



X.M.L.

Extensible Markup Language



XML – Introduction

- Syntaxe générique de format de données
 - Site Web
 - Echange de données
 - Dessins vectoriels
 - Représentation arborescente
 - ...
- Avantages
 - Méta-langage document texte
 - Chaînes de caractères délimitées par des balises
 - Balisage illimité
 - Svt contraintes métier
 - ...
- XML n'est pas
 - Un langage de programmation
 - Un protocole de transport
 - Une base de données



XML – Introduction

- **Portabilité**

- ✓ Réelle solution inter plate-forme
- ✓ Format de données pur (par marquage)
- ✓ Non propriétaire



XML – Fonctionnement

- **L'utilisation d'un parseur permet :**

- ✓ Séparation du document en éléments distincts
 - o Attributs
 - o Autres parties
- ✓ Assure le respect des règles XML
 - o Indique les erreurs
 - o Arrête le traitement

- **Utilisation d'une DTD (Définition de Type de Document)**

- ✓ Permet le respect de contraintes et règles

- **Applications recevant les données du parseur**

- ✓ Navigateur Web , Traitement de texte, Base de données,
Editeur graphique, Tableur, logiciel de gestion, programme de syndication,

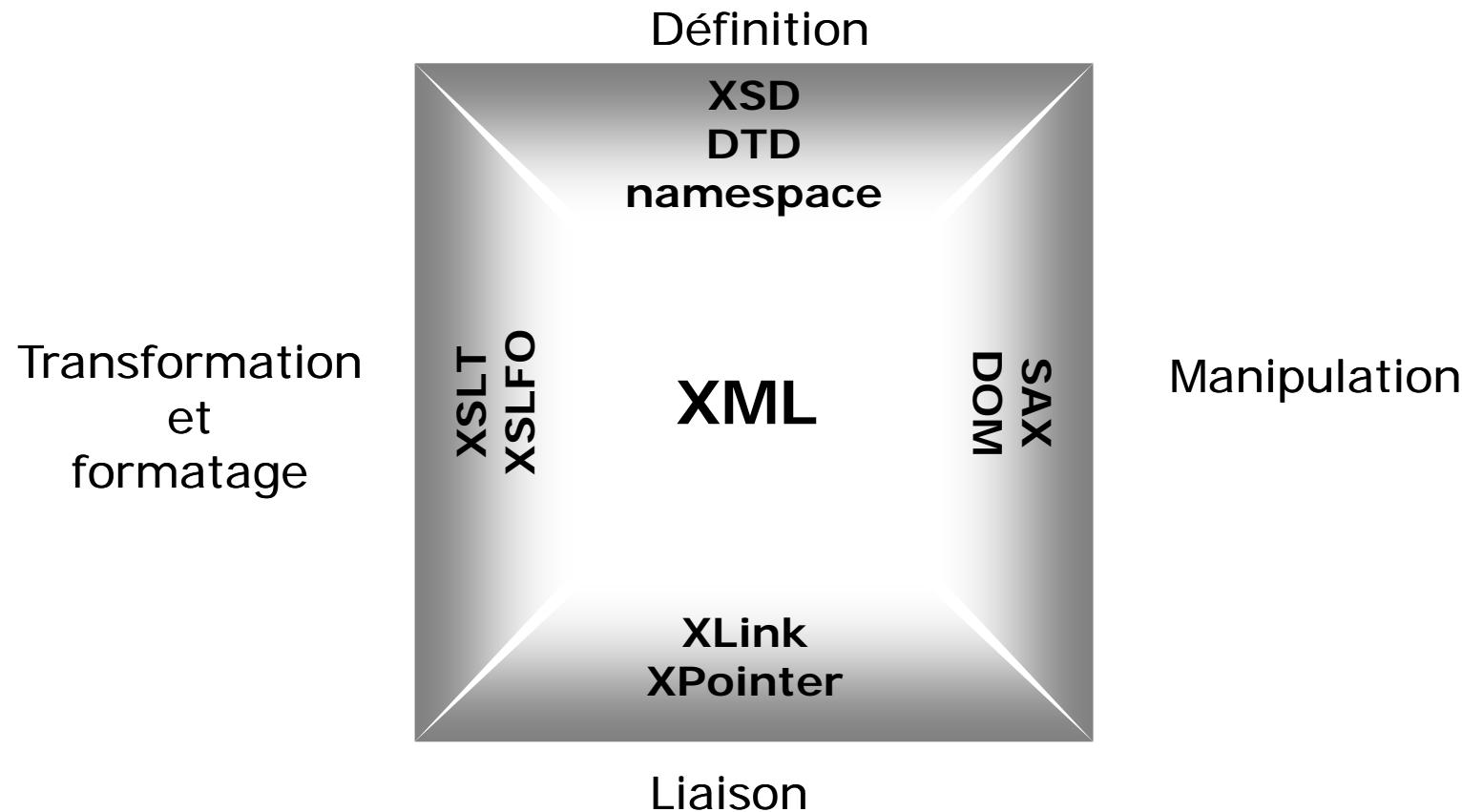
...

N'importe quel programme dans n'importe quel langage



XML – Fonctions

- Langage normalisé de structuration de l'information (W3C)
- Utilisé par n'importe quel langage





Principes de XML



XML – Documents & fichiers

- **Document XML**

- ✓ Document texte (pas de binaire)
- ✓ Document bien formé
 - o Respect de la syntaxe et des règles XML
- ✓ Nom de fichier suivant le parseur utilisé

```
<person>
```

Alan Turing

```
</person>
```

Personne.xml

- **Balises, élément**

- ✓ Début <nom_balise>
- ✓ Contenu de l'élément
- ✓ Fin </nom_balise>
- ✓ Le nom de la balise reflète son contenu
- ✓ Toujours une balise de début, avant une balise de fin
 - o Différent de HTML

- **Sensible à la casse**

- ✓ <Personne> différent de <PERSONNE>



XML – Arbre

personne.xml

- **Imbrication des balises (infini)**

- ✓ Définit une hiérarchie

- **Parent et enfants**

- ✓ Pour tous les éléments :
 - o 1 parent, n enfants
 - o Sauf élément racine
- ✓ Enfants inclus dans le parent

