XML

bstorm
TRAINING

Table des matières

Chapitre 1 : Le XML, qu'est-ce que c'est ?

Chapitre 2 : Syntaxe de base

Chapitre 3 : Définition de documents

Chapitre 4 : XPath

LE XML, QU'EST-CE QUE C'EST ?

XML ?

- eXtensible Markup Language
  - Langage de balisage extensible
  - Balise : <NomBalise> ... </NomBalise>

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<Person>
    <Name>Asimov</Name>
    <FirstName>Isaac</FirstName>
    <DateOfBirth>
        <Day>2</Day>
        <Month>January</Month>
        <Year>1920</Year>
    </DateOfBirth>
</Person>
```

Objectif

- Stockage de documents

Permet un stockage structuré d'une information ou d'un ensemble d'informations

- Transmission de données

Permet une communication structurée entre des applications

Historique

- 1986 – SGML
  - Standard Generalized Markup Language
  - Destiné aux documentations techniques
  - Trop complexe à utiliser dans la plupart des circonstances

- 1991 – HTML
  - HyperText Markup Language
  - Version (extrêmement) simplifiée du SGML
  - Utilisation ciblée

Historique

- 1998 – XML 1.0
  - Entre SGML et HTML
  - Simplification du SGML
  - Généralisation de l'HTML
  - Résultat : Langage plus simple et utilisable en toutes circonstances

- 2004 – XML 1.1
  - Mise à jour
  - Prise en compte des caractères spéciaux

Utilité

- Bureautique
  - Possibilité de stocker des documents Word ou Excel au format XML
- Dessins vectoriels
  - Le format SVG (Inkscape notamment) est stocké au format XML
- Fichiers de configuration
- AJAX (Bien que la tendance aille vers le JSON)
- Base de données XML

Intérêts

- Séparation du contenu de de la présentation
  - De la même manière que l'HTML avec le CSS
  - L'HTML contient encore des balises de présentation
- Simplicité (lisible même par un non-initié)
- Généricité
- Structuration forte
- Format libre

Quelques termes

Balise : <NomBalise>

Balise ouvrante : <NomBalise>

Balise fermante : </NomBalise>

Attribut : <NomBalise Attribut=«Valeur »>

Contenu : Texte entre une balise ouvrante et une fermante.
Peut contenir d'autres balises

Élément : Balises + Attributs + Contenu

Vocabulaire : Ensemble des balises existantes

Dialecte : Désigne un vocabulaire précis

Quelques uns de ces termes seront développés plus loin

Extensibilité

- Une des principales caractéristiques du XML

- XML != HTML :
  - Balises non-fixées
  - Ce sont les auteurs qui fixent leurs balises (et qui créent donc leur vocabulaire)
    - Nécessité pour les auteurs de s'accorder sur ce vocabulaire

- Contrepartie : Cette liberté dans la création du vocabulaire implique la création de règles afin de garder les documents cohérents

SYNTAXE DE BASE

Analyse d'un exemple

Fichier : FirstDocument.xml

```
1  <!-- Ceci est un commentaire -->
2  <!-- Document réalisé dans le cadre du cours XML -->
3
4  <?xml version="1.0" encoding="UTF-8" ?>
5  <!-- Première ligne réel d'un document XML
6  On y définit la version du XML utilisée
7  Ainsi que l'encodage du texte (Généralement ISOXXX ou UTF-8) -->
8
9  <!DOCTYPE [...]>
10 <!-- DTD, cette partie est tronquée, elle sera vue par la suite -->
11
12 <Character key="character01" >
13 <!-- On déclare un élément "Person" dont la clé est "Character01" -->
14     <Name>Majere</Name>
15     <!-- Le nom du personnage est "Majere" -->
16     <FirstName>Raistlin</FirstName>
17     <!-- Le prénom du personnage est "Raistlin" -->
18     <Class>Wizard</Class>
19     <Level>48</Level>
20
21 </Character>
22 <!-- Chaque balise ouverte doit être fermée sans chevauchement ! -->
```

Prologue

- Le prologue rassemble les lignes avant le nœud principal du document (lignes 1=>8)
- Il contient l'en-tête (ligne 4)
- Et également la déclaration du type de document (ligne 9)
- Et quelques commentaires, juste pour le plaisir

En-tête

- Première ligne réelle d'un document XML
- Format <?xml [...] ?>, ce n'est pas une balise
- Trois attributs :
  - version : Précise la version du XML utilisée
    - Valeurs possibles : « 1.0 », « 1.1 »
  - encoding : Type de codage
    - Valeurs possibles : « iso-8859 », « utf-8 », « utf-16 »
  - standalone : Précise si le fichier est autonome
    - Valeurs possibles : « yes », « no »

Déclaration de type de document

- Toujours dans le prologue
- Format : <!DOCTYPE [...]>
- Permet de définir les règles (DTD) du documents
- Cette partie sera développée plus loin dans ce cours

Corps du document

- Le corps du document contient les éléments :
  - <FirstName>Raistlin</FirstName>
  - Cette ligne forme un élément nommé «FirstName» et dont le contenu est «Raistlin»

- Pas d'espace entre le chevron ouvrant (<) et le nom de la balise :
  - < FirstName> : Cette balise n'est pas correcte
  - <FirstName > : Celle-ci l'est, les espaces après le nom sont autorisés

- Certains caractères réservés ne sont pas utilisables (pour l'instant) dans le texte du contenu (<, >, &)

- En cas de contenu vide tel quel : <FirstName></FirstName>
  - On peut utiliser une balise « auto-fermante » : <FirstName />

Corps du document

- Imbrication des éléments

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<Person>
    <Name>Asimov</Name>
    <FirstName>Isaac</FirstName>
    <DateOfBirth>
        <Day>2</Day>
        <Month>January</Month>
        <Year>1920</Year>
    </DateOfBirth>
</Person>
```

- Un élément peut contenir d'autres éléments

- Pas de chevauchement !
    - <Person><Name>Asimov</Person></Name>

Corps du document

- Tout le corps du document est compris dans le contenu d'un élément racine (ou nœud principale).

