from \_\_future\_\_ import absolute\_import  
#import requests  
from builtins import bytes  
from builtins import str  
from builtins import range  
from skybot.OF.lib.Utilities import dlp\_requests as requests  
import xml.etree.ElementTree as ET  
import time, datetime  
import timeout\_decorator  
import os  
import re  
from skybot.lib.logger import logger  
from skybot.OF.lib.Utilities.HealthMonitor import trackme  
from skybot.OF.lib.Utilities import Utils  
  
from skybot.OF.lib.core.SkyHighDashboard import Interface  
from .ServiceBase import CSP  
from robot.api.deco import keyword  
from robot.libraries.BuiltIn import BuiltIn  
import json  
import random  
import jinja2  
from skybot.lib import SHNInterface  
from skybot.OF.lib.core.SkyHighDashboard.ShnDlpInterface import ShnDlpUtil  
from skybot.AM.resources.locators import O365\_locators  
from skybot.lib.web\_automation.CommonHelper import CommonHelper  
from skybot.lib.web\_automation.ActionsHelper import ActionsHelper  
from skybot.lib.web\_automation.LocatorType import LocatorType  
from skybot.lib.web\_automation.SyncHelper import SyncHelper  
from skybot.OF.lib.Utilities.CustomError import AccesstokenError  
  
#requests.packages.urllib3.disable\_warnings()  
  
WS = ""  
if os.path.exists("OF/"):  
 WS = "OF/"  
  
class OneDrive(CSP):  
 known\_files = {}  
 attempt = 0  
 URL\_PATTERN\_TO\_FIND='-my.sharepoint'  
  
 def exclude\_external\_users(func):  
 def exclude\_users(instance):  
 internal\_user = []  
 users = func(instance)  
 for user in users:  
 if user is None or user.lower() == 'none':  
 continue  
 if instance.user.split('@')[1].lower() == user.split('@')[1].lower():  
 internal\_user.append(user)  
 logger.debug("Internal users list" +str(internal\_user))  
 return internal\_user  
 return exclude\_users  
  
 def \_\_init\_\_(self, shutil, qa\_env, tenantid, cspid, user=None, instance\_id = None):  
 logger.info("Initializing "+str(self.\_\_class\_\_.\_\_name\_\_)+" object...")  
 super(OneDrive, self).\_\_init\_\_(qa\_env)  
 self.user = user  
 self.password = None  
 # self.email\_flat , self.domain\_url = [None] \* 2  
 self.root\_folder = None  
 self.shutil = shutil  
 self.cspid = cspid  
 self.instance\_id = instance\_id  
 self.tenantid = tenantid  
 #logger.console("Inside onedrive: " + self.access\_token)  
 ##decrypt\_token is set as true to figure out the type of office365 application being installed###  
 self.access\_token = shutil.get\_access\_token(cspid, self.instance\_id)  
 self.access\_token\_graph = shutil.get\_access\_token(cspid,self.instance\_id, resource="https://graph.microsoft.com", decrypt\_token=True)  
 #xlogger.console("Inside onedrive: " + self.access\_token\_graph)  
 self.endpoint\_GetFolderByServerRelativeUrl = None  
 self.endpoint\_GetFileByServerRelativeUrl = None  
 self.endpoint\_users = None  
 self.endpoint\_contextinfo = None  
 self.domain\_name = None  
 self.list\_library\_name = self.get\_list\_name()  
 if self.\_get\_domain\_name():  
 self.\_get\_endpoints()  
 if self.access\_token is not None:  
 self.headers = {"Authorization": "Bearer " + self.access\_token, "Accept": "application/json"}  
 self.permission\_object = None  
 self.collaborated\_object = None  
 self.members\_to\_collaborate = []  
 self.multiple\_collaborators = None  
 self.external\_users = []  
 self.same\_domain\_external\_users = []  
 self.watchtower\_url = SHNInterface.myenv.get\_watchtower\_url()  
 self.stuck\_timeout = 45 # timeout value used esp. for methods that are getting stuck  
 # (self.request\_digest, self.cookies) = self.\_get\_request\_digest()  
 # Creating tmp directory in root folder to dump the user level XMl files  
 if not os.path.exists(WS+"tmp"):  
 os.mkdir(WS+"tmp")  
  
 if user is not None:  
 self.as\_user(user)  
 self.list\_guid = None  
 self.all\_collaborators = []  
 self.remaining\_collaborators =[]  
 self.isFlexiLink = None  
 self.site=None  
 self.response\_get\_all\_users=None  
  
  
 def as\_user(self, user):  
 self.user = user  
 #logger.console("Running onedrive with user: "+ self.user)  
 self.user\_flat , self.domain\_url = [None] \* 2  
 if self.domain\_name:  
 self.\_get\_endpoints()  
 (self.request\_digest, self.cookies) = self.\_get\_request\_digest()  
 # self.\_get\_endpoints()  
 # (self.request\_digest, self.cookies) = self.\_get\_request\_digest()  
  
 def \_refresh\_token(self):  
 logger.debug("Refreshing access token ...")  
 self.access\_token = self.shutil.get\_access\_token(self.cspid,self.instance\_id)  
 # logger.console("Inside onedrive: " + self.access\_token)  
 self.access\_token\_graph = self.shutil.get\_access\_token(self.cspid, self.instance\_id,resource="https://graph.microsoft.com")  
 if self.access\_token is not None:  
 self.headers = {"Authorization": "Bearer " + self.access\_token, "Accept": "application/json"}  
 self.user\_flat , self.domain\_url = [None] \* 2  
 self.\_get\_endpoints()  
 (self.request\_digest, self.cookies) = self.\_get\_request\_digest()  
  
 # Files method  
 def upload\_file(self, filename, parent\_id = 0, overwrite=True):  
 *"""  
 To upload the file to Onedrive.  
  
 Args:  
 parent\_id: Id of the folder where file required to be uploaded  
 filename: filename with location to be uploaded  
 overwrite (Optional) : Should we update file if it exists. Default is True  
  
 Returns:  
 Id: File ID of the uploaded file.  
  
 Raises:  
 None  
 """* super(OneDrive, self).upload\_file(filename)  
 file\_id = None  
 retry = 3  
 filename = str(filename)  
 if self.testdata not in filename:  
 filename = self.testdata + "/" + filename  
 if not overwrite:  
 file, ext = filename.split('/')[-1].split('.')  
 new\_file = file + str(time.time()).split('.')[0]+ '.' + ext  
 filename = '/'.join(filename.split('/')[:-1]) + '/' + new\_file  
 # headers = {"X-RequestDigest": self.request\_digest, "Content-Type": self.get\_mime\_type(filename=os.path.basename(filename)),  
 # "Accept": "application/json"}  
 parent\_id, self.mostrecentfolder = [0 if self.mostrecentfolder is None else self.mostrecentfolder] \* 2  
 if parent\_id:  
 endpoint\_GetFolderByServerRelativeUrl = re.sub('GetFolderByServerRelativeUrl(.\*)',  
 'GetFolderByServerRelativeUrl(\'' + parent\_id + '\')',  
 self.endpoint\_GetFolderByServerRelativeUrl)  
 else:  
 endpoint\_GetFolderByServerRelativeUrl = self.endpoint\_GetFolderByServerRelativeUrl  
 upload\_url = endpoint\_GetFolderByServerRelativeUrl + '/Files/add(url=\'' + os.path.basename(filename) \  
 + '\', overwrite=true)'  
 logger.info("URL to upload is " + upload\_url)  
  
 with open(filename, "rb") as fp:  
 if filename not in OneDrive.known\_files:  
 OneDrive.known\_files[filename] = fp.read()  
  
 # if filename not in OneDrive.known\_files:  
 # OneDrive.known\_files[filename] = open(filename, 'rb')#"".join(file\_handle.readlines())  
 # data\_to\_upload = OneDrive.known\_files[filename].read()  
  
 #logger.console("loading to memory")  
 #file\_handle = open(filename, "rb")  
  
 for i in range(retry):  
 headers = {"X-RequestDigest": self.request\_digest,"Content-Type": self.get\_mime\_type(filename=os.path.basename(filename)),"Accept": "application/json"}  
  
 logger.debug("Size of the data to uplaod >>>>"+ str(len(OneDrive.known\_files[filename])))  
 response\_upload\_file = requests.post(upload\_url, headers=headers, cookies=self.cookies, data=OneDrive.known\_files[filename])  
 self.response = response\_upload\_file  
  
 if (response\_upload\_file.status\_code == 200):  
 # file\_size\_before\_upload = os.path.getsize(OneDrive.known\_files[filename].name)  
 # logger.debug(">>>>[Before Upload] file {0} with file size {1}".format(OneDrive.known\_files[filename].name,file\_size\_before\_upload))  
  
 file\_size\_after\_upload = json.loads(response\_upload\_file.text)['Length']  
 logger.debug(">>>>[After Upload] file file size is {0}".format(file\_size\_after\_upload))  
 if (int(file\_size\_after\_upload) >0):  
 logger.info("Response for upload file is " + str(response\_upload\_file.text))  
 break  
 else:  
 file\_id=response\_upload\_file.json()["ServerRelativeUrl"]  
 self.delete\_file(file\_id)  
 logger.warn("File size is 0 byte so retrying .. ")  
  
 elif response\_upload\_file.status\_code in [401, 403]:  
 logger.debug("Got a 403 error refreshing access token...")  
 self.\_refresh\_token()  
  
 else:  
 logger.console("status code is not matched {0}".format(response\_upload\_file.status\_code))  
 break  
  
  
 #response\_upload\_file = requests.post(upload\_url, headers=headers, cookies=self.cookies, data=OneDrive.known\_files[filename], timeout=1)  
 if isinstance(response\_upload\_file, dict):  
 if response\_upload\_file.get('status\_code') == 504:  
 logger.warn('request timed out for {0} after {1} sec, however we will return True '  
 'assuming, post request is successful!!'.format("upload\_file", self.stuck\_timeout))  
 assumed\_file\_id = str(parent\_id + '/' + str(filename).split("/")[-1])  
 self.lastuploadedfiles.append(  
 {  
 "fileid": assumed\_file\_id,  
 "filename": str(filename).split("/")[-1],  
 "folderid": parent\_id,  
 "quarantineref": '/personal/' + self.user.replace("@", '\_').replace('.', '\_') + ':' + str(assumed\_file\_id),  
 "permissions\_object": {"id": self.mostrecentfolder, "permissions\_list": None}  
 }  
 )  
 test\_name = BuiltIn().replace\_variables('${TEST\_NAME}')  
 BuiltIn().set\_suite\_metadata(test\_name + "\_" + str(self.instance\_id) + "\_lastuploadedfiles",  
 self.lastuploadedfiles)  
 return assumed\_file\_id  
  
 logger.info("Response post upload file is: " + response\_upload\_file.text)  
 if "ServerRelativeUrl" in response\_upload\_file.json():  
 logger.info("File " + os.path.basename(filename) + " is successfully uploaded")  
 file\_id = response\_upload\_file.json()["ServerRelativeUrl"]  
 self.lastuploadedfiles.append(  
 {  
 "fileid":str(file\_id),  
 "filename":str(filename).split("/")[-1],  
 "folderid": parent\_id,  
 "quarantineref":'/personal/' + self.user.replace("@",'\_').replace('.', '\_') + ':' + str(file\_id),  
 "permissions\_object": {"id": self.mostrecentfolder, "permissions\_list": None}  
  
 }  
 )  
 logger.info("Files uploaded thus far: " + str(self.lastuploadedfiles))  
 test\_name = BuiltIn().replace\_variables('${TEST\_NAME}')  
 BuiltIn().set\_suite\_metadata(test\_name + "\_" + str(self.instance\_id) + "\_lastuploadedfiles", self.lastuploadedfiles)  
 return file\_id  
  
 def update\_file(self, file\_id, filename):  
 *"""  
 To update the file when we know file id  
  
 Args:  
 file\_id: Id of the file to be updated  
 filename: filename with location to be updated with  
  
 Returns:  
 Id: File ID of the updated file.  
  
 Raises:  
 None  
 """* return NotImplementedError  
  
 @keyword("download last uploaded file in")  
 def download\_file(self, file\_id=None):  
 *"""  
 To download the file when we know file id or name  
  
 Args:  
 file\_to\_check: structured information about the file to download filename, fileid and folderid  
 length: The number of bytes to receive, default or -1 is receive all data  
 Returns:  
 file: content of the file. as byte array  
  
 """* headers = {"X-RequestDigest": self.request\_digest}  
 if file\_id is None:  
 file\_id = self.lastuploadedfiles[-1].get('fileid')  
  
 logger.debug("Going to download file with file\_id: " + str(file\_id))  
  
 endpoint\_GetFileByServerRelativeUrl = re.sub('GetFileByServerRelativeUrl(.\*)',  
 'GetFileByServerRelativeUrl(\'' + file\_id + '\')',  
 self.endpoint\_GetFileByServerRelativeUrl)  
 get\_file\_data\_url = endpoint\_GetFileByServerRelativeUrl + "/$value"  
 request\_headers = self.headers.copy() # create a copy of the headers dict for use only in this function  
 r = requests.get(url=get\_file\_data\_url, headers=request\_headers, cookies=self.cookies, stream=True)  
 if r.status\_code in [401, 403]:  
 logger.warn("Got a 40X error with content: " + r.text)  
 self.\_refresh\_token()  
 return False  
 if r.status\_code in [200, 206]:  
 logger.info("file downloaded successfully: " + str(len(r.content)) + " bytes")  
 self.lastdownloadedfilecontents = bytes(r.content)  
 return True  
 else:  
 logger.debug("error while downloading file")  
 return False  
  
 @keyword("move last uploaded file to")  
 def move\_file(self, file\_id=None):  
 *"""  
  
 :param file\_id:  
 :return:  
 """* headers = {"X-RequestDigest": self.request\_digest}  
 filename = ""  
  
 if file\_id is None:  
 file\_id = self.lastuploadedfiles[-1].get('fileid')  
 filename = self.lastuploadedfiles[-1].get('filename')  
  
 move\_filename = filename  
 logger.debug("Going to move file with file\_id: " + str(file\_id))  
 move\_folder\_id = self.create\_folder("random")  
 endpoint\_GetFileByServerRelativeUrl = re.sub('GetFileByServerRelativeUrl(.\*)','GetFileByServerRelativeUrl(\'' + file\_id + '\')',self.endpoint\_GetFileByServerRelativeUrl)  
 file\_move\_url = endpoint\_GetFileByServerRelativeUrl + "/moveto" + "(newurl='" + str(move\_folder\_id) + "/" + move\_filename + "')"  
 response\_move\_file = requests.post(url=file\_move\_url, headers=headers, cookies=self.cookies)  
 if response\_move\_file.status\_code == 200:  
 logger.info("Response for upload file is " + str(response\_move\_file.text))  
 self.lastuploadedfiles.append(  
 {  
 "fileid":str(file\_id),  
 "filename":str(move\_filename).split("/")[-1],  
 "folderid": move\_folder\_id,  
 "quarantineref":'/personal/' + self.user.replace("@",'\_').replace('.', '\_') + ':' + str(file\_id),  
 "permissions\_object": {"id": self.mostrecentfolder, "permissions\_list": None}  
  
 }  
 )  
 elif response\_move\_file.status\_code in [401,403]:  
 logger.debug("Got a 403 error refreshing access token...")  
 self.\_refresh\_token()  
 raise Exception  
 logger.debug("Response post move file is: " + response\_move\_file.text)  
  
 return file\_id  
  
 @keyword("delete last uploaded file in")  
 def delete\_file(self, file\_id=None):  
 *"""  
 To delete the file using file id  
  
 Args:  
 file\_id: Id of the file to be deleted.  
 e.g. "/personal/admin\_ak5\_onmicrosoft\_com/Documents/ShareIt/DeleteMe.txt"  
  
