from \_\_future\_\_ import absolute\_import  
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  
import os  
import re  
from skybot.lib.logger import logger  
from skybot.OF.lib.core.SkyHighDashboard import Interface  
from robot.api.deco import keyword  
import json  
import random  
import string  
#from OneDrive import OneDrive  
from skybot.OF.lib.core.Services.OneDrive import OneDrive  
from robot.libraries.BuiltIn import BuiltIn  
from skybot.OF.lib.Utilities.HealthMonitor import trackme  
from skybot.OF.lib.Utilities import Utils  
  
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  
  
requests.packages.urllib3.disable\_warnings()  
retry = 3  
  
class SharePoint(OneDrive):  
 file\_to\_upload = {}  
 URL\_PATTERN\_TO\_FIND = 'sharepoint'  
  
 # Folders method  
  
 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  
 """* 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  
  
 if self.site:  
 endpoint\_GetFolderByServerRelativeUrl = re.sub('(.\*)\/\_api', self.domain\_url + '/sites/' + self.site + '/\_api',\  
 endpoint\_GetFolderByServerRelativeUrl)  
  
 logger.debug("Inside get\_folder\_info with 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"}  
 headers.update({"Accept": "application/json"})  
 response\_folder\_info = requests.get(url=get\_folder\_url, headers=headers, cookies=self.cookies)  
 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.debug("Got a 403 error..")  
 self.\_refresh\_token()  
 if self.site:  
 self.request\_digest = self.req\_digest(self.domain\_url + "/sites/" + self.site)  
 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  
  
# Link methods  
  
 @keyword("generate link in ${SERVICE} for last uploaded ${object}")  
 def create\_link(self, object=object, object\_id=None, password="", expiration="", direct="", link\_type="edit"):  
 *"""  
 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  
 """* link\_url = None  
 if object\_id is None:  
 if object == "file":  
 object\_id = self.lastuploadedfiles[-1]["fileid"]  
 elif object == "folder":  
 object\_id = self.mostrecentfolder  
 logger.debug("Inside create Link to create an anonymous Link for object : " + str(object\_id))  
 # headers = self.headers.copy()  
 headers = {"X-RequestDigest": self.request\_digest, "Accept": "application/json"}  
 headers.update({"Content-Type": "application/json"})  
 endpoint\_create\_link = self.domain\_url + "/\_api/SP.Web.CreateAnonymousLink"  
 data = {"url": self.domain\_url + object\_id, "isEditLink": True}  
 for attempt in range(1, 3):  
 response\_create\_link = requests.post(url=endpoint\_create\_link, headers=headers, data=json.dumps(data), cookies=self.cookies)  
 if response\_create\_link.status\_code == 200:  
 logger.debug("Response from generating Link is " + response\_create\_link.text)  
 break  
 elif response\_create\_link.status\_code in [403, 401]:  
 logger.debug("Got a 403 error..")  
 self.\_refresh\_token()  
 raise Exception  
 else:  
 logger.debug("Failed to create link, will retry after 10 sec: " + str(attempt) + "/3")  
 time.sleep(10)  
 continue  
 if "value" in response\_create\_link.json():  
 link\_url = response\_create\_link.json()["value"]  
 logger.debug("Link is generated with the link\_url: " + link\_url)  
 return link\_url  
  