- Il s'agit des balises englobant la totalité du contenu.

- Une seule racine par document !

Attributs

- Un attribut est un couple nom/valeur :
  - NomAttribut="ValeurAttribut"
- Les attributs se mettent dans la balise ouvrante (qui peut ne pas en contenir) :
  - <NomBalise Attribut1="Valeur1" Attribut2="Valeur2" >
- Ce qui ne change pas la balise fermante :
  - </NomBalise>
- Souvent utilisés pour les méta-données (données sur les données).
  - <date format="dd/mm/yyyy">01/03/1995</date>

Pour les noms XML

- Ils peuvent contenir des lettres/chiffres/caractères spéciaux (sauf ceux mentionnés plus haut)
- Ils doivent débuter par une lettre
- Ne pas commencer par les lettres « xml »
- Ne pas contenir d'espace

Exercice 1

- Réalisez un document XML représentant une liste de livres.
- Chaque livre a un titre, un auteur (juste le nom/prénom), un genre et une date de publication.
- Extra : Si le livre est le 2ième d'une série, on doit pouvoir retrouver le premier.

Exercice 2

- On veut créer un document XML structurant les données d'un annuaire téléphonique.

- L'annuaire doit comprendre au moins 2 personnes. Pour chaque personne, on souhaite connaître les informations suivantes :
  - Son sexe (homme ou femme).
  - Son nom.
  - Son prénom.
  - Son adresse.
  - Un ou plusieurs numéros de téléphone (téléphone portable, fixe, bureau, etc.).
  - Une ou plusieurs adresses e-mail (adresse personnelle, professionnelle, etc.).

DÉFINITION DE DOCUMENTS

Règles

- Afin de garder une cohérence dans le document → Règles

- Exemple :
  - Une balise <Date> doit contenir :
    - <Day>, <Month>, <Year>

- Règles → Modèles de documents

Règles

- Représentation de ces règles :
  - DTD (Document Type Définition)
  - Schémas

- Vérification d'un document XML :
  - Grâce aux modèles de documents
    - Notepad++ & XML Tool
    - Visual Studio Code & l'extension XML
    - XML Copy Editor

- Un document peut être bien formé mais invalide
  - Bien formé : Respect de la syntaxe XML
  - Valide : Respect des règles définies dans le DTD ou le schéma XML

DOCUMENT TYPE DEFINITION

Eléments

- Les règles s'appliquant aux éléments ont cette syntaxe :
  - <!ELEMENT balise (contenu)>
    - balise : on indique la balise concernée
    - contenu : on indique le contenu qu'elle doit avoir

- Exemples :
  - <!ELEMENT character (#PCDATA)>
    - La balise « character » doit contenir un ensemble de caractères (PCDATA = **P**arsed **C**haracter **DATA**)
    - Remarque : ne pas oublier les parenthèses autour de #PCDATA

Eléments

- Exemples :
  - <!ELEMENT character (name, firstname)>
    - La balise character DOIT contenir :
      une balise name (1 fois) ET une balise firstname (1 fois)

  - <!ELEMENT character (name | firstname)>
    - La balise character DOIT contenir
      une balise name (1 fois) OU une balise firstname (1 fois)

Remarque : les deux balises ne peuvent pas être présentes en même temps (~XOR)

Eléments

- Exemples :
  - <!ELEMENT character (name, firstname?)>
    - La balise character DOIT contenir
      une balise name (1 fois) et PEUT contenir une balise firstname (0 ou 1 fois)
  - <!ELEMENT character (name, firstname*)>
    - La balise character DOIT contenir
      une balise name (1 fois) et PEUT une balise firstname (0 ou plusieurs fois)
  - <!ELEMENT character (name, firstname+)>
    - La balise character DOIT contenir
      une balise name (1 fois) ET une balise firstname (1 ou plusieurs fois)

Eléments

- Exemples :
  - <!ELEMENT character EMPTY>
    - La balise character DOIT être vide
  - <!ELEMENT character ANY>
    - La balise character peut contenir n'importe quoi
      NB : l'utilisation de ANY est à éviter

Remarque : Ces deux mots-clés peuvent s'utiliser sans parenthèses

Attributs

- Evidemment les attributs ont aussi droit à leurs règles :
  - <!ATTLIST balise attribut (contenu) mode>
    - balise : on indique la balise concernée
    - attribut : on indique l'attribut concerné
    - contenu : on indique le contenu qu'elle doit avoir
    - mode : Permet d'indiquer si un élément est :
      - obligatoire (#REQUIRED)
      - facultatif (#IMPLIED)
      - fixé (#FIXED)

Attributs

- Exemples :
  - <!ATTLIST character sexe CDATA #REQUIRED>
    - La balise character DOIT contenir un attribut sexe qui doit contenir une chaîne de caractères (CDATA = Character DATA)
  - <!ATTLIST character sexe (h|f) #REQUIRED>
    - La balise character DOIT contenir un attribut sexe qui doit contenir soit "h" soit "f"
  - <!ATTLIST character sexe (h|f) #IMPLIED>
    - La balise character PEUT contenir un attribut sexe qui doit contenir soit "h", soit "f"