 Returns:  
 file: True if able to delete else False  
  
 Raises:  
 None  
 """* if file\_id is None:  
 file\_id = self.lastuploadedfiles[-1].get('fileid')  
  
 logger.debug('Going to delete filename with file\_id '+str(file\_id))  
 headers = {"X-RequestDigest": self.request\_digest, "Accept": "application/json"}  
 is\_file\_deleted = False  
  
 delete\_url = re.sub('GetFileByServerRelativeUrl(.\*)',  
 'GetFileByServerRelativeUrl(\'' + file\_id + '\')',  
 self.endpoint\_GetFileByServerRelativeUrl)  
 logger.info("Url to delete the file " + file\_id + " is " + delete\_url)  
 response\_delete\_file = requests.delete(delete\_url, headers=headers, cookies=self.cookies)  
 if response\_delete\_file:  
 is\_file\_deleted = True  
 return is\_file\_deleted  
  
 def get\_file\_info(self, file\_id, list\_item\_all\_fields=None):  
 *"""  
 get info of the file. will return json of the file property given it's file id  
 :param file\_id: file id  
 :return: JSON  
 """* logger.debug("Inside get\_file\_info")  
 headers = {"X-RequestDigest": self.request\_digest, "Accept": "application/json"}  
 endpoint\_GetFileByServerRelativeUrl = re.sub('GetFileByServerRelativeUrl(.\*)',  
 'GetFileByServerRelativeUrl(\'' + file\_id + '\')',  
 self.endpoint\_GetFileByServerRelativeUrl)  
 if self.site:  
 endpoint\_GetFileByServerRelativeUrl = re.sub('(.\*)\/\_api', self.domain\_url + '/sites/' + self.site + '/\_api', \  
 endpoint\_GetFileByServerRelativeUrl)  
 get\_file\_info\_url = endpoint\_GetFileByServerRelativeUrl + list\_item\_all\_fields  
 response\_file\_info = requests.get(url=get\_file\_info\_url, headers=headers, cookies=self.cookies)  
 if response\_file\_info.status\_code == 200:  
 logger.info("Response for getting file info is " + str(response\_file\_info.text))  
  
 elif response\_file\_info.status\_code in [401,403]:  
 logger.debug("Got a 403 error..")  
 self.\_refresh\_token()  
 if site.self:  
 req\_dig = self.req\_digest(self.domain\_url + "/sites/" + self.site)  
 headers["X-RequestDigest"] = req\_dig  
 raise Exception  
  
 logger.debug("Response from requesting file info for " + str(file\_id) + " + is " + str(response\_file\_info.text))  
 return response\_file\_info.json()  
  
 # Folders method  
  
 def create\_folder(self, folder\_name, parent\_id=None,site=None):  
 *"""  
 To create folder given parent id and folder name  
  
 Args:  
 parent\_id: Id of the folder where folder needs to be created e.g. ( folder1/folder2 )  
 folder\_name: Name of folder to be created  
  
 Returns:  
 folder id  
  
 Raises:  
 None  
 """* super(OneDrive, self).create\_folder(folder\_name, parent\_id)  
 headers = {"X-RequestDigest": self.request\_digest, "Accept": "application/json"}  
 logger.info("Request digest is " + str(self.request\_digest))  
 logger.info("Cookies is: " + str(self.cookies))  
 folder\_id = None  
 if folder\_name is None:  
 logger.error("Folder name is None")  
 return False  
 if str(folder\_name) == "random":  
 folder\_name = str(time.time())  
 logger.info("Inside create folder")  
 if parent\_id:  
 endpoint\_GetFolderByServerRelativeUrl = re.sub('GetFolderByServerRelativeUrl(.\*)',  
 'GetFolderByServerRelativeUrl(\'' + parent\_id + '\')',  
 self.endpoint\_GetFolderByServerRelativeUrl)  
 else:  
 endpoint\_GetFolderByServerRelativeUrl = self.endpoint\_GetFolderByServerRelativeUrl  
  
 if site:  
 self.site = site[type(self).\_\_name\_\_]  
 endpoint\_GetFolderByServerRelativeUrl = re.sub('(.\*)\/\_api',  
 self.domain\_url + '/sites/' + self.site + '/\_api',  
 endpoint\_GetFolderByServerRelativeUrl)  
 req\_dig = self.req\_digest(self.domain\_url + "/sites/" + self.site)  
 headers["X-RequestDigest"] = req\_dig  
 else:  
 self.site = None  
  
  
 create\_folder\_url = endpoint\_GetFolderByServerRelativeUrl + '/Folders/add(url=\'' + folder\_name + '\')'  
 logger.info("Create Folder url is " + create\_folder\_url)  
 response\_create\_folder = requests.post(create\_folder\_url, headers=headers, cookies=self.cookies)  
 self.response = response\_create\_folder  
 if response\_create\_folder.status\_code == 200:  
 logger.info("Response for creating folder is " + str(response\_create\_folder.text))  
  
 elif response\_create\_folder.status\_code in [401,403]:  
 logger.debug("Got a 403 error..")  
 self.\_refresh\_token()  
 if response\_create\_folder.status\_code==403 and OneDrive.attempt<3:  
 OneDrive.attempt = OneDrive.attempt + 1  
 self.create\_folder(folder\_name, parent\_id)  
 raise Exception  
  
 if "ServerRelativeUrl" in response\_create\_folder.json():  
 logger.info("Folder is created successfully with ServerRelativeUrl as " + response\_create\_folder.json()["ServerRelativeUrl"])  
 folder\_id = response\_create\_folder.json()["ServerRelativeUrl"]  
 self.mostrecentfolder = folder\_id  
 self.mostrecentfoldername = folder\_name  
 test\_name = BuiltIn().replace\_variables('${TEST\_NAME}')  
 BuiltIn().set\_suite\_metadata(test\_name + "\_" + str(self.instance\_id) + "\_mostrecentfolder", self.mostrecentfolder)  
 BuiltIn().set\_suite\_metadata(test\_name + "\_" + str(self.instance\_id) + "\_mostrecentfoldername", self.mostrecentfoldername)  
 return folder\_id  
  
 def delete\_folder(self, folder\_id):  
 *"""  
 Delete the folder given folder id. Yet to be implemented  
  
 Args:  
 folder\_id: Id of the folder to be deleted  
  
 Returns:  
 Boolean  
  
 Raises:  
 None  
 """* logger.debug("Going to delete the folder with id: " + folder\_id)  
 if not folder\_id:  
 folder\_id = self.mostrecentfolder  
 if folder\_id:  
 endpoint\_GetFolderByServerRelativeUrl = re.sub('GetFolderByServerRelativeUrl(.\*)',  
 'GetFolderByServerRelativeUrl(\'' + folder\_id + '\')',  
 self.endpoint\_GetFolderByServerRelativeUrl)  
 else:  
 endpoint\_GetFolderByServerRelativeUrl = self.endpoint\_GetFolderByServerRelativeUrl  
  
 headers = {"X-RequestDigest": self.request\_digest, "Accept": "application/json"}  
 response\_folder\_delete = requests.delete(url=endpoint\_GetFolderByServerRelativeUrl, headers=headers, cookies=self.cookies)  
 # response\_folder\_delete = requests.delete(url=folder\_id, headers=headers, cookies=self.cookies)  
 if response\_folder\_delete.status\_code == 200:  
 logger.info("Folder Deleted Success => " + str(folder\_id))  
  
 elif response\_folder\_delete.status\_code in [401,403]:  
 logger.debug("Got a 403 error..")  
 self.\_refresh\_token()  
 raise Exception  
  
 def get\_folder\_info(self, folder\_id=None, params=""):  
 *"""  
 Get info about the folder  
  
 Args:  
 folder\_id: Id of the folder to be deleted  
  
 Returns:  
 List containing JSON of file properties  
  
 Raises:  
 None  
 """* logger.debug("inside get\_folder\_info")  
 if not folder\_id:  
 folder\_id = self.mostrecentfolder  
 if folder\_id:  
 endpoint\_GetFolderByServerRelativeUrl = re.sub('GetFolderByServerRelativeUrl(.\*)',  
 'GetFolderByServerRelativeUrl(\'' + folder\_id + '\')',  
 self.endpoint\_GetFolderByServerRelativeUrl)  
 else:  
 endpoint\_GetFolderByServerRelativeUrl = self.endpoint\_GetFolderByServerRelativeUrl  
 logger.debug("Inside get\_folder\_info with params: folder\_id" + str(folder\_id))  
 get\_folder\_url = endpoint\_GetFolderByServerRelativeUrl + params  
 for attempt in range(1, 3):  
 headers = {"X-RequestDigest": self.request\_digest, "Accept": "application/json"}  
 response\_folder\_info = requests.get(url=get\_folder\_url, headers=headers, cookies=self.cookies)  
 logger.debug(response\_folder\_info.json())  
 if response\_folder\_info.status\_code == 200:  
 logger.debug(response\_folder\_info.json())  
 return response\_folder\_info.json()  
 elif response\_folder\_info.status\_code in [401, 403]:  
 logger.info("Token expired, refreshing and trying again")  
 self.\_refresh\_token()  
 else:  
 logger.warn("Failed to get folder info, will retry after 10 sec: " + str(attempt) + "/3")  
 time.sleep(10)  
 continue  
 logger.error("Failed to get folder info post attempt 3 times: " + str(folder\_id))  
 raise Exception  
  
 # Permission methods  
  
 def list\_permissions(self, object\_id):  
 *"""  
 List permissions of an object . Currently implemented for folder only ( yet to implement )  
  
 Args:  
 object\_id: Id of the object for which permissions required to be listed  
  
 Returns:  
 JSON of all permissions been part of the object e.g ["email": "user1@ak5.onmicrosoft.com", "role": "editor"]  
 RoleType mappings reference:  
 https://msdn.microsoft.com/en-us/library/microsoft.sharepoint.client.roletype%28v=office.14%29.aspx  
  
 Raises:  
 None  
 """* if str(object\_id) == "None":  
 object\_id = self.collaborated\_object  
  
 logger.debug("Inside list\_permissions to fetch permission for " + str(object\_id))  
 permissions\_list = []  
 permissions\_dict = {6: "none", 1: "viewer", 2: "editor", 3: "guest", 5: "owner"}  
 principaltype\_dict = {0 :"none", 1 :"user", 2 :"distributionlist", 4 :"securitygroup", 8 :"sharepointgroup", 15 :"all"}  
 headers = self.headers.copy()  
 headers.update({"Accept": "application/json"})  
  
 # headers = {"X-RequestDigest": self.request\_digest, "Accept": "application/json"}  
  
 if object\_id:  
 if self.lastuploadedfiles and self.lastuploadedfiles[-1].get('filename') == object\_id.split('/')[-1]:  
  
 endpoint\_GetByServerRelativeUrl = re.sub('GetFileByServerRelativeUrl(.\*)',  
 'GetFileByServerRelativeUrl(\'' + object\_id + '\')',  
 self.endpoint\_GetFileByServerRelativeUrl)  
 else:  
 endpoint\_GetByServerRelativeUrl = re.sub('GetFolderByServerRelativeUrl(.\*)',  
 'GetFolderByServerRelativeUrl(\'' + object\_id + '\')',  
 self.endpoint\_GetFolderByServerRelativeUrl)  
 else:  
 if self.lastuploadedfiles and self.lastuploadedfiles[-1].get('filename') == object\_id.split('/')[-1]:  
  
 endpoint\_GetByServerRelativeUrl = self.endpoint\_GetFileByServerRelativeUrl  
 else:  
 endpoint\_GetByServerRelativeUrl = self.endpoint\_GetFolderByServerRelativeUrl  
  
 self.get\_permissions\_url = endpoint\_GetByServerRelativeUrl + \  
 "?$expand=ListItemAllFields/ParentList,ListItemAllFields/RoleAssignments/Member," + \  
 "ListItemAllFields/RoleAssignments/RoleDefinitionBindings," + \  
 "ListItemAllFields/RoleAssignments/Member/Users"  
  
 response\_get\_permissions = requests.get(self.get\_permissions\_url, headers=headers, cookies=self.cookies)  
  
 if response\_get\_permissions.status\_code == 200:  
 #logger.info("Response for getting permission is " + str(response\_get\_permissions.text))  
 logger.info("Response for getting permission is successful")  
  
 elif response\_get\_permissions.status\_code in [401,403]:  
 logger.debug("Got a 403 error..")  
 logger.debug("Response for getting permission is " + str(response\_get\_permissions.text))  
 self.\_refresh\_token()  
  
 # logger.debug(json.dumps(response\_get\_permissions.json().get("ListItemAllFields").get("RoleAssignments")))  
 for item in response\_get\_permissions.json()["ListItemAllFields"]["RoleAssignments"]:  
 if "Email" in item["Member"] and not item["Member"]["IsSiteAdmin"]:  
 #logger.debug("item\_Member\_email is: " + item["Member"]["Email"])  
 #logger.debug("self.user is: " + self.user)  
 if item["Member"]["Email"] and item["Member"]["Email"] not in self.user:  
 member = {"email":"", "role": "", "type": ""}  
 member["email"] = str(item["Member"]["Email"])  
 # permissions\_list.append({"email": "", "role": ""})  
 logger.trace("User found with EmailID: " + str(item["Member"]["Email"]))  
 # permissions\_list[-1]["email"] = str(item["Member"]["Email"])  
 role\_type = []  
 for role\_list in item["RoleDefinitionBindings"]:  
 logger.trace("Role associated with the user is:" + str(role\_list["RoleTypeKind"]))  
 role\_type.append(role\_list["RoleTypeKind"])  
 logger.trace("Role type list contains " + str(role\_type))  
 # permissions\_list[-1]["role"] = permissions\_dict[max(role\_type)]  
 member["role"] = permissions\_dict[max(role\_type)]  
 if member["role"] is not "guest":  
 permissions\_list.append(member)  
 member["type"] = principaltype\_dict[int(item["Member"]["PrincipalType"])]  
 logger.trace("Response of retrieving all collab is " + str(permissions\_list))  
 return permissions\_list  
  
 def list\_permission(self, permission\_id):  
 *"""  
 List permission given the permission ID  
 :param permission\_id: Permission Id of an object for which info is required  
 :return: JSON representing the different attributes of a permission e.g. username, role  
 """* return NotImplementedError  
  
# @timeout\_decorator.timeout(90, use\_signals=False)  
 def add\_permission(self, object\_id, user\_attr, reset=True, file\_collaboration=False,collaborator=None,add\_to\_all\_collaborators=True):  
 *"""  
 Add a new permission given the object, and user attributes been provided  
 Role details : 1 = View, 2 = Edit, 3 = Owner, 0 = None  
  
 Args:  
 object\_id: Id of the object for which permissions required to be listed  
 user\_attr: List of user e.g. {"email":"skyhigh.blore@gmail.com", "role": 2}  
 If email is "default" , we get current user info and add +1 to the user name  
 e.g. skyhigh.blore@gmail.com will become skyhigh.blore+1@gmail.com  
 reset: Optional, if true it will re apply the same permission if found  
  
 Returns:  
 True : If permission is added successfully  
  
 Raises:  
 None  
 """* if (self.user not in self.remaining\_collaborators) and (self.\_\_class\_\_.\_\_name\_\_ == "SharePoint"):  
 self.remaining\_collaborators.append(self.user)  
 try:  
 modified\_permision = BuiltIn().replace\_variables('${permission\_type}')  
 except:  
 modified\_permision = ''  
  
 user = None  
 user\_attr\_dict = {"viewer": 1, "editor": 2, "owner": 3}  
 result = False  
 variables = BuiltIn().get\_variables()  
 if "${multiple\_collaborators}" in variables:  
 self.multiple\_collaborators = BuiltIn().replace\_variables('${multiple\_collaborators}')  
 if "${flexilink}" in variables:  
 self.isFlexiLink = BuiltIn().replace\_variables('${flexilink}')  
 else:  
 self.isFlexiLink = False  
  
 if not object\_id:  
 if file\_collaboration:  
 if self.mostrecentfolder and self.mostrecentfolder!=0:  
 self.collaborated\_object = self.mostrecentfolder + "/" + self.lastuploadedfiles[-1].get('filename')  
 else:  
 self.collaborated\_object = self.lastuploadedfiles[-1].get('fileid')  
 else:  
 self.collaborated\_object = self.mostrecentfolder  
 else:  
 self.collaborated\_object = object\_id  
  
 test\_name = BuiltIn().replace\_variables('${TEST\_NAME}')  
 BuiltIn().set\_suite\_metadata(test\_name + "\_" + str(self.instance\_id) + "\_collaborated\_object",  
 self.collaborated\_object)  
  
 logger.debug("Going to add permission for object: " + str(self.collaborated\_object))  
 # logger.debug("last created group onedrive" + str(self.lastcreatedgroup))  
 user\_attr = eval(str(user\_attr))  
 user\_attr["role"] = user\_attr\_dict[user\_attr["role"].lower()]  
 logger.debug("user\_attr contain " + str(user\_attr))  
 collaborators = list()  
  
 if user\_attr["email"] == "\*":  
 user = self.\_get\_another\_user\_for\_collab()  
 user\_attr["email"] = user  
 logger.debug("User to Collaborate with: " + user)  
 elif user\_attr["email"] == "external collaborator":  
 user = self.externalcollaborator  
 user\_attr["email"] = user  
 elif user\_attr["email"] == "group":  
 if self.lastcreatedgroup[-1]["email"] != None:  
 user = self.lastcreatedgroup[-1]["email"]  
 else:  
 user = self.lastcreatedgroup[-1]["groupname"]  
 logger.debug("User to Colloborate with: " + user)  
 if not self.isFlexiLink:  
 user\_attr["role"] = "role:1073741830" if user\_attr["role"] == 2 else "role:1073741826"  
 elif user\_attr["email"] == 'unaccepted invite collaborator':  
 collaborators = self.unacceptedinvitecollaborator.split(",")  
 user = collaborators[0]  
 else:  
 if isinstance(user\_attr["email"].split('@'), list) and \  
 user\_attr["email"].split('@')[1] != self.user.split('@')[1]:  
 user = user\_attr["email"]  
 user\_attr["email"] = 'unaccepted invite collaborator'  
 collaborators = [user]  
 else:  
 user = user\_attr["email"]  
  