 def \_get\_endpoints(self):  
 *"""  
 Going to build endpoints based on the email/domain  
 :return: None  
 """* default\_root\_folder = "/Shared Documents"  
 try:  
 for each in self.shutil.services.get("allservices"):  
 if each == "SharePoint":  
 self.root\_folder = self.shutil.services.get("allservices").get("SharePoint")\  
 .get("library", default\_root\_folder)  
 break  
 else:  
 if "\_" in each:  
 if each.split("\_")[1] == str(self.instance\_id):  
 self.root\_folder = self.shutil.services.get("allservices").get(each) \  
 .get("library", default\_root\_folder)  
 break  
 except Exception as e:  
 logger.warn("Not able to get default library due to: ${0}, using hardcoded default ${1}".format(e, str(default\_root\_folder)))  
 self.root\_folder = default\_root\_folder  
 self.user\_flat = self.user.replace("@", ".").replace(".", "\_")  
 self.domain\_url = "https://" + self.domain\_name + ".sharepoint.com"  
 self.domain\_admin\_url = "https://" + self.domain\_name + "-admin.sharepoint.com"  
 self.endpoint\_GetFolderByServerRelativeUrl = self.domain\_url + "/\_api/Web/GetFolderByServerRelativeUrl(\'" + \  
 self.root\_folder + "\')"  
 self.endpoint\_GetFileByServerRelativeUrl = self.domain\_url + "/sites/TestAuto/\_api/Web/GetFileByServerRelativeUrl(\'" + \  
 self.root\_folder + "\')"  
 self.endpoint\_users = "https://graph.microsoft.com/v1.0/" + self.domain\_name + ".onmicrosoft.com" + "/users"  
 self.endpoint\_retrieve\_links = "https://" + self.domain\_name + ".sharepoint.com" + \  
 "/\_vti\_bin/client.svc/ProcessQuery"  
 self.endpoint\_contextinfo = self.domain\_url + '/sites/QAAutomationPublic/\_api/contextinfo'  
 self.endpoint\_groups = "https://graph.microsoft.com/v1.0/" + self.domain\_name + ".onmicrosoft.com" + "/groups"  
 self.endpoint\_create\_field = self.domain\_url + "/\_api/web/lists/getbytitle(\'Documents\')/Fields"  
 self.endpoint\_create\_link = self.domain\_url + "/\_api/"  
 logger.debug("Sharepoint Library documents been used is: " + str(self.endpoint\_GetFolderByServerRelativeUrl))  
 self.endpoint\_retrieve\_flexilink = self.domain\_url + \  
 "/\_api/web/getlistitem(@url)/getsharinginformation/permissionsInformation/links?@url='%s'"  
 self.endpoint\_host\_web\_url = self.domain\_url  
 self.endpoint\_list = self.domain\_url + "/\_api/SP.AppContextSite(@target)/web/Lists"  
 self.endpoint\_create\_list = self.endpoint\_list + "?@target='" + self.domain\_url + "'"  
 self.endpoint\_GetFileByServerRelativePath = self.domain\_url + \  
 "/\_api/web/GetFileByServerRelativePath(decodedurl=@relativeUrl)/$value?@relativeUrl='"  
 #self.default\_root\_folder = default\_root\_folder  
 self.default\_root\_folder = self.root\_folder  
 self.endpoint\_DirectAccessSharing = self.domain\_url + "/\_api/SP.Sharing.DocumentSharingManager.UpdateDocumentSharingInfo"  
 self.endpoint\_sharepoint\_group = self.domain\_url + '/sites/QAAutomationPublic/\_api/web/sitegroups'  
 self.endpoint\_SPGroup\_add\_users = self.domain\_url + "/sites/QAAutomationPublic/\_api/web/sitegroups/GetById({0})/users"  
 #self.endpoint\_SPGroup\_add\_users = self.domain\_url + "/sites/QAAutomationPublic/\_api/web/sitegroups/GetById({0})/users"  
 self.endpoint\_UserCreated\_SPGroup\_add\_users = self.domain\_url + "/sites/{1}/\_api/web/sitegroups/GetById({0})/users"  
 self.endpoint\_SPGroup\_get\_users = self.domain\_url + "/\_api/web/sitegroups/GetByName('" + "{0}" + "')"  
 self.endpoint\_SPGroup\_delete = self.domain\_url + "/\_api/web/sitegroups/removebyid({0})"  
 self.endpoint\_groups\_url = "https://graph.microsoft.com/v1.0/groups/"  
 self.endpoint\_Flexilink = self.domain\_url + "/\_api/web/GetListItemUsingPath(decodedurl=@u)/ShareLink?@u='{0}'"  
 self.endpoint\_Folder\_DirectAccessSharing = self.domain\_url + \  
 "/sites/QAAutomationPublic/\_api/web/GetFolderByServerRelativeUrl(@relativeUrl)/ListItemAllFields/ShareObject?@relativeUrl='%s'"  
 self.endpoint\_File\_DirectAccessSharing = self.domain\_url + \  
 "/sites/QAAutomationPublic/\_api/web/GetFileByServerRelativeUrl(@relativeUrl)/ListItemAllFields/ShareObject?@relativeUrl='%s'"  
 self.endpoint\_DirectAccessSharing\_listId = self.domain\_url + "/sites/QAAutomationPublic/\_api/web/Lists(@a1)/GetItemById(@a2)/ShareObject?@a1='{%s}'&@a2='%s'"  
 self.endpoint\_GetFileListByServerRelativePathUrl = self.domain\_url + "/\_api/web/GetFileByS erverRelativePath(decodedurl=@relativeUrl)" + \  
 "/ListItemAllFields?@relativeUrl='%s'"  
 self.endpoint\_GetFolderListByServerRelativePathUrl = self.domain\_url + "/\_api/web/GetFolderByServerRelativePath(decodedurl=@relativeUrl)" + \  
 "/ListItemAllFields?@relativeUrl='%s'"  
 self.endpoint\_Flexilink\_bylistid = self.domain\_url + "/\_api/web/Lists(@a1)/GetItemById(@a2)/ShareLink?@a1='{%s}'&@a2='%s'"  
  