Attributs

- Exemples
  - <!ATTLIST character sexe CDATA #FIXED "h" >
    - La balise character DOIT contenir un attribut sexe qui doit contenir soit "h"
  - <!ATTLIST character key ID #REQUIRED>
    - La balise character DOIT contenir un attribut key qui doit être unique pour chaque balise character
  - <!ATTLIST character otherkey IDREF #IMPLIED>
    - La balise character PEUT contenir un attribut otherkey qui doit être correspondre à l'ID d'une des balises character

Emplacements

- Nous avons vu un certain nombre de règles jusqu'ici mais où les écrire ?
  - Soit directement dans le fichier XML :
    - <!DOCTYPE NoeudPrincipal [Règle1 Règle2 Règle3 ...]>
  - Soit dans un fichier séparé avec l'extension .dtd
    - Dans ce cas il faut référencer ce fichier dans le fichier XML
      - <!DOCTYPE NoeudPrincipal SYSTEM "Chemin" >
    - Il faut également préciser dans le prologue que le fichier n'est pas autonome
      - <?xml version="1.0" encoding="UTF-8" standalone="no"?>

Il vaut toujours mieux utiliser un fichier séparé mais on peut mettre temporairement les règles dans le fichier XML sans soucis

Exemple de DTD

- dates.xml :

```xml
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE dates SYSTEM "dates.dtd">
<dates>
  <date>
    <time format="12-hour">6:00PM</time>
    <day>17</day>
    <month>6</month>
    <year>2014</year>
  </date>
  <date>
    <time format="24-hour">17:00</time>
    <day>1</day>
    <month>1</month>
    <year>1970</year>
  </date>
  <date>
    <time format="24-hour">19:00</time>
    <day>25</day>
    <month>12</month>
    <year>0</year>
  </date>
</dates>
```

Exemple de DTD

- dates.dtd :

```
<!--Le noeud principal Dates peut contenir plusieurs Ã©lÃ©ments Date-->
<!ELEMENT dates (date+)>

<!--Chaque Ã©lÃ©ment Date est composÃ© des Ã©lÃ©ments Day, Month et Year-->
<!ELEMENT date (time, day, month, year)>

<!--L'Ã©lÃ©ment Time possÃ¨de un contenu non vide (#PCDATA)-->
<!ELEMENT time (#PCDATA)>

<!--Il possÃ¨de Ã©galement un attribut appelÃ© format qui indique le format
de l'heure. Cet attribut est obligatoire, et il peut uniquement prendre
les valeurs "12-hour" ou bien "24-hour"-->
<!ATTLIST time format (12-hour | 24-hour) #REQUIRED>

<!--L'Ã©lÃ©ment Day possÃ¨de un contenu non vide (#PCDATA)-->
<!ELEMENT day (#PCDATA)>

<!--L'Ã©lÃ©ment Month possÃ¨de un contenu non vide (#PCDATA)-->
<!ELEMENT month (#PCDATA)>

<!--L'Ã©lÃ©ment Year possÃ¨de un contenu non vide (#PCDATA)-->
<!ELEMENT year (#PCDATA)>
```

Exercice

- Prenez le fichier « listCharacterDTD.xml » et son fichier DTD associé :
  - Regardez si vous comprenez la structure du fichier et si vous en comprenez les règles
  - Regardez s'il est bien formé, sinon, corrigez-le
  - Regardez s'il est valide, sinon, corrigez-le
  - N'hésitez pas à l'améliorer

- Reprenez l'exercice 1 corrigé de la liste de livre (qui doit maintenant être bien formé) et écrivez toutes les règles nécessaires afin de le rendre valide

- De même pour l'exercice 2 sur l'annuaire téléphonique

Entités

- Les règles DTD apportent une notion jusqu'alors inconnue : les entités
- Une entité est comparable à une constante
  - Alias / Valeur
- On spécifie l'entité à un ou plusieurs endroits dans le fichier XML puis on écrit sa règle DTD
- (Il faut laisser la règle DTD dans le fichier XML et pas dans un fichier externe)

Entités

Les entités générales :

- Quelque part dans le fichier XML :
  - <class>&magic;</class> (magic est l'alias)
- Quelque part dans les règles :
  - <!ENTITY magic "Wizard" > (Wizard est la valeur)
- Dès lors, &magic; sera remplacé par "Wizard"
- Dans les entités générales, les alias sont précédés de & et suivi de ;

Entités

Avec les entités paramètres, les alias ne se trouvent pas dans le XML mais dans les règles

- Exemples :
  - <!ENTITY **% listClass** "class (Warrior|Wizard) #REQUIRED" >
  - <!ATTLIST character **%listClass;** >

- Résultat :
  - <!ATTLIST character class (Warrior|Wizard) #REQUIRED >

Attention : ne pas oublier l'espace entre % et listClass lors de la déclaration

Défauts

Les DTD (bien qu'encore très utilisés) ont quelques défauts :

- Nouveau format à apprendre (en plus du XML)
- Pas de typage :
  On peut préciser qu'un élément doit exister et s'il doit contenir quelque chose mais impossible de préciser si ce quelque chose doit être un int, un float, ...

SCHEMAS

Avantages

- Permet le typage des données
  - On peut préciser ce qui est un int, ce qui est du texte, ...

- Permet de définir des contraintes plus précises que les DTD

- XML : Les schémas sont également écrits en XML!
  Ça tombe bien, on le connaît !

