import os

from lxml import etree

import re

def fix\_empty\_tags(file\_path):

"""

Corrige les balises auto-fermantes dans un fichier XML en les transformant en balises ouvertes/fermées.

Args:

file\_path (str): Le chemin vers le fichier XML à corriger.

"""

with open(file\_path, 'r', encoding='utf-8') as file:

content = file.read()

# Rechercher les balises auto-fermantes comme <Lis/>

content\_fixed = re.sub(r'<(\w+)([^>]\*)/>', r'<\1\2></\1>', content)

# Écrire le contenu corrigé dans le fichier

with open(file\_path, 'w', encoding='utf-8') as file:

file.write(content\_fixed)

def update\_fields\_in\_folder(

folder\_path,

field\_paths,

base\_value="StressTestEBS",

start\_value=1,

output\_folder=None,

num\_duplicates=1,

exclude\_local\_name="ListedOption"

):

"""

Parcourt tous les fichiers XML, modifie les champs, duplique les fichiers

et sauvegarde avec des noms uniques, en respectant les hiérarchies directes de parents/enfants.

Args:

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

field\_paths (list): Liste des chemins XPath des champs à modifier (avec local name).

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)

global\_counter = start\_value # Compteur global pour chaque champ unique

# 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

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

# Crée un compteur spécifique pour chaque fichier dupliqué

file\_counter = global\_counter

global\_counter += 1 # Incrémente global pour toutes les duplications

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

for message\_elem in root.xpath("//\*[local-name()='Message']"):

# Vérification du parent direct pour ConversationId

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

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

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

if conversation\_id\_elems:

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

# Vérification du parent direct pour MessageId

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

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

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

if message\_id\_elems:

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']")

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 fields\_path uniquement si l'ordre exact des parents/enfants est respecté

for field\_path in field\_paths:

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

# Construction exacte du XPath en respectant la hiérarchie des parents/enfants

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

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

if field\_elems:

# Génère une nouvelle valeur unique pour chaque champ modifié

for elem in field\_elems:

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

elem.text = new\_value

global\_counter += 1 # Incrémente après chaque modification

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,

method="xml" # Ajout du paramètre pour éviter les balises auto-fermantes

)

# Corrige les balises auto-fermantes après l'écriture

fix\_empty\_tags(new\_file\_path)

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

field\_paths = [

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

]

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

update\_fields\_in\_folder