 def \_get\_SPgroups\_id(self,groupname):  
 headers = {"Authorization": "Bearer " + self.access\_token, "content-type": "application/json;odata=verbose","Accept": "application/json;odata=verbose"}  
 result = requests.get(url=self.endpoint\_sharepoint\_group, headers=headers)  
 if result.status\_code in [200,201]:  
 for group in json.loads(result.text)["d"]["results"]:  
 if group["Title"]==groupname:  
 return group["Id"]  
 logger.error("Group %s not Found!" %(groupname))  
 return False  
 elif result.status\_code in [401,403]:  
 logger.debug("Retrying after refreshing access token")  
 self.\_refresh\_token  
 r = self.\_get\_SPgroups\_id(groupname)  
 return r  
 else:  
 ("Unable to Fetch all groups due to " + str(result.text))  
 return False  
  
 @keyword("In ${SERVICE} get external members from ${members\_list}")  
 def get\_external\_users\_from\_group(self,members\_list):  
 external\_users=[]  
 for i in members\_list:  
 user\_domain=i.split("@")[1].split(".")[0]  
 logger.debug(user\_domain)  
 logger.debug(self.domain\_name)  
 if user\_domain==self.domain\_name:  
 continue  
 else:  
 external\_users.append(i)  
 return external\_users  
  
  
 @keyword("In ${SERVICE} create sharepoint group ${group\_name}")  
 def create\_sharepoint\_group(self,group\_name):  
  
 headers = {"Authorization": "Bearer " + self.access\_token, "Accept": "application/json;odata=verbose","content-type": "application/json;odata=verbose"}  
 data={  
 "\_\_metadata": {  
 "type": "SP.Group"  
 },  
 "Title": group\_name,  
 "Description": "Automation"  
 }  
 result=requests.post(url=self.endpoint\_sharepoint\_group, headers=headers, data=json.dumps(data))  
 if result.status\_code == 201:  
 json\_result = (json.loads(result.text))  
 GroupId=json\_result["d"]["Id"]  
 self.lastcreatedgroup.append(  
 {  
 "groupid": str(GroupId),  
 "groupname": str(group\_name),  
 "email":str(group\_name)  
 }  
 )  
 return True  
 elif result.status\_code == 500:  
 logger.debug("Group %s already exists, deleting and try again " %(group\_name))  
 deleted = self.delete\_sharepoint\_group(group\_name)  
 if deleted:  
 r = self.create\_sharepoint\_group(group\_name)  
 return r  
 else:  
 return False  
 elif result.status\_code in [401,403]:  
 logger.debug("Retrying after refreshing access token")  
 self.\_refresh\_token()  
 r = self.create\_sharepoint\_group(group\_name)  
 return r  
 else:  
 logger.error("Unable to create Sharepoint group due to " + str(result.text))  
 return False  
  