Base d'un schéma

- Voila le contenu de base d'un schéma :

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
3  </xsd:schema>
```

- La première ligne n'est plus à présenter

- La deuxième ligne est déjà un peu particulière, elle contient l'attribut « xmlns:xsd » qui n'est pas un attribut comme les autres

Espace de noms

- Lors de la création de nos propres fichiers XML, on crée notre propre vocabulaire

- Pour créer un schéma standard, on va se servir de noms déjà définis, en important un vocabulaire ou espace de nom

- Les espaces de noms permettent également d'éviter des conflits de noms entre différents vocabulaires

Espace de noms

- Pour importer un espace de noms, il faut mettre comme attribut, dans le nœud principal :
  - xmlns="Namespace"(xmlns = XML NameSpace)

- Où on ajoutera un préfixe pour être sûr d'utiliser le nom venant de l'espace de noms "Namespace" :
  - xmlns:xsd= "Namespace"
  - Cela signifie qu'avant d'utiliser chaque nom de cette espace de noms, il faut utiliser le préfixe xsd

Espace de noms

- Dès lors, l'instruction xmlns:xsd="http://www.w3.org/2001/XMLSchema" permet de définir un espace de noms qui aura comme préfixe xsd

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
3  </xsd:schema>
```

- La balise "schema" faisant partie de cet espace de noms, il faudra la précéder du préfixe xsd

Espace de noms : remarques

- Le choix du préfixe est complètement arbitraire
- De même pour le nom de l'espace de noms : il n'est pas obligatoire de mettre un URL ; il faut juste que l'espace de nom soit identifié de manière unique par ce nom
- Si on ne précise pas de préfixe pour un espace de noms, celui-ci sera considéré comme l'espace de noms par défaut
- L'espace de nom est accessible pour l'élément qui déclare celui-ci, ainsi que tous ses sous-éléments

Espace de noms : portée

- NS1.xml :

```xml
<?xml version="1.0" encoding="UTF-8"?>
<html xmlns="http://www.w3.org/1999/xhtml">
  <!-- L'espace de noms par defaut est celui de XHTML -->
  <head>
    <title>Espaces de noms</title>
  </head>
  <body>
    <mml:math xmlns:mml="http://www.w3.org/1998/Math/MathML">
      <mml:apply>
        <mml:eq/>
      </mml:apply>
    </mml:math>
    <!-- L'espace de noms MathML n'est maintenant plus disponible -->
  </body>
</html>
```

Espace de noms : portée

- NS2.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<html xmlns="http://www.w3.org/1999/xhtml">
  <!-- L'espace de noms par defaut est celui de XHTML -->
  <head>
    <title>Espaces de noms</title>
  </head>
  <body>
    <math xmlns="http://www.w3.org/1998/Math/MathML">
      <!-- L'espace de noms par defaut est maintenant celui de MathML -->
      <apply>
        <eq/>
      </apply>
    </math>
    <!-- L'espace de noms par défaut est à nouveau celui de XHTML -->
  </body>
</html>
```

Espace de noms : portée

- NS3.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<html xmlns="http://www.w3.org/1999/xhtml">
  <!-- L'espace de noms par défaut est maitenant l'espace de noms XHTML -->
  <!-- Tous les éléments html, head, title, body, ... appartiennent
  à l'espace de noms XHTML qui est l'espace de noms par défaut. -->
  <head>
    <title>Espaces de noms</title>
  </head>
  <body>
    <name xmlns="">
      <!-- L'espace de noms par défaut n'est plus spécifié -->
      <!-- Les trois éléments name, firstname et surname
      n'appartiennent à aucun espace de noms. -->
      <firstname>Gaston</firstname>
      <surname>Lagaffe</surname>
    </name>
  </body>
</html>
```

Structure d'un schéma

- Pour rappel, voici la structure de base d'un schéma

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
3  </xsd:schema>
```

- Pour définir le schéma, nous utiliserons les éléments qui sont définis dans l'espace de nom (associé au préfixe xsd ici)

- Le noeud principal d'un schéma sera toujours <xsd:schema></xsd:schema>

- Extension du fichier : .xsd

Référence à un schéma

- Dans le document XML, on ajoutera une instruction pour faire référence au schéma

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <noeudPrincipal xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3  xsi:noNamespaceSchemaLocation="nomDuSchema.xsd">
4  </noeudPrincipal>
```

Pour cela, il faudra utiliser un autre espace de noms (associé au préfixe xsi ici)

Exemple de schéma

- Dates.xml

```xml
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<dates xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="dates.xsd">
  <date>
    <time format="12-hour">6:00PM</time>
    <day>17</day>
    <month>6</month>
    <year>2014</year>
  </date>
  <date>
    <time format="24-hour">17:00</time>
    <day>1</day>
    <month>1</month>
    <year>1970</year>
  </date>
  <date>
    <time format="24-hour">19:00</time>
    <day>25</day>
    <month>12</month>
    <year>0</year>
  </date>
</dates>
```

Exemple de schéma

- Dates.xsd

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <!--Type Day-->
  <xsd:simpleType name="Day">
    <xsd:restriction base="xsd:positiveInteger">
      <xsd:maxInclusive value="31"/>
    </xsd:restriction>
  </xsd:simpleType>
  <!--Type Month-->
  <xsd:simpleType name="Month">
    <xsd:restriction base="xsd:positiveInteger">
      <xsd:maxInclusive value="12"/>
    </xsd:restriction>
  </xsd:simpleType>
  <!--Balise dates-->
  <xsd:element name="dates">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="date" maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <!--Balise date-->
  <xsd:element name="date">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="time"/>
        <xsd:element name="day" type="Day"/>
        <xsd:element name="month" type="Month"/>
        <xsd:element name="year" type="xsd:int"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <!--Balise time-->
  <xsd:element name="time">
    <xsd:complexType mixed="true">
      <xsd:attribute name="format" type="xsd:string" use="required"/>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

Déclaration d'éléments

- TOUS les éléments qui apparaissent dans le document XML devront être déclarés dans le schéma

- Pour déclarer un élément, nous aurons besoin de définir son "type". Celui-ci va définir ce que cet élément pourra contenir

Type nommé

- Déclaration :
  <xsd:element name="nom" type="type"/>
  - nom est le nom de l'élément
  - type est un type nommé c'est-à-dire soit un type prédéfini, soit un type que nous avons défini nous-même (voir suite du plus loin)

