#!/usr/bin/python

# this is gn\_iso19139\_to\_mef2.0\_example.py

"""

Script to create GeoNetwork's Metadata Exchange Format 2.0 (MEF2.0) archive from ISO 19139 metadata XML files

Usage: make sure to edit mef\_siteId, info\_xml, etc.

Change history:

2009-11-06 Wolfgang Grunberg, Arizona Geological Survey. Original Python 2.6 template

2012-05-23 TAC Changed paths to metadata locations, MEF final locations, and

temporary work locations. Changed siteID. Changed metadata dateTime

to just date. Major change to loop on MEF, include graphics files.

2012-06-06 TAC changes to generate MEF2 metadata

2013-08-13 TAC various changes to application

2014-07-19 TAC changed disk reference to "E:"

2014-10-24 TAC modified app for Levings lit refs

\*\*\* NOTE: MEF temporary folder is emptied at application start! \*\*\*

Upon 'build' completion, MEF files are ZIP archives with the following structure:

+-<uuid>.mef zip archive with the metadata's UUID (must be valid) as the file name

| -info.xml GeoNetwork (GN) specific metadata such as privileges,

related data & thumbnail image files, etc.

| +metadata directory which contains ISO 19139 metadata record

| +public directory with public thumbnail and data files - can be empty

| +private directory with private data files such as shape files etc.

- can be empty

Addresses to the source folders for all metadata, data and graphics to be processed.

"""

# Library Imports - not all may be needed

import os

import sys

import cProfile

import re

import shutil

import time

import mimetypes

import uuid

import logging

from xml.dom import minidom

from xml.dom import Node

from time import strftime

# module globals and constants

# globals

\_\_author\_\_ = "Wolfgang Grunberg"

\_\_copyright\_\_ = "Copyright 2009, Arizona Geological Survey"

\_\_credits\_\_ = ["Wolfgang Grunberg", "the Internets"]

\_\_license\_\_ = "GPL"

\_\_version\_\_ = "2.0.0"

\_\_email\_\_ = "wgrunberg@azgs.az.gov"

\_\_status\_\_ = "Development" # "Prototype", "Development", or "Production"

"""

\_\_author2\_\_ = "Terry Curran, P.Eng. (2013-08-16)"

\_\_email2\_\_ = "terry.curran@shaw.ca"

"""

#=========================================

# allow any image-type file for thumbnails

#=========================================

def is\_image\_file(filename, extensions=['.jpg', '.jpeg', '.gif', '.png']):

return any(filename.endswith(e) for e in extensions)

#==========================

# start of main application

#==========================

def createMef():

print "\*\*\*\*\*\n START Create MEF "+strftime("%Y-%m-%d %H:%M:%S")+" \n\*\*\*\*\*"

#==========================

# environment path settings

#==========================

# set PATH TO ROOT FOR ALL FILE LOCATIONS

main\_path = "E:\metadataData\\coastline\_bc\\"

# set incremental METADATA FOLDER PATH

mefName="metaGraphicsData"

path2meta = main\_path + mefName + "\\"

# set incremental THUMBNAIL FOLDER PATH

path2graphicsL = main\_path + "thumbnails\\"

path2graphicsS = main\_path + "thumbnails\\"

# set incremental DATA FOLDER PATH

path2data = main\_path + "data\\"

# set incremental TEMPORARY WORKSPACE FOLDER to place MEF content before zipping up

folderToZip = 'tmp'

path2tmp = main\_path + folderToZip + "\\"

# set incremental RESULT FOLDER PATH

path2result = main\_path + "mef\\"

# set LOG PATH to record packaging information

mefLogFile = path2result + strftime("%Y-%m-%dT%H-%M-%S") + "\_mef\_log.txt"

logging.basicConfig(filename=mefLogFile,level=logging.DEBUG,)

output = open(mefLogFile, 'w' )

output.write ("\*\*\*\*\* START Create MEF "+strftime("%Y-%m-%d %H:%M:%S")+" \*\*\*\*\*\n")