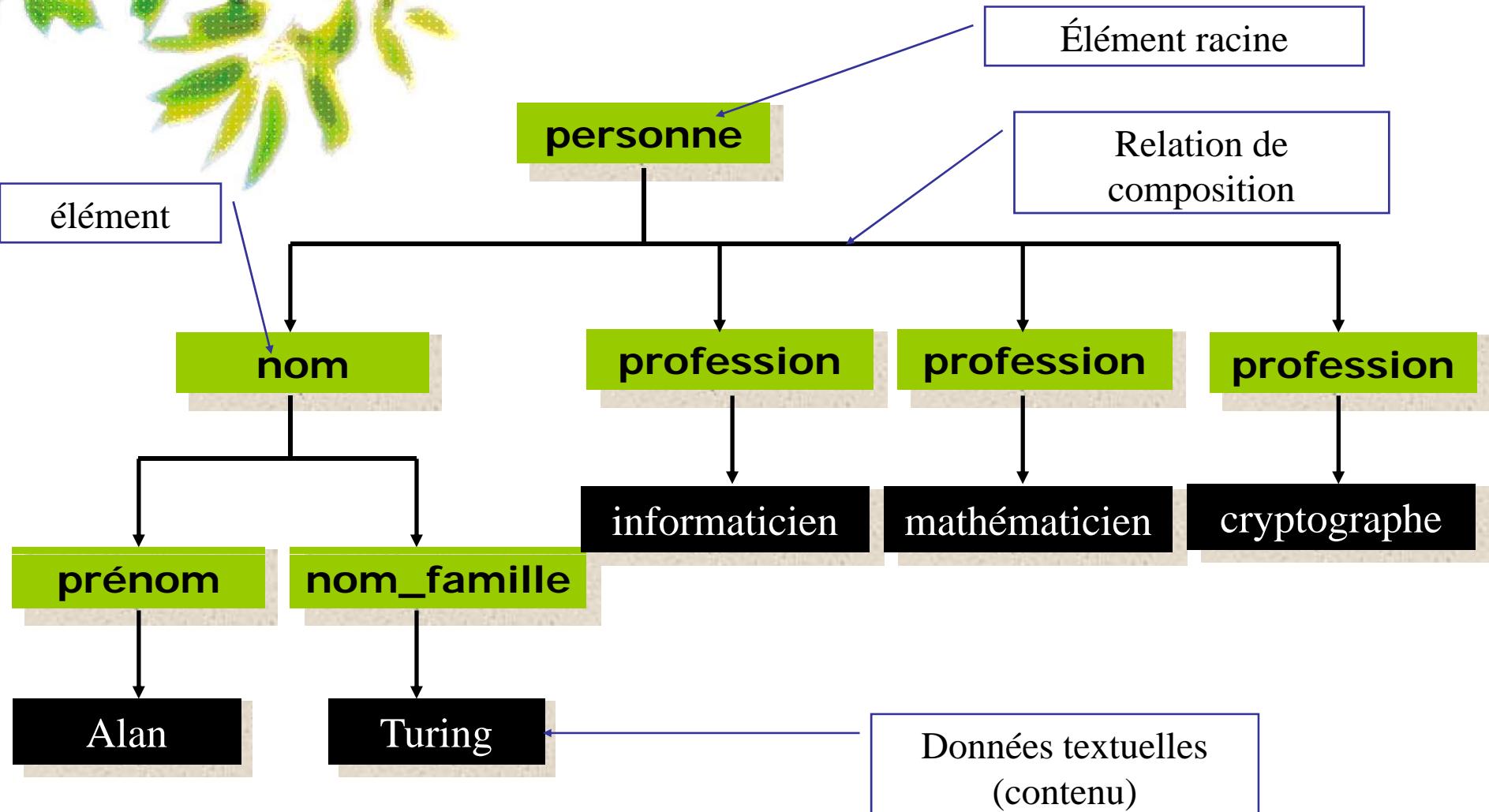
- **Elément racine / élément document**

- ✓ Seul élément sans parent
- ✓ Premier élément du document
- ✓ Contient tous les autres éléments

```
<personne>
  <nom>
    <prénom>Alan</prénom>
    <nom_famille>Turing</nom_famille>
  </nom>
  <profession>informaticien</profession>
  <profession>mathématicien</profession>
  <profession>cryptographe</profession>
</personne>
```



XML – Arbre





XML – Attribut

- **Paire nom-valeur balise de début**

- ✓ nom séparé de valeur par =
- ✓ valeur entourée par " ou '
- ✓ Exemple :

```
<personne naissance="23-06-1912" mort="07-06-1954">
```

Ou

```
<personne naissance='23-06-1912' mort='07-06-1954'>
```



XML – Noms XML

- **Même règles pour les différents objets**

- ✓ nom d'élément
- ✓ nom d'attribut
- ✓ Autres structures XML

- **Les noms peuvent contenir**

- ✓ N'importe quel caractères alphanumérique, ou chiffre standard ou non standard
- ✓ Les caractères de ponctuation suivants :
 - ✓ _ souligné - trait d'union . point)

- **Les noms ne peuvent pas contenir**

- ✓ les autres caractères de ponctuation
- ✓ de blancs
- ✓ La chaîne de caractère XML, xml ... (réservée)

- **Les noms commencent**

- ✓ lettre, symbole ou _
- ✓ leur longueur n'est pas limitée



XML – Appel d'entité

- Utilisation des caractères réservés pour balisage

<	→	<
&	→	&amp
>	→	>
"	→	"
'	→	'



XML – Section CDATA

- Permet le traitement de données textuelles brutes, non interprétables

- ✓ Balise :

```
<![CDATA[. . . . .]]>
```

- ✓ Exemple :

```
<test>
<p>Vous pouvez utiliser un attribut <code>xmlns</code> par defaut
pour ne pas avoir a ajouter le prefixe svg a tous vos elements: </p>
<![CDATA[
    <svg xmlns="http://www.w3.org/2000/svg"
        width="12cm" height="10cm">
        <ellipse rx="110" ry="130" cx="1cm" cy="1cm" />
        <rect x="4cm" y="1cm" width="3cm" height="6cm" />
    </svg>
]]>
</test>
```



XML – Commentaires

- **Similaires à HTML**

- ✓ Balise :

```
<!--. . . . . -->
```

- ✓ Exemple :

```
<!-- J'ai besoin de vérifier et de mettre à jour ces liens dès que j'en  
aurai l'occasion -->
```

- ✓ Peuvent apparaître n'importe où dans le document XML (avant ou après l'élément racine)
 - ✓ Ne peuvent apparaître à l'intérieur des balises
 - ✓ Ne peuvent apparaître dans un autre commentaire
 - ✓ Utilisable éventuellement par les parseurs (documentation)



XML – Instructions de traitement

- **Moyen de fournir des informations aux applications auxquelles sont destinées le document XML**

✓ Balise :

<? ?>

- ✓ tout de suite après <? , se trouve la cible, elle peut être :
- ✓ Nom de l'application destinatrice
 - ✓ Identifiant d'une instruction de traitement

✓ Exemple :

