1. Rappels généraux

N'oubliez pas de déposer votre travail sur Moodle à *chaque* séance. Vous pouvez le continuer chez vous après la séance et le re-déposer. Le travail doit rester entièrement personnel, mais vous pouvez demander de l'aide, conseils et informations.

Cette semaine nous allons découvrir deux aspects de la norme XML : la validation de documents et l'extraction d'informations d'un document. Ces deux concepts existent aussi avec la norme JSON (JSON schemas et JSONpath), mais sont nettement moins faciles à manipuler qu'avec XML Schemas et XPath à cause de la relative pauvreté du format JSON comparé à XML.

Ce TP n'est pas en lien direct avec la problématique du test automatique de logiciel, mais plusieurs outils des tests fonctionnels et d'API utilisent la norme XML, à la fois pour représenter des actions de test et aussi pour extraire des informations des résultats et les comparer à ce qui est attendu. Par exemple, Robot Framework (voir TP7) peut utiliser la norme XPath pour désigner un élément et récupérer son contenu dans le DOM de l'interface utilisateur à tester — c'est la manière la plus précise pour faire cela. Autre exemple en test d'API, Wiremock utilise XPath pour spécifier des comparaisons entre un document reçu d'un client et ce qui est attendu. Ce TP n'est présent dans le cours R4.02 uniquement parce qu'il n'est pas fait ailleurs et que ces connaissances sur XML sont jugées indispensables au métier que vous préparez.

👉 Créez un nouveau dossier pour tous les fichiers du TP : `tp6`.

2. Validation d'un document XML (max 1 heure)

👉 Téléchargez massifs1.xml et schema_massifs.xsd (clic droit puis enregistrer la cible du lien sous...).

Le fichier `massifs1.xml` décrit des massifs montagneux en France, mais il est très incomplet. C'est volontaire pour rendre le TP intéressant. Étudiez sa structure, l'imbrication des éléments et leurs attributs.

Les altitudes sont définies en mètres, les longueurs en kilomètres et les surfaces en kilomètres carrés.

Le fichier `schema_massifs.xsd` est un schéma de validation pour `massifs1.xml`. Il permet de vérifier si ce dernier est correct.

👉 Tapez les deux commandes suivantes. Ces commandes reposent sur la même librairie sous-jacente, `libxml2`, et elles donnent le même résultat :

```
xmllint --schema schema_massifs.xsd --noout massifs1.xml
xmlstarlet val --xsd schema_massifs.xsd -e massifs1.xml
```

Si aucun de ces logiciels n'est installé ou ne fonctionne sur votre poste, téléchargez xml-valid et xml-valid.jar . C'est une application Java (sources) qui fait le même travail. Faites ensuite `chmod u+x xml-valid` pour le rendre exécutable. On la lance par :

```
./xml-valid --schema schema_massifs.xsd massifs1.xml
```

👉 Ouvrez `schema_massifs.xsd` dans un éditeur et regardez comment il est constitué. Ce sont des définitions d'éléments et de types, `Elem...` et `Type...`. Les éléments utilisent les types à

la fois pour définir leur contenu et pour les attributs. Regardez par exemple comment les types `ElemMassif` et `TypeAltitude` sont définis et où ils sont employés.

2.1. Vérifications de la validation

On va maintenant légèrement altérer ces fichiers pour voir comment la validation peut échouer.

👉 Ouvrez `massifs1.xml` dans un éditeur, puis essayez successivement les modifications suivantes, vérifiez que la validation ne passe plus, puis annulez les changements pour revenir à la version initiale à chaque fois :

- Enlevez le `>` de l'une des balises ou enlevez l'un des `"` de l'un des attributs. Ça rend le document « mal formé », pas au format XML.
- Copiez l'un des éléments `<glacier>` (n'importe lequel) et placez-le directement sous la racine, ligne 3.
- Copiez l'un des éléments `<glacier>` d'un massif et mettez-le avant tous les `<sommet>` du même massif.
- Changer l'une des balises `<glacier>` en `<lac>`.
- Enlevez l'attribut `nom` de l'un des massifs.
- Ajoutez l'attribut `nature="calcaire"` au massif du Jura.
- Ajoutez un chiffre à l'une des altitudes, ex: `<sommet nom="Le Mont Blanc" altitude="94808">`

👉 Ouvrez `schema_massifs.xsd` dans un éditeur, puis essayez une par une chacune des modifications suivantes, vérifiez que la validation ne passe plus et surtout étudiez un peu le message d'erreur (les lignes signalées en erreur), puis annulez tous les changements avant de passer au point suivant :

