

## *XPath : Naviguer dans un document XML*

Modéliser en XMLSchema et stocker des données au format XML c'est bien ; pouvoir les manipuler, c'est mieux ! Dans les chapitres qui suivront celui-ci, nous apprendrons à écrire des feuilles de transformation XSLT et à lire et écrire des documents XML. Dans les deux cas, il est nécessaire de naviguer dans le document, c'est-à-dire de sélectionner des éléments ou attributs particuliers, ou même des listes d'éléments, de passer d'un élément à un autre, ... de la même manière que l'on navigue dans l'arborescence des fichiers d'un disque dur.

Le langage XPath offre un moyen d'identifier un ensemble de nœuds dans un document XML. Toutes les applications (écrites en Java, Javascript, C++, C#...) ayant besoin de récupérer un fragment de document XML peuvent utiliser ce langage. Nous verrons dans des chapitres ultérieurs de nombreux exemples d'usage avec des technologies utilisant XPath (XSLT, DOM) ; d'autres existent comme XQuery, etc.

XPath est fondé sur la représentation d'un document XML sous forme d'arbre, comme illustré dans les exemples du chapitre précédent: [II.7](#) et [III.2](#).

Une expression XPath (encore appelée chemin XPath ou encore *location path*) est **une expression non XML**, avec une syntaxe compacte. Elle s'évalue sur un nœud courant dans un contexte donné.

Il s'agit d'un ensemble d'étapes de navigation (*location steps*) séparées par des '/'. Par exemple, l'expression `/bagagesPassagers/bagage/ref` appliquée sur le document XML des figures [II.7](#) et [III.2](#) donne le résultat expliqué dans le tableau suivant<sup>1</sup>

---

<sup>1</sup>voir également l'animation proposée sur la plateforme de cours.

<div style="border: 1px solid red; padding: 2px; display: inline-block;"> <code>/bagagesPassagers/bagage/ref</code> </div> <div style="display: flex; justify-content: space-around; width: 100%; margin-top: 5px;"> <span>1</span><span>2</span><span>3</span><span>4</span> </div>	
1- <code>/</code> sélectionne l'arbre entier (il s'agit d'un chemin absolu)	2- <code>/bagagesPassagers</code> sélectionne tout l'élément racine <code>bagagesPassagers</code> (c'est-à-dire l'élément racine et ses sous-éléments).
3- <code>bagage</code> sélectionne les nœuds <code>bagage</code> , fils direct de la racine <code>bagagesPassagers</code> .	4- A partir des nœuds <code>bagage</code> sélectionnés, <code>ref</code> sélectionne les nœuds fils <code>ref</code> .

Une expression XPath désigne un ou plusieurs *chemins* dans l'arbre à partir du nœud courant. Elle a pour résultat soit un ensemble de nœuds (*node-set*), soit une valeur (numérique, booléenne ou alphanumérique).

*Remarque:* comme vous le verrez en TD et TP, déboguer une application dépendant d'une expression xpath peut s'avérer compliqué. Fort heureusement, il existe de nombreux sites sur Internet permettant de tester des expressions xpath sur des documents XML valides, par exemple <http://www.utilities-online.info/xpath>.

## 1 Node-set

Un nœud est un élément<sup>2</sup> + ses attributs.

A partir d'un nœud courant, une étape de chemin XPath (*location step*) renvoie **un ensemble de nœuds**, c'est-à-dire une *node-set*. Chaque nœud de ce *node-set* devient à son tour le nœud courant pour l'évaluation du *location-step* suivant.

On commence par la racine dénotée par `/`.

## 2 Les axes de recherche

Un axe est un ensemble de nœuds relativement à un autre nœud. On distingue les *axes verticaux* et les *axes horizontaux*.

### 2.a Axes verticaux

Parmi les axes verticaux, les axes *avant* permettent de descendre dans l'arbre:

<sup>2</sup>Rappel: un élément est soit une balise ouvrante + un contenu + une balise fermante, soit une balise ouvrante/fermante.