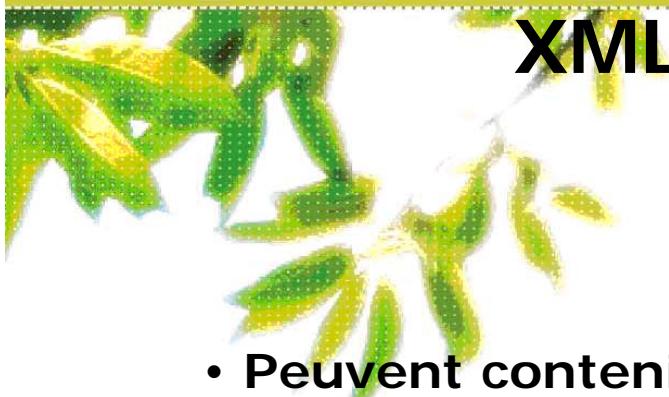
En HTML, utilisation balise META Robots pour indiquer aux moteurs de recherche ou robots s'ils doivent indexer une page et comment le faire
Équivalence XML :

<?Robots index="yes" follow=no"?>

Cible : Robots

Attributs : index, follow, précisent :

- ✓ La page doit être indexée
- ✓ Non exploration des liens relatifs au document (si yes, exploration)



XML – Instructions de traitement

- **Peuvent contenir une quantité de texte illimitée**
 - ✓ Utilisé par langages comme php, intégration de portions de code
- **Ce sont des balises, pas des éléments**
 - ✓ Même règles que les commentaires
- **La plus fréquente : xmlstylesheet**
 - ✓ Permet la liaison des feuilles de styles CSS à un document
 - ✓ Exemple :

```
<?xmlstylesheet href="person.css" type="text/css"?>
<person>
    Alan Turing
</person>
```



XML – Déclarations

- **Les documents peuvent commencer par une déclaration :**

- ✓ Pas obligatoire
- ✓ Idem instruction de traitement
- ✓ Contenu :
 - ✓ nom xml
 - ✓ attributs :
 - o version, version XML
 - o type de codage, type codage du jeu de caractères
 - o standalone, permet la lecture de la DTD
 - no : la DTD est dans un fichier autre que celui qui est lu
 - yes : DTD interne
- ✓ Si non indiqué :
 - ✓ Version 1.0
 - ✓ type de codage, Unicode affecté, mais utilisation par le parseur des premiers octets pour définir type.
 - ✓ standalone=no