- Ligne marquée `<!--ICI 1 -->` : au lieu de `maxOccurs="unbounded"`, mettez `maxOccurs="2"`. Regardez bien quelle est la ligne de `massifs1.xml` qui est en erreur (pourquoi n'a-t-il pas signalé les 5 massifs des Alpes ?)
- Ligne `<!--ICI 2 -->` : au lieu de `minOccurs="0"`, mettez `minOccurs="1"`
- Ligne `<!--ICI 3 -->` : au lieu de `xs:string`, mettez `xs:integer`.
- Ligne `<!--ICI 4 -->` : enlevez l'attribut `mixed="true"`. Cet attribut autorise la présence de texte, *character content*, dans l'élément. Si on l'enlève, alors il ne doit pas y avoir de texte entre les balises de cet élément.
- Ligne `<!--ICI 5 -->` : ajoutez l'attribut `use="required"`.

2.2. Augmentation des fichiers

On continue avec un nouveau fichier XML, contenant des ajouts au document initial. Téléchargez massifs2.xml (clic droit, enregistrer la cible du lien sous...).

👉 Ce nouveau document ne se valide pas avec `schema_massifs.xsd`. Votre travail consiste à modifier ce dernier, le schéma, pour qu'il valide `massifs2.xml`.

Vous allez avoir des difficultés pour modéliser l'élément `<département>`, pas à cause du é, mais parce qu'il mélange attribut et contenu texte obligatoire. Évidemment, on pourrait faire comme

pour `<sommet>` et `<glacier>`, mais ces éléments peuvent être vides, ce qui n'est pas souhaitable pour `<département>`.

Relisez le transparent 50 « Élément texte avec attribut » du cours CM2 sur XML . Il faut adapter un peu, car dans ce transparent, l'attribut est de type `xs:string`. Ici, ça sera `xs:integer`. Et le texte est du type `xs:integer` dans le transparent, tandis qu'ici, ça sera `xs:string`.

Vous devrez rendre tous les attributs obligatoires sauf si ça empêche la validation de `massifs2.xml`. Par exemple, dans `massifs2.xml`, tous les lacs ont un attribut `altitude` donc il doit être obligatoire, et certains glaciers n'ont pas l'attribut `longueur`, donc il est optionnel.

👉 Ensuite, vous allez rendre la validation plus stricte, en définissant des types plus précis :

- `TypeLongueur` = décimal entre 0.5 et 15, 15 étant exclus
- `TypeSurface` = décimal compris entre 0 et 5, et trois chiffres après la virgule (facette `fractionDigits`)
- `TypeNumeroDept` = entier à deux chiffres, ou expression régulière `[0-9][0-9]|2[AB]` (facette `pattern`)
- `TypeNomDept` = chaîne non vide (longueur > 0)
- `TypeLacNaturel` = chaîne valant soit `oui` soit `non`. Sa valeur par défaut est `oui`.

👉 Voici maintenant un problème complexe, s'il vous reste du temps (1h max sur la validation). Est-ce possible de modifier le schéma pour que chaque massif sans sous-massif doive avoir au moins un sommet, glacier ou lac ; et inversement que chaque massif avec des sous-massifs ne doit avoir aucun sommet, uniquement des sous-éléments massif ? C'est à dire que le massif du Jura n'a pas de sous-massif, donc il doit avoir au moins un sommet. Inversement, le massif des Alpes est composé de sous-massifs, donc ne doit pas avoir de sommets, uniquement des éléments `<massif>`.

La solution passe par une imbrication subtile de `<xs:sequence>` et `<xs:choice>`, relire le CM2 transparents 40 à 43. On rappelle que `<xs:sequence>` impose des éléments dans un certain ordre, dont `minOccurs` et `maxOccurs` fixent le nombre à la suite, et `<xs:choice>` indique une alternative de possibilités exclusives.

Donc la structure pourrait être la suivante. L'élément `<massif>` doit être composé de :

- d'une séquence commençant par zéro ou plus `<département>`.
- une alternative entre :
  - soit un ou plusieurs éléments `<massif>`
  - soit une séquence d'éléments `<sommet>`, `<glacier>` et `<lac>`.

Avant de tenter cette évolution, copiez votre fichier `schema_massifs.xsd`. Si vous n'y arrivez pas, revenez à la solution plus simple validant toutes les combinaisons. Dans tous les cas, votre `schema_massifs.xsd` doit valider `massifs2.xml`.

