.debug("The file `%s' metadata size did not change.", inpath)

payload = test\_payload

if outpath:

try:

fo = open(outpath, 'wb')

except IOError, (errno, strerror):

log.error("Failed to open `%s': %s", outpath, strerror)

return False

else:

try:

fd, temppath = tempfile.mkstemp()

# preserve the permission bits

shutil.copymode(inpath, temppath)

fo = os.fdopen(fd, 'wb')

except EnvironmentError, (errno, strerror):

log.error("Failed to create temporary file: %s", strerror)

return False

log.debug("Creating the output file")

try:

fo.write(create\_flv\_header(has\_audio=flv.has\_audio,

has\_video=flv.has\_video))

fo.write(payload)

f.seek(flv.first\_media\_tag\_offset)

shutil.copyfileobj(f, fo)

except IOError, (errno, strerror):

log.error("Failed to create the indexed file: %s", strerror)

if not outpath:

# remove the temporary file

force\_remove(temppath)

return False

f.close()

fo.close()

if not outpath:

# If we were not writing directly to the output file

# we need to overwrite the original

try:

shutil.move(temppath, inpath)

except EnvironmentError, (errno, strerror):

log.error("Failed to overwrite the original file "

"with the indexed version: %s", strerror)

return False

return True

def process\_options():

usage = "%prog [-U] file [outfile|file2 file3 ...]"

description = ("Finds keyframe timestamps and file offsets "

"in FLV files and updates the onMetaData "

"script tag with that information. "

"With the -U (update) option operates on all parameters, "

"overwriting the original file. Without the -U "

"option accepts one input and one output file path.")

version = "%%prog flvlib %s" % \_\_versionstr\_\_

parser = OptionParser(usage=usage, description=description,

version=version)

parser.add\_option("-U", "--update", action="store\_true",

help=("update mode, overwrites the given files "

"instead of writing to outfile"))

parser.add\_option("-r", "--retimestamp", action="store\_true",

help=("rewrite timestamps in the files before indexing, "

"identical to running retimestamp-flv first"))

parser.add\_option("-R", "--retimestamp-inplace", action="store\_true",

help=("same as -r but avoid creating temporary files at "

"the risk of corrupting the input files in case "

"of errors"))

parser.add\_option("-v", "--verbose", action="count",

default=0, dest="verbosity",

help="be more verbose, each -v increases verbosity")

options, args = parser.parse\_args(sys.argv)

if len(args) < 2:

parser.error("You have to provide at least one file path")

if not options.update and len(args) != 3:

parser.error("You need to provide one infile and one outfile "

"when not using the update mode")

if options.retimestamp and options.retimestamp\_inplace:

parser.error("You cannot provide both -r and -R")

if options.verbosity > 3:

options.verbosity = 3

log.setLevel({0: logging.ERROR, 1: logging.WARNING,

2: logging.INFO, 3: logging.DEBUG}[options.verbosity])

return options, args

def index\_files():

options, args = process\_options()

clean\_run = True

retimestamp\_mode = None

if options.retimestamp:

retimestamp\_mode = 'atomic'

elif options.retimestamp\_inplace:

retimestamp\_mode = 'inplace'

if not options.update:

clean\_run = retimestamp\_and\_index\_file(args[1], args[2],

retimestamp=retimestamp\_mode)

else:

for filename in args[1:]:

if not retimestamp\_and\_index\_file(filename,

retimestamp=retimestamp\_mode):

clean\_run = False

return clean\_run

def main():

try:

outcome = index\_files()

except KeyboardInterrupt:

# give the right exit status, 128 + signal number

# signal.SIGINT = 2

sys.exit(128 + 2)

except EnvironmentError, (errno, strerror):

try:

print >>sys.stderr, strerror

except StandardError:

pass

sys.exit(2)

if outcome:

sys.exit(0)

else:

sys.exit(1)