 #Next set of if else conditions are to invoke collaboration methods as per the collaboration types  
 if user\_attr["email"] == 'unaccepted invite collaborator' or (user\_attr["email"] == user and self.isFlexiLink):  
 '''  
 role:1073741830(previously 1073741827) is for edit role and role:1073741826 is for view  
 '''  
 if (user\_attr["email"] == user and self.isFlexiLink):  
 collaborators = [user]  
 collaboration\_details = {  
 "collaborators": collaborators,  
 "add\_to\_all\_collaborators": add\_to\_all\_collaborators,  
 "modified\_permision": modified\_permision,  
 }  
 result = self.flexiLinkUserCollaboration(user\_attr, collaboration\_details)  
 elif user\_attr["email"] == "group" and self.isFlexiLink:  
 user\_attr["role"] = "2"  
 if add\_to\_all\_collaborators:  
 self.all\_collaborators.append(self.lastcreatedgroup[-1]["groupname"])  
 if add\_to\_all\_collaborators and modified\_permision.lower() == 'viewer' or add\_to\_all\_collaborators is False:  
 self.remaining\_collaborators.append(self.lastcreatedgroup[-1]["groupname"])  
 result = self.flexiLinkGroupCollaboration(user\_attr,file\_collaboration)  
 elif user\_attr["email"] == "group":  
 collaboration\_details = {  
 "collaborator": collaborator,  
 "add\_to\_all\_collaborators": add\_to\_all\_collaborators,  
 "modified\_permision": modified\_permision,  
 "file\_collaboration": file\_collaboration  
 }  
 result = self.directAccessGroupCollaboration(collaboration\_details, user\_attr)  
 else:  
 if add\_to\_all\_collaborators:  
 self.all\_collaborators.append(user\_attr["email"])  
 if add\_to\_all\_collaborators and modified\_permision.lower() == 'viewer' or add\_to\_all\_collaborators is False:  
 self.remaining\_collaborators.append(user\_attr["email"])  
 result = self.directAccessUserCollaboration(user\_attr, file\_collaboration)  
  
 BuiltIn().set\_suite\_metadata(test\_name + "\_" + str(self.instance\_id) + "\_permission\_object",  
 self.permission\_object)  
 return result  
  
 def get\_listid\_itemId(self,item,file=None):  
 *"""  
 get info listId and itemId of the file/folder.  
 :param file/folder  
 :return: tuple  
 """* retry = 3  
 logger.debug("Inside get\_listid\_itemId")  
 headers = {"X-RequestDigest": self.request\_digest, "Accept": "application/json"}  
  
 endpoint\_GetListByServerRelativeUrl = self.endpoint\_GetFileListByServerRelativePathUrl if file \  
 else self.endpoint\_GetFolderListByServerRelativePathUrl  
  
  
 endpoint\_GetListByServerRelativeUrl = endpoint\_GetListByServerRelativeUrl % item  
  
 while retry > 0:  
 response = requests.get(url=endpoint\_GetListByServerRelativeUrl, headers=headers, cookies=self.cookies)  
 if response.status\_code == 200:  
 logger.debug("Response for ListAllItems: " + str(response.text))  
 m = re.search("guid'(.\*)'\)/Items\((.\*)\)", response.json()["odata.editLink"])  
 id\_tuple = m.groups()  
 break  
 elif response.status\_code in [401, 403]:  
 logger.debug("Got a 403 error..")  
 self.\_refresh\_token()  
 retry = retry - 1  
  
 return id\_tuple  
  
 def directAccessGroupCollaboration(self, collaboration\_details, user\_attr):  
 newApi = None  
 try:  
 newApi = BuiltIn().replace\_variables('${listApi}')  
 except:  
 pass  
  
 if newApi is not None and newApi:  
 listId, itemId = self.get\_listid\_itemId(self.collaborated\_object, collaboration\_details["file\_collaboration"])  
 endpoint\_sharing = self.endpoint\_DirectAccessSharing\_listId % (listId,itemId)  
 else:  
 endpoint\_sharing = self.endpoint\_File\_DirectAccessSharing if collaboration\_details["file\_collaboration"] \  
 else self.endpoint\_Folder\_DirectAccessSharing  
 endpoint\_sharing = endpoint\_sharing % self.collaborated\_object  
 logger.debug("URL for sharing is " + endpoint\_sharing)  
  
 # Create the list based on a specific collaborator or all  
 if collaboration\_details["collaborator"]:  
 for group in self.lastcreatedgroup:  
 if group["groupname"] == collaboration\_details["collaborator"]:  
 if collaboration\_details["add\_to\_all\_collaborators"]:  
 self.all\_collaborators.append(group["groupname"])  
 if collaboration\_details["add\_to\_all\_collaborators"] and collaboration\_details["modified\_permision"].lower() == 'viewer' \  
 or collaboration\_details["add\_to\_all\_collaborators"] == False:  
 self.remaining\_collaborators.append(group["groupname"])  
 break  
 else:  
 if collaboration\_details["add\_to\_all\_collaborators"]:  
 self.all\_collaborators.append(group["groupname"] for group in self.lastcreatedgroup)  
 if collaboration\_details["add\_to\_all\_collaborators"] and collaboration\_details["modified\_permision"].lower() == 'viewer' \  
 or collaboration\_details["add\_to\_all\_collaborators"] == False:  
 self.remaining\_collaborators.append(group["groupname"] for group in self.lastcreatedgroup)  
  
 picker\_input = list()  
 collaborated\_group\_emails = list()  
 group\_attributes = list()  
 group\_attibute = {'role': "", 'email': ""}  
  
 # Create the permission request payload based on a specific collaborator or all  
 for group in self.lastcreatedgroup:  
 if collaboration\_details["collaborator"] and collaboration\_details["collaborator"] == group["email"]:  
 if newApi is not None and newApi:  
 picker\_key = "{\"Key\": \"" + group["fid"] + "\"," + \  
 "\"DisplayText\": \"" + group['apiDisplayText'] + "\"," + \  
 "\"IsResolved\": true," + \  
 "\"Description\":\"" + group["groupname"] + "\"," + \  
 "\"EntityType\":\"SecGroup\"," + \  
 "\"EntityData\":{\"Email\":\"" + group["email"] + "\"," + \  
 "\"DisplayName\":\"" + group["apiDisplayName"] + "\"}," + \  
 "\"MultipleMatches\":[]," + \  
 "\"ProviderName\":\"FederatedDirectoryClaimProvider\"," + \  
 "\"ProviderDisplayName\":\"Federated Directory\"}"  
 else:  
 picker\_key = "{\"Key\": \"" + group["email"] + "\"}"  
 picker\_input.append(picker\_key)  
 collaborated\_group\_emails.append(group["email"])  
 group\_attibute['role'] = user\_attr['role']  
 group\_attibute['email'] = group['email']  
 group\_attributes.append(group\_attibute)  
 break  
 else:  
 if newApi is not None and newApi:  
 picker\_key = "{\"Key\": \"" + group["fid"] + "\"," + \  
 "\"DisplayText\": \"" + group['apiDisplayText'] + "\"," + \  
 "\"IsResolved\": true," + \  
 "\"Description\":\"" + group["groupname"] + "\"," + \  
 "\"EntityType\":\"SecGroup\"," + \  
 "\"EntityData\":{\"Email\":\"" + group["email"] + "\"," + \  
 "\"DisplayName\":\"" + group["apiDisplayName"] + "\"}," + \  
 "\"MultipleMatches\":[]," + \  
 "\"ProviderName\":\"FederatedDirectoryClaimProvider\","+ \  
 "\"ProviderDisplayName\":\"Federated Directory\"}"  
  
 else:  
 picker\_key = "{\"Key\": \"" + group["email"] + "\"}"  
 picker\_input.append(picker\_key)  
 collaborated\_group\_emails.append(group["email"])  
 group\_attibute['role'] = user\_attr['role']  
 group\_attibute['email'] = group['email']  
 group\_attributes.append(group\_attibute)  
  
 peoplePicker = '[' + ', '.join(picker\_input) + ']'  
 data = {  
 "emailBody": None,  
 "includeAnonymousLinkInEmail": False,  
 "sendEmail": True,  
 "propagateAcl": True,  
 "useSimplifiedRoles": True,  
 "roleValue": user\_attr["role"],  
 "peoplePickerInput": peoplePicker  
 }  
 logger.debug("Users to collaborate with :" + str(self.all\_collaborators))  
 logger.debug("Data is " + str(data))  
 result = list()  
 for n in range(1, 4):  
 logger.info("Trying to add permission to the file... iteration %s of 3" % n)  
 headers = {"X-RequestDigest": self.request\_digest, "Accept": "application/json",  
 "Content-Type": "application/json"}  
 response\_add\_permission = requests.post(url=endpoint\_sharing, headers=headers,  
 data=json.dumps(data), cookies=self.cookies,  
 timeout=self.stuck\_timeout)  
 logger.debug("The response action is ",response\_add\_permission)  
 if response\_add\_permission.status\_code == 200:  
 logger.debug("Response of setting permission for folder is " + str(response\_add\_permission.text))  
 invitedUsers = response\_add\_permission.json()["UniquelyPermissionedUsers"]  
 for invitedUser in invitedUsers:  
 logger.debug(invitedUser["Email"])  
 if invitedUser["Email"] in collaborated\_group\_emails or invitedUser['User'] in collaborated\_group\_emails :  
 result.append(True)  
 else:  
 result.append(False)  
 break  
  
 elif response\_add\_permission.status\_code in [401, 403, 504]:  
 logger.debug("Got not success code: %s so retrying" % response\_add\_permission.status\_code)  
 logger.debug("Response for getting permission is " + str(response\_add\_permission.text))  
 self.\_refresh\_token()  
  
 if self.lastuploadedfiles:  
 if "object" not in self.lastuploadedfiles[-1]:  
 self.lastuploadedfiles[-1]["permissions\_object"]["id"] = self.collaborated\_object  
 self.lastuploadedfiles[-1]["permissions\_object"]["permissions\_list"] = [group\_attributes]  
 else:  
 self.lastuploadedfiles[-1]["permissions\_object"]["permissions\_list"] = [group\_attributes]  
 else:  
 self.permission\_object = {"permissions\_object": {"id": self.collaborated\_object, "permissions\_list": [group\_attributes]}}  
 logger.debug("permission object is %s" %(self.permission\_object))  
  
 return False if len(result) == 0 else all(result)  
  
 def flexiLinkGroupCollaboration(self, user\_attr,file\_collaboration=None):  
 newApi = None  
 try:  
 newApi = BuiltIn().replace\_variables('${listApi}')  
 except:  
 pass  
  
 if newApi is not None and newApi:  
 listId, itemId = self.get\_listid\_itemId(self.collaborated\_object, file\_collaboration)  
 endpoint\_sharing = self.endpoint\_Flexilink\_bylistid % (listId,itemId)  
 else:  
 endpoint\_sharing = self.endpoint\_Flexilink.format(self.collaborated\_object)  
  
 #endpoint\_sharing = self.endpoint\_Flexilink.format(self.collaborated\_object)  
 logger.debug("URL for sharing is " + endpoint\_sharing)  
 picker\_input = list()  
 collaborated\_group\_emails = list()  
 group\_attributes = list()  
 group\_attibute = {'role': "", 'email': ""}  
 if newApi is not None and newApi:  
 for group in self.lastcreatedgroup:  
 picker\_key = "{\"Key\": \"" + group["fid"] + "\"," + \  
 "\"DisplayText\": \"" + group['apiDisplayText'] + "\"," + \  
 "\"IsResolved\": true," + \  
 "\"Description\":\"" + group["groupname"] + "\"," + \  
 "\"EntityType\":\"SecGroup\"," + \  
 "\"EntityData\":{\"Email\":\"" + group["email"] + "\"," + \  
 "\"DisplayName\":\"" + group["apiDisplayName"] + "\"}," + \  
 "\"MultipleMatches\":[]," + \  
 "\"ProviderName\":\"FederatedDirectoryClaimProvider\"," + \  
 "\"ProviderDisplayName\":\"Federated Directory\"}"  
 picker\_input.append(picker\_key)  
 collaborated\_group\_emails.append(group["email"])  
 group\_attibute['role'] = user\_attr['role']  
 group\_attibute['email'] = group['email']  
 group\_attributes.append(group\_attibute)  
  
 peoplePicker = '[' + ', '.join(picker\_input) + ']'  
 data = {  
 "request": {  
 "createLink": True,  
 "settings": {  
 "linkKind": 6,  
 "expiration": None,  
 "role": user\_attr['role'],  
 "restrictShareMembership": True,  
 "updatePassword": False,  
 "password": ""  
 },  
 "peoplePickerInput": peoplePicker,  
 }  
 }  
 else:  
 for group in self.lastcreatedgroup:  
 picker\_key = "{\"Key\": \"" + group["email"] + "\"}"  
 picker\_input.append(picker\_key)  
 collaborated\_group\_emails.append(group["email"])  
 group\_attibute['role'] = user\_attr['role']  
 group\_attibute['email'] = group['email']  
 group\_attributes.append(group\_attibute)  
  
 peoplePicker = '[' + ', '.join(picker\_input) + ']'  
 data = {  
 "request": {  
 "createLink": True,  
 "settings": {  
 "linkKind": 6,  
 "expiration": None,  
 "role": user\_attr['role'],  
 "restrictShareMembership": True,  
 "updatePassword": False,  
 "password": ""  
 },  
 "peoplePickerInput": peoplePicker,  
 "emailData": {  
 "body": "",  
 "subject": ""  
 }  
 }  
 }  
 logger.debug("Users to collaborate with :" + str(self.all\_collaborators))  
 logger.debug("Data is " + str(data))  
  
 result = list()  
 for n in range(1, 4):  
 logger.info("Trying to add permission to the file... iteration %s of 3" % n)  
 headers = {"X-RequestDigest": self.request\_digest, "Accept": "application/json",  
 "Content-Type": "application/json"}  
 response\_add\_permission = requests.post(url=endpoint\_sharing, headers=headers, data=json.dumps(data),  
 cookies=self.cookies, timeout=self.stuck\_timeout)  
 if response\_add\_permission.status\_code == 200:  
 logger.debug("Response of setting permission for resource is " + str(response\_add\_permission.text))  
 invitations = response\_add\_permission.json()['sharingLinkInfo']['Invitations']  
 for invitation in invitations:  
 if invitation['invitee']['email'] in collaborated\_group\_emails:  
 result.append(True)  
 else:  
 result.append(False)  
 break  
 elif response\_add\_permission.status\_code in [401, 403, 504]:  
 logger.debug("Got not success code: %s so retrying" % response\_add\_permission.status\_code)  
 logger.debug("Response for getting permission is " + str(response\_add\_permission.text))  
 self.\_refresh\_token()  
  
 if self.lastuploadedfiles:  
 if "object" not in self.lastuploadedfiles[-1]:  
 self.lastuploadedfiles[-1]["permissions\_object"]["id"] = self.collaborated\_object  
 self.lastuploadedfiles[-1]["permissions\_object"]["permissions\_list"] = [group\_attributes]  
 else:  
 self.lastuploadedfiles[-1]["permissions\_object"]["permissions\_list"] = [group\_attributes]  
 else:  
 self.permission\_object = {"permissions\_object": {"id": self.collaborated\_object, "permissions\_list": [group\_attributes]}}  
 logger.debug("permission object is %s" %(self.permission\_object))  
  
 return False if len(result) == 0 else all(result)  
  
 def flexiLinkUserCollaboration(self, user\_attr, collaboration\_details):  
  
 endpoint\_sharing = self.endpoint\_Flexilink.format(self.collaborated\_object)  
 logger.debug("URL for sharing is " + endpoint\_sharing)  
 role = "role:1073741830" if user\_attr["role"] == 2 else "role:1073741826"  
 collaborators = collaboration\_details['collaborators']  
 if self.multiple\_collaborators:  
 picker\_input = list()  
 for i in range(len(collaborators)):  
 picker\_key = "{\"Key\": \"" + collaborators[i] + "\"}"  
 picker\_input.append(picker\_key)  
 peoplePicker = '[' + ', '.join(picker\_input) + ']'  
 self.all\_collaborators.extend(collaborators)  
 user=collaborators  
 else:  
 user = collaborators[0]  
 peoplePicker = "[{\"Key\": \"" + user + "\"}]"  
 if collaboration\_details['add\_to\_all\_collaborators']:  
 self.all\_collaborators.append(user)  
  
 if collaboration\_details['add\_to\_all\_collaborators'] and collaboration\_details['modified\_permision'].lower() == 'viewer' \  
 or collaboration\_details['add\_to\_all\_collaborators'] == False:  
 self.remaining\_collaborators.append(user)  
 data = {  
 "request": {  
 "createLink": True,  
 "settings": {  
 "linkKind": 6,  
 "expiration": None,  
 "role": user\_attr['role'],  
 "restrictShareMembership": True,  
 "updatePassword": False,  
 "password": ""  
 },  
 "peoplePickerInput": peoplePicker,  
 "emailData": {  
 "body": "",  
 "subject": ""  
 }  
 }  
 }  
 logger.debug("Users to collaborate with :" + str(self.all\_collaborators))  
 logger.debug("Data is " + str(data))  
 user\_attr["email"] = user  
 result = list()  
 for n in range(1, 4):  
 logger.info("Trying to add permission to the file... iteration %s of 3" % n)  
 headers = {"X-RequestDigest": self.request\_digest, "Accept": "application/json",  
 "Content-Type": "application/json"}  
 response\_add\_permission = requests.post(url=endpoint\_sharing, headers=headers, data=json.dumps(data),  
 cookies=self.cookies, timeout=self.stuck\_timeout)  
 if response\_add\_permission.status\_code == 200:  
 logger.debug("Response of setting permission for resource is " + str(response\_add\_permission.text))  
 invitations = response\_add\_permission.json()['sharingLinkInfo']['Invitations']  
 for invitation in invitations:  
 if invitation['invitee']['email'] in collaborators:  
 result.append(True)  
 else:  
 result.append(False)  
 break  
 elif response\_add\_permission.status\_code in [401, 403, 504]:  
 logger.debug("Got not success code: %s so retrying" % response\_add\_permission.status\_code)  
 logger.debug("Response for getting permission is " + str(response\_add\_permission.text))  
 self.\_refresh\_token()  
  
 if self.lastuploadedfiles:  
 if "object" not in self.lastuploadedfiles[-1]:  
 self.lastuploadedfiles[-1]["permissions\_object"]["id"] = self.collaborated\_object  
 self.lastuploadedfiles[-1]["permissions\_object"]["permissions\_list"] = [user\_attr]  
 else:  
 self.lastuploadedfiles[-1]["permissions\_object"]["permissions\_list"] = [user\_attr]  
 else:  
 self.permission\_object = {"permissions\_object": {"id": self.collaborated\_object, "permissions\_list": [user\_attr]}}  
 logger.debug("permission object is %s" %(self.permission\_object))  
  
 return False if len(result) == 0 else all(result)  
  