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
<personne>
    Alan Turing
</personne>
```



XML – Vérification contraintes de forme

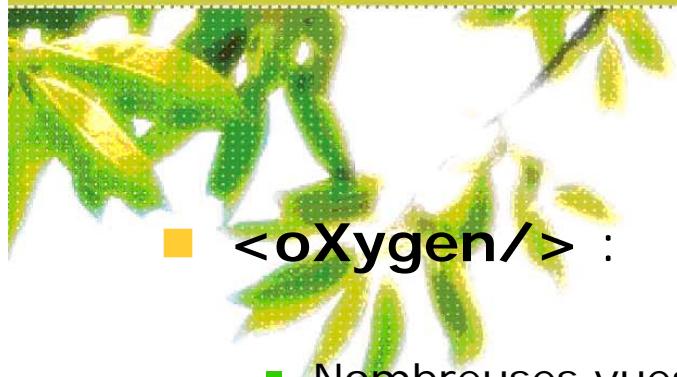
- **Document XML → Document Bien Formé, règles :**

- ✓ à chaque balise de début doit correspondre une balise de fin
- ✓ les éléments peuvent être imbriqués, ils ne doivent pas se recouvrir;
- ✓ il ne doit y avoir qu'un seul élément racine;
- ✓ les valeurs des attributs doivent être entre guillemets;
- ✓ un élément ne doit pas avoir deux attributs avec le même nom;
- ✓ les commentaires et instructions de traitement ne doivent pas apparaître à l'intérieur de balises;
- ✓ aucun caractère < ou & non échappé ne doit apparaître dans les données textuelles d'un élément ou d'un attribut.



Editeurs X.M.L.

Commerciaux



Editeurs XML - Commerciaux



■ <oxygen/> :

- Nombreuses vues : éditeur de source, débogueur XSLT
- Visualisation/édition d'arbre
- Edition et validation DTD, XML Schema
- Différence/fusion XML
- Support XQuery, XSLT, XPath
- Support Xalan, Saxon, MSXML, XSLTProc

■ Plates-formes :

- Windows
- Mac OS X
- Linux
- Unix
- Java
- plugin Eclipse (requiert Java)



Editeurs XML - Commerciaux

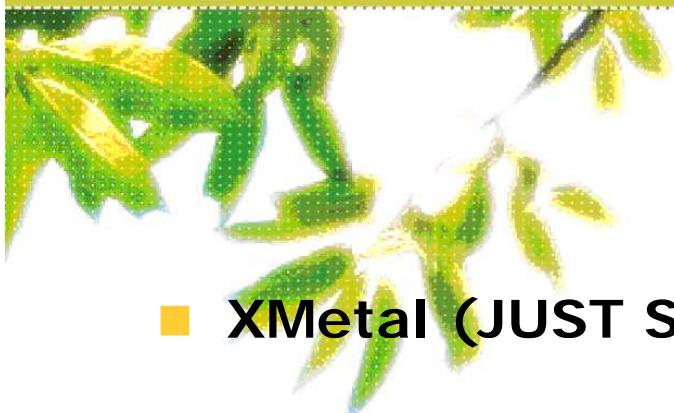


■ XML Spy (Altova) :

- Editeur de DTD, WSDL,
- Schema, XPath,
- XQuery, SOAP
- Complétion de code
- Liaison à une base de données
- Support Java et .NET

■ Plates-formes :

- Windows
- plugin Eclipse (requiert Java)
- Module visual Studio .NET



Editeurs XML - Commerciaux

JUST.
SYSTEMS

■ XMetal (JUST SYSTEM) :

- Support XPath,
- MathML,
- SGML,
- DOM, feuilles de style...
- Scripts utilisateur
- Intégration CMS

■ Plates-formes :

- Windows ActiveX requiert .NET



Editeurs XML - Commerciaux

Progress. | DataDirect.

■ Stylus Studio (DATADIRECT) :

- Support SOAP,
- Schema, DTD,
- XHTML, XML Mapping...
- Support Java et .NET
- Intégration à un SGBD
- Outils services Web et conversion de l'existant

■ Plates-formes :

- Windows



Editeurs XML - Commerciaux



■ Exchanger (Cladonia) :

- Editeurs Schema, RelaxNG, DTD
- Débogueur XSLT
- Support XPath, XQuery, WSDL, XML Signature, XSL:FO,
- MathML, DocBook...
- Outils WebDAV et FTP

■ Plates-formes :

- Windows
- Linux
- OS X
- Unix



Editeurs XML - Commerciaux

**TOPOLOGI
XML JUDGE**

- **XML Judge (Topologi) :**

- Editeurs Schema, RelaxNG, DTD
- Support SGML

- **Plates-formes :**

- Windows java JRE 1.4, 1.5



Editeurs XML - Commerciaux



■ XML Writer (Wattle Software) :

- Utilise MSXML (**Microsoft XML Core Services (MSXML)**)
- Utilise MSXML
- Support DTD, XSD Schema, XSLT
- Assistance intelligente
- Exemples de code
- Conversion de format
-
- **Plates-formes :**
 - Windows



Editeurs X.M.L.

Open Source



Editeurs XML – Open Source



Vex

A Visual Editor for XML

■ VEX – A Visual Editor for Xml :

- Licence LGPL
- Largement extensible
- Intégration Eclipse
- Outils WebDAV et FTP
- Support DTD et CSS

- Plates-formes :
 - Windows
 - Linux
 - Plugin Eclipse

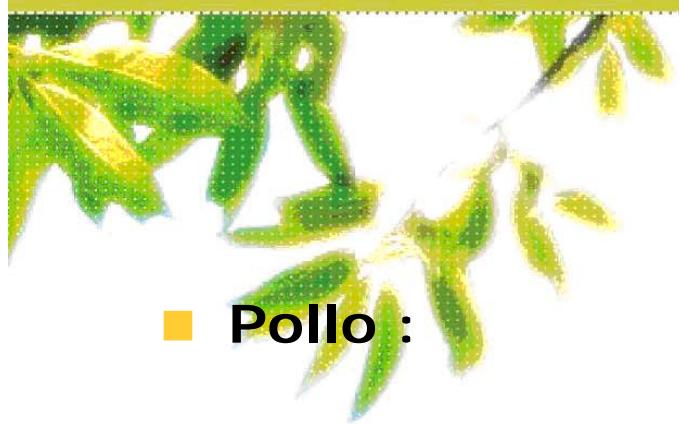


Editeurs XML – Open Source

BXE

■ BitFlux Editor (BXE) :

- Licence Apache
 - Support RelaxNG, CSS...
 - Système de plugins
-
- Plates-formes :
 - Toutes (requiert Mozilla/Firefox)



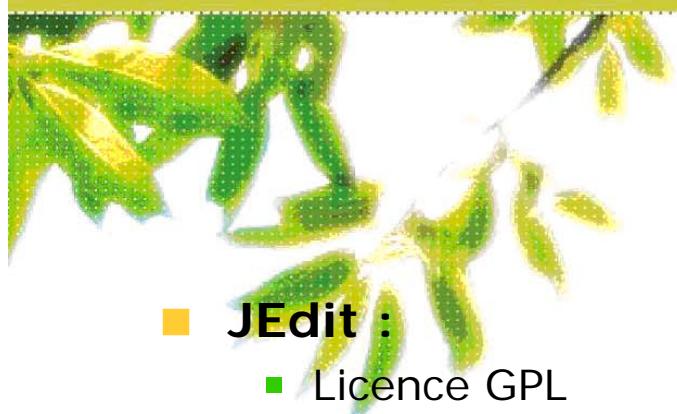
Editeurs XML – Open Source

■ Pollo :

- Licence MIT
- Visualisation/édition d'arbre
- Support XPath, Schema, DTD, RelaxNG...
- Glisser/déposer

■ Plates-formes :

- Windows, OS X, Linux, Unix (requiert Java)



Editeurs XML – Open Source



- **JEdit :**

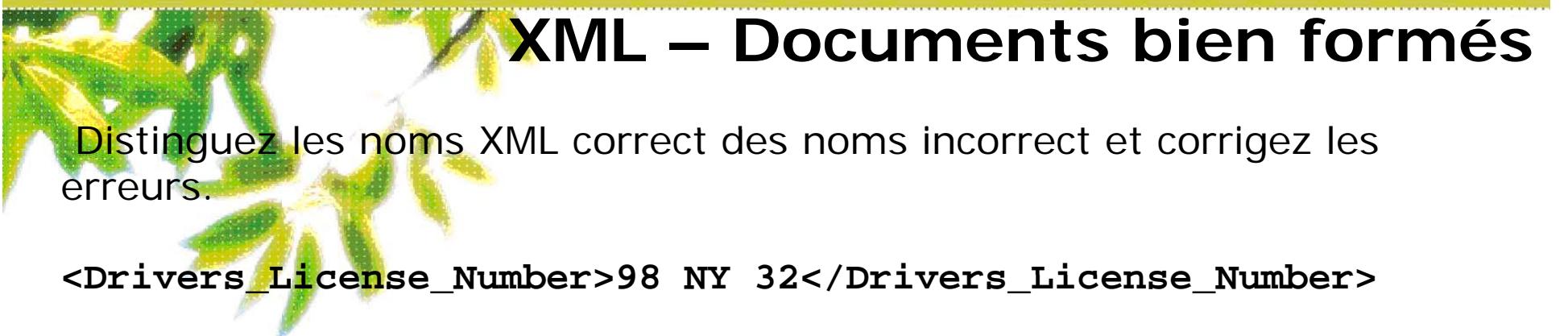
- Licence GPL
- Complétion de code
- Editeur graphique

- **Plates-formes :**

- Windows
- OS X
- Linux
- Unix
- OS/2
- VMS...
- (requiert Java)



Exemples



XML – Documents bien formés

Distinguez les noms XML correct des noms incorrect et corrigez les erreurs.

<Drivers_License_Number>98 NY 32</Drivers_License_Number>

<Driver's_License_Number>98 NY 32</Driver's_License_Number>

<month-day-year>7/23/2001</month-day-year>

<first name>Alan</first name>

<àçttûä>øåú</àçttûä>

<first_name>Alan</first_name>

<month/day/year>7/23/2001</month/day/year>

<_4-lane>I-610</_4-lane>

<téléphone>011 33 91 55 27 55 27</téléphone>

<4-lane>I-610</4-lane>



XML – Documents bien formés

```
<Drivers_License_Number>98 NY 32</Drivers_License_Number>
```

Correction: Correct

```
<Driver's_License_Number>98 NY 32</Driver's_License_Number>
```

Correction: Incorrect (apostrophe)

```
<month-day-year>7/23/2001</month-day-year>
```

Correction: Correct

```
<first name>Alan</first name>
```

Correction: Incorrect (présence d'un espace)

```
<àçttûä>øåú</àçttûä>
```

Correction: Correct

```
<first_name>Alan</first_name>
```

Correction: Correct



XML – Documents bien formés

```
<month/day/year>7/23/2001</month/day/year>
```

Correction: Incorrect (à cause des /)

```
<_4-lane>I-610</_4-lane>
```

Correction: Correct

```
<téléphone>011 33 91 55 27 55 27</téléphone>
```

Correction: Correct

```
<4-lane>I-610</4-lane>
```

Correction: Incorrect (un nom XML ne commence pas par un chiffre)



XML – Documents bien formés

Lisez les exemples suivants et vérifier si les documents XML sont bien formés (et expliquez pourquoi si ce n est pas le cas) :

