# Tutoriel - XML, XPath et XQuery

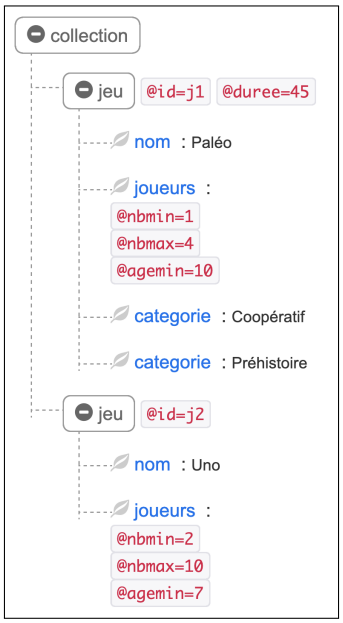
Fabien Duchateau (Université Claude Bernard Lyon 1) - 2025



**1.** eXtensible Markup Language (XML) est un **langage de balisage, organisé selon un modèle d'arbre**.
Ci-contre une collection décrivant des jeux de société. Une balise décrit une entité (e.g., un jeu, une catégorie), et on peut y imbriquer d'autres balises pour ajouter des informations (e.g., la balise `<jeu>` imbrique une balise `<nom>` ). La balise de plus haut niveau (ici `<collection>`) est appelée racine. Enfin une balise (ouvrante) peut inclure des attributs sous la forme `nom="valeur"`, comme `nbmin` qui détaille la balise `<joueurs>`.

Il est possible de visualiser des données XML sous forme d'arbre, par exemple sur Xmlviewer ou sur CodeBeautify. On peut alors déplier ou cacher certains éléments.

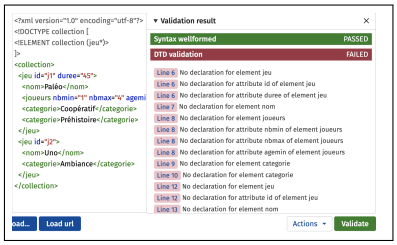
```xml
<collection>
  <jeu id="j1" duree="45">
    <nom>Paléo</nom>
    <joueurs nbmin="1" nbmax="4"
    agemin="10" />
    <categorie>Coopératif</categorie>
    <categorie>Préhistoire</categorie>
  </jeu>
  <jeu id="j2">
    <nom>Uno</nom>
    <categorie>Ambiance</categorie>
  </jeu>
</collection>
```




**2.** Une Document Type Definition (DTD) est un **langage de schéma pour des données XML**, dans lequel on décrit chaque balise et attribut. Dans la première ébauche de DTD ci-contre, l'élément `<!DOCTYPE` précise l'élément racine. Les `<!ELEMENT` définissent un attribut et son contenu. Ici, on précise qu'une balise `collection` contient 0 ou plusieurs balises `jeu` (symbole `*`).
Le site Truugo XML permet de valider un document XML par rapport à sa DTD. Copiez-collez d'abord la première version de la DTD, puis les données. Quand on appuie sur le bouton `Validate`, l'outil informe que la syntaxe est correcte, mais que les données ne sont pas (encore) conformes au schéma.

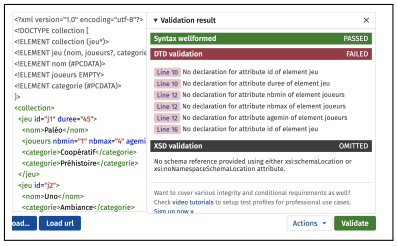
```dtd
<!DOCTYPE collection [
<!ELEMENT collection (jeu*)>
]>
```

```xml
<collection>
  <jeu id="j1" duree="45">
    <nom>Paléo</nom>
    <joueurs nbmin="1" nbmax="4"
    agemin="10" />
    <categorie>Coopératif</categorie>
    <categorie>Préhistoire</categorie>
  </jeu>
  <jeu id="j2">
    <nom>Uno</nom>
    <categorie>Ambiance</categorie>
  </jeu>
</collection>
```




**3.** Désormais le code montre uniquement la DTD, mais n'oubliez pas de copier-coller aussi les données ! Ajoutons dans notre schéma les autres **éléments décrivant des balises**. Un `jeu` imbrique une seule balise `nom` obligatoire, zéro ou une balise `joueurs` (symbole `?`), et au moins une `catégorie` (symbole `+`). On décrit chacune de ces nouvelles balises, en indiquant ici qu'elle contient du texte (`#PCDATA`) ou pas (`EMPTY`).