 @keyword("In ${SERVICE} add specific members ${users\_ids} to ${group\_id}")  
 def add\_members(self, usersIds, group\_id):  
 ids\_to\_add\_to\_group = []  
 ids\_to\_add\_to\_group.extend(usersIds)  
 logger.debug(ids\_to\_add\_to\_group)  
 headers = {"Authorization": "Bearer " + self.access\_token\_graph, "Content-type": "application/json"}  
 logger.debug(group\_id)  
 endpoint\_group\_member\_url = self.endpoint\_groups\_url + str(group\_id)  
 members\_list = []  
 for i in range(len(ids\_to\_add\_to\_group)):  
 members\_list.append("https://graph.microsoft.com/v1.0/directoryObjects/" + str(ids\_to\_add\_to\_group[i]))  
 data = {"members@odata.bind": members\_list}  
 try:  
 response\_add\_members = requests.patch(url=endpoint\_group\_member\_url, headers=headers,  
 data=json.dumps(data))  
 logger.debug(response\_add\_members.content)  
 return True  
 except Exception as e:  
 logger.error(" ======member not added====== " + str(e))  
 raise Exception  
 return False  
  
 @keyword("In ${SERVICE} get ${domain\_name} domain ${count} users")  
 def get\_same\_domain\_external\_users(self, domain, number):  
 self.same\_domain\_external\_users = []  
 headers = {"Authorization": "Bearer " + self.access\_token\_graph, "Accept": "application/json"}  
 self.response\_get\_all\_users = requests.get(url=self.endpoint\_users, headers=headers)  
 if self.response\_get\_all\_users.status\_code == 200:  
 for user in self.response\_get\_all\_users.json().get("value"):  
 logger.debug(user)  
 # if "ext" in user.get("userPrincipalName").lower():  
 if len(self.same\_domain\_external\_users) < int(number):  
 if domain in user.get("userPrincipalName").lower():  
 self.same\_domain\_external\_users.append(user.get("mail"))  
 else:  
 continue  
 member\_ids\_to\_add = self.\_get\_user\_ids(self.same\_domain\_external\_users)  
 return member\_ids\_to\_add  
  
 @keyword("In ${SERVICE} get o365 group ${value}")  
 def get\_members\_from\_group(self, value):  
 logger.debug(self.lastcreatedgroup)  
 return self.lastcreatedgroup[0][value]  
  
 def directAccessUserCollaboration(self, user\_attr, file\_collaboration):  
  
 endpoint\_sharing = self.endpoint\_File\_DirectAccessSharing if file\_collaboration \  
 else self.endpoint\_Folder\_DirectAccessSharing  
 endpoint\_sharing = endpoint\_sharing % self.collaborated\_object  
 logger.debug("URL for sharing is " + endpoint\_sharing)  
  
 user\_attr["role"] = "role:1073741827" if user\_attr["role"] == 2 else "role:1073741826"  
 peoplePicker = '[{"Key": "' + user\_attr["email"] + '"}]'  
 data = {  
 "emailBody": None,  
 "includeAnonymousLinkInEmail": False,  
 "sendEmail": True,  
 "propagateAcl": True,  
 "useSimplifiedRoles": True,  
 "roleValue": user\_attr["role"],  
 "peoplePickerInput": peoplePicker  
 }  
 logger.debug("Users to collaborate with :" + user\_attr["email"])  
 logger.debug("Data is " + str(data))  
  
 result = list()  
 for n in range(1, 4):  
 logger.info("Trying to add permission to the file... iteration %s of 3" % n)  
 headers = {"X-RequestDigest": self.request\_digest, "Accept": "application/json",  
 "Content-Type": "application/json"}  
 response\_add\_permission = requests.post(url=endpoint\_sharing, headers=headers,  
 data=json.dumps(data), cookies=self.cookies,  
 timeout=self.stuck\_timeout)  
 if response\_add\_permission.status\_code == 200:  
 logger.debug("Response of setting permission for folder is " + str(response\_add\_permission.text))  
 invitedUsers = response\_add\_permission.json()["UniquelyPermissionedUsers"]  
 for invitedUser in invitedUsers:  
 if invitedUser["Email"] in [user\_attr["email"]]:  
 result.append(True)  
 else:  
 result.append(False)  
 break  
  
 elif response\_add\_permission.status\_code in [401, 403, 504]:  
 logger.debug("Got not success code: %s so retrying" % response\_add\_permission.status\_code)  
 logger.debug("Response for getting permission is " + str(response\_add\_permission.text))  
 self.\_refresh\_token()  
  
 if self.lastuploadedfiles:  
 if "object" not in self.lastuploadedfiles[-1]:  
 self.lastuploadedfiles[-1]["permissions\_object"]["id"] = self.collaborated\_object  
 self.lastuploadedfiles[-1]["permissions\_object"]["permissions\_list"] = [user\_attr]  
 else:  
 self.lastuploadedfiles[-1]["permissions\_object"]["permissions\_list"] = [user\_attr]  
 else:  
 self.permission\_object = {"permissions\_object": {"id": self.collaborated\_object, "permissions\_list": [user\_attr]}}  
 logger.debug("permission object is %s" %(self.permission\_object))  
  
 return False if len(result) == 0 else all(result)  
  
  
 @keyword("Compare permissions in ${SERVICE} ${before} and ${after}")  
 def compare\_permissions(self,permissions\_before,permissions\_after):  
 logger.debug ("permissions before " + str(permissions\_before))  
 logger.debug("permissions after " + str(permissions\_after))  
 from skybot.lib.ObjectPathHelper import object\_path\_helper  
 collaborators\_before = object\_path\_helper.get\_object\_path\_value(permissions\_before, "$..\*.email")  
 collaborators\_after = object\_path\_helper.get\_object\_path\_value(permissions\_after, "$..\*.email")  
 logger.debug(collaborators\_after,collaborators\_before)  
 if collaborators\_before == collaborators\_after:  
 logger.debug("Collaborators have been restored")  
 return True  
 else:  
 logger.error("Collaborators not restored successfully")  
 return False  
  
 def edit\_permission(self, permission\_id, role):  
 *"""  
 Edit the permission by recognizing it with permission ID . Can only update role  
 :param permission\_id: permission ID to edit  
 :param role: role of the email to change to . Available options are :-  
 # editor, viewer, previewer, uploader, previewer uploader, viewer uploader, co-owner, or owner  
 :return: Boolean  
 """* return NotImplementedError  
  
 def delete\_permission(self, permission\_id):  
 *"""  
 Delete permission using Permission Id  
  
 Args:  
 permission\_id: Permission Id  
  
 Returns:  
 Boolean  
  
 Raises:  
 None  
 """* return NotImplementedError  
  
 # Link methods  
 @keyword("generate link in ${SERVICE} for last uploaded ${object}")  
 @timeout\_decorator.timeout(90, use\_signals=False)  
 def create\_link(self, object=object):  
 *"""  
 Create the link for an object. yet to be implemented  
  
 Args:  
 object: File or Folder  
 object\_id: Object id for which link to be created  
 password (Optional): password to be set  
 expiration (Optional): expiration to be set  
 direct ( Optional): Boolean specifies it is a direct link or not  
 link\_type ( Optional): edit or view # specific to sharepoint or onedrive  
 Returns:  
 Link Id or link that gets generated  
  
 Raises:  
 None  
 """* return self.\_create\_link(link\_type='external',object=object,role='editor',allowFileDiscovery=None)  
  
 @keyword("For ${service} generate ${link\_type} link for ${object} having ${role} with ${allowFileDiscovery}")  
 def internally\_shared\_link\_file(self,link\_type,object,role,allowFileDiscovery):  
 logger.debug("1. Creating a %s shared link for last uploaded object %s with role %s" % (link\_type,object,role))  
 return self.\_create\_link(link\_type,object,role,allowFileDiscovery)  
  
 def \_create\_link(self,link\_type,object,role,allowFileDiscovery, object\_id=None):  
  
 link\_url = None  
 link\_endpoint=''  
 data = {}  
 if object\_id is None:  
 if object == "file":  
 object\_id = self.lastuploadedfiles[-1]["fileid"]  
 elif object == "folder":  
 object\_id = self.mostrecentfolder  
 headers = {"X-RequestDigest": self.request\_digest, "Accept": "application/json"}  
 headers.update({"Content-Type": "application/json"})  
 if link\_type=='internal':  
 link\_endpoint = 'SP.Web.CreateOrganizationSharingLink'  
 elif link\_type=='external':  
 link\_endpoint= 'SP.Web.CreateAnonymousLink'  
 endpoint\_create\_link = self.endpoint\_create\_link + link\_endpoint  
 if role == 'editor' or role == 'any':  
 data = {"url": self.domain\_url + object\_id, "isEditLink": True}  
 elif role == 'viewer':  
 data = {"url": self.domain\_url + object\_id, "isEditLink": False}  
 response\_create\_link = requests.post(url=endpoint\_create\_link, headers=headers, data=json.dumps(data), cookies=self.cookies, timeout=self.stuck\_timeout)  
 if isinstance(response\_create\_link, dict):  
 if response\_create\_link.get('status\_code') == 504:  
 logger.warn('request timed out for {0} after {1} sec, however we will return True '  
 'assuming, post request is successful!!'.format("create\_link", self.stuck\_timeout))  
 return True  
 else:  
 logger.debug("The response is %s" % (response\_create\_link.json()))  
 if response\_create\_link.status\_code == 200:  
 logger.debug("Response from generating Link is " + response\_create\_link.text)  
  
 elif response\_create\_link.status\_code in [401,403]:  
 logger.debug("Got a 403 error..")  
 self.\_refresh\_token()  
 raise Exception  
 elif (re.search("Blocked by policy response",str(response\_create\_link.json()["odata.error"]["message"]["value"]))):  
 logger.console("Permission is not set due to block permissions rule")  
 result=True  
  
 elif (re.search("Please review the file for sensitive or confidential content",str(response\_create\_link.json()["odata.error"]["message"]["value"]))):  
 logger.console("Permission is not set due to block permissions rule")  
 result=True  
  
 if "value" in response\_create\_link.json():  
 link\_url = response\_create\_link.json()["value"]  
 logger.debug("Link is generated with the link\_url: " + str(link\_url))  
  
 return link\_url  
  
 def update\_link(self, link\_id, active, expiration, password):  
 *"""  
 Update the link with link id. yet to be implemented  
  
 Args:  
 link\_id: link to be updated  
 active (Optional): Boolean  
 expiration (Optional): expiration to be set  
 password (Optional): password to be set  
  
 Returns:  
 Link Id or link that gets generated  
  
 Raises:  
 None  
 """* return NotImplementedError  
  
 def retrieve\_link(self, link\_id, object\_id):  
 *"""  
 Retrieve the link by link id. yet to be implemented  
  
 Args:  
 link\_id: link id of link to be retrieved  
  
 Returns:  
 Link URl  
  
 Raises:  
 None  
 """* return NotImplementedError  
  
 def delete\_link(self, link\_id):  
 *"""  
 Delete the link by link id. yet to be implemented  
  
 Args:  
 link\_id: link id to be deleted  
  
 Returns:  
 Boolean  
  
 Raises:  
 None  
 """* return NotImplementedError  
  
 def retry\_handler(func):  
 def wrapper(instance, \*args, \*\*kwargs):  
 result = None  
 try:  
 result = func(instance, \*args, \*\*kwargs)  
 if result == False:  
 instance.\_refresh\_token()  
 result = func(instance, \*args, \*\*kwargs)  
 except:  
 instance.\_refresh\_token()  
 result = func(instance, \*args, \*\*kwargs)  
  
 return result  
  
 return wrapper  
  
 def get\_all\_links(self, object\_id, type="file", link=None):  
 *"""  
 Get all the links for an object  
 :param object\_id:  
 :param type: file or folder  
 :return: link details  
 LinkKind - 1 - restricted link 2 - view link with sign in 3  
 edit link with sign in 4 - view link with no sign in 5 - edit link with no sign in  
 """* linkKind\_dict = {1: "restricted\_link", 2: "view\_link\_with\_sign\_in", 3: "edit\_link\_with\_sign\_in",  
 4: "view\_link\_with\_no\_sign\_in", 5: "edit\_link\_with\_no\_sign\_in"}  
 logger.debug("Going to retrieve all links associated with object: " + str(object\_id))  
 # headers = self.headers.copy()  
 headers = {"X-RequestDigest": self.request\_digest, "Accept": "application/json"}  
 headers.update({"Content-Type": "application/json"})  
 response\_object\_json = None  
 if type == "file":  
 response\_object\_json = self.get\_file\_info(object\_id, list\_item\_all\_fields="/ListItemAllFields")  
 elif type == "folder" or type == "nested folder":  
 response\_object\_json = self.get\_folder\_info(object\_id, params="/ListItemAllFields")  
 editlink = response\_object\_json["odata.editLink"]  
 editlink\_re = re.match(r'Web/Lists\(guid\'(.\*)\'\)/Items\((.\*)\)', editlink)  
 parent\_guid = editlink\_re.group(1)  
 file\_id = editlink\_re.group(2)  
 logger.debug("parent GUID is: " + parent\_guid + "& file ID is: " + file\_id)  
 sharing\_info\_file\_handler = open(WS+ "cfg/Office365/GetListItemSharingInfo.xml")  
 sharing\_info\_file\_content = str(sharing\_info\_file\_handler.read())  
 sharing\_info\_file\_content = sharing\_info\_file\_content.replace("{Guid}","{"+parent\_guid+"}")  
 sharing\_info\_file\_content = sharing\_info\_file\_content.replace("{Int32}",file\_id)  
  
 if link == 'flexilink':  
 url = self.domain\_url + object\_id  
 retrieve\_flexilink\_url = eval('self.endpoint\_retrieve\_flexilink % url')  
 response\_get\_all\_links = requests.post(retrieve\_flexilink\_url, headers=headers, cookies=self.cookies)  
 else:  
 response\_get\_all\_links = requests.post(url=self.endpoint\_retrieve\_links, data=str(sharing\_info\_file\_content), headers=headers, cookies=self.cookies)  
  
 if response\_get\_all\_links.status\_code == 200:  
 logger.info("Response for retrieve links is " + str(response\_get\_all\_links.text))  
  
 elif response\_get\_all\_links.status\_code in [401, 403]:  
 logger.debug("Got a 403 error..")  
 self.\_refresh\_token()  
 else:  
 logger.debug("Failed to get response of get all links: " + str(response\_get\_all\_links.json()))  
 raise Exception  
  
 logger.debug("Response of get all links is: " + str(response\_get\_all\_links.json()))  
 return response\_get\_all\_links.json()  
  
