```
import requests
import json
import shutil
import time
import urllib3
import gzip
import os
import sys
import pytz
import datetime
from datetime import datetime
from datetime import date
import datetime as dt
import time
from datetime import datetime, timedelta, timezone
import time
import calendar
from io import StringIO
import pandas as pd
```

Use own usernam, password and web proxies

```
useAws = True
os.environ['http_proxy'] = "http://hybrid-web.xxxxx.xxxxxxxxx.com:x0xx"
os.environ['https_proxy'] = "http://hybrid-web.xxxxx.xxxxxxxx.com:x0xx"


proxyServers =  {'http': os.environ['http_proxy'],
   'https': os.environ['https_proxy']}
userName = 'NNNNNNNNN'
passWord = 'NNNNNNNNN2'


# Define filePath and fileNameRoot
```

```
filePath = "C:\\Data\\FXtickdata\\"

fileNameRoot = "USDZAR"



token = '_OdKbJdylkhwlMjKQsIKeiPqbcw78uCCD2qQox3Kp_v_-
NSPuAcf85WCUIe42g_7HzS_oTpyht2cLMuLkS-GuyMHmiHnIWB0sK9-uwC67MvP8byDPTytXp9Zz9-
gGJxnZaSjn5FuaLnAb3DRb15bXQ2nC0pWAGP6H9KXRHP9qZt56SdIQzdzgei7f2FFOB3cE1ViyKfpMKfD
wq7sSBg8EPFjvey2m20TaqCB36Saacr7eCrcHOmRgui3nTPqR0o-pL6u7EuU-
IeHwdWn3CxwbJFZgkirQrX4-4_1BGEdWzZI'


def NewToken(un, pw):
    if not (un and pw):
        print ('Username or password is empty. Please type in your username and password in the Step1
section of the code and try again.')
        sys.exit()


    requestUrl = 'https://selectapi.datascope.refinitiv.com/RestApi/v1/Authentication/RequestToken'
    requestHeaders = {
        'Prefer':'respond-async',
        'Content-Type':'application/json'
        }
    requestBody = {
        'Credentials': {
        'Username': un,
        'Password': pw
      }
    }
    r1 = requests.post(requestUrl, json = requestBody, headers = requestHeaders, proxies =
proxyServers)


    if r1.status_code == 200 :
        jsonResponse = json.loads(r1.text.encode('ascii', 'ignore'))
        return jsonResponse["value"]
```

```
    else:
        print ('Please type in valid username and password in the Step1 section of the code and try again.')
        sys.exit()

if token:
    #check the validity of the authentication token
    requestUrl = 'https://selectapi.datascope.refinitiv.com/RestApi/v1/Users/Users(' + userName + ')'
    requestHeaders = {
        'Prefer':'respond-async',
        'Content-Type':'application/json',
        'Authorization': 'token ' + token
    }
    r1 = requests.get(requestUrl, headers = requestHeaders, proxies = proxyServers)
    #if the response status is 200 OK then the token is valid, otherwise request new token
    if r1.status_code != 200:
        token = NewToken(userName, passWord)
else:
    #token is empty, request new token.
    token = NewToken(userName, passWord)

print('using authorization token:')
print(token)

timestamp = pd.Timestamp('now')
today = timestamp.tz_localize('UTC').tz_convert(pytz.timezone('Africa/Johannesburg'))
today_str = today.strftime('%Y-%m-%d')

yesterday = today - pd.Timedelta(1, "d")
yesterday_str = yesterday.strftime('%Y-%m-%d')
```

```
start_date = pd.to_datetime('2010-01-01', format='%Y-%m-%d')

start_date_str = start_date.strftime('%Y-%m-%d')


filedate = (timestamp - pd.Timedelta(1, "d")).strftime('%Y%m%d')


QueryStartDate = start_date.strftime("%Y-%m-%dT%H:%M:%S.000Z")

QueryEndDate = yesterday.strftime("%Y-%m-%dT%H:%M:%S.000Z")


print("QueryStartDate:", QueryStartDate)

print("QueryEndDate:", QueryEndDate)


requestUrl='https://selectapi.datascope.refinitiv.com/RestApi/v1/Extractions/ExtractRaw'


requestHeaders={"Prefer":"respond-async","Content-Type":"application/json","Authorization":
"token " + token}


requestBody={

  "ExtractionRequest": {

    "@odata.type":
"#DataScope.Select.Api.Extractions.ExtractionRequests.TickHistoryIntradaySummariesExtractionRequ
est",

    "ContentFieldNames": ["Close Ask","Close Bid","High Ask","High Bid","Low Ask","Low Bid","No.
