# Copyright (c) 2019 Digital Asset (Switzerland) GmbH and/or its affiliates. All rights reserved.

# SPDX-License-Identifier: Apache-2.0

import argparse

import csv

import json

import requests

import uuid

import time

import os

isLocalDev = True

owner = "Alice"

client = "Client1"

broker1 = "Broker1"

broker2 = "Broker2"

localToken = "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJsZWRnZXJJZCI6ImhlbGxvY2RtIiwiYXBwbGljYXRpb25JZCI6ImZvb2JhciIsInBhcnR5IjoiQWxpY2UifQ.SY9x-Eh\_mnPJwKzn4UXvHgtDSbFCRWZFqv0HgaGeXNI"

epoch = 0 # millis from epoch

localEndpoint = "http://localhost:7575"

dablEndpoint = "https://api.projectdabl.com/data"

metadataFileName = "../../resources/CDM.json"

defaultCdmEventFileName = "UC1\_Block\_Trade\_BT1.json"

endpoint = ''

ledgerID = ''

partyMap = {}

partyNameMap = {}

tradeDetails = {}

def loadCDMFile(fileName):

  """Opens a file containing a CDM JSON instance, and decodes into a Python dictionary."""

  fileName = "trade2/" + fileName

  with open(fileName) as cdmJsonString:

    return json.load(cdmJsonString)

def convertCDMJsonToDAMLJson(cdmDict):

  """Given a CDM dict, convert it into a dict that can be understood by the

     DAML HTTP REST service"""

  from message\_integration.metadata.cdm.cdmMetaDataReader import CdmMetaDataReader

  from message\_integration.metadata.damlTypes import Record

  from message\_integration.strategies.jsonCdmDecodeStrategy import JsonCdmDecodeStrategy

  from message\_integration.strategies.jsonCdmEncodeStrategy import JsonCdmEncodeStrategy

  with open(metadataFileName) as metadataRaw:

    metadata = CdmMetaDataReader().fromJSON(json.load(metadataRaw))

    return JsonCdmDecodeStrategy(metadata).decode(cdmDict, Record("Event"))

def writeDAMLJsonToLedger(damlDict, contractName, signatoryName, arguments, httpEndpointPrefix):

  """Given a dict containing a DAML contract, load to the ledger via the HTTP

     REST service. Return resulting HTTP response."""

  singatoryParty = partyNameToParty(signatoryName)

  token = partyNameToToken(signatoryName)

  tokenHeader = {'Authorization': f'Bearer {token}'}

  return requests.post(

    f"{endpoint}/command/create",

    headers = tokenHeader,

    json = {

      "templateId" : {

        "moduleName": "Main",

        "entityName": contractName

      },

      "meta" : {

        "ledgerEffectiveTime": epoch # Wall time unsupported on DABL

      },

      "argument": arguments

    },

    verify=False

  )

def readDAMLJsonFromLedger(contractName, signatoryName, httpEndpointPrefix):

  """Given the contract name, query ledger for all such contracts, returning

     the HTTP response, with a monkey patched `contract` accessor."""

  token = partyNameToToken(signatoryName)

  tokenHeader = {'Authorization': f'Bearer {token}'}

  response = requests.post(

    f"{endpoint}/contracts/search",

    headers = tokenHeader,

    json = {

      "%templates" : [

        {

          "moduleName" : "Main",

          "entityName" : contractName

        }

      ]

    },

    verify=False

  )

  if response.status\_code == 200:

    result = response.json()["result"]

    response.contractId = result[0]["contractId"] if result else None

  return response

def exerciseChoice(signatoryName, contractIdToExerciseOn, choiceName, choiceArguments, httpEndpointPrefix):

  """Exercises 'SayHello' on a CashTransfer contract.

  This sets the `contract.eventIdentifier.assignedIdentifier.identifier.value`

  to the given text, and increments the `version` by one.