 @keyword("In ${SERVICE} get ${domain\_name} domain ${count} users")  
 def get\_same\_domain\_external\_users(self, domain, number):  
 self.same\_domain\_external\_users = []  
 headers = {"Authorization": "Bearer " + self.access\_token\_graph, "Accept": "application/json"}  
 self.response\_get\_all\_users = requests.get(url=self.endpoint\_users, headers=headers)  
 if self.response\_get\_all\_users.status\_code == 200:  
 for user in self.response\_get\_all\_users.json().get("value"):  
 logger.debug(user)  
 #if "ext" in user.get("userPrincipalName").lower():  
 if len(self.same\_domain\_external\_users) < int(number):  
 if domain in user.get("userPrincipalName").lower():  
 self.same\_domain\_external\_users.append(user.get("mail"))  
 else:  
 continue  
 member\_ids\_to\_add = self.\_get\_user\_ids(self.same\_domain\_external\_users)  
 return member\_ids\_to\_add  
  
 # User related methods  
 @retry\_handler  
 @exclude\_external\_users  
 def get\_all\_users(self):  
 *"""  
 Get list of current users info whose access token is fetched from Skyhigh console  
  
 Args:  
 None  
  
 Returns:  
 List of users  
  
 Raises:  
 None  
 """* logger.debug("Inside get all users")  
 users = []  
 self.external\_users = []  
 headers = {"Authorization": "Bearer " + self.access\_token\_graph, "Accept": "application/json"}  
 self.response\_get\_all\_users = requests.get(url=self.endpoint\_users, headers=headers)  
 if self.response\_get\_all\_users.status\_code == 200:  
 for user in self.response\_get\_all\_users.json().get("value"):  
 if user is None or user == 'None':  
 logger.debug("User is none.Not appending in the list of user ")  
 elif "ext" in user.get("userPrincipalName").lower():  
 # user is external  
 self.external\_users.append(user.get("mail"))  
 users.append(str(user.get("mail")))  
 else:  
 # user is internal  
 users.append(str(user.get("mail")))  
 elif self.response\_get\_all\_users.status\_code in [403,401]:  
 logger.debug("Got a %s error.." %(self.response\_get\_all\_users.status\_code))  
 self.\_refresh\_token()  
 raise Exception  
 else:  
 logger.debug("Error in getting members info " + self.response\_get\_all\_users.text)  
 return users  
  
 # Miscelleneous methods  
 @keyword("check if file is tombstoned in")  
 @trackme  
 def isFileTombstoned(self, filename=None, template=None, HTMLWrap=False):  
 *"""  
 Check whether file is tombstoned or not by specifying filename. ( Current logic required to be enhanced )  
 we cannot use file id because DLP updates the tombstoned file and we don't have a reference of file\_id  
 To verify if file is tombstoned, we are checking AppEditor field. Currently, we are not able to distinguish  
 if file is Delete tombstoned or Quarantine tombstoned. yet to implement  
 Args:  
 filename: original filename for which required to be checked if tombstone file is generated ?  
 template: check whether the placeholder file has template of delete or qurantine  
 Returns:  
 Boolean  
 Raises:  
 None  
 """* logger.debug("Inside OneDrive.isFileTombstoned: " + str(self.lastuploadedfiles))  
 if template.lower() not in ['drm\_protected', 'seclore\_drm\_protected']:  
 ext = ".pdf"  
 elif template.lower() == "seclore\_drm\_protected" and HTMLWrap:  
 ext = ".html"  
 else:  
 ext = ""  
 folder\_id = self.mostrecentfolder  
 file\_to\_check = None  
 result = False  
 if filename is None:  
 file\_to\_check = self.lastuploadedfiles[-1]  
 else:  
 for each in self.lastuploadedfiles:  
 id, name = each.get("fileid"), each.get("filename")  
 if name == filename:  
 file\_to\_check = each  
  
 if file\_to\_check:  
 fileid, filename = file\_to\_check.get("fileid"), file\_to\_check.get("filename")  
  
 logger.debug("Inside IsFileTombstoned, checking for " + filename +" " + str(template))  
  
  
 if (fileid is None) or (filename is None) or (folder\_id is None):  
 logger.error("Found a None value", str(fileid)+str(filename)+str(folder\_id))  
 return False  
 if not filename:  
 logger.error("File is not known to be uploaded ", filename, self.lastuploadedfiles)  
 return False  
 for n in range(1, 20):  
 logger.info("Trying to find the tombstone file")  
 file\_id = self.find\_file\_inside\_folder\_by\_name(folder\_id, str(filename) + str(ext))  
 # if file\_id is None:  
 # logger.debug("Error in getting the folder contents")  
 if file\_id:  
 logger.info("Found the file, id is: " + str(file\_id))  
 file\_info = self.get\_file\_info(file\_id, list\_item\_all\_fields="/ListItemAllFields/FieldValuesAsText?$select=AppEditor")  
 logger.debug("App editor " + str(file\_info))  
  
 if template.lower() in ["quarantine", "delete"]:  
 app\_editor\_info = file\_info.get("AppEditor")  
 if app\_editor\_info:  
 logger.debug("Verification is completed")  
 result = True  
 break  
 elif template.lower() in ["drm\_protected"]:  
 result = self.\_validate\_drm\_action(file\_id, filename)  
 if result:  
 logger.debug("Found file as DRM Protected")  
 break  
 else:  
 logger.debug("File is not yet DRM protected...will re-verify after 10 secs")  
 time.sleep(10)  
 elif template.lower() in ["seclore\_drm\_protected"]:  
 result = self.isFileSecloreDRMProtected(file\_id, filename)  
 if result:  
 logger.debug("Found file as Seclore DRM Protected")  
 break  
 else:  
 logger.debug("File is not yet Seclore DRM protected...will re-verify after 10 secs")  
 time.sleep(10)  
  
 else:  
 logger.info("Did not find the file yet, sleeping for 10 sec - " + str(n) + "/20")  
 time.sleep(15)  
 if not result:  
 logger.error("File "+str(filename)+" is not Tombstoned in O365")  
 return False  
 return result  
  
 @keyword("verify changed permission for folder in ${Service} as role ${permission\_type}")  
 def verify\_folder\_permissions(self, permission\_type, wait\_time=60):  
 *"""  
  
 :param permission\_type:  
 :param wait\_time:  
 :return:  
 """* result = False  
 # folder\_id = self.mostrecentfolder  
 logger.debug("Inside verify\_folder\_permissions for OneDrive")  
 if self.lastuploadedfiles:  
 if self.lastuploadedfiles[-1].get("permissions\_object").get("permissions\_list"):  
 expected\_permissions\_list = self.lastuploadedfiles[-1].get("permissions\_object").get("permissions\_list")  
 # logger.console("Inside Expected Permissions List:" +str(expected\_permissions\_list))  
 else:  
 expected\_permissions\_list = self.list\_permissions(self.permission\_object.get("permissions\_object").get("id"))  
 else:  
 expected\_permissions\_list = self.list\_permissions(self.permission\_object.get("permissions\_object").get("id"))  
  
 logger.debug("Expected permission List is " + str(expected\_permissions\_list))  
 for n in range(1, 20):  
 current\_permissions\_list = self.list\_permissions(self.collaborated\_object)  
 if current\_permissions\_list is None:  
 break  
 logger.debug("Current permission list is " + str(current\_permissions\_list))  
 if not permission\_type:  
 logger.debug("Going to match None as Permission Type")  
 if len(current\_permissions\_list)==1:  
 logger.debug("Going to find if only one user is present")  
 creds = self.shutil.services  
 if current\_permissions\_list[0]['role'] == 'owner' and (current\_permissions\_list[0]['email']).split('@')[1] == (creds['allservices']['SharePoint']['users']).split('@')[1]:  
 logger.debug("internal user only. hence revoke collaboration for external users is verified")  
 return True  
 break  
 if not current\_permissions\_list:  
 logger.debug("Did not find any collaborated user, hence will return True")  
 result = True  
 break  
 else:  
 #logger.debug("Going to match for Permission Type: " + permission\_type)  
 for list in expected\_permissions\_list:  
 list["role"] = permission\_type  
 logger.debug("Expected permission list is:" + str(expected\_permissions\_list))  
 if sorted(current\_permissions\_list, key=lambda i: i['email']) == \  
 sorted(expected\_permissions\_list, key=lambda i: i['email']):  
 logger.debug("Current and Expected permission list is matched!! , returning True")  
 result = True  
 break  
 logger.debug("Verify permissions is not succeeded yet, sleeping for 10 sec - " + str(n) + "/20")  
 time.sleep(10)  
 if result is False:  
 logger.error("Verifying Permission fails for object: " + str(self.collaborated\_object))  
 return result  
  
 @keyword("Check ${Service} if ${type} ${link} expired")  
 def check\_link(self,type='', link=''):  
  
 result=self.verify\_link\_expiry(type=type,link=link)  
 return result  
  
 @keyword("verify whether in ${Service} the ${type} link is expired or not")  
 def verify\_link\_expiry(self, type='', link='', expired=True):  
 link\_kind\_dict = { 'internal\_view':2,  
 'internal\_edit':3,  
 'external\_view':4,  
 'external\_edit':5,  
 'internal\_any':3,  
 'external\_any':5  
 }  
 object\_id = []  
  
 logger.debug("Object\_id passed is " + type)  
 if str(type) == "file":  
 logger.debug("Matched")  
 logger.debug(self.lastuploadedfiles)  
 object\_id.append(self.lastuploadedfiles[-1]["fileid"])  
 elif str(type) == "folder":  
 object\_id.append(self.mostrecentfolder)  
 elif str(type) == "nested folder":  
 object\_id = self.nestedfolders  
 for obj in object\_id:  
 logger.debug("Going to verify if link is expired on a object: " + str(obj))  
 for n in range(1, 20):  
 '''  
 In case of collaboration with the external user who is not part of O365 users, O365 creates  
 the collaboration as a shared link. So I have added the code to verify link expiry in this method  
 instead of adding in verify\_folder\_permissions method  
 '''  
 if str(link) == 'flexilink':  
 all\_links\_json = self.get\_all\_links(obj, type, link)  
 else:  
 all\_links\_json = self.get\_all\_links(obj, type)  
 result=[]  
 if str(link) in list(link\_kind\_dict.keys()):  
 link\_kind\_id=link\_kind\_dict.get(link)  
  
 for linktype in all\_links\_json[-1]["SharingLinks"]:  
 if linktype["LinkKind"] == link\_kind\_id:  
 if linktype["IsActive"] == False:  
 logger.debug("%s type link has expired" %(link))  
 result.append(True)  
 else:  
 result.append(False)  
 logger.debug("%s type link has not expired" %(link))  
 else:  
 try:  
 for linktype in all\_links\_json[-1]["SharingLinks"]:  
 if linktype["LinkKind"] == 4 or linktype["LinkKind"] == 5:  
 logger.debug("Matched" + str(linktype))  
 if linktype["IsActive"] == False:  
 result.append(True)  
 else:  
 result.append(False)  
 except:  
 logger.debug("This is for flexilink sharing")  
 for linktype in all\_links\_json["value"]:  
 '''flexi link has linkkind 6 so checking for exact type'''  
 # if 4 <= linktype["linkDetails"]["LinkK ind"] <= 6:  
 if linktype["linkDetails"]["LinkKind"] == 6:  
 logger.debug("Matched" + str(linktype["linkDetails"]))  
 # if linktype["linkDetails"]["IsActive"] == False:  
 # result.append(True)  
 # else:  
 # result.append(False)  
 invitees = [invitation['invitee']['email'] for invitation in linktype["linkDetails"]["Invitations"]]  
 for user in self.all\_collaborators:  
 result.append(True) if user not in invitees else result.append(False)  
  
 logger.debug("Result of verifying links is: " + str(result))  
 if all(result) == True:  
 break  
 else:  
 logger.debug("Verify links is not succeeded yet, sleeping for 10 sec - " + str(n) + "/20")  
 time.sleep(10)  
 return all(result)  
  
 def \_get\_domain\_name(self):  
 domain = re.match(r'.\*@(.\*)\.onmicrosoft\.com', self.user)  
 if domain:  
 self.domain\_name = domain.group(1)  
 logger.debug("Sharepoint domain name is: " + self.domain\_name)  
 elif self.access\_token\_graph is not None:  
 logger.debug("Cannot get Sharepoint domain name from username: " + self.user)  
 logger.info("Getting the Sharepoint domain name from MS Graph")  
 header = {"Authorization": "Bearer " + self.access\_token\_graph}  
 url = "https://graph.microsoft.com/v1.0/users/" + self.user + "/drive/root"  
 try:  
 r = requests.get(url, headers=header)  
 Utils.request\_trace(r)  
 onedriveurl = r.json().get("webUrl")  
 logger.debug("webUrl from Graph: " + str(onedriveurl))  
 domain = re.match(r'https://(.+)-my\.sharepoint\.com', onedriveurl)  
 self.domain\_name = domain.group(1)  
 except Exception as e:  
 logger.error("Exception while extracting domain name from Graph response " + str(e))  
  
 if self.domain\_name is None:  
 logger.error("Could not get Sharepoint domain name")  
 return False  
  
 logger.debug("Sharepoint and OneDrive domain name is " + self.domain\_name)  
 return True  
  
 def \_get\_endpoints(self):  
 *"""  
 Going to build endpoints based on the email/domain  
 :return: None  
 """* logger.debug("Building O365 endpoints...")  
  
 self.user\_flat = self.user.replace("@", ".").replace(".", "\_")  
 self.domain\_url = "https://" + self.domain\_name + "-my.sharepoint.com"  
 self.domain\_admin\_url = "https://" + self.domain\_name + "-admin.sharepoint.com"  
 self.root\_folder = "/personal/" + self.user\_flat + "/" + self.list\_library\_name  
 self.endpoint\_GetFolderByServerRelativeUrl = self.domain\_url + "/personal/" + self.user\_flat + \  
 "/\_api/Web/GetFolderByServerRelativeUrl(\'" + self.root\_folder + "\')"  
 self.endpoint\_GetFileByServerRelativeUrl = self.domain\_url + "/personal/" + self.user\_flat + \  
 "/\_api/Web/GetFileByServerRelativeUrl(\'" + self.root\_folder + "\')"  
 self.endpoint\_GetFileListByServerRelativePathUrl = self.domain\_url + "/personal/" + self.user\_flat + \  
 "/\_api/web/GetFileByServerRelativePath(decodedurl=@relativeUrl)" + \  
 "/ListItemAllFields?@relativeUrl='%s'"  
 self.endpoint\_GetFolderListByServerRelativePathUrl = self.domain\_url + "/personal/" + self.user\_flat + \  
 "/\_api/web/GetFolderByServerRelativePath(decodedurl=@relativeUrl)" + \  
 "/ListItemAllFields?@relativeUrl='%s'"  
 self.endpoint\_users = "https://graph.microsoft.com/v1.0/" + self.domain\_name + ".onmicrosoft.com" + "/users"  
 self.endpoint\_retrieve\_links = "https://" + self.domain\_name + "-my.sharepoint.com" + "/personal/" + \  
 self.user\_flat + "/\_vti\_bin/client.svc/ProcessQuery"  
 self.endpoint\_contextinfo = self.domain\_url + '/personal/' + self.user\_flat + '/\_api/contextinfo'  
 self.endpoint\_create\_field = self.domain\_url + '/personal/' + self.user\_flat + \  
 '/\_api/web/lists/getbytitle(\'Documents\')/Fields'  
 # self.endpoint\_update\_field = self.domain\_url + '/personal/' + self.user\_flat + \  
 # '/\_api/web/lists/getbytitle(\'Documents\')/items(2)'  
 self.endpoint\_groups = "https://graph.microsoft.com/v1.0/" + self.domain\_name + ".onmicrosoft.com" + "/groups"  
 self.endpoint\_groups\_url = "https://graph.microsoft.com/v1.0/groups/"  
 self.endpoint\_create\_link = self.domain\_url + "/personal/" + self.user\_flat + "/\_api/"  
 self.endpoint\_retrieve\_flexilink = self.domain\_url + "/personal/" + self.user\_flat + \  
 "/\_api/web/getlistitem(@url)/getsharinginformation/permissionsInformation/links?@url='%s'"  
 self.endpoint\_host\_web\_url = self.domain\_url + "/personal/" + self.user\_flat  
 self.endpoint\_list = self.domain\_url + "/\_api/SP.AppContextSite(@target)/web/Lists"  
 self.endpoint\_create\_list = self.endpoint\_list + "?@target='" + self.endpoint\_host\_web\_url + "'"  
 self.endpoint\_GetFileByServerRelativePath = self.domain\_url + '/personal/' + self.user\_flat + \  
 "/\_api/web/GetFileByServerRelativePath(decodedurl=@relativeUrl)/$value?@relativeUrl=" + \  
 "'/personal/" + self.user\_flat  
 self.default\_root\_folder = "/Documents/"  
 self.endpoint\_DirectAccessSharing = self.domain\_url + "/personal/" + self.user\_flat + \  
 "/\_api/SP.Sharing.DocumentSharingManager.UpdateDocumentSharingInfo"  
 self.endpoint\_Flexilink = self.domain\_url + '/personal/' + self.user\_flat + \  
 "/\_api/web/GetListItemUsingPath(decodedurl=@u)/ShareLink?@u='{0}'"  
 self.endpoint\_Folder\_DirectAccessSharing = self.domain\_url + "/personal/" + self.user\_flat + \  
 "/\_api/web/GetFolderByServerRelativeUrl(@relativeUrl)/ListItemAllFields/ShareObject?@relativeUrl='%s'"  
 self.endpoint\_File\_DirectAccessSharing = self.domain\_url + "/personal/" + self.user\_flat + \  
 "/\_api/web/GetFileByServerRelativeUrl(@relativeUrl)/ListItemAllFields/ShareObject?@relativeUrl='%s'"  
 self.endpoint\_DirectAccessSharing\_listId = self.domain\_url + "/personal/" + self.user\_flat + "/\_api/web/Lists(@a1)/GetItemById(@a2)/ShareObject?@a1='{%s}'&@a2='%s'"  
 self.endpoint\_Flexilink\_bylistid = self.domain\_url + '/personal/' + self.user\_flat + \  
 "/\_api/web/Lists(@a1)/GetItemById(@a2)/ShareLink?@a1='{%s}'&@a2='%s'"  
  