 @keyword("In ${SERVICE} validate for external users {usersList}")  
 def validate\_for\_external\_users(self,usersList):  
 logger.debug(usersList)  
 domains\_list=[]  
 user\_domain = self.domain\_url.split("//")[1]  
 for user in usersList:  
 domain=user.split('@')[1].split(".")[0]  
 domains\_list.append()  
 domain==user\_domain  
  
  
 @keyword("In ${SERVICE} get members of sharepoint group ${groupname}")  
 def get\_members\_of\_sharepoint\_group(self,groupname=None):  
 for group in self.lastcreatedgroup:  
 if group['groupname'] == groupname:  
 groupid = group["groupid"]  
 break  
 url = self.endpoint\_SPGroup\_get\_users.format(groupname) + "/users?$select=Email,Id"  
 logger.debug(url)  
 headers = {"Authorization": "Bearer " + self.access\_token, "Accept": "application/json;odata=nometadata","content-type": "application/json;odata=nometadata"}  
 result = requests.get(url=url, headers=headers)  
 if result.status\_code in [200, 201]:  
 my\_json = result.content.decode('utf8')  
 data = json.loads(my\_json)  
 users = []  
 results = data['value']  
 for i in results:  
 users.append(i['Email'])  
 return users  
 elif result.status\_code in [401, 403]:  
 logger.debug("Retrying after refreshing access token")  
 self.\_refresh\_token  
 result = requests.get(url=url, headers=headers)  
 return result.content  
 else:  
 ("Unable to Fetch members of group due to " + str(result.text))  
 return False  
  
 @keyword("In ${SERVICE} add members to sharepoint site ${sitename} default group ${groupid}")  
 def add\_members\_to\_sharepoint\_site\_default\_group(self,sitename=None, groupid=None ):  
 add\_result = []  
 users = BuiltIn().replace\_variables('${o365\_users}')  
 self.members\_to\_collaborate = self.get\_users\_for\_O365Group(users)  
 if sitename == "":  
 url = self.endpoint\_SPGroup\_add\_users.format(groupid)  
 else:  
 url = self.endpoint\_UserCreated\_SPGroup\_add\_users.format(groupid,sitename)  
 headers = {"Authorization": "Bearer " + self.access\_token, "content-type": "application/json;odata=verbose"}  
 for user in self.members\_to\_collaborate:  
 login\_name = self.\_get\_loginname(user)  
 data = {  
 '\_\_metadata': {  
  
 'type': 'SP.User'  
 },  
 'LoginName': login\_name  
 }  
  
 response=requests.post(url=url, headers=headers, data=json.dumps(data))  
 if response.status\_code == 201:  
 add\_result.append({user:True})  
 elif response.status\_code in [401,400]:  
 self.\_refresh\_token()  
 response = requests.post(url=url, headers=headers, data=json.dumps(data))  
 if response.status\_code == 201:  
 add\_result.append({user: True})  
 else:  
 add\_result.append({user: False})  
 logger.debug(add\_result)  
 logger.error("Adding member %s failed " %(user))  
 return False  
  