```
<?xml version="1.0" ?>
<top>
    <item>Question 1<item answer="a">
    <item>Question 2<item answer="b">
    <item>Question 3<item answer="c">
</top>
```

```
<?xml version="1.0" ?>
<text>
    <font size='8pt'>petite police</font>
    <font size='24pt'>grande police</font>
</text>
```



XML – Documents bien formés

Lisez les exemples suivants et vérifier si les documents XML sont bien formés (et expliquez pourquoi si ce n est pas le cas) :

```
<?xml version="1.0" ?>
<top>
    <item>Question 1<item answer="a">
    <item>Question 2<item answer="b">
    <item>Question 3<item answer="c">
</top>
```

Correction: <item> n est pas fermé des attributs ne se mettent pas dans la balise fermante

```
<?xml version="1.0" ?>
<text>
    <font size='8pt'>petite police</font>
    <font size='24pt'>grande police</font>
</text>
```

Correction: Bien formé



XML – Documents bien formés

Lisez les exemples suivants et vérifier si les documents XML sont bien formés (et expliquez pourquoi si ce n est pas le cas) :

```
<?xml version="1.0" ?>
<top>
  <item val=2/>
  <item val=3/>
  <item val=12/>
</top>
```

```
<?xml version="1.0" ?>
<text>
  <font small>Un petit texte</font>
  <font big>Un grand texte</font>
</text>
```



XML – Documents bien formés

Lisez les exemples suivants et vérifier si les documents XML sont bien formés (et expliquez pourquoi si ce n est pas le cas) :

```
<?xml version="1.0" ?>
  <top>
    <item val=2/>
    <item val=3/>
    <item val=12/>
  </top>
```

Correction: Manque des guillemets pour les attributs

```
<?xml version="1.0" ?>
  <text>
    <font small>Un petit texte</font>
    <font big>Un grand texte</font>
  </text>
```

Correction: pas d'espaces dans les noms de balise (ou alors small est un attribut mais doit être suivi par affectation à une valeur)



XML – Documents bien formés

Lisez les exemples suivants et vérifier si les documents XML sont bien formés (et expliquez pourquoi si ce n est pas le cas) :

```
<?xml version="1.0"?>
<a>
  <b a="toto">Et hop</b>
</a>
<a>
  <b a="titi">Voil_a</b>
</a>
```

```
<?xml version="1.0"?>
<programme titre="Internet">
  <ul>
    <li>XML</li>
    <li>DTD</li>
    <li>API</li>
    <li>XSL</li>
  </ul>
</programme>
```



XML – Documents bien formés

Lisez les exemples suivants et vérifier si les documents XML sont bien formés (et expliquez pourquoi si ce n est pas le cas) :

```
<?xml version="1.0"?>
<a>
  <b a="toto">Et hop</b>
</a>
<a>
  <b a="titi">Voil_a</b>
</a>
```

Correction: Manque un élément Racine

```
<?xml version="1.0"?>
<programme titre="Internet">
  <ul>
    <li>XML</li>
    <li>DTD</li>
    <li>API</li>
    <li>XSL</li>
  </ul>
</programme>
```

**Correction: les balises fermantes ne sont pas bonnes **



XML – Documents bien formés

Lisez les exemples suivants et vérifier si les documents XML sont bien formés (et expliquez pourquoi si ce n est pas le cas) :

```
<?xml version="1.0"?>
<a><b><c/></b><d></b></a>
```

```
<?xml version="1.0" ?>
<niveaux>
<truc>chose</truc>
<niveau index="1">
<truc> </truc>
</niveau>
<niveau index="2">
<truc attribut="chose"> </truc>
</niveau> </niveaux>
```



XML – Documents bien formés

Lisez les exemples suivants et vérifier si les documents XML sont bien formés (et expliquez pourquoi si ce n est pas le cas) :

```
<?xml version="1.0"?>
<a><b><c/></b><d></b></a>
```

Correction: Mauvaise balise </d>

```
<?xml version="1.0" ?>
<niveaux>
<truc>chose</truc>
<niveau index="1">
<truc> </truc>
</niveau>
<niveau index="2">
<truc attribut="chose"> </truc>
</niveau> </niveaux>
```

**Correction: Bien Formé
(truc peut être utilisé à plusieurs endroits)**



X.M.L. - DTD

Définition de Type de Document

- Permet la vérification des documents XML, explique :

- ✓ Éléments, entités apparaissant dans le document
- ✓ Quels sont les contenus des éléments et attributs
- ✓ Utilisée par les parseurs pour validation
 - ✓ soit rejet
 - ✓ soit correction (rejet des éléments non valides)
- ✓ Généralement document externe
- ✓ suffixe.**dtd** (non normalisé)



XML – DTD Validation

- **La DTD ne précise pas :**

- ✓ Quel est l'élément racine du document
- ✓ Combien d'instances de chaque élément apparaissent dans le document
- ✓ A quoi ressemblent les données textuelles des éléments
- ✓ Quel est le sens d'un éléments

- o par exemple :

- nom d'une personne

- une date

**Le caractère bien formé est requis pour tout document XML,
la DTD est optionnelle**

```

<personne>
  <nom>
    <prénom>Alan</prénom>
    <nom_famille>Turing</nom_famille>
  </nom>
  <profession>informaticien</profession>
  <profession>mathématicien</profession>
  <profession>cryptographe</profession>
</personne>

```

```

<!ELEMENT personne (nom, profession*)>
<!ELEMENT nom (prénom, nom_famille)>
<!ELEMENT prénom (#PCDATA)>
<!ELEMENT nom_famille (#PCDATA)>
<!ELEMENT profession (#PCDATA)>

```

XML – DTD Exemple

personne.dtd

Déclaration de l'élément personne (1)

Déclaration de l'élément nom (2)

Déclaration des éléments prénom,
nom_famille, profession

(1) : Chaque élément personne contient au moins un nom et 0 à n profession * = 0,n

(2) : Chaque élément nom contient un prénom et un nom_famille

(3) : Elément type PCDATA (Parsed Character DATA)

Le document ne contient pas d'autres éléments que ceux définis dans la DTD



```
<!ELEMENT personne (nom, profession*)>
<!ELEMENT nom (prénom, nom_famille)>
<!ELEMENT prénom (#PCDATA)>
<!ELEMENT nom_famille (#PCDATA)>
<!ELEMENT profession (#PCDATA)>
```

```
<personne>
  <nom>
    <prénom>Alan</prénom>
    <nom_famille>Turing</nom_famille>
  </nom>
</personne>
```

valide

```
<personne>
  <profession>informaticien</profession>
  <profession>mathématicien</profession>
  <profession>cryptographe</profession>
</personne>
```

invalide

Pas de
nom !!

Profession avant
 élément nom !!