Asks","No. Bids",

      "Open Ask","Open Bid"],

    "IdentifierList": {

      "@odata.type": "#DataScope.Select.Api.Extractions.ExtractionRequests.InstrumentIdentifierList",

      "InstrumentIdentifiers": [{"Identifier": "ZAR=","IdentifierType": "Ric"},

                                #{"Identifier": "GBP=","IdentifierType": "Ric"}

                               ],

      "UseUserPreferencesForValidationOptions":"false"

    },

    "Condition": {
```

```
      "MessageTimeStampIn": "GmtUtc",
      "ReportDateRangeType": "Range",
      "QueryStartDate":QueryStartDate ,
      "QueryEndDate": QueryEndDate,
      "SummaryInterval": "OneMinute",
      "TimebarPersistence":"true",
      "DisplaySourceRIC":"true"
    }
  }
}

r2 = requests.post(requestUrl, json=requestBody,headers=requestHeaders)

#Display the HTTP status of the response
#Initial response status (after approximately 30 seconds wait) is usually 202
status_code = r2.status_code
print ("HTTP status of the response: " + str(status_code))

if status_code == 200 :
    r2Json = json.loads(r2.text.encode('ascii', 'ignore'))
    jobId = r2Json["JobId"]
    print ('\njobId: ' + jobId + '\n')
    notes = r2Json["Notes"]
    print ('Extraction notes:\n' + notes[0])

#If status is 202, display the location url we received, and will use to poll the status of the extraction request:
if status_code == 202 :
    requestUrl = r2.headers["location"]
    print ('Extraction is not complete, we shall poll the location URL:')
    print (str(requestUrl))
```

```
    requestHeaders={
        "Prefer":"respond-async",
        "Content-Type":"application/json",
        "Authorization":"token " + token
    }

#As long as the status of the request is 202, the extraction is not finished;
#we must wait, and poll the status until it is no longer 202:
while (status_code == 202):
    print ('As we received a 202, we wait 30 seconds, then poll again (until we receive a 200)')
    time.sleep(30)
    r3 = requests.get(requestUrl,headers=requestHeaders)
    status_code = r3.status_code
    print ('HTTP status of the response: ' + str(status_code))

#When the status of the request is 200 the extraction is complete;
#we retrieve and display the jobId and the extraction notes (it is recommended to analyse their content)):
if status_code == 200 :
    r3Json = json.loads(r3.text.encode('ascii', 'ignore'))
    jobId = r3Json["JobId"]
    print ('\njobId: ' + jobId + '\n')
    notes = r3Json["Notes"]
    print ('Extraction notes:\n' + notes[0])

#If instead of a status 200 we receive a different status, there was an error:
if status_code != 200 :
    print ('An error occured. Try to run this cell again. If it fails, re-run the previous cell.\n')
```

```
requestUrl = "https://selectapi.datascope.refinitiv.com/RestApi/v1/Extractions/RawExtractionResults" + "('" + jobId + "')" + "/$value"


#AWS requires an additional header: X-Direct-Download
if useAws:
   requestHeaders={
      "Prefer":"respond-async",
      "Content-Type":"text/plain",
      "Accept-Encoding":"gzip",
      "X-Direct-Download":"true",
      "Authorization": "token " + token
   }
else:
   requestHeaders={
      "Prefer":"respond-async",
      "Content-Type":"text/plain",
      "Accept-Encoding":"gzip",
      "Authorization": "token " + token
   }

r5 = requests.get(requestUrl,headers=requestHeaders,stream=True)
#Ensure we do not automatically decompress the data on the fly:
r5.raw.decode_content = False
if useAws:
   print ('Content response headers (AWS server): type: ' + r5.headers["Content-Type"] + '\n')
   #AWS does not set header Content-Encoding="gzip".
else:
   print ('Content response headers (LSEG Tick History server): type: ' + r5.headers["Content-Type"] + ' - encoding: ' + r5.headers["Content-Encoding"] + '\n')


#Next 2 lines display some of the compressed data, but if you uncomment them save to file fails
```

```
#print ('20 bytes of compressed data:')
#print (r5.raw.read(20))

# Add filedate to fileNameRoot
fileNameRoot = fileNameRoot + filedate

# Define fileName
fileName = filePath + fileNameRoot + ".csv.gz"
print ('Saving compressed data to file:' + fileName + ' ... please be patient')



chunk_size = 1024
rr = r5.raw
with open(fileName, 'wb') as fd:
    shutil.copyfileobj(rr, fd, chunk_size)
fd.close

print ('Finished saving compressed data to file:' + fileName + '\n')

#Now let us read and decompress the file we just created.
#For the demo we limit the treatment to a few lines:
maxLines = 10
print ('Read data from file, and decompress at most ' + str(maxLines) + ' lines of it:')

uncompressedData = ""
count = 0
with gzip.open(fileName, 'rb') as fd:
```

```
    for line in fd:
        dataLine = line.decode("utf-8")
        #Do something with the data:
        #print (dataLine)
        uncompressedData = uncompressedData + dataLine
        count += 1
        if count <= maxLines:
            #break
            print (dataLine)
fd.close()

from io import StringIO
import pandas as pd

dataprices = pd.read_csv(StringIO(uncompressedData))
dataprices.head()
```