 return True  
  
 @keyword("In ${SERVICE} add members to sharepoint group ${groupname}")  
 def add\_members\_to\_sharepoint\_group(self,groupname=None ):  
 add\_result = []  
 users = BuiltIn().replace\_variables('${o365\_users}')  
 self.members\_to\_collaborate = self.get\_users\_for\_O365Group(users)  
 index=0  
 if groupname:  
 for group in self.lastcreatedgroup:  
 if group['groupname']== groupname:  
 groupid=group["groupid"]  
 self.lastcreatedgroup[index]['members'] = self.members\_to\_collaborate  
 index +=1  
 else:  
 groupid = self.lastcreatedgroup[-1].get("groupid")  
 if not groupid:  
 logger.error("Group name %s is not created" % (groupname))  
 return False  
 url = self.endpoint\_SPGroup\_add\_users.format(groupid)  
 headers = {"Authorization": "Bearer " + self.access\_token, "content-type": "application/json;odata=verbose"}  
 for user in self.members\_to\_collaborate:  
 login\_name = self.\_get\_loginname(user)  
 data = {  
 '\_\_metadata': {  
  
 'type': 'SP.User'  
 },  
 'LoginName': login\_name  
 }  
  
 response=requests.post(url=url, headers=headers, data=json.dumps(data))  
 if response.status\_code == 201:  
 add\_result.append({user:True})  
 elif response.status\_code in [401,400]:  
 self.\_refresh\_token()  
 response = requests.post(url=url, headers=headers, data=json.dumps(data))  
 if response.status\_code == 201:  
 add\_result.append({user: True})  
 else:  
 add\_result.append({user: False})  
 logger.debug(add\_result)  
 logger.error("Adding member %s failed " %(user))  
 return False  
  
 return True  
  
 @keyword("In ${SERVICE} Get groupID from ${groupname}")  
 def get\_groupid(self,groupname=None):  
 groupid = None  
 groupid = self.\_get\_SPgroups\_id(groupname)  
 return groupid  
  
  
 @keyword("In ${SERVICE} delete latest sharepoint group")  
 def delete\_sharepoint\_group(self,groupname=None):  
 headers = {"Authorization": "Bearer " + self.access\_token, "content-type": "application/json;odata=verbose"}  
 delete\_result = []  
 groupid = None  
 retry = 0  
 if groupname:  
 for each in self.lastcreatedgroup:  
 if each['groupname'] == groupname:  
 groupid = each.get("groupid")  
 break  
  
 if not groupid:  
 groupid = self.\_get\_SPgroups\_id(groupname)  
  
 if groupid:  
 url = self.endpoint\_SPGroup\_delete.format(groupid)  
 response = requests.post(url=url, headers=headers)  
 if response.status\_code in [200, 201]:  
 delete\_result.append(True)  
 elif response.status\_code in [401,403] and retry == 0:  
 self.\_refresh\_token()  
 response = requests.post(url=url, headers=headers)  
 if response.status\_code in [200, 201]:  
 delete\_result.append(True)  
 else:  
 delete\_result.append(False)  
 else:  
 for group in self.lastcreatedgroup:  
 retry = 0  
 groupid = group.get("groupid")  
 url = self.endpoint\_SPGroup\_delete.format(groupid)  
 result = requests.post(url=url, headers=headers)  
 if result.status\_code in [200, 201]:  
 delete\_result.append(True)  
 elif result.status\_code in [401, 403] and retry == 0:  
 self.\_refresh\_token()  
 result = requests.post(url=url, headers=headers)  
 if result.status\_code in [200, 201]:  
 delete\_result.append(True)  
 else:  
 delete\_result.append(False)  
 else:  
 logger.error("Unable to delete SP Group %s due to %s and status %s " % (  
 groupid, result.text, result.status\_code))  
 delete\_result.append(False)  
  
 return all(delete\_result)  
  
 def \_get\_loginname(self,username):  
 if username:  
 for user in self.response\_get\_all\_users.json().get("value"):  
 if not user.get("mail"):  
 continue  
 if username == user.get("mail"):  
 loginname="i:0#.f|membership|"+str(user.get("userPrincipalName").lower())  
 return loginname  
 logger.debug("user %s not found in Sharepoint" %(username))  
 return False  
 else:  
 logger.debug("username sent is None")  
 return False  
  
