"""

The script basically generates the request\_data\_object and posts to BT.

"""

import getpass

import json

import logging

from sqlalchemy.exc import DBAPIError

from etl.bbg\_transport.dto import RequestDataItem, RequestItem, RequestOptionItem

from etl.core import da\_config

from etl.core import util

from etl.core.util import uri\_post

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

from etl.repo.pim\_pm.pl\_bbg\_batch\_series\_vw import PlBbgBatchSeriesVwRepo

USAGE = ['Queuer Agent',

('--source\_code',

{

'help': 'Source code',

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

}

),

]

class QueuerAgent(object):

"""

"""

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

logging.info('QueuerAgent')

self.USERNAME = getpass.getuser()

self.config = da\_config.get\_etl\_cfg()

cfg\_section = self.config.get('pl\_auto')

self.base\_url = cfg\_section.get('base\_url')

self.description = cfg\_section.get('request\_description')

self.response\_format = cfg\_section.get('response\_format')

self.requestor\_code = cfg\_section.get('requestor\_code')

self.etl\_source\_code = self.options.get('source\_code')

@staticmethod

def get\_request(repo):

"""

:param repo:

:return:

"""

model = repo.model

try:

data = repo.query.filter(model.batch\_status\_code == 'IN\_QUEUE').all()

return data

except DBAPIError as err:

logging.error(err)

def get\_priority\_list(self, result):

"""

:param result:

:return:

"""

data\_list = list()

history\_list = list()

for i in result:

if i.bbg\_program\_code == 'GETDATA':

data\_list.append(i)

else:

history\_list.append(i)

data\_list = self.get\_priority\_list\_by\_interface\_code(data\_list)

history\_list = self.get\_priority\_list\_by\_interface\_code(history\_list)

plist = data\_list+history\_list

return plist

@staticmethod

def get\_priority\_list\_by\_interface\_code(result):

"""

:param result:

:return:

"""

plist = []

for i in result:

if i.bbg\_interface\_code == 'SAPI':

plist.insert(0, i)

else:

plist.append(i)

return plist

def get\_request\_object(self, objdata, result\_series):

"""

:param objdata:

:param result\_series:

:return:

"""

data\_items = [RequestDataItem(tag=i.pl\_series\_code, yellow\_key=i.bbg\_yellow,

ticker=i.bbg\_ticker) for i in result\_series]

headers = self.get\_headers(objdata)

fields = self.get\_request\_fields(result\_series)

request\_options = [RequestOptionItem(option\_name=key, option\_value=headers[key])

for key in headers]

request = RequestItem(request\_description=self.description,

requestor\_code=self.requestor\_code,

program\_code=objdata.bbg\_program\_code,

interface\_code=objdata.bbg\_interface\_code,

response\_format\_code=self.response\_format,

request\_data\_items=data\_items,

request\_options=request\_options,

request\_fields=fields)

payload = json.dumps(request.to\_json())

return payload

@staticmethod

def get\_headers(objdata):

"""

:param objdata:

:return:

"""

headers = dict()

if objdata.bbg\_program\_code == "GETDATA":

headers['DATERANGE'] = str(objdata.asof\_end\_date\_key) + \

"|" + str(objdata.asof\_end\_date\_key)

elif objdata.bbg\_program\_code == "GETHISTORY":

headers['DATERANGE'] = str(objdata.asof\_start\_date\_key) + \

"|" + str(objdata.asof\_end\_date\_key)

return headers

@staticmethod

def get\_request\_fields(result\_series):

"""

:param result\_series:

:return:

"""

request\_fields\_list = []

for i in result\_series:

request\_fields\_list.append(i.bbg\_mnemonic)

return list(set(request\_fields\_list))

def post\_to\_bt(self, payload):

"""

:param payload:

:return:

"""

response = uri\_post(self.base\_url + 'request\_data', payload)

return response

@staticmethod

def update\_request(batch\_id, bt\_request\_id, bt\_status\_code,

request\_obj, batch\_status\_code, repo):

"""

:param batch\_id:

:param bt\_request\_id:

:param bt\_status\_code:

:param request\_obj:

:param batch\_status\_code:

:param repo:

:return:

"""

model = repo.model

try:

update\_row = repo.query.filter(model.batch\_status\_code == 'IN\_QUEUE',

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

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

update\_row[0].bt\_request\_id = bt\_request\_id

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

update\_row[0].bt\_request\_payload = request\_obj

repo.save(update\_row)

except DBAPIError as err:

logging.error(err)

def run(self):

"""

:return:

"""

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

priority\_list = self.get\_priority\_list(result)

for i in priority\_list:

repo = PlBbgBatchSeriesVwRepo()

model = repo.model

try:

result\_batch = repo.query.filter(model.batch\_id == i.batch\_id).all()

obj = self.get\_request\_object(i, result\_batch)

response = self.post\_to\_bt(obj)

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

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

str(obj), 'SENT\_TO\_BT', PlBbgBatchRepo())

except Exception as err:

logging.error(err)

def main():

"""

Delegates all processing to Agent instance.

"""

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

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

try:

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

logging.info(cmd\_line)

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

args.source\_code = ''

args.etl\_audit\_id = ''

logging.info("Agent started")

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

agent.run()

except Exception as ex:

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

return -1

else:

logger.info("Agent completed successfully.")

return 0

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

sys.exit(main())