```
<personne>
  <profession>informaticien</profession>
  <nom>
    <prénom>Alan</prénom>
    <nom_famille>Turing</nom_famille>
  </nom>
  <profession>mathématicien</profession>
  <profession>cryptographe</profession>
</personne>
```



```
<!ELEMENT personne (nom, profession*)>
<!ELEMENT nom (prénom, nom_famille)>
<!ELEMENT prénom (#PCDATA)>
<!ELEMENT nom_famille (#PCDATA)>
<!ELEMENT profession (#PCDATA)>
```

```
<personne>
  <nom>
    <prénom>Alan</prénom>
    <nom_famille>Turing</nom_famille>
  </nom>
  <profession>mathématicien</profession>
  <profession>cryptographe</profession>
  <publication>sur les nombres
    calculables...</publication>
</personne>
```

Elément publication
non déclaré !!

Texte ajouté hors des
éléments

invalide

```
<personne>
  <nom>
    <prénom>Alan</prénom,>
    <nom_famille>Tuxing</nom_famille>
  </nom>
  fut un <profession>informaticien</profession>,
  un <profession>mathématicien</profession> et
  un <profession>cryptographe</profession>
</personne>
```



XML – DOCTYPE

- **Contenu dans le document XML :**

- ✓ Indique l'élément racine
- ✓ L'URI – Uniform Ressource Identifier, où se trouve la DTD
- ✓ Exemple :

```
<!DOCTYPE personne SYSTEM "http://www.divae.fr/formation/xml/personne.dtd">
```

- ✓ URI relative – même site que document :

```
<!DOCTYPE personne SYSTEM "/formation/xml/personne.dtd">
```

- ✓ même répertoire que document :

```
<!DOCTYPE personne SYSTEM "personne.dtd">
```



XML – DTD Exemple

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE personne SYSTEM
"http://www.divae.fr/formation/xml/personne.dtd">
<personne>
    <nom>
        <prénom>Alan</prénom>
        <nom_famille>Turing</nom_famille>
    </nom>
    <profession>informaticien</profession>
    <profession>mathématicien</profession>
    <profession>cryptographe</profession>
</personne>
```



XML – Sous-ensemble interne

- **DTD interne au document XML :**

- ✓ Déclaration du type de document contient la DTD
- ✓ Exemple

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE personne [
    <!ELEMENT prénom (#PCDATA)>
    <!ELEMENT nom_famille (#PCDATA)>
    <!ELEMENT profession (#PCDATA)>
    <!ELEMENT nom (prénom, nom_famille)>
    <!ELEMENT personne (nom, profession*)>
]>
<personne>
    <nom>
        <prénom>Alan</prénom>
        <nom_famille>Turing</nom_famille>
    </nom>
    <profession>informaticien</profession>
    <profession>mathématicien</profession>
    <profession>cryptographe</profession>
</personne>
```



XML – Validation de document

- **Navigateur :**

- ✓ Ne valident pas les documents (en général)
 - ✓ Vérifie les contraintes de forme

- **Parseur – exemple Xerces, classe `java.sax.SAXcount` :**

- ✓ Valide et vérifie la forme

- **Validation en ligne :**

- ✓ Formulaire de validation XML du Brown University Scholarly Technology Group :

<http://www.stg.brown.edu/service/xmlvalid/>



Déclaration d'éléments



XML – Déclaration d'éléments

- Chaque élément utilisé doit être déclaré dans la DTD

<!ELEMENT nom élément (modèle_contenu)>

- ✓ nom quelconque autorisé
- ✓ Modèle :
 - ✓ Sous-élément de l'élément (éventuel)
 - ✓ Ordre d'apparition de l'élément



XML – Eléments PCDATA

- **Modèle de contenu le plus simple :**

- ✓ Ne doit contenir que des données textuelles
- ✓ Ne contient pas de sous-élément
 - ✓ Exemple :
`<!ELEMENT numéro_téléphone (#PCDATA)>`



XML – Eléments / sous élément

- **Modèle de contenu simple, définissant :**

- ✓ Contenu d'un élément
- ✓ Type du sous-élément
- ✓ Exemple :

```
<!ELEMENT fax (numéro_téléphone)>
```

- o Un élément fax ne peut rien contenir en dehors d'un élément numéro_téléphone
- o Il ne peut en contenir plus ou moins de un



XML – Eléments / séquence

- **Modèle de contenu, définissant :**

- ✓ Un ensemble de sous-éléments
- ✓ Les sous-élément sont séparés par une virgule
- ✓ Exemple :

<!ELEMENT nom (prénom, nom_famille)>

- o Un élément nom contient un sous-élément prénom, exactement suivi d'un élément nom_famille



XML – Eléments / Nb éléments

- Permettre de moduler l'existence ou non de sous-éléments :
- Utilisation de suffixes :

? Autorise zéro ou un élément

* Autorise zéro ou plusieurs éléments

+ Autorise un ou plusieurs éléments

Exemple :

Déclaration permettant qu'un élément nom doit contenir un prénom, un ou plusieurs second_prenom et éventuellement un nom_famille

<!ELEMENT nom (prénom, second_prénom*, nom_famille?)>



XML – Eléments / choix

- Permettre de définir des structures de sous-éléments différentes
- Le choix est modélisé par une barre verticale | :

Exemple :

Cette déclaration indique qu'un élément méthodeRéponse contient soit un sous-élément param soit un sous-élément faute :

```
<!ELEMENT méthodeRéponse (param | faute)>
```



XML – Exercice

client.xml

- **Créer un document XML, possédant un DTD interne et répondant aux principes suivants :**

Un client possède un nom complet décomposé en nom, prénom

Il habite à une adresse constituée d'un n° de rue, d'une rue dans une ville

On peut le joindre par le biais de son contact composé :

- * de numéros de téléphone (0 ou plusieurs)
- * de fax (0 ou plusieurs)
- * d' email (0 ou plusieurs)

```
<client>
  <nom_complet>
    <nom>Fino</nom>
    <prenom>Jean-Marie</prenom>
  </nom_complet>

  <adresse>
    <num_rue>24</num_rue>
    <rue>Grand chemin</rue>
    <ville>Montbéliard</ville>
  </adresse>