  
  
 def upload\_file(self, filename, parent\_id=0, overwrite=True, site=None):  
  
 headers = {"X-RequestDigest": self.request\_digest, "Content-Type": self.get\_mime\_type(filename=os.path.basename(filename)),  
 "Accept": "application/json"}  
  
 filename = str(filename)  
 if self.testdata not in filename:  
 filename = self.testdata + "/" + filename  
 file = str(os.path.basename(filename))  
  
 with open(filename, "rb") as fp:  
 if filename not in SharePoint.file\_to\_upload:  
 SharePoint.file\_to\_upload[file] = fp.read()  
 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=\'' + requests.encode\_url(os.path.basename(filename)) \  
 + '\', overwrite=true)'  
 if site:  
 self.site = site[type(self).\_\_name\_\_]  
 req\_dig = self.req\_digest(self.domain\_url + "/sites/" + self.site)  
 headers["X-RequestDigest"] = req\_dig  
 upload\_url = re.sub('(.\*)\/\_api', self.domain\_url + '/sites/' + self.site + '/\_api', upload\_url)  
 else:  
 self.site = None  
  
 logger.info("Create File url is " + upload\_url)  
  
 for i in range(retry):  
 response = requests.post(upload\_url, headers=headers,cookies=self.cookies, data=SharePoint.file\_to\_upload[file])  
 self.response = response  
 if response.status\_code == 200:  
 break  
 if response.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/" + self.site)  
 headers["X-RequestDigest"] = req\_dig  
  
 if response.status\_code != 200:  
 raise Exception  
  
 logger.info("Response post upload file is: " + response.text)  
 if "ServerRelativeUrl" in response.json():  
 logger.info("File " + os.path.basename(filename) + " is successfully uploaded")  
 file\_id = response.json()["ServerRelativeUrl"]  
  
 if self.site:  
 quarantineref = "/sites/" + self.site + ":" + str(file\_id)  
 else:  
 quarantineref = "/:" + str(file\_id)  
  
 self.lastuploadedfiles.append(  
 {  
 "fileid":str(file\_id),  
 "filename":str(file),  
 "folderid": parent\_id,  
 "quarantineref": quarantineref,  
 "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 req\_digest(self, site):  
 url = site + '/\_api/contextinfo'  
 headers = {"Accept": "application/json", "Content-Length": '0'}  
 response = requests.post(url, headers=headers, cookies=self.cookies)  
 return response.json()['FormDigestValue']  
  
  
 @keyword("get different user from ${SERVICE}")  
 def get\_different\_user\_for\_collab(self):  
 users = self.get\_all\_users()  
 users.remove(str(self.user))  
 return random.choice(users)  
  
 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['SharePoint']['adminResourceURL']).send\_keys(  
 str(params['resourceURL']))  
 driver\_obj.find\_element\_by\_xpath(api.page\_elements\_dict['Jive']['JiveSubmit']).click()  
 time.sleep(5)  
 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  
  
 @keyword("In ${SERVICE} create o365 group with ${visibility} ${name}")  
 def create\_o365\_group(self,visibility,name,type=None):  
 result = True  
 #name = name+".sharepoint"  
 if not type:  
 type = ["Unified"]  
 logger.debug("type of group is === " + str(type))  
 else:  
 logger.debug("type of group in else is === " + str(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": 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  
 @keyword("In ${SERVICE} verify retry count in case of rate limits")  
 def verify\_retrycount\_for\_ratelimits(self):  
 import redis  
 redis\_server = self.shutil.get\_redis\_ip()  
 redis\_client = redis.Redis(redis\_server)  
 redis\_key = 'offlinedlp:event\_error\_metrics:{0}:{1}:{2}:event\_retry\_count'.format(self.tenantid, self.cspid, self.instance\_id)  
 file\_path = self.lastuploadedfiles[-1]['quarantineref']  
 retries = int(redis\_client.hget(redis\_key, file\_path))  
 if retries >= 2:  
 logger.debug("Retrying in case of rate limits")  
 return True  
 else:  
 logger.debug("Not retrying in case of rate limits")  
 return False  
  