  Return the updated contract:

  """

  token = partyNameToToken(signatoryName)

  tokenHeader = {'Authorization': f'Bearer {token}'}

  return requests.post(

    f"{endpoint}/command/exercise",

    headers = tokenHeader,

    json = {

      "meta" : {

        "ledgerEffectiveTime": epoch # Wall time unsupported on DABL

      },

      "templateId" : {

        "moduleName" : "Main",

        "entityName" : "UC2",

      },

      "contractId": contractIdToExerciseOn,

      "choice": choiceName,

      "argument": choiceArguments,

    },

    verify=False

  )

def partyToPartyName(party):

  partyData = partyMap.get(party, None)

  if not partyData:

    raise Exception(f'Could not translate party "{party}" to a party name')

  return partyData['partyName']

def partyNameToParty(partyName):

  partyData = partyNameMap.get(partyName, None)

  if not partyData:

    raise Exception(f'Could not translate party name "{partyName}" to a party')

  return partyData['party']

def partyToToken(party):

  partyData = partyMap.get(party, None)

  if not partyData:

    raise Exception(f'Could not fetch party JWT from party "{party}"')

  return partyData['jwt']

def partyNameToToken(partyName):

  partyData = partyNameMap.get(partyName, None)

  if not partyData:

    raise Exception(f'Could not fetch party JWT from party name "{partyName}"')

  return partyData['jwt']

def buildTradeDetails(cdmData):

  client = cdmData['party'][0]['name']['value']

  tradeType = cdmData['primitive']['execution'][0]['after']['execution']['executionType']

  tradeAmount = cdmData['primitive']['execution'][0]['after']['execution']['quantity']['amount']

  currency = cdmData['primitive']['execution'][0]['after']['execution']['price']['grossPrice']['currency']['value']

  executionKey = cdmData['primitive']['execution'][0]['after']['execution']['meta']['globalKey']

  tradeDetails['executionKey'] = executionKey

  tradeDetails['client'] = client

  tradeDetails['tradeType'] = tradeType

  tradeDetails['tradeAmount'] = tradeAmount

  tradeDetails['currency'] = currency

  print(tradeDetails)

  if not client:

    raise Exception(f'Could not fetch party JWT from party name ')

  return tradeDetails

if \_\_name\_\_ == '\_\_main\_\_' :

  parser = argparse.ArgumentParser("Hello CDM")

  parser.add\_argument('-d', '--local\_dev', action='store\_true')

  parser.add\_argument('-l', '--ledger\_id', type=str, help="The DABL Ledger ID")

  parser.add\_argument('-p', '--party\_map', type=str, help="Path to a .csv file containing a list of DABL partyName,party,jwt (no header)")

  parser.add\_argument('-c', '--cdm\_file',  type=str, help="CDM File")

  args = parser.parse\_args()

  lis = os.listdir(r"./trade2")

  print(lis)

# for line1 in lis:

#   #print(line1)

#   print(f"#### Loading CDM JSON from {line1} ####")

#   cdmJson = loadCDMFile(line1)

#   print("Loaded the following JSON object:")

#   print(cdmJson)

  for line1 in lis:

    print(f"#### Loading CDM JSON from {line1} ####")

    cdmJson = loadCDMFile(line1)

    #cdmJson = loadCDMFile(line1)

    cdmJson["meta"]["globalKey"] = str(uuid.uuid4()) # We overwrite the globalKey, to avoid DAML key clashes, allowing us to reload the same contract many times.

    #print("Loaded the following JSON object:")

    #print(cdmJson)

    tradeDetailsData = buildTradeDetails(cdmJson)

    print("#### Converting to DAML JSON, wrapping in an 'UC1 ' contract ####")

    damlJson = convertCDMJsonToDAMLJson(cdmJson)

    #print("Resulting JSON object:")

    #rint(damlJson)

    #if tradeDetailsData['tradeAmount'] != tradeDetailsData['allocation\_1'] + tradeDetailsData['allocation\_2']:

    #  raise Exception(f'Trade allocation is not matcing with trade amount !')

    print("#### Sending Transfer contract to ledger ####")

    arguments = {

          "event": damlJson,

          "owner": owner,

          "client": tradeDetailsData['client'],

          "broker1": broker1,

          "broker2": broker2,

          "obs": tradeDetailsData['client'],

          "executionKey": tradeDetailsData['executionKey'],

          "tradeType": tradeDetailsData['tradeType'],

          "tradeAmount": tradeDetailsData['tradeAmount'],

          "currency": tradeDetailsData['currency'],

          "tradeStage": "Allocation",

        }