"""

"""

import getpass

import logging

import os

from shutil import copyfile

from sqlalchemy import or\_

from etl.core.util import uri\_get

from etl.repo.fnd\_cfdw.etl\_config import EtlConfigRepo

from etl.repo.pim\_pm.pl\_bbg\_batch import PlBbgBatchRepo

from sqlalchemy.exc import DBAPIError

import os

from shutil import copyfile

USAGE = ['Fetcher Agent',

# ('--source\_code',

# {

# 'help': 'Source code',

# 'default': 'PL\_BT\_POLL\_AGENT'

# }

# ),

]

class FetcherAgent(object):

"""

"""

def \_\_init\_\_(self, logger=None, options=None):

logging.info('Fetching the values of config code from EtlConfigRepo')

logging.info('FetcherAgent')

self.USERNAME = getpass.getuser()

md = EtlConfigRepo.instance.get\_by\_config\_code('PL\_BT\_ENDPOINT')

self.base\_url = md.config\_value

@staticmethod

def get\_requests(repo):

"""

:param repo:

:return:

"""

model = repo.model

try:

logging.info('Getting all the requests from pl\_bbg\_batch table which are in SENT\_TO\_BT status')

data = repo.query.filter(

or\_(model.batch\_status\_code == 'SENT\_TO\_BT',

model.batch\_status\_code == 'PENDING')).all()

return data

except DBAPIError as err:

logging.error(err)

def get\_request\_status\_by\_url(self, obj):

"""

:param obj:

:return:

"""

logging.info('Sending the request to BT')

logging.info('GET: %s, \r\n\t%s', self.base\_url, str(obj.bt\_request\_id))

try:

response = uri\_get(self.base\_url + 'check\_status' + '/' + str(obj.bt\_request\_id))

except Exception as e:

logging.error(e)

logging.info('response: %s \r\nresponse:\t%s', self.base\_url, response)

return response

@staticmethod

def update\_request(batch\_id, data\_file\_path, bt\_request\_id, bt\_status\_code, repo):

"""

:param batch\_id:

:param data\_file\_path:

:param bt\_request\_id:

:param bt\_status\_code:

:param repo:

:return:

"""

logging.info('Updating the staging the table with the status of the request')

model = repo.model

try:

update\_row = repo.query.filter(model.bt\_request\_id == bt\_request\_id,

model.batch\_id == batch\_id).all()

update\_row[0].bt\_status\_code = bt\_status\_code

if update\_row[0].bt\_status\_code == 'BBGERROR':

update\_row[0].batch\_status\_code = 'BBG\_ERROR'

elif update\_row[0].bt\_status\_code == 'BTERROR':

update\_row[0].batch\_status\_code = 'BT\_ERROR'

update\_row[0].bt\_response\_file\_path = data\_file\_path

repo.save(update\_row)

except DBAPIError as err:

logging.error(err)

def run(self):

"""

:return:

"""

result = self.get\_requests(PlBbgBatchRepo())

for i in result:

response = self.get\_request\_status\_by\_url(i)

self.update\_request(i.batch\_id, response['data\_file\_path'], response['request\_id'],

response['request\_status'], PlBbgBatchRepo())

class LoaderAgent(object):

"""

"""

def \_\_init\_\_(self, logger=None, options=None):

logging.info('Fetching the values of config code from EtlConfigRepo')

try:

logging.info('LoaderAgent')

self.USERNAME = getpass.getuser()

md = EtlConfigRepo.instance.get\_by\_config\_code('PL\_BT\_DATA\_FILE\_PATH')

self.destination = md.config\_value

except:

pass

@staticmethod

def copy\_file(src, dst, program\_code):

if os.path.isdir(dst):

destination\_file = os.path.basename(src)

if program\_code == 'GETDATA':

ext = '.inc'

else:

ext = '.full'

name\_ext = os.path.splitext(destination\_file)

destination\_file = name\_ext[0]+ext+name\_ext[1]

dst = os.path.join(dst, destination\_file)

copyfile(src, dst)

logging.info("Copying {0} to {1}".format(src, dst))

return True

else:

logging.info("Can't copy {0} to destination: {1}".format(src, dst))

return False

@staticmethod

def get\_requests(repo):

"""

:param repo:

:return:

"""

model = repo.model

try:

logging.info('Getting all the requests from pl\_bbg\_batch table which are in SENT\_TO\_BT status')

data = repo.query.filter(model.batch\_status\_code == 'SENT\_TO\_BT').all()

return data

except DBAPIError as err:

logging.error(err)

@staticmethod

def update\_request(batch\_id, batch\_status\_code, repo):

"""

:param batch\_id:

:param batch\_status\_code:

:param repo:

:return:

"""

logging.info('Updating the staging the table with the status of the request')

model = repo.model

try:

update\_row = repo.query.filter(model.batch\_id == batch\_id).all()

update\_row[0].batch\_status\_code = batch\_status\_code

repo.save(update\_row)

except DBAPIError as err:

logging.error(err)

def run(self):

"""

:return:

"""

result = self.get\_requests(PlBbgBatchRepo())

for i in result:

if i.bt\_status\_code == 'SUCCESS':

if self.copy\_file(i.bt\_response\_file\_path.strip(), self.destination, i.bbg\_program\_code):

self.update\_request(i.batch\_id, 'BT\_DONE', PlBbgBatchRepo())

def main():

"""

Delegates all processing to Agent instance.

"""

logger = logging.getLogger("{}".format(

os.path.splitext(os.path.basename(\_\_file\_\_))[0]))

cmd\_line = sanitize\_cmd\_line(copy.copy(sys.argv))

logging.info(cmd\_line)

args = util.parse\_args(\*USAGE)

logging.info("Fetcher Agent started")

try:

with FetcherAgent(logger=logger, options=args) as agent:

agent.run()

except Exception as ex:

logger.critical("Fetcher Agent exited with error: %s", ex)

return -1

else:

logger.info("Fetcher Agent completed successfully.")

logging.info("Loader Agent started")

try:

with LoaderAgent(logger=logger, options=args) as agent:

agent.run()

except Exception as ex:

logger.critical("Loader Agent exited with error: %s", ex)

return -1

else:

logger.info("Loader Agent completed successfully.")

return 0

if \_\_name\_\_ == "\_\_main\_\_":

sys.exit(main())