# ensure temporary folder is empty

for file\_object in os.listdir(path2tmp):

file\_object\_path = os.path.join(path2tmp, file\_object)

if os.path.isfile(file\_object\_path):

os.unlink(file\_object\_path)

else:

shutil.rmtree(file\_object\_path)

# must be \*VALID\* metadata site and UUID

mef\_siteId = "b2a4aa65-4bad-4564-bffe-4f7e43db82fb"

siteName = "Marine data for BC"

#======================================================

# get list of filenames to process (to be folder names)

#======================================================

try:

metaDir = os.listdir(path2meta)

# print metaDir #debug

except os.error:

print " EXCEPTION: directory containing metadata files does not exist ("+path2meta+") "

return

#reset uuid to dummy

recordId = "00000000-0000-0000-0000-000000000000"

#========================================================================

# loop through source metadata, moving each file to temp location,

# then modifying and saving

#========================================================================

for metaFile in metaDir:

recordName = os.path.splitext(metaFile)[0]

print '\n\*\*\*MEF Folder: ', recordName

output.write ("\n")

output.write ("mef folder: " + recordName )

#=====================================================================

# select the metadata record, retrieve key metadata contents, form mef

#

#=====================================================================

f = open(path2meta + metaFile, 'r')

# Load ISO metadata string into XML object

xmldoc = minidom.parseString(f.read() )

# print xmldoc.toxml() # debug

# get metadata title (first 60 chars)

title = xmldoc.getElementsByTagName('gmd:citation')[0].getElementsByTagName('gmd:CI\_Citation')[0].getElementsByTagName('gmd:title')[0].getElementsByTagName('gco:CharacterString')[0].firstChild.data

print "title (60 chars): ", title[0:60]

output.write ("\n" + title[0:55] + "\n")

# get metadata creation date

try:

create\_date = xmldoc.getElementsByTagName('gmd:dateStamp')[0].getElementsByTagName('gco:Date')[0].firstChild.data

except:

create\_date = xmldoc.getElementsByTagName('gmd:dateStamp')[0].getElementsByTagName('gco:DateTime')[0].firstChild.data

# print "createDate: ", create\_date # debug

# get metadata uuid => recordId

recordId = xmldoc.getElementsByTagName('gmd:fileIdentifier')[0].getElementsByTagName('gco:CharacterString')[0].firstChild.data

print "uuid: ", recordId # debug

# get thumbnail names in the metadata record

# create temporary folder

mef\_dir = recordId + "\\" # Temporary folder that is created and deleted

### tmp paths for mef folders and files

mefMetaPath = path2tmp + mef\_dir + "metadata\\"

mefPrivatePath = path2tmp + mef\_dir + "private\\"

mefPublicPath = path2tmp + mef\_dir + "public\\"

mefApplischemaPath = path2tmp + mef\_dir + "applischema\\"

mefMetadata = mefMetaPath + "metadata.xml"

mefInfoFile = path2tmp +mef\_dir + "info.xml"

## create mef directory structure

try:

os.mkdir(path2tmp + mef\_dir)

except OSError, err:

print >>sys.stderr, " EXCEPTION: ", err

return

## create empty 'Application Schema' directory (not used presently)

if os.path.exists(mefApplischemaPath) == False:

try:

os.mkdir(mefApplischemaPath)

except OSError, err:

print >>sys.stderr, " EXCEPTION: ", err

return

## create empty 'public' (graphics) directory

if os.path.exists(mefPublicPath) == False:

try:

os.mkdir(mefPublicPath)

except OSError, err:

print >>sys.stderr, " EXCEPTION: ", err

return

# ... then add graphics files to mef public folder

Thumbnails = "False"

for root, dirs, files in os.walk(path2graphicsL):

for filename in filter(is\_image\_file, files):

if (filename.split('.')[0] == recordName):

graphics\_l = recordName + '.png'

shutil.copy(path2graphicsL + filename, mefPublicPath + filename)