  <contact>
    <telephone>03-81-90-99-00</telephone>
    <telephone>03-81-90-99-10</telephone>
    <fax>03-81-90-99-01</fax>
    <fax>03-81-90-99-02</fax>
    <email>jmfino@divae.fr</email>
    <email>jmfino@wanadoo.fr</email>
  </contact>
</client>
```

XML – Exercice - Correction

client.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE client [
    <!ELEMENT client (nom_complet, adresse, contact)>
    <!ELEMENT nom_complet (nom,prenom)>
    <!ELEMENT adresse (num_rue, rue, ville)>
    <!ELEMENT contact (telephone*, fax*, email*)>
    <!ELEMENT nom (#PCDATA)>
    <!ELEMENT prenom (#PCDATA)>
    <!ELEMENT num_rue (#PCDATA)>
    <!ELEMENT rue (#PCDATA)>
    <!ELEMENT ville (#PCDATA)>
    <!ELEMENT telephone (#PCDATA)>
    <!ELEMENT fax (#PCDATA)>
    <!ELEMENT email (#PCDATA)>
]>
<client>
    <nom_complet>
        <nom>Fino</nom>
        <prenom>Jean-Marie</prenom>
    </nom_complet>

    <adresse>
        <num_rue>24</num_rue>
        <rue>Grand chemin</rue>
        <ville>Montbéliard</ville>
    </adresse>

    <contact>
        <telephone>03-81-90-99-00</telephone>
        <telephone>03-81-90-99-10</telephone>
        <fax>03-81-90-99-01</fax>
        <fax>03-81-90-99-02</fax>
        <email>jmfino@divae.fr</email>
        <email>jmfino@wanadoo.fr</email>
    </contact>
</client>
```



XML – Eléments / parenthèses

- Permettre de combiner les déclaration précédentes
- Utilisation combinée des choix et des séquences :

Exemple :

Un élément cercle contient un élément centre suivi soit par un élément radian, soit diamètre

<!ELEMENT cercle (centre, (radian | diamètre))>

Expression du centre soit en coordonnées cartésiennes ou polaires.

Chaque centre contient soit un élément x et un élément y, soit un élément r et un élément q (utilisation de séquences, chacune d'elles étant mise entre parenthèses et associée à un choix):

< ! ELEMENT centre ((x, y) | (r, q))>



XML – Eléments / Contenu mixte

- Permettre de définir des documents hétérogènes contenant :

- ✓ des sous-éléments
- ✓ des données non-balisees
- ✓ des données textuelles

```
<!ELEMENT définition (#PCDATA | terme)*>
```

```
<définition>
```

La **Machine de Turing** est un automate abstrait à états finis comportant une mémoire infinie dont il peut être prouvé qu'il peut être équivalent à n'importe quel autre automate à états finis, contenant une quantité de mémoire quelconque mais fixe. Ainsi, ce qui est vrai pour une machine de Turing est vrai pour toute machine équivalente quelque soit son implémentation.

```
</définition>
```



XML – Eléments / Contenu mixte

- **On peut ajouter un nombre quelconque de sous-éléments :**

- ✓ #PCDATA doit toujours être indiqué en premier
- ✓ Exemple :

Un élément paragraphe peut contenir n'importe quel nombre d'éléments nom, profession, note et date, dans n'importe quel ordre et entrecoupés des données textuelles analysées

```
<!ELEMENT paragraphe (#PCDATA | nom | profession | note )*>
```



XML – Elément vide

- Ces éléments ne possèdent aucun contenu :

- ✓ Sont décrit par une balise < et />
- ✓ DTD : Empty , document <élément />
- ✓ Exemple DTD :

```
<!ELEMENT image EMPTY>
```

Exemple document

```
<image source="bus.jpg" width="152" height="345"  
alt="Alan Turing debout devant un bus"/>
```

Que des attributs, pas de contenu



XML – Eléments / Any

- Permet d'indiquer l'existence d'un élément, sans indiquer :

- ✓ Ce qu'il doit ou ne doit pas contenir
- ✓ Exemple ANY :

<!ELEMENT page ANY>

Cette déclaration indique qu'un élément page peut contenir n'importe quel contenu, y compris un contenu mixte, des sous-éléments et même d'autres éléments page

- ✓ Utilise lors de la conception de DTD
- ✓ Attention à une sur-utilisation de ANY, **permettant des changements incontrôlables**



Déclaration d'attributs



XML – Déclaration d'attributs

- Permet la déclaration des attributs aux différents éléments :

- ✓ Utilisation des déclarations ATTLIST
- ✓ Un seul ATTLIST peut déclarer plusieurs attributs pour un seul type d'élément
- ✓ Si le même attribut est répété plusieurs fois → déclaration séparée
- ✓ Exemple :

<!ATTLIST image source CDATA #REQUIRED>

- o L'élément image a un attribut appelé source
- o La valeur de l'attribut est une donnée textuelle
- o Sa valeur est obligatoire



XML – Déclaration attributs - Exemple

```
<!ATTLIST image source CDATA #REQUIRED  
        width CDATA #REQUIRED  
        height CDATA #REQUIRED  
        alt CDATA #IMPLIED>
```

- ✓ Les attributs source, width et height sont obligatoires.
- ✓ L'attribut alt est optionnel
- ✓ Ces 4 attributs sont déclarés contenir de la donnée textuelle, le type d'attribut le plus générique.

Cette déclaration a le même effet et le même sens que quatre déclarations ATTLIST différentes, une pour chaque attribut.



XML – Type attribut

- **Dans tous les documents XML, les valeurs d'attributs :**

- ✓ Peuvent être n'importe quelle chaîne de caractères
- ✓ Les occurrences de < ou de & doivent être échappés par < et &, ainsi que les " et '
- ✓ 10 types d'attributs existant dans XML :

CDATA

NMTOKEN

NMTOKENS

ENUMERATION

ENTITY

ENTITIES

ID

IDREF

IDREFS

NOTATION



XML – Attribut CDATA

- **Une valeur d'attribut CDATA, peut contenir :**

- ✓ N'importe quelle chaîne de caractères
- ✓ C'est l'attribut le plus général
- ✓ Exemple :

```
<!ATTLIST image alt CDATA #IMPLIED>
```



XML – Attribut NMTOKEN

- **Unité lexicale nominale XML (ressemble à un nom XML)**
- **Doit se composer des mêmes caractères qu'un nom XML :**
 - ✓ Alphanumérique
 - ✓ et/ou caractères graphiques
 - ✓ des signes de ponctuation _ , - , :
 - ✓ Ne contient pas d'espace
 - ✓ Tous les caractères peuvent être le premier caractère
 - ✓ Exemple : 12 .cshrc

Chaque nom XML est une unité lexicale nominative

Toutes les unités lexicales nominatives ne sont pas des noms XML



XML – Attribut NMTOKEN

- Exemple :

- ✓ Valeur type NMTOKEN, permettant de déclarer l'attribut année d'un élément journal.