- Liste complète des types prédéfinis :
  http://www.w3.org/TR/xmlschema-2/#built-in-primitive-datatypes

Type nommé

- Quelques exemples de types prédéfinis :
  - numérique : xsd:boolean, xsd:int, xsd:long, xsd:integer, xsd:float, xsd:double, xsd:decimal, ...
  - chaînes de caractères : xsd:string, ...
  - heures et dates :
    - xsd:time (format hh:mm:ss[.sss][TZ]), ex : 14:07:23, 14:07:23-07:00
    - xsd:date (format YYYY-MM-DD), ex : 2008-01-16
    - xsd:dateTime (format YYYY-MM-DDThh:mm:ss), ex : 2008-01-16T14:07:23
    - xsd:duration (format P[nY][nM][nDT][nH][nM][nS]), ex: P1Y6M, P1M12DT2H et P1YD3H10S
    - ...
- Exemples

```
1  <xsd:element name="nomPerso" type="xsd:string"/>
2  <xsd:element name="endurance" type="xsd:int"/>
```

.

Valeur par défaut et valeur fixe

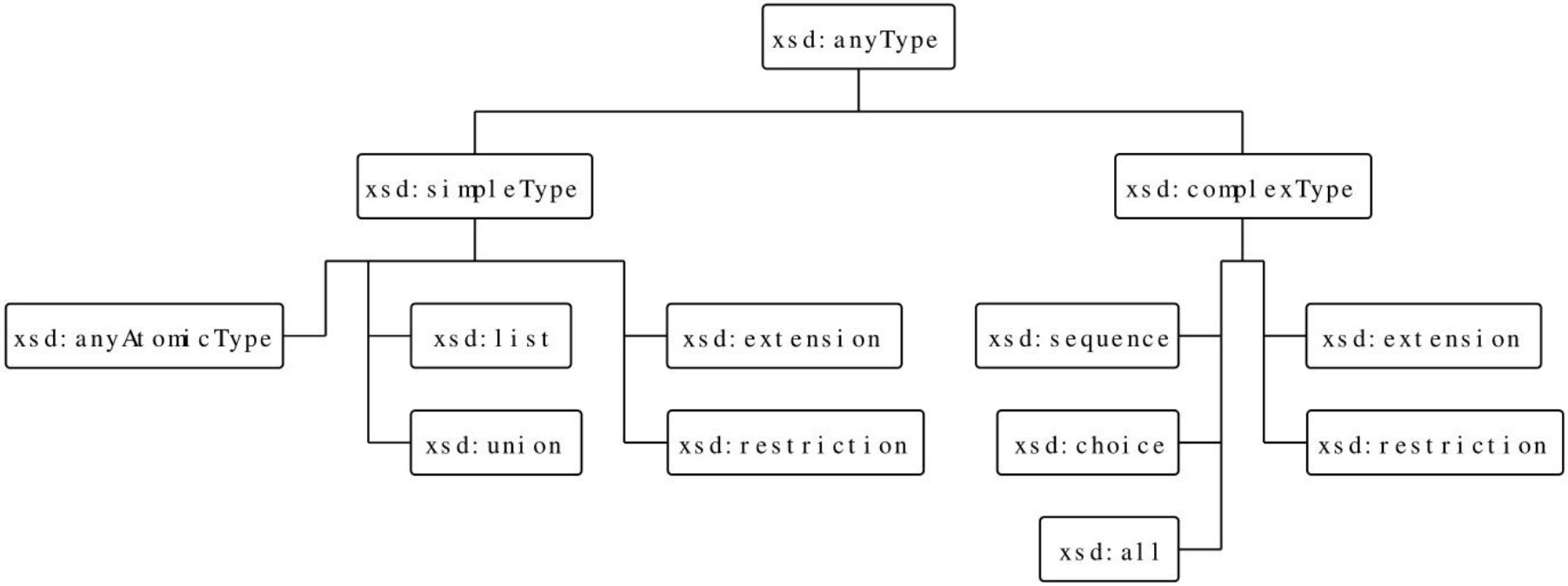
- Il est également possible de donner des valeurs par défaut avec l'attribut default, ainsi que des valeurs fixes avec l'attribut fixed

- Exemples

```
1  <xsd:element name="nomPerso" type="xsd:string" default="Nouveau Personnage"/>
2  <xsd:element name="endurance" type="xsd:int" fixed="100"/>
```

Définir son type

- Hiérarchie de construction des types



Types simples

- Les types simples (xsd:simpleType) définissent uniquement des contenus textuels. lls ne peuvent donc pas contenir d'autres éléments

- lls peuvent être utilisés pour les éléments et les attributs, et il est possible de modifier un type existant avec les éléments xsd:restriction et xsd:extension et l'attribut base (voir plus loin)

Types simples

- Exemple
  - xml

```xml
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<month xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="test1.xsd">
  5
</month>
```

  - xsd

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:simpleType name="Month">
    <xsd:restriction base="xsd:positiveInteger">
      <xsd:maxInclusive value="12"/>
    </xsd:restriction>
  </xsd:simpleType>
  <xsd:element name="month" type="Month"/>
</xsd:schema>
```

Type anonyme

- Au lieu d'utiliser un type nommé, on peut déclarer le type en même temps que l'élément

```
1  <xsd:element name="element">
2    <xsd:simpleType>
3      <!--Definition du type simple ici-->
4    </xsd:simpleType>
5  </xsd:element>
6
7
8
9  <xsd:element name="element">
10   <xsd:complexType>
11     <!--Definition du type complexe ici-->
12   </xsd:complexType>
13 </xsd:element>
```

Type anonyme

- Exemple précédent avec un type anonyme
  - xml