**self** désigne le nœud courant. Par exemple `/bagagesPassagers/bagage/type/self::type[format="sac à dos"]` désigne le seul nœud `type` dont l'attribut `format` a pour valeur `sac à dos`.

**child** désigne les fils du nœud courant (c'est l'axe par défaut). L'expression `bagage/child::type` est équivalente à l'expression `bagage/type`.

**descendant** désigne les descendants (les fils, et les fils des fils et les fils des fils des fils, etc.) du nœud courant. par exemple l'expression `/bagagesPassagers/descendant::ref` désigne les 2 éléments `ref` de l'arbre.

**descendant-or-self** désigne les descendants du nœud courant ainsi que le nœud courant lui-même.

Les axes *arrière* permettent de remonter dans l'arbre:

**parent** désigne le nœud parent du nœud courant

**ancestor** désigne les ascendants du nœud courant. Par exemple si le nœud courant est le premier élément `ref`, l'élément `ancestor::bagagesPassagers/compagnie` fait référence à l'élément `compagnie`.

**ancestor-or-self** désigne les ascendants du nœud courant ainsi que le nœud courant lui-même

## 2.b Axes horizontaux

**following-sibling** désigne les nœuds frères (i.e. issus du même parent) placés **après** le nœud courant. Par exemple, `/bagagesPassagers/date/following-sibling::node` désigne les 2 éléments `bagage`

**preceding-sibling** désigne les nœuds frères placés **avant** le nœud courant.

Par exemple, `/bagagesPassagers/date/preceding-sibling::node` désigne l'élément `compagnie`

## 2.c Axe pour les attributs

**attribute** désigne les attributs du nœud courant

## 2.d Abréviation des Axes

Afin d'éviter une trop grande lourdeur, les simplifications suivantes sont autorisées:

abréviation	Axe
(rien)	<code>child::</code>
@	<code>attribute::</code>
.	<code>self::node()</code>
//	<code>desendant-or-self::node()/</code>
//X	<code>desendant-or-self::node()/child::X</code>
..	<code>parent::node()</code>
../X	<code>parent::node/child::X</code>

## 2.e Remarque sur les Chemins

On peut faire un parallèle entre les chemins XPath et les chemins sur les disques durs des ordinateurs. Par exemple, sur l'arborescence d'un ordinateur, `/` désigne la racine du disque. De même, en XPath, `/bagagesPassagers` désigne la racine de l'arbre.

De même, sur un disque dur, `/cours/XML` désigne le fichier XML dans le répertoire `cours` à la racine du disque ; le chemin `/bagagesPassagers/date` désigne l'élément `date`, fils de l'élément racine `bagagesPassagers`.

Dans les 2 cas, si le chemin commence par `/` il s'agit d'un chemin *absolu* (qui part de la racine), sinon, il s'agit d'un chemin *relatif* (qui part de l'endroit/du nœud courant).

Attention cependant, cette comparaison a des limites, car dans le cas d'un chemin XPath, un chemin absolu (ou relatif) peut sélectionner plusieurs nœuds !

## 3 Sélectionner des parties de l'arbre

### 3.a Les filtres

**:: (nom)** le filtre *nom* sélectionne les nœuds de l'axe qui portent ce nom.

Par exemple, `/bagagesPassagers/child::bagage` sélectionne les 2 nœuds `bagage` fils de `bagagesPassagers`.

**\*** le filtre `*` sélectionne les nœuds de l'axe qui ont un nom.

Par exemple, `/bagagesPassagers/*/type/attribute::*` sélectionne les 4 attributs des 2 éléments `type` de l'arbre.

**text()** le filtre `text()` sélectionne les nœuds de type texte.

Par exemple, `/bagagesPassagers/date/text()` renvoie "2012-01-06".