 @keyword("Log into ${service} as ${user}")  
 def login\_to\_service\_ui(self, user):  
 logger.console("Logging into SharePoint UI now")  
 if super(SharePoint, self).login\_to\_service\_ui(user):  
 logger.console("Successfully Logged into SharePoint!")  
 return True  
 else:  
 logger.console("Could not login to SharePoint")  
 return False  
  
  
 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, SharePoint.URL\_PATTERN\_TO\_FIND)  
 except Exception as e:  
 logger.console("Link received is None")  
 return False  
  
 for link in link\_to\_click:  
 if ('-my.' in link)== False and '.sharepoint' in link:  
 link\_to\_click=link  
 if link\_to\_click:  
 logger.console("Link to click in SharePoint= " + 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("SharePoint link is expired! The user does not have permission to access this file")  
 return True  
 else:  
 logger.console("SharePoint link accessible")  
 return False  
 else:  
 logger.error("No SharePoint link received")  
 return False  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 from skybot.lib import SHNInterface  
 SHNInterface.myenv = SHNInterface.Util("qaautoregression", "dlpqap1@gmail.com", "Welcome2dlp#")  
 from skybot.OF.lib.core.SkyHighDashboard.ShnDlpInterface import ShnDlpUtil  
 shutil = ShnDlpUtil("qaautoregression", 5642, "Welcome2dlp#", "dlpqap1@gmail.com", None, None, use\_token=True)  
 SHNInterface.myenv = shutil  
 shutil.current\_service = "SharePoint"  
 os.environ.setdefault("office365\_password", "")  
  
 Od = SharePoint(shutil, "qaautoregression", 5642,16131, "admin@shnqaeu4.onmicrosoft.com", instance\_id=12997)  
  
 Od.as\_user("user1@shnqaeu4.onmicrosoft.com")  
 #Od.create\_folder('Test7',site='automationgroupsite')  
 #Od.upload\_file('Confidential.docx',site='automationgroupsite')  
 #Od.create\_folder('FoldertoCheckCollaboration3')  
 #Od.upload\_file('/Users/siddharth/Documents/DlpProjectOF/trunk/DLPRobotFramework/data/files/forbidden.txt')  
 #Od.add\_permission(None,user\_attr={'role':'editor','email':'\*'},file\_collaboration=False)  
 # Od.as\_user("admin@ak001.onmicrosoft.com")  
 # Od.create\_folder("myfolder", "/personal/admin\_ak001\_onmicrosoft\_com/Documents")  
 # Od.create\_folder("Ashish1\_Folder")  
 # Od.upload\_file("/Users/ashishk/Documents/Skyhigh/svn/automation/automation/DLPFramework\_Current/DLPRobotFramework/data/files/test.txt")  
 # Od.list\_permissions("/personal/admin\_ak5\_onmicrosoft\_com/Documents/My Folder")  
 object\_id = "/Shared Documents/1596647468.433655/Confidential.docx"  
 user\_attr = {"email": "\*", "role": "editor"}  
 Od.add\_permission(object\_id, user\_attr,add\_to\_all\_collaborators=False)  
 # # # Od.get\_drive()  
 # print Od.upload\_file("/Users/ashishk/Documents/Skyhigh/svn/automation/automation/DLPFramework\_Current/DLPRobotFramework/data/files/test.txt")  
 # Od.delete\_file(file\_id="my folder/test.txt")