Thumbnails = "True"

break

for root, dirs, files in os.walk(path2graphicsS):

for filename in filter(is\_image\_file, files):

if (filename.split('.')[0] == recordName + '\_s'):

graphics\_s = recordName + '\_s.png'

shutil.copy(path2graphicsS + filename, mefPublicPath + filename)

Thumbnails = "True"

break

if (Thumbnails == "False"):

print(" # # # NO GRAPHICS FILES LIKE: " + recordName + " # # # #")

output.write (" \* \* \* no thumbnail files \* \* \*\n" )

# ... then set graphics names in metadata record

print "graphics\_l\_s: ", graphics\_l," ", graphics\_s # debug

## create empty 'private' (data) directory

try:

os.mkdir(mefPrivatePath)

except OSError, err:

print >>sys.stderr, " EXCEPTION: ", err

return

# ... then copy download data file(s) to 'private' folder, if data files exist

onLineRecs = 0

try:

onLine = xmldoc.getElementsByTagName('gmd:MD\_DigitalTransferOptions')[0].getElementsByTagName('gmd:onLine')

onLineRecs = onLine.length

except:

print

print "onLine sections: ", onLineRecs

output.write (" onLine sections: " + str(onLineRecs) )

recordData = ""

for i in range(onLineRecs):

Proto = xmldoc.\

getElementsByTagName('gmd:MD\_DigitalTransferOptions')[0].\

getElementsByTagName('gmd:onLine')[i].\

getElementsByTagName('gmd:CI\_OnlineResource')[0].\

getElementsByTagName('gmd:protocol')[0].\

getElementsByTagName('gco:CharacterString')[0].firstChild.data

if (Proto == 'WWW:DOWNLOAD-1.0-http--download'):

recordData = xmldoc.\

getElementsByTagName('gmd:MD\_DigitalTransferOptions')[0].\

getElementsByTagName('gmd:onLine')[i].\

getElementsByTagName('gmd:CI\_OnlineResource')[0].\

getElementsByTagName('gmd:name')[0].\

getElementsByTagName('gco:CharacterString')[0].firstChild.data

output.write ("\n " + str(i) + ":" + Proto + " " + recordData )

# copy data record to mef private folder

print i, Proto, " ",recordData

if os.path.exists(path2data + recordData):

shutil.copy2(path2data + recordData, mefPrivatePath + recordData)

# elif os.path.exists(path2data2 + recordData):

# shutil.copy2(path2data2 + recordData, mefPrivatePath + recordData)

else:

print i, Proto

if (recordData == ""):

print recordData, " : ??? no download file in metadata record ???"

## create empty 'metadata' directory

if os.path.exists(mefMetaPath) == False:

try:

os.mkdir(mefMetaPath)

except OSError, err:

print >>sys.stderr, " EXCEPTION: ", err

return

# ... then copy metadata record to mef metadata folder

if os.path.exists(path2meta + metaFile) == True:

shutil.copy2( path2meta + metaFile, mefMetaPath+"metadata.xml")

# cleanup DOM

xmldoc.unlink()

f.close()

#====================

# construct info file

#====================

info\_xml = "<?xml version=\"1.0\" encoding=\"UTF-8\"?>\n"

info\_xml += "<info version=\"2.0\">\n<general>\n"

info\_xml +="<uuid>"+recordId+"</uuid>\n"

# universally unique identifier assigned to the metadata and must be a valid UUID. This element is optional and, when omitted, the reader should generate one

# when the metadata was created

info\_xml +="<createDate>"+create\_date+"</createDate>\n"

# most recent change to the metadata

info\_xml +="<changeDate>"+strftime("%Y-%m-%d" +"T" + "%H:%M:%S")+"</changeDate>\n"

info\_xml +="<siteId>" +mef\_siteId +"</siteId>\n" # This is an UUID that identifies the actor that created the metadata and must be a valid UUID. When the UUID element is missing, this element should be missing too. If present, it will be ignored.