**comment()** le filtre `comment()` renvoie les nœuds de type commentaire de l'axe.

Par exemple, `/comment()` renvoie "Bagagerie".

### 3.b Les critères

On peut ajouter des critères de sélection en ajoutant des crochets aux *location-steps*. `Chemin[critère]` ne sélectionne que les nœuds du `Chemin` qui correspondent à la sélection par critère. On peut lire ce chemin xpath ainsi : "renvoie les noeuds du Chemin *tel que ce critère est vrai*".

#### Critères d'attributs

On peut sélectionner tous les nœuds de l'arbre ayant/n'ayant pas un certain attribut:

`//*[@format]` sélectionne tous le nœuds qui ont un attribut `format`.

`//*[not(@format)]` sélectionne tous les nœuds de l'arbre qui n'ont pas d'attribut `format`.

`//bagage/*[@*]` sélectionne tous les nœuds fils de `bagage` qui ont un attribut (quel qu'il soit)

`//bagage/*[not(@*)]` sélectionne tous les nœuds fils de `bagage` qui n'ont aucun attribut

`//type[@couleur='noir']` sélectionne tous les nœuds `type` qui ont un attribut `couleur` de valeur `noir`

#### Critères d'ordre

`/bagagesPassagers/bagage[1]` sélectionne le premier nœud `bagage`.

`//bagage[2]` sélectionne le deuxième nœud `bagage`.

`/bagagesPassagers/bagage[last()]` sélectionne le dernier nœud `bagage`.

### 3.c Expressions arithmétiques

On peut également utiliser des expressions arithmétiques pour les sélections:

Opérations	
+	addition
-	soustraction
*	multiplication
div	division (attention, ce n'est pas /)
mod	modulo (reste de la division entière)
=	égalité
!=	inégalité
Opérateurs Booléens	
and	<i>et puis</i>
or	<i>ou bien</i>
not()	négation
Opérateurs de comparaison	
<	strictement inférieur à
<=	inférieur ou égal à
>	strictement supérieur à
>=	supérieur ou égal à

En XPath, une expression est vraie s'il a au moins un nœud sélectionné. Par exemple, l'expression `//type[@couleur='noir']` sélectionne tous les nœuds `type` dont l'attribut `couleur` a la valeur `noir`. L'expression `//type[@couleur!='noir']` sélectionne tous les nœuds `type` dont l'attribut `couleur` n'a pas la valeur `noir`. Ces deux expressions sont *vraies*. En effet, il existe un nœud dont l'attribut `couleur` a la valeur `noir` et un nœud dont l'attribut `couleur` n'a pas la valeur `noir`.

### 3.d Fonctions

#### Fonctions sur les chaînes de caractères

<code>string-length(s: string) : int</code>	renvoie la longueur (en nombre de caractères) de la chaîne de caractères <code>s</code> Exemple <code>string-length('bonjour')</code> renvoie 7.
<code>concat(s1: string, s2: string, ...): string</code>	renvoie la chaîne composée de <code>s1</code> , <code>s2</code> , ... à la suite Exemple: <code>concat('bonjour', 'toi')</code> renvoie la chaîne <code>'bonjourtoi'</code> .
<code>contains(s1: string, s2: string): boolean</code>	renvoie vrai si la chaîne <code>s1</code> contient la chaîne <code>s2</code> Exemple: <code>contains('bonjour', 'njo')</code> renvoie <code>true</code> .
<code>starts-with(s1: string, s2: string): boolean</code>	renvoie vrai si la chaîne <code>s1</code> commence par la chaîne <code>s2</code> Exemple: <code>starts-with('bonjour', 'njo')</code> renvoie <code>false</code> .
<code>substring(s: string, d:int, [, f:int]): string</code>	renvoie la sous-chaîne de <code>s</code> à partir de l'indice <code>d</code> (éventuellement jusqu'à l'indice <code>f</code> ) Exemple: <code>substring('bonjour', 3)</code> renvoie <code>'jour'</code> .