 #@trackme  
 def \_get\_request\_digest(self):  
 *"""  
 This method is required to get request digest to be used by upload file method. Request digest is fetched by a series of steps  
 a) Build and copy xml based on the prototype xml saved in cfg/GetOfficeTokenCommand.xml  
 b) Get Office token and cookies  
 :return: Tuple of Request Digest and Cookies  
 """* # this function requires the password for ofice 365, this was hardcoded before,  
 # now read it from environment variable office365\_password or the cfg file  
 # and default to "Abcd\_1234" when not found in cfg file  
  
 if not self.shutil.current\_service:  
 servicename = "OneDrive"  
 else:  
 servicename=self.shutil.current\_service  
  
 #servicename = "onedrive"  
 #default value in case of email verification  
 for group,services in list(self.shutil.services.items()):  
 if group == 'outageservices':  
 continue  
 for service in list(services.keys()):  
 if self.instance\_id == self.shutil.services.get(group).get(service).get("instanceid"):  
 servicename=service  
 break  
  
 default\_password = "Abcd\_1234"  
 #o365servicename = self.\_\_class\_\_.\_\_name\_\_  
 o365servicename=servicename  
 if os.environ.get("office365\_password"):  
 password = os.environ.get("office365\_password")  
 logger.debug("Took "+o365servicename+" password from environment variable office365\_password")  
 else:  
 try:  
 if o365servicename.lower() == "exchange":  
 password = self.shutil.services.get("mailservices").get(o365servicename).get("password", default\_password)  
 else:  
 password = self.shutil.services.get("allservices").get(o365servicename).get("password", default\_password)  
 except Exception as e:  
 logger.warn("could not get "+o365servicename+" password from config file, using hardcoded default")  
 password = default\_password  
 self.password = password  
  
 tree = ET.parse(WS+"cfg/Office365/GetOfficeTokenCommand.xml")  
 office\_token = None  
 cookies = None  
 request\_digest = None  
 for value in tree.find('.//{http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd}UsernameToken'):  
 if(value.text) == "OneDriveUserName":  
 value.text = self.user  
 if(value.text) == "OneDrivePassword":  
 value.text = password  
 for value in tree.find('.//{http://www.w3.org/2005/08/addressing}EndpointReference'):  
 if(value.text) == "OneDriveBaseUrl":  
 value.text = self.domain\_url  
 tree.write(WS+"tmp/" + self.user\_flat + "\_"+o365servicename+".xml")  
 f = open(WS+"tmp/" + self.user\_flat + "\_"+o365servicename+".xml", "r")  
 r = requests.post("https://login.microsoftonline.com/extSTS.srf", data=f.read())  
 Utils.request\_trace(r)  
 logger.info("Response from extSTS.srf for "+o365servicename+" is: " + r.text)  
  
 tree = ET.fromstring(r.content)  
  
 for value in tree.findall('.//{http://schemas.xmlsoap.org/ws/2005/02/trust}RequestedSecurityToken'):  
 token = value.find('.//{http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd}BinarySecurityToken')  
 office\_token = token.text  
 logger.info(o365servicename + " token is " + str(office\_token))  
  
 r = requests.post(self.domain\_url + '/\_forms/default.aspx?wa=wsignin1.0', data=office\_token)  
 Utils.request\_trace(r)  
 cookies = r.cookies  
 logger.info("Cookies for "+o365servicename+" is " + str(cookies))  
 r = requests.post(self.endpoint\_contextinfo, cookies=cookies)  
 Utils.request\_trace(r)  
 if r.status\_code != 200:  
 logger.error("Error authenticating to " + o365servicename + " with status: " + str(  
 r.status\_code) + " message: " + r.text)  
 return (None, None)  
 tree = ET.fromstring(r.content)  
 for value in tree.findall('.//{http://schemas.microsoft.com/ado/2007/08/dataservices}FormDigestValue'):  
 request\_digest = value.text  
 logger.info("Request Digest for "+ o365servicename +" is " + str(request\_digest))  
 return (request\_digest, cookies)  
  
 def \_get\_another\_user\_for\_collab(self):  
 users = self.get\_all\_users()  
 users.remove(str(self.user))  
 users.remove(str(self.admin))  
 return random.choice(users)  
  
 def get\_users(self, users=None):  
 logger.debug("inside get users")  
 self.members\_to\_collaborate = self.get\_users\_for\_O365Group(users)  
 """  
 # if not self.members\_to\_collaborate:  
 internal\_users\_set = set(self.get\_all\_users()) - set(self.external\_users)  
 external\_users\_set = self.external\_users  
 logger.debug("Internal users set is: " + str(internal\_users\_set))  
 logger.debug("External users set is: " + str(external\_users\_set))  
  
 if "\*" in users:  
 try:  
 if "i" not in users or "e" not in users:  
 logger.error("Please use correct syntax to select users . e.g. i1\*e1\*, due to incorrect synatx "  
 "this will use by default \*")  
 internal\_groups\_choice, external\_groups\_choice = re.match(r'i(\d)\\*e(\d)\\*', users, re.M|re.I).groups()  
 internal\_users = random.sample(internal\_users\_set, int(internal\_groups\_choice))  
 external\_users = random.sample(external\_users\_set, int(external\_groups\_choice))  
 except Exception as e:  
 logger.warn("Error in randomly picking Users" + str(e))  
  
 self.members\_to\_collaborate = ",".join(internal\_users+external\_users)  
 self.members\_to\_collaborate=self.members\_to\_collaborate.split(",")  
 logger.console("users to collaborate:" + str(self.members\_to\_collaborate))  
 # else:  
 # logger.debug("Members to collaborate already exists.. hence using " + str(self.members\_to\_collaborate))  
 """  
  
 def get\_mime\_type(self, filename):  
 *"""  
  
 :param filename: filename for which mime type requires to be returned  
 :return:  
 """* ext = os.path.splitext(filename)[1]  
 if "doc" in ext:  
 return "application/msword"  
 if "docx" in ext:  
 return "application/vnd.openxmlformats-officedocument.wordprocessingml.document"  
 if "ppt" in ext:  
 return "application/vnd.ms-powerpoint"  
 if "pptx" in ext:  
 return "application/vnd.openxmlformats-officedocument.presentationml.presentation"  
 if "xls" in ext:  
 return "application/vnd.ms-excel"  
 if "xlsx" in ext:  
 return "application/vnd.openxmlformats-officedocument.spreadsheetml.sheet"  
 if "txt" in ext:  
 return "text/plain"  
  
 @keyword("check if ${filename} is present in folder")  
 def find\_file\_inside\_folder\_by\_name(self, parent\_id, file\_name):  
 *"""  
 Private method  
 Get file id by specifying file name and parent id  
  
 Args:  
 parent\_id: Parent Id of the folder  
 file\_name: Name of the file for which file id to be found  
  
 Returns:  
 Boolean  
  
 Raises:  
 None  
 """* file\_name = str(file\_name)  
  
 if parent\_id is None:  
 parent\_id = self.mostrecentfolder  
  
 logger.debug("Going to find " + str(file\_name) + " inside folder ID: " + str(parent\_id))  
 file\_id = None  
 list\_files = self.get\_folder\_info(parent\_id, params="/Files").get("value")  
 if list\_files is None:  
 return None  
 if list\_files:  
 for file\_properties in list\_files:  
 if file\_properties["Name"] == file\_name:  
 file\_id = file\_properties["ServerRelativeUrl"]  
 return file\_id  
 if file\_id is None:  
 logger.debug("Unable to find the file: "+ str(file\_name))  
  
 return None  
  
 def cleanup(self):  
 *"""  
 going to clean onedrive account . Delete recursively all files and folders in account  
 :return:  
 """* logger.debug("Going to delete folder for user: " + self.user)  
 folder\_info\_json = self.get\_folder\_info(self.root\_folder, params="/Folders").get("value")  
 for item in folder\_info\_json:  
 # if item.get("ServerRelativeUrl", None) is None:  
 # if "Forms" in item.get("odata.id"):  
 # continue  
 # self.delete\_folder(item.get("odata.id"))  
 # else:  
 # logger.debug(item.get("ServerRelativeUrl"))  
 # if "Forms" in item.get("ServerRelativeUrl"):  
 # continue  
 # self.delete\_folder(item.get("ServerRelativeUrl"))  
 logger.debug(item.get("ServerRelativeUrl"))  
 if "Forms" in item.get("ServerRelativeUrl"):  
 continue  
 self.delete\_folder(item.get("ServerRelativeUrl"))  
 file\_info\_json = self.get\_folder\_info(self.root\_folder, params="/Files").get("value")  
 for item in file\_info\_json:  
 logger.debug(item.get("ServerRelativeUrl"))  
 self.delete\_file(item.get("ServerRelativeUrl"))  
  
 def verify\_permission\_on\_files\_inside\_folder(self, permission\_type, folder\_id=None):  
 *"""  
 To verify permission on the files present in the the folder  
 Args:  
 permission\_type: Changed permission to be verified. Ex. Editor, Viewer, None  
 folder\_id: Id of the folder  
  
 Returns:  
 Bool. Return True if verification is successfull for all the files  
  
 """* results = []  
  
 if folder\_id is None:  
 folder\_id = self.mostrecentfolder  
  
 files\_list = self.list\_all\_files\_inside\_folder(folder\_id=folder\_id)  
 files\_list = files\_list["value"]  
  
 if files\_list:  
 for files in files\_list:  
 self.collaborated\_object = files['ServerRelativeUrl']  
 result = self.verify\_folder\_permissions(permission\_type)  
 results.append(result)  
  
 return all(results)  
  
 else:  
 logger.info("No files found inside folder:: %s" % folder\_id)  
  
 def list\_all\_files\_inside\_folder(self, folder\_id):  
 *"""  
 To get all the files present in the folder  
 Args:  
 folder\_id: Id of the folder  
  
 Returns:  
 List of all the files present inside folder  
  
 """* files\_list = self.get\_folder\_info(folder\_id, params="/Files")  
  
 return files\_list  
  
 def add\_metadata\_tag(self, name, value, parent\_id=0):  
 *"""  
 To add metadata tag to last updated file  
 Args:  
 name: name of the tag to set on the file  
 value: tag value  
 """* file\_id = self.lastuploadedfiles[-1]["fileid"]  
 if file\_id is None:  
 return  
 endpoint\_GetFileByServerRelativeUrl = re.sub('GetFileByServerRelativeUrl(.\*)',  
 'GetFileByServerRelativeUrl(\'' + file\_id + '\')',  
 self.endpoint\_GetFileByServerRelativeUrl)  
  
 endpoint\_set\_metadata = endpoint\_GetFileByServerRelativeUrl + '/ListItemAllFields'  
  
 create\_field\_headers = {"X-RequestDigest": self.request\_digest, "Content-Type": 'application/json',  
 "Accept": "application/json"}  
  
 create\_field = { 'Title': name,  
 'FieldTypeKind': 2,  
 'Required': 'false',  
 'EnforceUniqueValues': 'false',  
 'StaticName': name  
 }  
  
 update\_headers = {"X-RequestDigest": self.request\_digest, "Content-Type": 'application/json',  
 "Accept": "application/json", "If-Match": "\*"}  
  
 tag\_internal\_name = self.\_get\_fields\_by\_title(name, headers=create\_field\_headers)  
  
 if tag\_internal\_name is None:  
 res\_create\_field = requests.post(self.endpoint\_create\_field, headers=create\_field\_headers, cookies=self.cookies, json=create\_field)  
 if res\_create\_field.status\_code == 201:  
 logger.info("Response for create field is " + str(res\_create\_field.text))  
 else:  
 raise Exception("Failed to create metadata tag")  
 tag\_internal\_name = self.\_get\_fields\_by\_title(name, headers=create\_field\_headers)  
 update\_data = {tag\_internal\_name: value}  
 else:  
 update\_data = {tag\_internal\_name: value}  
  
 res\_update\_field = requests.patch(endpoint\_set\_metadata, headers=update\_headers, cookies=self.cookies, json=update\_data)  
 if res\_update\_field.status\_code == 204:  
 logger.info("Response for update data is " + str(res\_update\_field.text))  
 else:  
 raise Exception("Failed to add tag to the metadata field")  
 return res\_update\_field.status\_code  
  
 def validate\_metadata\_tag(self, name, value, parent\_id=0):  
 *"""  
 To add metadata tag to last updated file  
 Args:  
 name: name of the tag to set on the file  
 value: tag value  
 """* file\_id = self.lastuploadedfiles[-1]["fileid"]  
 if file\_id is None:  
 return  
 endpoint\_GetFileByServerRelativeUrl = re.sub('GetFileByServerRelativeUrl(.\*)',  
 'GetFileByServerRelativeUrl(\'' + file\_id + '\')',  
 self.endpoint\_GetFileByServerRelativeUrl)  
  
 endpoint\_set\_metadata = endpoint\_GetFileByServerRelativeUrl + '/ListItemAllFields'  
  
 update\_headers = {'Authorization': "Bearer " + str(self.access\_token) , 'Content-Type': 'application/json',  
 'Accept': "application/json"}  
  
 #update\_data = {name: value}  
  
 res\_update\_field = requests.get(endpoint\_set\_metadata, headers=update\_headers)  
 res\_update\_field\_json = res\_update\_field.json()  
 logger.debug("response is %s and response type is %s" %(res\_update\_field,type(res\_update\_field\_json)))  
  
 if res\_update\_field.status\_code in (204,200):  
 logger.info("Response for fetching metadata is " + str(res\_update\_field.text))  
 elif res\_update\_field.status\_code in [401,403]:  
 logger.debug("Token expired, refreshing and trying again")  
 self.\_refresh\_token()  
 update\_headers['Authorization'] = "Bearer " + str(self.access\_token)  
 res\_update\_field = requests.get(endpoint\_set\_metadata, headers=update\_headers)  
 res\_update\_field\_json = res\_update\_field.json()  
 else:  
 raise Exception("Failed to fetch metadata field" + str(res\_update\_field))  
  
 for key in list(res\_update\_field\_json.keys()):  
 if res\_update\_field\_json[key]==value and key==name:  
 logger.debug("found the metadata tag %s and value %s" %(name, value))  
 return True  
 logger.console("Metadata not found")  
  
 return False  
  
 @keyword("In ${SERVICE} create o365 group with ${visibility} ${name}")  
 def create\_o365\_group(self, visibility, name, group\_type=None):  
 logger.debug("Inside create\_o365\_group" + str(name))  
 result = True  
 #name = name+".onedrive"  
 if not group\_type:  
 group\_type = ["Unified"]  
 logger.debug("type of group is === " + str(group\_type))  
 else:  
 logger.debug("type of group in else is === " + str(group\_type))  
 existing\_groups = self.get\_o365\_groups()  
 logger.debug("existing groups === " + str(existing\_groups))  
  
 for group in existing\_groups["value"]:  
 logger.debug("group is ==" + str(group))  
 if group.get("displayName") == name:  
 logger.debug("O365 groups named %s already exists " % (name))  
 self.lastcreatedgroup.append(  
 {  
 "groupid":str(group.get("id")),  
 "groupname":str(group.get("displayName")),  
 "email":str(group.get("mail")),  
 "id": str(group.get("id")),  
 'fid': "c:0o.c|federateddirectoryclaimprovider|" + str(group.get("id")),  
 'apiDisplayText': str(group.get("displayName")) + " Members",  
 'apiDisplayName': str(group.get("displayName")) + " Members"  
  