info\_xml +="<siteName>" +siteName +"</siteName>\n" # Site Name

# metadata type: dublin-core, fgdc-std, iso19115, iso19139

info\_xml +="<schema>iso19139</schema>\n"

# MEF format: simple, partial, full

if os.path.exists(mefPrivatePath+recordData):

info\_xml +="<format>full</format>\n"

elif os.path.exists(mefPublicPath+graphics\_s):

info\_xml +="<format>partial</format>\n"

else:

info\_xml +="<format>simple</format>\n"

info\_xml +="<isTemplate>false</isTemplate>\n" # A boolean field that indicates if this metadata is a template used to create new ones. There is no real distinction between a real metadata and a template but some actors use it to allow fast metadata creation.

info\_xml +="<rating>0</rating>\n" # If present, indicates the users' rating of the metadata ranging from 1 (a bad rating) to 5 (an excellent rating). The special value 0 means that the metadata has not been rated yet. Can be used to sort search results.

info\_xml +="<popularity>0</popularity>\n" # If present, indicates the popularity of the metadata. The value must be positive and high values mean high popularity. The criteria used to set the popularity is left to the writer. Its main purpose is to provide a metadata ordering during a search.

info\_xml +="</general>\n<categories>\n\

<category name=\"datasets\" />\n"

info\_xml +="</categories>\n"

info\_xml +="<privileges>\n" # and privileges

info\_xml +="<group name=\"all\">\n\

<operation name=\"view\" />\n\

<operation name=\"download\"/>\n\

<operation name=\"dynamic\"/>\n\

<operation name=\"featured\"/>\n</group>\n</privileges>"

# GN public files - thumbnails

info\_xml +="\n<public>\n"

if os.path.exists(mefPublicPath+graphics\_s):

info\_xml +="\

<file name=\"" +graphics\_s +"\" changeDate=\"" +strftime('%Y-%m-%d' +'T' + '%H:%M:%S',time.localtime(os.path.getmtime(mefPublicPath+graphics\_s))) +"\" />\n"

if os.path.exists(mefPublicPath+graphics\_l):

info\_xml +="\

<file name=\"" +graphics\_l +"\" changeDate=\"" +strftime('%Y-%m-%d' +'T' + '%H:%M:%S',time.localtime(os.path.getmtime(mefPublicPath+graphics\_l))) +"\" />\n"

info\_xml +="</public>\n"

# GN private file - data; require authentication during download request

if os.path.exists(mefPrivatePath+recordData):

info\_xml +="<private>\n"

for recordData in os.listdir(mefPrivatePath):

info\_xml +=" <file name=\"" + recordData +"\" changeDate=\"" +strftime('%Y-%m-%d' +'T' + '%H:%M:%S',time.localtime(os.path.getmtime(mefPrivatePath+recordData))) +"\" />\n"

info\_xml +="</private>"

info\_xml +="\n</info>"

# print info\_xml # debug

#=====================================================

# save 'info\_xml' in path2tmp\mef folder as 'info.xml'

#======================================================

try:

f = open(mefInfoFile,'w')

f.write(info\_xml)

f.close()

True

except:

print " EXCEPTION: failed to write " + mefInfoFile

#===============================================

# save whole \_tmp folder in mef2 (zipped) format

#===============================================

# go to folder above temporary directory

os.chdir( main\_path )

print "\ncompressing to mef2:", os.getcwd() +"\\" +folderToZip

mefFile = mefName

shutil.make\_archive(path2result + mefFile, "zip", folderToZip)

print "\nsaved as:", path2result +mefFile +".zip"

#===================================

# end of creation of all mef2 folder

#===================================

print "\n\*\*\*\*\*\n END Create MEF "+strftime("%Y-%m-%d %H:%M:%S")+" \n\*\*\*\*\*"

output.write ("\n\*\*\*\*\* END Create MEF "+strftime("%Y-%m-%d %H:%M:%S")+"\*\*\*\*\*")

#=======================

# trigger the processing

#=======================

createMef()

# execution performance information

# cProfile.run('createMef()')