3. Requêtes XPath

XPath permet d'interroger un document XML, c'est à dire extraire des éléments, attributs ou textes correspondant à une spécification basée sur des chemins et des conditions.

3.1. Tutoriel

On commence par une série d'exemples pour comprendre les principes. Téléchargez le document `annonces.xml` (clic droit, enregistrer la cible du lien sous...).

Pour l'exécution d'une requête, vous avez le choix, soit de travailler en ligne de commande, soit avec ce formulaire HTML .

- en ligne de commande, pour afficher des éléments :

```
xmllint --xpath 'expression' annonces.xml
```

- avec xmlstarlet, l'affichage des textes est un peu meilleur :

```
xmlstarlet sel --template --value-of 'expression' annonces.xml
```

Voici maintenant des requêtes de complexité croissante à essayer et à comprendre. Elles sont dans `exemples_xpath` . À vous d'exprimer mentalement en français ce que calculent ces requêtes quand ce n'est pas expliqué. Il n'est pas demandé de rédiger les réponses. C'est seulement pour comprendre XPath. N'hésitez pas à demander des explications si quelque chose vous semble étrange.

- Requêtes de base
  - `/annonces/avion/prix` sélectionne tous les éléments `<prix>` qui correspondent au chemin
  - `/annonces/avion/prix/text()` seulement les contenus textes des éléments
  - `//description` tous les éléments `<description>` où qu'ils soient
  - `//avion/@année` tous les attributs `année` des éléments `<avion>`
  - `//avion/@année/text()` rien n'est affiché car un attribut n'a pas de contenu texte
  - `//avion/@Annee` le nom de l'attribut n'est pas bon, rien n'est affiché, mais ce n'est pas une erreur, seulement une requête sans résultat dans ce document
- Requêtes simples avec des conditions
  - `//avion[@id=159]` élément `<avion>` entier ayant cet attribut avec cette valeur
  - `//avion[@année>=1990]/fabricant` condition sur l'avion puis recherche des sous-éléments
  - `//avion[couleur="jaune"]`
  - `//avion[couleur="jaune"][@année>1990]`
  - `//avion[couleur="jaune" and @année>1990]` équivalent à la précédente
  - `//avion[couleur="bleu" or @année>1990]`
  - `//avion[prix[@monnaie="$"]>50000]` conditions imbriquées
- Requêtes sur la position des éléments
  - `//avion[position()=1]/@id`
  - `//avion[2]/@id` vérifiez que c'est bien celui du 2e du fichier
  - `//avion[position()=last()]/@id`
  - `//avion[last()]/@id` ici, `last()` retourne le numéro du dernier élément de la liste, on peut s'en servir comme indice

  - `//modèle[last()]` : piège car il retourne les éléments `<modèle>` qui sont les derniers *dans leur parent*, et non pas le dernier `<modèle>` du document. Les éléments restent attachés à leur parent. Ça peut compliquer certaines requêtes.
  - `//avion[last()]/couleur[last()]` dernière `<couleur>` du dernier `<avion>`

- Présence ou absence d'attributs et de sous-éléments
  - `//avion[not(prix)]` affiche les éléments `<avion>` qui n'ont pas l'élément `<prix>`.
  - `//avion[prix=""]` élément `<avion>` ayant un sous-élément `<prix>` présent mais vide
  - `//avion[prix!="" and description=""]`
  - `//avion[lieu and not(lieu/pays)]` `<avion>` qui ont un sous-élément `<lieu>` sans sous-sous-élément `<pays>`

- Utilisation des axes et des jokers
  - `//prix[.>50000]/..`
  - `//ville[text()="Morlaix"]/../../@année` ici `.` et `text()` sont équivalents
  - `/descendant::ville[text()="Morlaix"]/ancestor::camion/child::fabricant`
  - `//*[@id<400]` éléments quelconques ayant un attribut `id` valant moins de 400
  - `//avion[@id=456]/following-sibling::avion[1]` premier élément `<avion>` suivant l'avion id=456
  - `//avion[@id=159]/preceding-sibling::*[1]` premier élément quelconque précédant le `<avion>` qui a un attribut id valant 159

- Fonctions
  - `//avion[count(couleur) > 1]` le comptage des éléments `<couleur>` se fait indépendamment dans chaque élément `<avion>`
  - `//*[string-length(modèle)=1]`
  - `//avion[contains(description, "refait")]`