```xml
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<month xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="test1.xsd">
  5
</month>
```

  - xsd

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="month">
    <xsd:simpleType>
      <xsd:restriction base="xsd:positiveInteger">
        <xsd:maxInclusive value="12"/>
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:element>
</xsd:schema>
```

Types complexes

- Les types complexes (xsd:complexType) définissent des contenus purs (contient uniquement des sous-éléments), des contenus textuels ou des contenus mixtes
- Ils ne peuvent être utilisés que pour les éléments
- Comme les types simples, il est possible de dériver d'un type existant avec l'attribut base
- Pour le construire, il faut utiliser les opérateurs de séquence (xsd:sequence), de choix (xsd:choice) ou d'ensemble (xs:all) que nous verrons plus en détails juste après

Types complexes

- Exemple
  - xml

```xml
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<date xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="test1.xsd">
   <day>1</day>
   <month>1</month>
   <year>1970</year>
</date>
```

  - xsd

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="date">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="day" type="xsd:int"/>
        <xsd:element name="month" type="xsd:int"/>
        <xsd:element name="year" type="xsd:int"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

Opérateur de séquence

- L'opérateur de séquence xsd:sequence permet de former une suite d'éléments

```
1  <xsd:element name="date">
2    <xsd:complexType>
3      <xsd:sequence>
4        <xsd:element ref="time"/>
5        <xsd:element name="day" type="Day"/>
6        <xsd:element name="month" type="Month"/>
7        <xsd:element name="year" type="xsd:int"/>
8      </xsd:sequence>
9    </xsd:complexType>
10 </xsd:element>
```

- "Équivalent" DTD:

<!ELEMENT date (time, day, month, year)>

Opérateur de séquence

- Remarque : les éléments doivent apparaître dans le même ordre que celui défini dans l'élément xsd:sequence !

- Il est également possible de définir le nombre min et max d'apparition de chacun des éléments avec les attributs minOccurs et maxOccurs

```
24  <xsd:element name="date">
25    <xsd:complexType>
26      <xsd:sequence>
27      <!--La valeur unbounded signifie pas de limites-->
28        <xsd:element ref="time" minOccurs="0" maxOccurs="unbounded"/>
29        <xsd:element name="day" type="Day" minOccurs="2" maxOccurs="3"/>
30        <xsd:element name="month" type="Month" minOccurs="2"/>
31        <xsd:element name="year" type="xsd:int" maxOccurs="3"/>
32      </xsd:sequence>
33    </xsd:complexType>
34  </xsd:element>
```

Opérateur de choix

- Le contenu doit être l'un des éléments de l'élément xsd:choice

```
1 <xsd:element name="publication">
2   <xsd:complexType>
3     <xsd:choice>
4       <xsd:element ref="book"/>
5       <xsd:element ref="article"/>
6       <xsd:element ref="report"/>
7     </xsd:choice>
8   </xsd:complexType>
9 </xsd:element>
```

- "Équivalent" DTD:

<!ELEMENT publication (book | article | report)>

Opérateur de choix

- On peut également utiliser les attributs minOccurs et maxOccurs

```
24  <xsd:element name="date">
25    <xsd:complexType>
26      <xsd:choice>
27      <!--La valeur unbounded signifie pas de limites-->
28        <xsd:element ref="time" maxOccurs="unbounded"/>
29        <xsd:element name="day" type="Day" maxOccurs="unbounded"/>
30        <xsd:element name="month" type="Month" maxOccurs="unbounded"/>
31        <xsd:element name="year" type="xsd:int" maxOccurs="unbounded"/>
32      </xsd:choice>
33    </xsd:complexType>
34  </xsd:element>
```

Attention : cela signifie que l'on peut avoir plusieurs éléments time OU plusieurs éléments day OU plusieurs éléments month OU plusieurs éléments year

Imbrication

- On peut imbriquer plusieurs types. Que signifie le bout de schéma suivant ?

```
<xsd:element name="book" minOccurs="1" maxOccurs="unbounded">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="title" type="xsd:string"/>
      <xsd:choice>
        <xsd:element name="author" type="xsd:string"/>
        <xsd:element name="authors">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="author" type="xsd:string"
              minOccurs="2" maxOccurs="unbounded"/>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
      </xsd:choice>
      <xsd:element name="year" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Opérateur d'ensemble

- L'opérateur d'ensemble est donné par l'élément xsd:all
- Il définit une liste d'éléments, chacun devant apparaître 1 fois (ou entre 0 et 1 fois si minOccurs="0")
- Le maxOccurs ne peut pas être différent de 1
- L'ordre d'apparition des éléments n'a pas d'importance
- On ne peut pas imbriquer les opérateur xsd:all dans d'autres constructeurs (xsd:sequence, xsd:choice et xsd:all)

Opérateur d'ensemble

- Exemple

```
1  <xsd:element name="book">
2    <xsd:complexType>
3      <xsd:all minOccurs="0">
4        <xsd:element name="title" type="xsd:string"/>
5        <xsd:element name="author" type="xsd:string"/>
6        <xsd:element name="year" type="xsd:string"/>
7        <xsd:element name="publisher" type="xsd:string"/>
8      </xsd:all>
9    </xsd:complexType>
10 </xsd:element>
```

Référencer un élément global

- Un élément global est un élément qui est enfant direct de l'élément xsd:schema
- Il est possible de référencer un élément global déjà défini avec l'attribut ref
- Exemple
  - xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<person xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="test1.xsd">
  <date>
    <day>1</day>
    <month>1</month>
    <year>1970</year>
  </date>
</person>
```