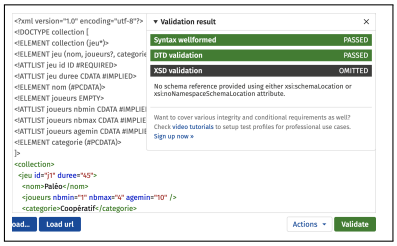
```dtd
<!DOCTYPE collection [
<!ELEMENT collection (jeu*)>
<!ELEMENT jeu (nom, joueurs?, categorie+)>
<!ELEMENT nom (#PCDATA)>
<!ELEMENT joueurs EMPTY>
<!ELEMENT categorie (#PCDATA)>
]>
```



**4.** Plus que quelques erreurs de validation concernant la **définition des attributs** ! Ajoutons des instructions `<!ATTLIST` pour chacun des attributs, en spécifiant le nom de sa balise, son nom d'attribut, un type (comme `ID`, `CDATA`, cf une liste détaillée), et éventuellement une contrainte (e.g., `#REQUIRED` pour obligatoire).

Et voilà, nos données XML sont conformes au schéma de la DTD !

```
<!DOCTYPE collection [
<!ELEMENT collection (jeu*)>
<!ELEMENT jeu (nom, joueurs?, categorie+)>
<!ATTLIST jeu id ID #REQUIRED>
<!ATTLIST jeu duree CDATA #IMPLIED>
<!ELEMENT nom (#PCDATA)>
<!ELEMENT joueurs EMPTY>
<!ATTLIST joueurs nbmin CDATA #IMPLIED>
<!ATTLIST joueurs nbmax CDATA #IMPLIED>
<!ATTLIST joueurs agemin CDATA #IMPLIED>
<!ELEMENT categorie (#PCDATA)>
]>
```




**5.** Interrogeons maintenant les données XML avec le langage d'interrogation XPath basé sur les chemins (comme sous Unix). On peut indiquer un **chemin complet vers une balise** depuis la racine. On peut également utiliser `//` qui cherche dans toutes les balises descendantes.
L'interface Videlibri Xidel permet de tester des requêtes. Copiez-collez les données à gauche, et votre requête XPath à droite. Dans les options (sous le bouton `Envoyer`), sélectionnez `Node format: xml` pour avoir des résultats avec balises. À vous !

```
(: commentaire (et pas des smileys) :)
(: toutes les balises nom :)
/collection/jeu/nom
//nom
```

```
(: nom du deuxième jeu :)
//jeu[2]/nom
```

```
(: le texte des balises enfant des balises
du deuxième jeu :)
//jeu[2]/*/text()
```

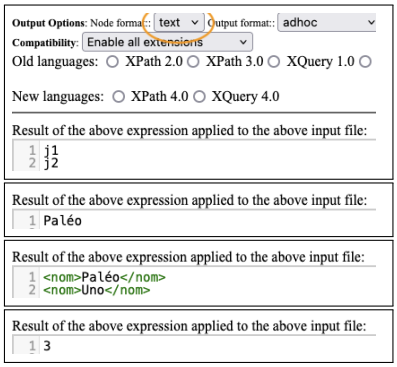

**6.** Le langage XPath permet de manipuler les **attributs**, d'utiliser des **fonctions** ou de vérifier des **conditions**.
Le symbole `@` donne accès aux attributs (choisir `Node format : text` pour voir le résultat de la première requête). Un test s'exprime entre crochets, et filtre les éléments qui ne satisfont pas la condition. Enfin il existe des fonctions comme `count` ou `contains` (cf liste de fonctions).
Essayez d'écrire d'autres requêtes (e.g., l'identifiant des jeux dont le nom contient un *n*, ou le nombre de jeux d'ambiance).

```
(: attributs id des jeux :)
//jeu/@id
```

```
(: texte des noms de jeux de durée < 60 :)
//jeu[@duree<60]/nom/text()
```

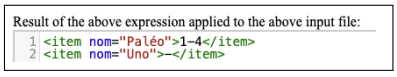
```
(: nom des jeux dont le nom contient 'o'
ou d'âge minimal supérieur à 8 :)
//jeu[contains(nom/text(), "o") or
joueurs/@agemin > 8]/nom
```

```
(: nombre de catégories :)
count(//categorie)
```




**7.** Le langage XQuery permet d'**interroger et construire un nouveau document XML**. Dans l'exemple, la requête itère sur chaque balise `jeu`, définit 3 variables (nom du jeu et valeurs d'attribut des âges), et fabrique en sortie des morceaux de XML avec ces variables.
Entrainez-vous en imaginant vos requêtes !

```
for $j in //jeu
let $n := $j/nom/text()
let $jmin := data($j/joueurs/@nbmin)
let $jmax := data($j/joueurs/@nbmax)
return <item
nom="{$n}">{$jmin}-{$jmax}</item>
```




Ce tutoriel est volontairement limité pour en faciliter la prise en main (détails supplémentaires sur la page de l'UE).