- Requêtes imbriquées
  - `//avion[@année and not(@année > //avion/@année)]` affiche l'avion le plus ancien. On ne peut pas faire `//avion[@année = min(//avion/@année)]` car la fonction `min` n'existe pas en XPath 1.0, seulement en XPath 2.0. Mais avec une astuce, on peut compenser ce manque. L'idée est que l'année minimale est celle qui n'est jamais supérieure à aucune autre. On a donc une requête imbriquée pour chercher toutes les années et on refuse les avions dont l'année est plus grande qu'une autre.
  - `//avion[fabricant = //avion[modèle="CAP 10"]/fabricant ]` on cherche les `<avion>` dont le fabricant est le même que celui de l'`<avion>` dont le modèle est "CAP 10" c'est à dire quels sont les avions fabriqués par le constructeur qui a fait le CAP 10. On compare le nom du fabricant à celui qui est donné par la requête imbriquée `//avion[modèle="CAP 10"]/fabricant`.

3.2. Exercices

On reprend le fichier massifs2.xml en tant que source de données.

Téléchargez ReponsesTP6.txt . C'est là que vous devrez enregistrer chacune de vos réponses aux questions. Si vous voyez plusieurs solutions vraiment différentes ou que vous n'avez pas tout

à fait la bonne réponse mais plusieurs propositions proches, vous pouvez les mettre l'une après l'autre. Par exemple, mais ce ne sont pas du tout les bonnes réponses :

```
Question 4.1.a massifs2.xml : massifs qui ont des sommets
/massif//glacier[hauteur=largeur]
/sommets/hauteur>0

Question 4.1.b massifs2.xml : massifs glaciaires
//glacier[massif="sommet"]/..
```

IMPORTANT : ne changez ni le nom, ni la structure du fichier `ReponsesTP6.txt`. Vous devez seulement placer vos réponses en dessous des noms des questions. N'insérez surtout pas de retour à la ligne dans une requête si elle est trop longue. Laissez-la comme elle est, sur une seule ligne, sinon elle serait considérée comme deux requêtes différentes pour la même question, toutes deux fausses car pas complètes.

Si vous avez un message pour le correcteur, placez-le seul sur une ligne commençant par `##`.

Pour vérifier ce que vous allez déposer sur Moodle, on vous conseille d'utiliser CheckTP6.html . C'est une page HTML contenant un script capable d'analyser vos réponses. Il ne vous donnera pas votre note, mais vous saurez si vos réponses ont une chance d'être correctes.

Écrire les requêtes XPath répondant aux questions suivantes.

a. Afficher l'altitude de la Meije (sommet tel que nom=« La Meije ») : `altitude="3983"`.

b. Afficher les noms des glaciers situés dans un massif du département Haute-Savoie : 2 résultats (Le Glacier d'Argentière et La Mer de Glace).

c. Afficher les noms des glaciers qui n'ont pas d'attribut `longueur` : 5 résultats.

d. Afficher les noms des sommets situés dans le massif « Pyrénées » et ayant une description (un contenu texte), 2 résultats.

e. Afficher les noms des massifs qui sont dans au moins cinq départements : 3 résultats.

f. Afficher les noms des lacs situés à plus de 2500m d'altitude dans le massif des Écrins : 2 résultats.

g. Afficher le nom du plus haut lac du document : `nom="Lac de Combeynot"`. On obtient la réponse avec une requête imbriquée.

h. Afficher les noms des massifs qui contiennent au moins un lac et un glacier : 3 résultats.

i. Afficher le nom du sommet qui est juste avant celui appelé « L'Aiguille du Midi » : `nom="Le Mont Brouillard"`

j. Afficher le nom des massifs qui portent le même nom que l'un de leurs départements : 2 résultats, `nom="Jura"`, `nom="Vosges"`.

k. Afficher les noms des sommets plus hauts que « La Barre des Écrins » : 4 résultats.

l. Afficher les noms des lacs situés dans le département 38 : 7 résultats.

4. Rendu du TP

Vous devrez remettre votre travail à la fin de chaque séance. C'est ainsi que c'est noté, en contrôle continu.

Ouvrez un terminal bash dans le dossier `tp6` de votre TP :

```
cd ..
tar cfvz tp6.tgz tp6
```

Puis déposez le fichier `tp6.tgz` dans la zone de dépôt du TP6 sur Moodle R4.02 Qualité de développement .

ATTENTION votre travail est personnel. Si vous copiez pour quelque raison que ce soit le travail d'un autre, vous serez tous les deux pénalisés par un zéro.

En cas de souci quelconque, envoyez un mail à `pierre.nerzic@univ-rennes1.fr` pour expliquer le problème.