 }  
 )  
 logger.debug("last created group when group already exists == " + str(self.lastcreatedgroup))  
 return result  
  
 owner\_id = self.\_get\_user\_ids([self.admin])  
 logger.debug("===Group not found, creating new one===")  
 mail\_nick\_name=name.replace(" ","")  
 headers = {"Authorization": "Bearer " + self.access\_token\_graph, "content-type": "application/json"}  
 data={  
 "groupTypes": group\_type,  
 "displayName": name,  
 "mailNickname": mail\_nick\_name,  
 "mailEnabled": "true",  
 "securityEnabled": "false",  
 "visibility":visibility,  
 "owners@odata.bind": ["https://graph.microsoft.com/v1.0/users/" + owner\_id[0]]  
 }  
 logger.debug("endpoint is %s, headers are %s. data is %s " % (self.endpoint\_groups,headers,data))  
 #logger.debug("Data type is" + str(type(data)) )  
 #logger.debug("headers type is" + str(type(headers)) )  
 response\_create\_group = requests.post(url=self.endpoint\_groups, headers=headers, data=json.dumps(data))  
 new\_group=response\_create\_group.json()  
 logger.debug("response is " + str(new\_group))  
 if response\_create\_group.status\_code in (200,201):  
  
 self.lastcreatedgroup.append(  
 {  
 "groupid":str(new\_group.get("id")),  
 "groupname":str(new\_group.get("displayName")),  
 "email":str(new\_group.get("mail")),  
 "id": str(new\_group.get("id")),  
 'fid': "c:0o.c|federateddirectoryclaimprovider|" + str(new\_group.get("id")),  
 'apiDisplayText': str(new\_group.get("displayName")) + " Members",  
 'apiDisplayName': str(new\_group.get("displayName")) + " Members"  
  
 }  
 )  
 logger.debug("last created group when new group is created == " + str(self.lastcreatedgroup))  
 else:  
 logger.error("Group creation failed due to " + str(response\_create\_group.\_content))  
 result=False  
 return result  
  
 @retry\_handler  
 def get\_o365\_groups(self):  
 headers = {"Authorization": "Bearer " + self.access\_token\_graph, "Accept": "application/json"}  
 response\_get\_all\_groups = None  
 response\_get\_all\_groups = requests.get(url=self.endpoint\_groups, headers=headers)  
  
 if response\_get\_all\_groups.status\_code in [200,201]:  
 return response\_get\_all\_groups.json()  
 elif response\_get\_all\_groups.status\_code in [401,403]:  
 logger.debug("Refresh token and retry again")  
 raise Exception  
 else:  
 logger.debug("Failed to get the token")  
  
  
 @retry\_handler  
 def \_get\_user\_ids(self,mail):  
  
 logger.debug("Inside \_get\_user\_ids and mail value is %s" % (mail))  
 user\_id=[]  
 headers = {"Authorization": "Bearer " + self.access\_token\_graph, "Accept": "application/json"}  
 response\_get\_all\_users = requests.get(url=self.endpoint\_users, headers=headers)  
 if response\_get\_all\_users.status\_code == 200:  
 for user in response\_get\_all\_users.json().get("value"):  
 # logger.debug("user is %s" % (user))  
 if user.get("mail") in mail:  
 user\_id.append(user.get('id'))  
 # logger.debug("user id list is %s ==== " % (user\_id))  
  
 elif response\_get\_all\_users.status\_code in [403,401]:  
 logger.debug("Got a %s error.." %(response\_get\_all\_users.status\_code))  
 self.\_refresh\_token()  
 raise Exception  
 else:  
 logger.debug("Error in getting member id " + response\_get\_all\_users.text)  
 return None  
 return user\_id  
  
 @keyword("In ${SERVICE} add o365 group members")  
 def add\_members\_to\_o365\_group(self):  
 result = False  
 members\_to\_add\_to\_group = []  
 group\_id = self.lastcreatedgroup[-1]["groupid"]  
 endpoint\_group\_member\_url = self.endpoint\_groups\_url + str(group\_id) + "/members/$ref"  
 o365\_users = BuiltIn().replace\_variables('${o365\_users}')  
 group\_users = self.get\_users\_for\_O365Group(o365\_users)  
 # get all users  
 all\_users = self.get\_all\_users()  
  
 # finding diff of users from what is already selected to be part of policy  
 # setdiff = set(all\_users)-set(self.members\_to\_collaborate)  
 # setdiff = set(all\_users) - set(group\_users)  
 # randomly chosing another user to collaborate  
 # int\_mem = random.choice(list(setdiff))  
  
 # adding to the variable members\_added  
 # members\_to\_add\_to\_group.append(int\_mem)  
 members\_to\_add\_to\_group.extend(group\_users)  
 self.lastcreatedgroup[-1]["members"]=members\_to\_add\_to\_group  
  
 # for member in self.members\_to\_collaborate:  
 # members\_to\_add\_to\_group.append(member)  
  
 member\_ids\_to\_add = self.\_get\_user\_ids(members\_to\_add\_to\_group)  
 logger.debug("adding members now %s======" % (member\_ids\_to\_add))  
 headers = {"Authorization": "Bearer " + self.access\_token\_graph, "content-type": "application/json"}  
 for member\_id in member\_ids\_to\_add:  
 data = {  
 "@odata.id": "https://graph.microsoft.com/v1.0/directoryObjects/"+str(member\_id)  
 }  
 retry = 3  
 try:  
 if self.\_isMemberPresentInGroup(group\_id, member\_id):  
 result = True  
 continue  
 response\_add\_members = requests.post(url=endpoint\_group\_member\_url , headers=headers, data=json.dumps(data))  
 except Exception as e:  
 logger.error(" ======member not added====== " + str(e) )  
 raise Exception  
  
 logger.debug("response for adding member id %s is %s" % (member\_id,response\_add\_members))  
 if response\_add\_members.status\_code == 204:  
 logger.debug("member added successfully")  
 result = True  
 else:  
 if response\_add\_members.status\_code == 403:  
 self.access\_token\_graph = shutil.get\_access\_token(self.cspid, self.instance\_id,  
 resource="https://graph.microsoft.com",  
 decrypt\_token=True)  
 logger.debug("Member not added due to error, so retrying")  
 #logger.error("Member not added due to " + str(response\_add\_members.\_content))  
 while retry > 0:  
 time.sleep(30)  
 response\_add\_members = requests.post(url=endpoint\_group\_member\_url, headers=headers,  
 data=json.dumps(data))  
 if response\_add\_members.status\_code == 204:  
 logger.debug("member added successfully")  
 result = True  
 break  
 else:  
 result = False  
 retry = retry - 1  
 if result:  
 return result  
 else:  
 raise Exception  
  
 @keyword("In ${SERVICE} delete latest O365 group")  
 def delete\_o365\_group(self):  
 result = []  
 for group in self.lastcreatedgroup:  
 headers = {"Authorization": "Bearer " + self.access\_token\_graph, "content-type": "application/json"}  
 group\_delete\_endpoint = self.endpoint\_groups + "/" + str(group.get("groupid"))  
 group\_delete\_response = requests.delete(url=group\_delete\_endpoint , headers=headers)  
 if group\_delete\_response.status\_code == 204:  
 logger.debug("Deleted group " + str(group.get("groupname")))  
 result.append(True)  
 else:  
 logger.error("Could not delete group %s due to following error %s " % (group.get("name"),group\_delete\_response.\_content))  
 result.append(False)  
 logger.debug("No group to delete")  
 return all(result)  
  
 @keyword("In ${SERVICE} ${enable} ${feature} feature")  
 def enable\_shared\_link\_feature(self, enable, feature):  
  
 self.shared\_link\_url = self.watchtower\_url + "/v1/" + str(self.shutil.gettenantid()) + "/config/" +str(feature)  
  
  
 if enable == 'enable':  
 logger.debug("Going to enable %s feature with url %s" % (feature, self.shared\_link\_url))  
 try:  
 o365\_enable\_shared\_link = requests.request("POST", self.shared\_link\_url)  
 except Exception as e:  
 logger.debug(e)  
 return False  
  
 if o365\_enable\_shared\_link.status\_code == 200:  
 logger.debug("The feature %s is enabled %s:- " % (feature,o365\_enable\_shared\_link.text))  
  
 return True  
 else:  
 return False  
 elif enable == 'disable':  
 logger.debug("Going to disable %s feature with url %s" % (feature, self.shared\_link\_url))  
 try:  
 o365\_enable\_shared\_link = requests.request("DELETE", self.shared\_link\_url)  
 except Exception as e:  
 logger.debug(e)  
 return False  
  
 if o365\_enable\_shared\_link.status\_code == 200:  
 logger.debug("The %s Feature disabled %s :- " % (feature, o365\_enable\_shared\_link.text))  
 return True  
 else:  
 return False  
  
  
 @keyword("fetch event stats from")  
 def fetch\_event\_stats(self,Sleep\_Enable=False):  
 *''' Fetch the Event stats for the CSP provided '''* logger.console("Fetch event counts for all the CSPs of a tenant")  
 zeus\_ip = self.shutil.get\_zeus\_ip()  
 if Sleep\_Enable:  
 logger.console("Waiting for 10 mins for EventStats to get updated")  
 time.sleep(600)  
 url = 'http://'+zeus\_ip+':9000/v1/tenant-health/'+str(self.tenantid)  
 response = requests.get(url)  
 if response.status\_code == 200:  
 logger.console("Able to get tenant health")  
 else:  
 logger.console("Failed to get the tenant health with status code"+ response.status\_code)  
 return False  
 try:  
 api\_data = json.loads(response.text)  
 except Exception:  
 logger.console("Failed to parse json for api fetch event counts")  
 return False  
 event\_stats = []  
 for i in api\_data:  
 if int(i['instance\_id'] != int(self.instanceid)):  
 continue  
 else:  
 event\_stats = i['event\_stat\_list']  
 break  
 csp\_event\_stats = {}  
 for item in event\_stats:  
 name = item['name']  
 csp\_event\_stats[name] = item  
 return csp\_event\_stats  
  
 def \_get\_fields\_by\_title(self, tag\_name, headers):  
  
 response = requests.get(self.endpoint\_create\_field, headers=headers, cookies=self.cookies, verify=False)  
 if response.status\_code == 200:  
 logger.console("Able to get fileds by title")  
 else:  
 logger.console("Failed to get the fields by title"+ response.status\_code)  
 return None  
  
 json\_res = json.loads(response.text)  
 fields = [item for item in json\_res['value'] if item['Title'] == tag\_name]  
 if len(fields) == 0:  
 logger.console("Did not find tag :%s on the root folder" % tag\_name)  
 return None  
  
 property\_name = fields[0].get('EntityPropertyName', None)  
 return property\_name  
  
 @keyword("In ${SERVICE} ${action} external sharing in O365")  
 def external\_sharing(self, action="enable"):  
 *"""  
 Enable Or Disable External Sharing For The SharePoint Site  
  
 Args:  
 action: enable or disable  
  
 Returns:  
 List containing JSON of file properties  
  
 Raises:  
 None  
 """* sharesettings\_file\_path = self.shutil.env\_info.get("TEMPLATES\_FOLDER")  
 sharesettings\_file = 'ShareSetting.xml'  
 setting = 2 if action == "disable" else 3  
 kwargs = {  
 'sharesetting': int(setting),  
 'domain\_url':str(self.domain\_url),  
 }  
  
 env = jinja2.Environment(loader=jinja2.FileSystemLoader(searchpath=sharesettings\_file\_path),  
 trim\_blocks=True,  
 lstrip\_blocks=True  
 )  
  
 template = env.get\_template(sharesettings\_file)  
 template\_after\_render = template.render(kwargs)  
 sharesettings\_data = template\_after\_render  
 cspid = "16131" # tenant administration in OneDrive can be done only with SharePoint token  
 from skybot.lib.GetInstances import get\_instance  
 instance\_id = get\_instance.get\_instance\_id(cspid)  
 try:  
 self.access\_token\_admin = self.shutil.get\_access\_token(cspid, resource=self.domain\_admin\_url, instance\_id=instance\_id)  
 except Exception as e:  
 logger.error(e)  
 raise Exception  
 endpoint\_sharing = self.domain\_admin\_url + "/\_vti\_bin/client.svc/ProcessQuery"  
 headers = {"Authorization": "Bearer " + self.access\_token\_admin, "Accept": "application/xml"}  
  
 response\_sharing = requests.post(url=endpoint\_sharing, headers=headers, data=str(sharesettings\_data))  
 logger.debug("The response is %s" % (response\_sharing.json()))  
  
 if response\_sharing.status\_code in [401,403]:  
 logger.debug("Got a %s error.." %(response\_sharing.status\_code))  
 raise Exception  
 elif response\_sharing.status\_code == 200:  
 logger.console("Successfully set the sharing on the site")  
 ext\_share = response\_sharing.json()[4]["DefaultSharingLinkType"]  
 if ext\_share == 3:  
 logger.debug("External Sharing is enabled on the site")  
 return True  
 elif ext\_share == 2:  
 logger.debug("Internal Sharing is enabled on the site")  
 return True  
 return False  
  
 @keyword("In ${SERVICE} update last uploaded file with data ${data}")  
 def update\_file(self,data='NULL'):  
 *"""  
 To overwrite the last uploaded file with input data  
  
 Args:  
 data : Data that needs to be updated in to the document  
  
 Returns:  
 Returns the file\_id of last uploaded file  
  
 Raises:  
 Exception in case of any token refresh issues  
 """* filename = self.lastuploadedfiles[-1].get('filename')  
 filecontent = data  
 file\_id = self.lastuploadedfiles[-1]["fileid"]  
 if file\_id is None:  
 return  
 headers = {"X-RequestDigest": self.request\_digest, "Content-Type": self.get\_mime\_type(filename=os.path.basename(filename)),  
 "Accept": "application/json"}  
 parent\_id, self.mostrecentfolder = [0 if self.mostrecentfolder is None else self.mostrecentfolder] \* 2  
 if parent\_id:  
 endpoint\_GetFolderByServerRelativeUrl = re.sub('GetFolderByServerRelativeUrl(.\*)',  
 'GetFolderByServerRelativeUrl(\'' + parent\_id + '\')',  
 self.endpoint\_GetFolderByServerRelativeUrl)  
 else:  
 endpoint\_GetFolderByServerRelativeUrl = self.endpoint\_GetFolderByServerRelativeUrl  
 update\_url = endpoint\_GetFolderByServerRelativeUrl + '/Files/add(url=\'' + os.path.basename(filename) \  
 + '\', overwrite=true)'  
 logger.info("URL to update is " + update\_url)  
  
 response\_update\_file = requests.post(update\_url, headers=headers, cookies=self.cookies, data=filecontent)  
  
 if response\_update\_file.status\_code == 200:  
 logger.info("Response for upload file is " + str(response\_update\_file.text))  
  
 elif response\_update\_file.status\_code in [401,403]:  
 logger.debug("Got a 403 error refreshing access token...")  
 self.\_refresh\_token()  
 raise Exception  
  
 logger.info("Response post upload file is: " + response\_update\_file.text)  
 return file\_id  
  
 @keyword("get version count of last uploaded file")  
 def get\_version\_details(self,filename=None,site=None):  
 *"""  
 To get the version count for last uploaded file  
  