#### Fonctions sur les node-set

<code>count([node-set]) : int</code>	nombre de nœuds [contenu dans le node-set].
<code>last(): int</code>	taille du node-set courant.
<code>name([node-set]): string</code>	Nom du nœud courant [1er du node-set].
<code>position(): int</code>	Position du nœud courant.

## 4 Ecrire correctement du XPath

Vous connaissez la syntaxe XPath ? C'est bien ! Vous découvrirez vite cependant que l'écriture de chemins complexes en XPath peut vite devenir un casse-tête sans fin.

Pour éviter au maximum de se retrouver dans une situation inextricable, **il FAUT que vous travailliez méthodiquement**, à savoir :

### 4.a Formuler correctement le problème

Prenons tout de suite un exemple avec le document XML suivant qu stocke les données de la bagagerie d'un aéroport pour les bagages hors format :

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <bagagerie
3   xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
4   xmlns='http://www.timc.fr/nicolas.glade/bagages'
5   xsi:schemaLocation='http://www.timc.fr/nicolas.glade/bagages bagagerieHF.
   xsd'>
6   <passagers>
7     <passager nom="Gator" prenom="Ali" id="AG18300427"></passager>
8     <passager nom="Nancière" prenom="Sophie" id="SF23890311"></passager>
9     <passager nom="Kass" prenom="Oscar" id="OK17401023"></passager>
10  </passagers>
11  <bagages>
12    <bagage idPassager="OK17401023">
13      <ref>AF677793</ref>
14      <type format="autre" couleur="blanc"/>
15      <poids>12.1</poids>
16      <description>skis</description>
17    </bagage>
18    <bagage idPassager="OK17401023">
19      <ref>AF677793</ref>
20      <type format="autre" couleur="bleu"/>
21      <poids>8.7</poids>
22      <description>skis</description>
23    </bagage>
24    <bagage idPassager="AG18300427">
25      <ref>AF48717</ref>
26      <type format="autre" couleur="marron"/>
27      <poids>6.4</poids>
28      <description>tam-tam</description>
29    </bagage>
30    <bagage idPassager="SF23890311">
31      <ref>AF67840</ref>
32      <type format="autre" couleur="blanc"/>
33      <poids>10.4</poids>
34      <description>skis</description>
35    </bagage>
36  </bagages>
37 </bagagerie>

```

et le schéma associé :

```

1 <?xml version="1.0"?>
2 <schema version="1.0"
3   xmlns="http://www.w3.org/2001/XMLSchema"
4   xmlns:bg="http://www.timc.fr/nicolas.glade/bagages"
5   targetNamespace="http://www.timc.fr/nicolas.glade/bagages"
6   elementFormDefault="qualified">
7   <include schemaLocation="bagages.xsd"/>
8
9   <element name="bagagerie" type="bg:Bagagerie"/>
10
11  <complexType name="Bagagerie">

```