Référencer un élément global

- Exemple
  - xsd

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="date">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="day" type="xsd:int"/>
        <xsd:element name="month" type="xsd:int"/>
        <xsd:element name="year" type="xsd:int"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="person">
    <xsd:complexType mixed="true">
      <xsd:sequence>
        <xsd:element ref="date"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

Éléments locaux

- Il est possible de définir deux éléments ayant le même nom, mais des types différents pour autant qu'ils ne soient pas globaux

- Exemple
  - xml

```xml
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
<lists xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="test1.xsd">
  <strings>
    <local>Une chaîne</local>
    <local>A string</local>
  </strings>
  <integers>
    <local>-1</local>
    <local>1</local>
  </integers>
</lists>
```

Éléments locaux

- Exemple
  - xsd

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="strings">
    <xsd:complexType>
      <xsd:sequence>
        <!-- Déclaration du premier élément local -->
        <xsd:element name="local" type="xsd:string" maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="integers">
    <xsd:complexType>
      <xsd:sequence>
        <!-- Déclaration du second élément local -->
        <xsd:element name="local" type="xsd:integer" maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="lists">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="strings"/>
        <xsd:element ref="integers"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

Contenu mixte

- Le contenu d'un élément est pur s'il contient uniquement des sous-éléments
- Le contenu d'un élément est mixte s'il contient à la fois des sous-éléments et du contenu textuel
- Il faut alors rajouter l'attribut mixed="true" pour les types complexes
- Remarque : le cas ne se présente pas pour les types simples étant donné qu'il ne peuvent pas contenir de sous-éléments

Contenu mixte

- Exemple
  - xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<person xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="test1.xsd">
Prénom :<firstname>Albert</firstname>,
Nom :<lastname>Einstein</lastname>.
</person>
```

  - xsd

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="person">
    <xsd:complexType mixed="true">
      <xsd:sequence>
        <xsd:element name="firstname" type="xsd:string"/>
        <xsd:element name="lastname" type="xsd:string"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

Déclaration d’attributs

- Les attributs sont toujours des types simples (xsd:simpleType) car ils ne contiennent que du contenu textuel

- Exemple de déclaration
  <xsd:attribute name=“nom" type="type"/>
  - nom est le nom de l’attribut
  - type est son type (type nommé)

Déclaration d'attributs

- Exemple (avec un attribut age)
  - xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<person xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="test1.xsd" age="15">
Prénom :<firstname>Albert</firstname>,
Nom :<lastname>Einstein</lastname>.
</person>
```

  - xsd

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="person">
    <xsd:complexType mixed="true">
      <xsd:sequence>
        <xsd:element name="firstname" type="xsd:string"/>
        <xsd:element name="lastname" type="xsd:string"/>
      </xsd:sequence>
      <xsd:attribute name="age" type="xsd:int"/>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

Déclaration d'attributs

- Ils peuvent également être déclarés avec un type que nous avons défini (toujours xsd:simpleType pour les attributs !)

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:simpleType name="Age">
    <xsd:restriction base="xsd:positiveInteger">
      <xsd:maxInclusive value="150"/>
    </xsd:restriction>
  </xsd:simpleType>
  <xsd:element name="person">
    <xsd:complexType mixed="true">
      <xsd:sequence>
        <xsd:element name="firstname" type="xsd:string"/>
        <xsd:element name="lastname" type="xsd:string"/>
      </xsd:sequence>
      <xsd:attribute name="age" type="Age">
      </xsd:attribute>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

Déclaration d'attributs

- Même remarque avec un type anonyme

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="person">
    <xsd:complexType mixed="true">
      <xsd:sequence>
        <xsd:element name="firstname" type="xsd:string"/>
        <xsd:element name="lastname" type="xsd:string"/>
      </xsd:sequence>
      <xsd:attribute name="age">
        <xsd:simpleType>
          <xsd:restriction base="xsd:positiveInteger">
            <xsd:maxInclusive value="150"/>
          </xsd:restriction>
        </xsd:simpleType>
      </xsd:attribute>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

Déclaration d'attributs

- Grâce à l'attribut use, on peut préciser si l'attribut obligatoire (required)

- On peut aussi donner des valeurs par défaut ou des valeurs fixes

```
1  <xsd:attribute name="id" type="xsd:ID" use="required"/>
2  <xsd:attribute name="NomPerso" type="xsd:string" default="Nouveau Perso"/>
3  <xsd:attribute name="Force" type="xsd:int" fixed="100"/>
```

Extension de types

- Extension ~ héritage en orienté objet : rajouter des attributs ou des éléments
- Utilise l'élément xsd:extension
- Trois cas :
  - Types simples
  - (Types complexes à contenu simple)
  - (Types complexes à contenu complexe)

Types simples

- Rappel : type simple = uniquement contenu textuel
- Avec l'extension on peut uniquement ajouter des attributs au type simple
- Le type obtenu est alors un type complexe à contenu simple
- Remarque : ce type simple peut être un type prédéfini ou bien un type qui nous avons défini (simpleType)

Types simples

- Exemple (extension de xsd:decimal)
  - xml (partiel)

```xml
<price currency="euro">3.14</price>
```

  - xsd (partiel)

```xml
<xsd:complexType name="Price">
  <xsd:simpleContent>
    <xsd:extension base="xsd:decimal">
      <!-- Attribut ajouté -->
      <xsd:attribute name="currency" type="xsd:string"/>
    </xsd:extension>
  </xsd:simpleContent>
</xsd:complexType>
<xsd:element name="price" type="Price"/>
```

Types complexes

- Si un type complexe est obtenu par extension ou restriction d'un autre type, il faut rajouter l'éléments xsd:simpleContent si le contenu est purement textuel, et l' élément xsd:complexContent sinon

