import os

from lxml import etree

def update\_fields\_based\_on\_xsi\_type(

folder\_path,

field\_paths\_by\_xsi\_type, # Dictionary with xsi:type values as keys and corresponding field paths as values

base\_value="StressTestEBS",

start\_value=1,

output\_folder=None,

num\_duplicates=1,

exclude\_local\_name="ListedOption"

):

"""

Parcourt tous les fichiers XML, modifie les champs en fonction de l'attribut xsi:type, duplique les fichiers et

sauvegarde avec des noms uniques.

Args:

folder\_path (str): Chemin du dossier contenant les fichiers XML.

field\_paths\_by\_xsi\_type (dict): Dictionnaire où la clé est la valeur de l'attribut xsi:type et la valeur est

la liste des chemins XPath des champs à modifier.

base\_value (str): Base de la nouvelle valeur (par défaut "StressTestEBS").

start\_value (int): Valeur de départ pour l'incrémentation.

output\_folder (str): Chemin vers le dossier de sortie. Si None, les fichiers sont enregistrés dans le dossier source.

num\_duplicates (int): Nombre de duplications souhaitées pour chaque fichier XML.

exclude\_local\_name (str): Exclut les fichiers contenant cet élément (par défaut "ListedOption").

"""

if output\_folder is None:

output\_folder = folder\_path

if not os.path.exists(output\_folder):

os.makedirs(output\_folder)

# Liste tous les fichiers dans le dossier

for file\_name in os.listdir(folder\_path):

file\_path = os.path.join(folder\_path, file\_name)

# Vérifie si c'est un fichier XML

if os.path.isfile(file\_path) and file\_name.endswith('.xml'):

print(f"Traitement du fichier : {file\_name}")

try:

# Charge le fichier XML avec lxml

parser = etree.XMLParser(remove\_blank\_text=True)

tree = etree.parse(file\_path, parser)

root = tree.getroot()

# Exclut les fichiers contenant "ListedOption"

if root.xpath(f"//\*[local-name()='{exclude\_local\_name}']"):

print(f"Fichier {file\_name} contient '{exclude\_local\_name}', exclu du traitement.")

continue

# Cherche le tag Message et vérifie la valeur de l'attribut xsi:type

message\_elems = root.xpath("//\*[local-name()='Message']")

if not message\_elems:

print(f"Pas de balise <Message> trouvée dans {file\_name}.")

continue

for message\_elem in message\_elems:

xsi\_type = message\_elem.get("{http://www.w3.org/2001/XMLSchema-instance}type")

if xsi\_type not in field\_paths\_by\_xsi\_type:

print(f"xsi:type '{xsi\_type}' non trouvé dans le dictionnaire pour {file\_name}.")

continue

# Récupère les field\_paths correspondant à ce xsi:type

field\_paths = field\_paths\_by\_xsi\_type[xsi\_type]

print(f"xsi:type '{xsi\_type}' trouvé dans {file\_name}, modification des champs...")

for duplicate\_num in range(1, num\_duplicates + 1):

# Incrémente à chaque duplication

count = start\_value + duplicate\_num - 1

# 1. Modifie ConversationId et MessageId si leur parent est Message (direct parent)

conversation\_id\_elems = message\_elem.xpath("./\*[local-name()='ConversationId']")

if conversation\_id\_elems:

for conversation\_id\_elem in conversation\_id\_elems:

conversation\_id\_elem.text = "$ConversationId"

print(f"ConversationId modifié en '$ConversationId' dans {file\_name}")

message\_id\_elems = message\_elem.xpath("./\*[local-name()='MessageId']")

if message\_id\_elems:

for message\_id\_elem in message\_id\_elems:

message\_id\_elem.text = "$MessageId"

print(f"MessageId modifié en '$MessageId' dans {file\_name}")

# 2. Modifie l'élément User si son parent est Message (ou si Message est la racine)

user\_elems = message\_elem.xpath("./\*[local-name()='User']")

if user\_elems:

for user\_elem in user\_elems:

# Remplacement de <User> par le contenu modifié

new\_user = etree.Element("User")

name\_elem = etree.SubElement(new\_user, "Name")

name\_elem.text = "MANAGER"

password\_elem = etree.SubElement(new\_user, "Password", encryption="NONE")

user\_elem.getparent().replace(user\_elem, new\_user)

print(f"Élément User modifié dans {file\_name}")

# 3. Modifie les champs dans field\_paths en fonction du xsi:type

for field\_path in field\_paths:

# Construction correcte du XPath

path\_parts = field\_path.split('/')

if len(path\_parts) == 1:

# Si c'est un élément de la racine

xpath\_expr = f"\*[local-name()='{path\_parts[0]}']"

else:

# Sinon, construire le chemin avec les parents et l'élément cible

xpath\_expr = "/".join([f"\*[local-name()='{part}']" for part in path\_parts])

field\_elems = message\_elem.xpath(f"./{xpath\_expr}")

if field\_elems:

new\_value = f"{base\_value}-{path\_parts[-1]}-{count:011d}"

for elem in field\_elems:

elem.text = new\_value

print(f"Champ {field\_path} modifié -> {new\_value}")

# Crée le nouveau nom de fichier avec un suffixe unique pour chaque duplication

new\_file\_name = file\_name.replace('.xml', f'\_new{duplicate\_num}.xml')

new\_file\_path = os.path.join(output\_folder, new\_file\_name)

# Écrit dans le nouveau fichier en conservant le namespace

tree.write(new\_file\_path, pretty\_print=True, encoding="utf-8", xml\_declaration=True)

print(f"Fichier modifié et dupliqué enregistré sous : {new\_file\_path}")

except etree.XMLSyntaxError as e:

print(f"Erreur de parsing dans le fichier {file\_name}: {e}")

# Exemple d'utilisation :

folder\_path = '/chemin/vers/le/dossier' # Remplacez par le chemin de votre dossier contenant les fichiers XML

output\_folder = '/chemin/vers/dossier/sortie' # Dossier où enregistrer les fichiers modifiés

# Dictionnaire où les clés sont les valeurs de xsi:type, et les valeurs sont les chemins XPath des champs à modifier

field\_paths\_by\_xsi\_type = {

"ProductCreationRequestType": [

'Swap/Name', # Exemple de chemin avec parent Swap et élément Name

'MirroredTrade/Trade/Instrument/SEDProduct/FlowPayoff/SwapReference/ExternalId' # Exemple plus complexe

],

"OtherRequestType": [

'SomeOtherField/Name',

]

}

# Appel de la fonction avec duplication de 5 fois chaque fichier XML

update\_fields\_based\_on\_xsi\_type(

folder\_path,

field\_paths\_by\_xsi\_type,

base\_value="StressTestEBS",

start\_value=1,

output\_folder=output\_folder,

num\_duplicates=5

)