```

12      <sequence>
13          <element name="passagers">
14              <complexType>
15                  <sequence>
16                      <element name="passager" type="bg:Passager" maxOccurs="
17                          unbounded"/>
18                  </sequence>
19              </complexType>
20          </element>
21          <element name="bagages">
22              <complexType>
23                  <sequence>
24                      <element name="bagage" type="bg:BagageHF" maxOccurs="
25                          unbounded"/>
26                  </sequence>
27              </complexType>
28          </element>
29      </sequence>
30  </complexType>
31  <complexType name="Passager">
32      <attribute name="nom" type="string" use="required"/>
33      <attribute name="prenom" type="string" use="required"/>
34      <attribute name="id" type="bg:IDPassager" use="required"/>
35  </complexType>
36
37  <complexType name="BagageHF">
38      <complexContent>
39          <extension base="bg:Bagage">
40              <sequence>
41                  <element name="description" type="string"/>
42              </sequence>
43              <attribute name="idPassager" type="bg:IDPassager"/>
44          </extension>
45      </complexContent>
46  </complexType>
47
48  <simpleType name="IDPassager">
49      <restriction base="string">
50          <pattern value="[A-Z]{2}[0-9]{8}"/>
51      </restriction>
52  </simpleType>
53
54 </schema>

```

Supposons que l'on souhaite connaître le nom de famille de tous les passagers dont les bagages hors format sont des skis.

D'abord, on commence par reformuler cette phrase sous cette forme : [Ce qu'on veut] [tel que] [la ou les conditions]. Cela donne dans notre cas : [Le nom de famille des passagers] [tel que] [les bagages sont des skis].

Mais si l'on regarde mieux, on voit que ce que l'on doit faire, c'est relier l'identifiant des bagages qui sont des skis, à l'identifiant passager. Donc on peut reformuler la condition ainsi : [les identifiants des bagages qui sont des skis]  $\Rightarrow$  [les identifiants des bagages] [tel que] [ce sont des skis]

On voit donc qu'on va devoir mettre en relation un 1er chemin (les noms des passagers) avec un deuxième chemin (les identifiants des bagages)

#### 4.b Exprimer ce que l'on veut et ce que sont les conditions

Que veut-on finalement ?  $\Rightarrow$  les noms des passagers ... donc on commence par écrire le chemin XPath qui donne les noms de tous les passagers : `//passager/@nom`. Comme attendu, on obtient la liste de tous les noms du document : { Gator, Nancière, Kass}

Pour les bagages, que veut-on ?  $\Rightarrow$  **les identifiants des bagages** ... donc on écrit cela sous la forme d'un autre chemin XPath : `//bagage/@idPassager` . Cette fois, on obtient la liste de tous les identifiants passagers dans les bagages : { OK17401023, OK17401023, AG18300427, SF23890311}. Vous remarquerez que l'on peut obtenir plusieurs fois le même identifiant.

Ensuite on se pose la question de comment réunir les deux. Ce qui lie les bagages aux passagers, ce sont les identifiants des passagers. Mais avant de réunir les deux chemins, on va tester des choses simples, par exemple des identifiants fixes. Reprenons les noms des passagers et ajoutons une sélection liée à l'identifiant : je veux les noms des passagers dont l'identifiant est X : `//passager[@id='AG18300427']/@nom` ce qui se lit *les noms des passagers tel que l'identifiant passager est AG18300427*. On obtient la liste contenant 1 seul nom (mais il s'agit bien d'une liste) : {Gator}

De même, pour les bagages, récupérons les identifiants des bagages qui sont des skis : `//bagage[contains(./description/text(),'skis')]/@idPassager` . On obtient la liste d'identifiant suivante : { OK17401023, OK17401023, SF23890311 }

Ensuite seulement, on peut relier les 2 chemins :

`//passager[@id=//bagage[contains(./description/text(),'skis')]/@idPassager]/@nom`, ce qui renvoie bien la liste de noms attendus { Nancière, Kass} qui sont les deux seules personnes à partir au ski en avion.

#### 4.c Procéder par petites étapes qu'on teste

De la même manière qu'on n'écrit pas un programme entier avant de le tester, on n'écrit pas une ligne complexe de XPath sans procéder par petites étapes. Chacune des étapes écrite au dessus doit être testée individuellement soit avec un outil en ligne (par exemple <http://xpather.com/> qui marche bien) soit avec un programme appelant en Java ou autre.

Autrement dit, vous devez tester dans l'ordre les expressions suivantes :

- `//passager/@nom`
- `//bagage/@idPassager`
- `//passager[@id='AG18300427']/@nom`
- `//bagage[contains(./description/text(),'skis')]/@idPassager`
- `//passager[@id=//bagage[contains(./description/text(),'skis')]/@idPassager]/@nom`