- Exemples

```
<!-- Type dérivé à contenu textuel -->
<xsd:complexType ...>
  <xsd:simpleContent>
    <!-- Extension ou restriction -->
  </xsd:simpleContent>
</xsd:complexType>


<!-- Type dérivé à contenu pur ou mixte -->
<xsd:complexType ...>
  <xsd:complexContent>
    <!-- Extension ou restriction -->
  </xsd:complexContent>
</xsd:complexType>
```

Restriction de types

- Restriction : on dérive d'un type de base pour rajouter des restrictions grâce à l'élément **xsd:restriction** et on utilise l'attribut base pour spécifier de quel type on dérive

- Types simples
  - Restriction par intervalle
  - Restriction par énumération

- (Types complexes à contenu simple)

- (Types complexes à contenu complexe)

Restriction de types

- Pour les types simples, l'élément **xsd:restriction** est enfant direct de l'élément **xsd:simpleType**

- Pour les types complexes :
  - enfant direct de **xsd:simpleContent** si contenu simple (= uniquement contenu textuel + attributs éventuels)

  - enfant direct de **xsd:complexContent** si contenu complexe (= contenu textuel + sous-éléments + attributs éventuels)

Type simple : restriction par intervalle

- On utilise les éléments (chacun prend un attribut value)
  - xsd:minInclusive : valeur minimale inclusive
  - xsd:minExclusive : valeur minimale exclusive
  - xsd:maxInclusive : valeur maximale inclusive
  - xsd:maxExclusive : valeur maximale exclusive

- Exemple pour un élément (type simple)

```
<xsd:element name="year">
  <xsd:simpleType>
    <xsd:restriction base="xsd:integer">
      <xsd:minInclusive value="1970"/>
      <xsd:maxInclusive value="2050"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>
```

Type simple : restriction par intervalle

- Exemple pour un attribut (toujours type simple pour attribut)

```
<xsd:attribute name="date">
  <xsd:simpleType>
    <xsd:restriction base="xsd:date">
      <!-- Date après le 31 décembre 2001 exclus -->
      <xsd:minExclusive value="2001-12-31"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:attribute>
```

Type simple : restriction par énumération

- On utilise l'élément xsd:enumeration (et l'attribut value)

```
<xsd:simpleType name="PhoneType">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="home"/>
    <xsd:enumeration value="work"/>
  </xsd:restriction>
</xsd:simpleType>
<xsd:element name="phone">
  <xsd:complexType>
    <xsd:simpleContent>
      <xsd:extension base="xsd:string">
        <xsd:attribute name="type"/>
      </xsd:extension>
    </xsd:simpleContent>
  </xsd:complexType>
</xsd:element>
```

Type simple : autres restrictions

- Quelques autres éléments (chacun avec un attribut value) que l'on peut mettre dans xsd:restriction :
  - xsd:length, xsd:minLengthet xsd:maxLength : restriction sur la taille des strings
  - xsd:fractionDigitset xsd:totalDigits : nombre de chiffres après la virgule, et nombre total de chiffres respectivement
  - xsd:pattern : pour définir une expression régulière (regexp)

Exercices

- Reprenez l'exercice 1 de la liste de livre et écrivez le schéma le plus complet possible

- De même pour l'exercice 2 sur l'annuaire téléphonique

XPATH

XPATH

- XPATH (XML PATH = Chemin XML)

- Technologie permettant d'extraire des données d'un documents XML :
  - Elements
  - Attributs
  - Commentaires
  - ...

Arbre

- Il faut considérer le document XML comme un arbre :
  - Parent : Nœud directement au dessus du nœud courant
  - Enfants : Nœuds directement en dessous du nœud courant
  - Descendants : Tous les nœuds en dessous du nœud courant (peu importe le niveau)
  - Ancêtres : Tous les nœuds au dessus du nœud courant peu importe le niveau
  - Frères : Nœuds se trouvant sur le même niveau

Recherche par étape

- On parcourt l'arbre par étapes successives
  - /child::listCharacter/Character[attribute::key="character02"]/FirstName

- On peut avoir autant d'étapes que nécessaire

Type de chemin

Le type de chemin départ du départ :

- Chemin absolu :
  Le nœud de départ est la racine de l'arbre XML (Commence par « / »)

- Chemin relatif :
  Le nœud de départ peut être n'importe quel nœud de l'arbre XML

Etape

- Une étape est décomposée en trois partie :
  - Axe
  - Nœud
  - Prédicats

- Le format d'une étape :
  - Axe::nœud[prédicat][prédicat]...

Axe

- Détermine le sens de la recherche (haut, bas, niveau, ...) :
    - Parmi les valeurs possibles on retrouvera notamment :
      ancestor, attribute, child, descendant, parent, preceding, following, self

Noeuds

- Une fois la « direction » de la recherche définie, on va déterminer l'objet de la recherche :

- Parmi les valeurs possibles :
  - Le nom d'un nœud précis : indique quel nœud doit être recherché
  - * : recherche dans tous les types de nœuds
  - node() : recherche dans tous les nœuds contenant simplement du texte
  - Comment() : recherche vers les nœuds de commentaire

Prédicat

- Permet d'affiner la recherche :
  - Il peut y en avoir plusieurs à la suite mais également aucun

- Type de prédicat :
  - attribute : permet une recherche en fonction de l'attribut
  - count() : permet de compter le nombre de nœud
  - last() : permet de sélectionner le dernier nœud
  - position() : permet de sélectionner un nœud en fonction de sa position

Références

- L'essentiel de XML – Olivier Carton : http://www.liafa.jussieu.fr/~carton/Enseignement/XML/Cours/support.pdf

- Lien intéressant : http://fr.openclassrooms.com/informatique/cours/structurez-vos-donnees-avec-xml

Merci pour votre attention.

bstorm TRAINING