 Args:  
 None  
  
 Returns:  
 version count: returns the version count of the last uploaded file  
  
 Raises:  
 Exception in case of any token refresh issues  
 """* retry = 3  
 if not filename:  
 file\_id = self.lastuploadedfiles[-1]["fileid"]  
 else:  
 file\_id = os.path.join(self.mostrecentfolder,filename)  
 headers = {"X-RequestDigest": self.request\_digest, "Accept": "application/json"}  
  
 endpoint\_GetFileByServerRelativeUrl = re.sub('GetFileByServerRelativeUrl(.\*)','GetFileByServerRelativeUrl(\'' + file\_id + '\')',self.endpoint\_GetFileByServerRelativeUrl)  
 if site:  
 req\_dig = self.req\_digest(self.domain\_url + "/sites/" + site)  
 headers["X-RequestDigest"] = req\_dig  
 endpoint\_GetFileByServerRelativeUrl = re.sub('(.\*)\/\_api', self.domain\_url + '/sites/' + site + '/\_api', endpoint\_GetFileByServerRelativeUrl )  
 get\_versions\_url = endpoint\_GetFileByServerRelativeUrl + '?$expand=Versions'  
  
 for i in range(retry):  
 response\_get\_versions = requests.get(get\_versions\_url, headers=headers, cookies=self.cookies)  
 logger.debug(response\_get\_versions.json())  
 if response\_get\_versions.status\_code == 200:  
 logger.info("Response for getting version count is " + str(response\_get\_versions.text))  
 version\_dict = {}  
 version\_dict['latest\_version'] = int(eval(response\_get\_versions.json()["UIVersionLabel"]))  
 version\_history = [float(x['VersionLabel']) for x in response\_get\_versions.json()['Versions']]  
 version\_dict['version\_history'] = version\_history  
 version\_dict['version\_count'] = len(version\_history) + 1  
 # version\_count = int(eval(response\_get\_versions.json()["UIVersionLabel"]))  
 return version\_dict  
 elif response\_get\_versions.status\_code in [401,403]:  
 logger.debug("Got a 403 error refreshing access token...")  
 self.\_refresh\_token()  
 if site:  
 req\_dig = self.req\_digest(self.domain\_url + "/sites/" + site)  
 headers["X-RequestDigest"] = req\_dig  
 if retry == 3:  
 raise AccesstokenError  
 else:  
 raise Exception("Failed to retrieve version of the file")  
  
 def \_isMemberPresentInGroup(self, groupid, member):  
 *"""  
 Check if member is present in the group or not  
 :param groupid: O365 group id  
 :param member: User id  
 :return: Boolean  
 """* retry = 3  
 result = False  
 headers = {"Authorization": "Bearer " + self.access\_token\_graph, "content-type": "application/json"}  
 endpoint\_group\_member\_url = self.endpoint\_groups\_url + str(groupid) + "/members/$ref"  
 while retry > 0:  
 response = requests.get(url=endpoint\_group\_member\_url , headers=headers)  
 retry = retry - 1  
 if response.status\_code == 200:  
 try:  
 members\_list = json.loads(response.text)["value"]  
 if isinstance(members\_list, list) and len(members\_list) == 0:  
 return False  
 member\_ids = [item['@odata.id'].split("/")[-2] for item in members\_list]  
 return True if member in member\_ids else False  
 except KeyError:  
 logger.error("Response does not contain value filed")  
 return False  
 else:  
 logger.debug("Get group members call did not go through .. retrying")  
 time.sleep(30)  
  
 if not result:  
 raise Exception  
  
 def \_validate\_drm\_action(self, file\_id, filename):  
 *"""  
 drm\_action validation is three step process:  
 1. Verify file size before and after applying drm. File size should increase after drm  
 2. Verify checksume before and after applying drm. It should not be same  
 3. After applying drm action, list the file tags and it should have 'DRM\_PROTECTED' as one of the tags  
 :param file\_id:  
 :return: Boolean  
 """* result = False  
  
 self.download\_file(file\_id)  
 download\_content\_to = WS + 'tmp/' + str(time.time()) + '.' + filename.split('.')[-1]  
 with open(download\_content\_to, 'wb') as f:  
 f.write(self.lastdownloadedfilecontents)  
 from tika import parser  
 try:  
 content = parser.from\_file(download\_content\_to)  
 except Exception:  
 if os.path.exists(download\_content\_to):  
 os.remove(download\_content\_to)  
 return False  
  
 if 'PROTECTED BY IONIC SECURITY' in content['content']:  
 logger.debug("File is DRM protected")  
 result=True  
  
 if os.path.exists(download\_content\_to):  
 os.remove(download\_content\_to)  
  
 return result  
  
 @keyword("In ${SERVICE} create ${listType} in o365 with name ${listName}")  
 def create\_list(self, listType, listName):  
 *"""  
 Create generic list or document library in O365  
 :param listType:Generic List = 0; Document Library = 1;  
 :param listName: Name of the list or doclibrary  
 """* if listType.lower() == "document library":  
 BaseType = 1  
 BaseTemplate = 101  
 elif listType.lower() == "generic list":  
 BaseType = 0  
 BaseTemplate = 100  
 else:  
 logger.debug("list type is not supported")  
 raise Exception  
 logger.debug("Going to Create: " + str(listType))  
 headers = {"Authorization": "Bearer " + self.access\_token, "Accept": "application/json;odata=verbose"}  
 headers.update({"Content-Type": "application/json;odata=verbose"})  
 payload = {  
 "\_\_metadata": {"type": "SP.List"},  
 "AllowContentTypes": 'true',  
 "BaseType": BaseType,  
 "BaseTemplate": BaseTemplate,  
 "ContentTypesEnabled": 'true',  
 "Description": "Creating list via API",  
 "Title": listName  
 }  
  
 response\_create\_list = requests.post(self.endpoint\_create\_list, headers=headers, json=payload)  
 if response\_create\_list.status\_code == 500 and \  
 "title already exists" in response\_create\_list.json()['error']['message']['value']:  
 logger.info("List or Doc Lib already exists " + str(response\_create\_list.json()))  
 self.\_get\_list\_guid(listName)  
 elif response\_create\_list.status\_code in [200,201]:  
 self.list\_guid = response\_create\_list.json()["d"]["Id"]  
 logger.debug("Created the List: " + listName + "& GUID is: " + self.list\_guid)  
 else:  
 logger.error("Failed to Create the List")  
 raise Exception  
 return response\_create\_list.json()  
  
 @keyword("In ${SERVICE} delete o365 list ${listName}")  
 def delete\_list(self, listName=None):  
 *"""  
 Delete generic list or document library in O365  
 :param listName: Name of the list or doclibrary  
 """* if listName is not None:  
 logger.debug("Going to get GUID of list: " + str(listName))  
 self.\_get\_list\_guid(listName)  
 if self.list\_guid is not None:  
 logger.debug("Deleting the List with GUID: " + self.list\_guid)  
 headers = {"Authorization": "Bearer " + self.access\_token, "X-HTTP-Method": "DELETE", "IF-MATCH": "\*"}  
 self.endpoint\_delete\_list = self.endpoint\_list + "(guid'" + self.list\_guid + "')?@target='" + self.endpoint\_host\_web\_url + "'"  
 response\_delete\_list = requests.post(self.endpoint\_delete\_list, headers=headers)  
 logger.debug("Deleted the list with GUID: " + self.list\_guid)  
 return True  
 else:  
 logger.warn("No existing list to be Deleted")  
 return None  
  
 def \_get\_list\_guid(self, listName):  
 *"""  
 Gets the Guid of generic list or document library in O365  
 :param listName: Name of the list or doclibrary  
 """* headers = {"Authorization": "Bearer " + self.access\_token, "Accept": "application/json;odata=verbose"}  
 headers.update({"Content-Type": "application/json;odata=verbose"})  
 self.endpoint\_get\_list = self.endpoint\_list + "/getbytitle('" + listName + "')?@target='" + self.endpoint\_host\_web\_url + "'"  
 response\_get\_list = requests.get(self.endpoint\_get\_list, headers=headers)  
 if response\_get\_list.status\_code != 200 and \  
 "does not exist" in response\_get\_list.json()['error']['message']['value']:  
 logger.error("Error:" + str(response\_get\_list.json()['error']['message']['value']))  
 raise Exception  
 self.list\_guid = response\_get\_list.json()["d"]["Id"]  
 return self.list\_guid  
  
 # @keyword("In ${SERVICE} validate aip label ${label\_info}")  
 # def validate\_aip\_label(self, label\_info):  
 # """  
 # To verify the applied AIP label name on the given file  
 # Args:  
 # file\_path: path of the file  
 # label\_info: Colon separated AIP instance id, name, type of label, label id, name (ex- 2067:AIP-shneuqa:62e5eb5c-da1f-412a-ba31-04818b4c53d7:passport-label)  
 # """  
 # label = str(label\_info).split(':')  
 # label\_id = label[2]  
 # label\_name = label[3]  
 # root\_file\_path = self.default\_root\_folder + self.mostrecentfoldername + '/' + self.lastuploadedfiles[-1]["filename"]  
 # aip\_label\_details = "AIP Label ID: " + label\_id + ", AIP Label Name: " + label\_name  
 # logger.debug("File Path: " + root\_file\_path + ", " + aip\_label\_details)  
 # file\_download\_url = self.endpoint\_GetFileByServerRelativePath + root\_file\_path + "'"  
 # request\_headers = {'Authorization': "Bearer " + str(self.access\_token), 'Content-Type': 'application/json',  
 # 'Accept': "application/json"}  
 # response = requests.get(file\_download\_url, headers=request\_headers)  
 # if response.status\_code in (204, 200):  
 # logger.info("Successfully retrieved the file content to verify AIP labels")  
 # elif response.status\_code in [401, 403]:  
 # logger.debug("Token expired, refreshing and trying again")  
 # self.\_refresh\_token()  
 # raise Exception  
 # else:  
 # raise Exception("Failed to fetch file content " + str(response.content))  
  
 # if str(response.content).find(str(label\_id)) and str(response.content).find(str(label\_name)):  
 # logger.debug("Got " + aip\_label\_details + " in the file " + root\_file\_path)  
 # return True  
 # logger.debug("Didn't find " + aip\_label\_details + " in the file " + root\_file\_path)  
 # return False  
  
 @keyword("Log into ${service} as ${user}")  
 def login\_to\_service\_ui(self, user):  
 app\_name = "OneDrive"  
 super(OneDrive, self).login\_to\_service\_ui(user)  
 self.driver.set\_window\_size(1600, 1200)  
 if user == 'user':  
 username = self.user  
 password = self.password  
 elif user == 'external user':  
 username = self.externalcollaborator  
 password = self.externalcollaborator\_password  
 try:  
 serviceurl = "https://login.microsoftonline.com"  
 CommonHelper.go\_to\_url(self.driver, serviceurl)  
 CommonHelper.wait\_for\_seconds(3)  
 CommonHelper.wait\_for\_element\_and\_sendkeys(self.driver, LocatorType.XPATH, O365\_locators['login\_username'],  
 username)  
 CommonHelper.wait\_for\_element\_and\_click(self.driver, LocatorType.XPATH, O365\_locators['login\_next\_button'])  
 CommonHelper.wait\_for\_seconds(5)  
 CommonHelper.wait\_for\_element\_and\_sendkeys(self.driver, LocatorType.XPATH, O365\_locators['login\_password'],  
 password)  
 CommonHelper.wait\_for\_element\_and\_click(self.driver, LocatorType.XPATH, O365\_locators['login\_signIn\_button'])  
 CommonHelper.wait\_for\_seconds(4)  
  
 keep\_sign\_ele = CommonHelper.get\_elements\_from\_locator\_type(self.driver, LocatorType.XPATH,  
 O365\_locators['login\_keep\_signedIn'])  
 if len(keep\_sign\_ele) > 0:  
 CommonHelper.wait\_for\_element\_and\_click(self.driver, LocatorType.XPATH,  
 O365\_locators['login\_keep\_signedIn'])  
 CommonHelper.wait\_for\_seconds(15)  
  
 if CommonHelper.is\_element\_displayed(self.driver, LocatorType.XPATH, str(O365\_locators['homepage\_login']), 120):  
 logger.console("Succesfully logged into o365 account")  
 return True  
 else:  
 return False  
  
 except Exception as e:  
 ss\_name = time.time()  
 logger.console("Login to O365 failed, taking screenshot")  
 self.driver.save\_screenshot("%s.png" % ss\_name)  
 return False  
  
 def enable\_api\_access(self, params, driver\_obj, api, wait, EC, By):  
 time.sleep(5)  
 wait.until(EC.visibility\_of\_element\_located((By.XPATH, api.page\_elements\_dict["common"]["preReqCheck"])))  
 driver\_obj.find\_element\_by\_xpath(api.page\_elements\_dict['common']['preReqCheck']).click()  
 logger.debug("Clicked Prerequisites")  
 time.sleep(3)  
 wait.until(EC.element\_to\_be\_clickable((By.XPATH, api.page\_elements\_dict["common"]["nextButton"])))  
 driver\_obj.find\_element\_by\_xpath(api.page\_elements\_dict['common']['nextButton']).click()  
 logger.debug("Clicked Next")  
 time.sleep(5)  
 driver\_obj.find\_element\_by\_xpath(api.page\_elements\_dict['common']['credsButton']).click()  
 logger.debug("Clicked Provide Credentials")  
 time.sleep(5)  
 handles=driver\_obj.window\_handles  
 current=driver\_obj.current\_window\_handle  
 driver\_obj.switch\_to.window(handles[1])  
  
 driver\_obj.find\_element\_by\_xpath(api.page\_elements\_dict['OneDrive']['oneDriveEmail']).send\_keys(str(params['email']))  
 driver\_obj.find\_element\_by\_xpath(api.page\_elements\_dict['OneDrive']['nextButton']).click()  
 time.sleep(10)  
 driver\_obj.find\_element\_by\_xpath(api.page\_elements\_dict['OneDrive']['password']).send\_keys(str(params['password']))  
 driver\_obj.find\_element\_by\_xpath(api.page\_elements\_dict['OneDrive']['nextButton']).click()  
 time.sleep(10)  
 driver\_obj.find\_element\_by\_xpath(api.page\_elements\_dict['OneDrive']['acceptButton']).click()  
 time.sleep(10)  
 driver\_obj.switch\_to.window(current)  
 return True  
  
 def click\_url(self, url):  
 url = [x for sublist in url for x in sublist]  
 url = list(dict.fromkeys(url))  
 urls= [value for value in url if value is not False]  
 try:  
 link\_to\_click = self.select\_url(urls, OneDrive.URL\_PATTERN\_TO\_FIND)[0]  
 except Exception as e:  
 logger.console("Link received is None")  
 return False  
  
 if link\_to\_click:  
 logger.console("Link to click in OneDrive= " + str(link\_to\_click))  
 CommonHelper.go\_to\_url(self.driver, link\_to\_click)  
 #waiting for the page to load  
 CommonHelper.wait\_for\_seconds(8)  
 if CommonHelper.is\_element\_displayed(self.driver, LocatorType.XPATH,  
 O365\_locators['onedrive\_item\_removed\_page']):  
 logger.console("OneDrive link is expired- The user doesnot have permission to access this file")  
 return True  
 else:  
 logger.console("OneDrive link accessible")  
 return False  
 else:  
 logger.error("No OneDrive link received")  
 return False  
  
 def get\_users\_for\_O365Group(self, users):  
 internal\_users\_set = set(self.get\_all\_users()) - set(self.external\_users)  
 external\_users\_list = self.external\_users  
 internal\_users\_list = [i\_user for i\_user in list(internal\_users\_set) if i\_user.lower() not in (self.user.lower(), self.email.lower())]  
 #internal\_users\_list = [i\_user for i\_user in list(internal\_users\_set)]  
 logger.debug("Internal users list is: " + str( ))  
 logger.debug("External users list is: " + str(external\_users\_list))  
 o365\_group\_users = []  
  
 if "\*" in users:  
 try:  
 if "i" not in users or "e" not in users:  
 logger.error("Please use correct syntax to select users . e.g. i1\*e1\*, due to incorrect synatx "  
 "this will use by default \*")  
 internal\_groups\_choice, external\_groups\_choice = re.match(r'i(\d)\\*e(\d)\\*', users, re.M|re.I).groups()  
 internal\_users = sorted(internal\_users\_list)[:int(internal\_groups\_choice)]  
 external\_users = sorted(external\_users\_list)[:int(external\_groups\_choice)]  
 # internal\_users = random.sample(internal\_users\_set, int(internal\_groups\_choice))  
 # external\_users = random.sample(external\_users\_set, int(external\_groups\_choice))  
 o365\_group\_users.extend(internal\_users)  
 o365\_group\_users.extend(external\_users)  
 logger.debug("O365 group users:" + str(o365\_group\_users))  
 except Exception as e:  
 logger.warn("Error in randomly picking Users" + str(e))  
 return o365\_group\_users  
  
 def get\_list\_name(self):  
 if getattr(self, 'library', None) is None:  
 title = "Documents"  
 else:  
 title = self.library  
 return title  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 #from skybot.OF.lib.core.SkyHighDashboard.ShnDlpInterface import ShnDlpUtil  
 #shutil = ShnDlpUtil("qaautoregression", 80560, "Welcome2dlp#", "dlpp1qa@gmail.com", None, None, use\_token=True)  
 from skybot.lib import SHNInterface  
 SHNInterface.myenv.project = "OF"  
 SHNInterface.myenv.zeus\_admin\_uname = "admin@spop.com"  
 SHNInterface.myenv.zeus\_admin\_pwd = "admin"  
 # SHNInterface.myenv = SHNInterface.Util('qaautoregression', 80560, 'Welcome2dlp#', 'dlpp1qa@gmail.com', None, None, use\_token=True)  
 from skybot.OF.lib.core.SkyHighDashboard.ShnDlpInterface import ShnDlpUtil  
 shutil = ShnDlpUtil('qaautoregression', 79164, 'Skyhigh\_1234', 'vidisha\_aggarwal@mcafee.com', None, None, use\_token=True)  
 # SHNInterface.myenv = shutil  
 Od = OneDrive(shutil, "qaautoregression", 79164, 3210, "user2@skyhightest2.onmicrosoft.com", instance\_id=10844)  
 Od.as\_user("user1@skyhightest3.onmicrosoft.com")  
 Od.cleanup()  
 # Od.delete\_file("7BE27B6A9B-F2BF-46AC-85D7-35C091A2E81E")  
 #Od.create\_folder('Test7')