Cet attribut doit contenir un entier comme 1990 ou 2015

<!ATTLIST journal année NMTOKEN #REQUIRED>

Cette déclaration ne nous prémunit pas contre les valeurs : **99** ou **Mars**

Elle évite celles comme : **1990 CE ou Bonjour à tous** qui contiennent des blancs



XML – Attribut NMTOKENS

- Contient une ou plusieurs unités lexicales nominales XML :

- ✓ Séparées par des blancs
- ✓ Exemple :

Attribut dates d'un élément concerts, de la forme mm-jj-aa

```
<concerts dates="08-21-2001 08-23-2001 08-27-2001">
```

Kat and the Kings

```
</concerts>
```

Déclaration :

```
<!ATTLIST concerts dates NMTOKENS #REQUIRED>
```



XML – Attribut ENUMERATION

- **Non représenté par un mot-clé XML :**

- ✓ Liste de valeurs possibles pour un attribut
- ✓ Pas de valeurs permises en dehors de la liste
- ✓ Valeurs séparées par des barres verticales
- ✓ Chaque valeur possible est une unité lexicale nominale
- ✓ Exemple :
 - ✓ La valeur de l'attribut mois de l'élément date doit être un des 12 noms de mois

```
<!ATTLIST date mois (Janvier | Février | Mars | Avril | Mai | Juin | Juillet  
| Août | Septembre | Octobre | Novembre | Décembre) #REQUIRED>
```



XML – Attribut ENUMERATION

- Exemple :

La valeur de l'attribut jour doit être un nombre compris entre 1 et 31

```
<!ATTLIST date jour (1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23
| 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31) #REQUIRED>
```

La valeur de l'attribut année doit être un entier entre 1970 et 2009

```
<!ATTLIST date année (1970 | 1971 | 1972 | 1973 | 1974 | 1975 |
1976 | 1977 | 1978 | 1979 | 1980 | 1981 | 1982 | 1983 | 1984 |
1985 | 1986 | 1987 | 1988 | 1989 | 1990 | 1991 | 1992 | 1993 |
1994 | 1995 | 1996 | 1997 | 1998 | 1999 | 2000 | 2001 | 2002 |
2003 | 2004 | 2005 | 2006 | 2007 | 2008 | 2009 ) #REQUIRED>
```



XML – Attribut ID

- **Contient un nom XML (identifiant unique aux éléments) :**
 - ✓ Ce nom est unique dans le document
 - ✓ Aucun attribut de type ID dans le document ne peut avoir la même valeur
 - ✓ Chaque élément ne peut avoir plus d'un attribut de type ID
 - ✓ Exemple : Chaque élément employé doit avoir un attribut ID numéro_sécu :

<!ATTLIST employé numéro_sécu ID #REQUIRED>

- ✓ ATTENTION :

Un nombre n'est pas un ID XML autorisé, la solution classique préfixe les valeurs avec un caractère souligne ou une lettre normale.

Par exemple: **<employé numéro_sécu="_123-45-6789"/>**



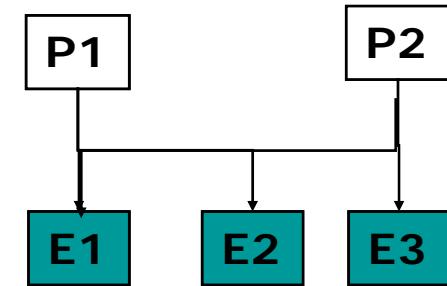
XML – Attribut IDREF

- Référence à un attribut de type ID :

- ✓ Doit être un nom XML
- ✓ Permet l'établissement de relations plusieurs à plusieurs, lorsque l'imbrication ne suffit pas → Références croisées



XML – Attribut IDREF



Chaque **projet (P)** a un attribut de type ID projet_id

Tout **employé (E)** a un attribut de type ID numéro_sécu.

Chaque projet a un sous-élément membre équipe qui identifie qui travaille sur le projet

Chaque élément employé a un sous-élément tâches qui indique quels projets sont assignés à cet employé.

Vu que chaque projet est assigné à plusieurs employés et que certains employés sont affectés à plusieurs projets, il n'est pas possible de mettre :

- ✓ les employés sous-éléments des projets
- ✓ ou les projets sous-éléments des employés.

```

<!ATTLIST employé numéro_sécu ID #REQUIRED>
<!ATTLIST projet projet_id ID #REQUIRED>
<!ATTLIST membre_équipe employé IDREF #REQUIRED>
<!ATTLIST tâches projet_id IDREF #REQUIRED>

```

DTD

Fichier XML

Projet
Projet_id = "p1"
But : Plan de développement stratégique

Projet
Projet_id = "p2"
But : Déployez Linux

Employé
numéro_secu : "ss123-45-6789"
nom : "Fred Smith"

Employé
numéro_secu : "ss876-54-3210"
nom : "Jill Jones"

Employé
numéro_secu : "ss876-12-3456"
nom : "Sidney Lee"

```

<projet projet_id="p1">
<but>Plan de développement stratégique</but>
<membre_équipe employé="ss123-45-6789"/>
<membre_équipe employé="ss876-54-3210"/></projet>

```

```

<projet projet_id="p2">
<but>Déployez Linux</but>
<membre_équipe employé="ss123-45-6789"/>
<membre_équipe employé="ss876-12-3456"/></projet>

```

```

<employé numéro_sécu="ss123-45-6789">
    <nom>Fred Smith</nom>
    <tâches projet_id="p1"/>
    <tâches projet_id="p2"/></employé>

```

```

<employé numéro_sécu="ss876-54-3210">
    <nom>Jill Jones</nom>
    <tâches projet_id="p1"/></employé>

```

```

<employé numéro_sécu="ss876-12-3456">
    <nom>Sydney Lee</nom>
    <tâches projet_id="p2"/></employé>

```

```
<!ATTLIST employé numéro_sécu ID #REQUIRED>
<!ATTLIST projet projet_id ID #REQUIRED>
<!ATTLIST membre_équipe employé IDREF #REQUIRED>
<!ATTLIST tâches projet_id IDREF #REQUIRED>
```



Exercice :

A partir du fichier lesProjets.xml,
créer sa DTD

DTD

Fichier XML

```
<projet projet_id="p1">
  <but>Plan de développement stratégique</but>
  <membre_équipe employé="ss123-45-6789"/>
  <membre_équipe employé="ss876-54-3210"/></projet>

<projet projet_id="p2">
  <but>Déployez Linux</but>
  <membre_équipe employé="ss123-45-6789"/>
  <membre_équipe employé="ss876-12-3456"/></projet>

<employé numéro_sécu="ss123-45-6789">
  <nom>Fred Smith</nom>
  <tâches projet_id="p1"/>
  <tâches projet_id="p2"/></employé>

<employé numéro_sécu="ss876-54-3210">
  <nom>Jill Jones</nom>
  <tâches projet_id="p1"/></employé>

<employé numéro_sécu="ss876-12-3456">
  <nom>Sydney Lee</nom>
  <tâches projet_id="p2"/></employé>
```



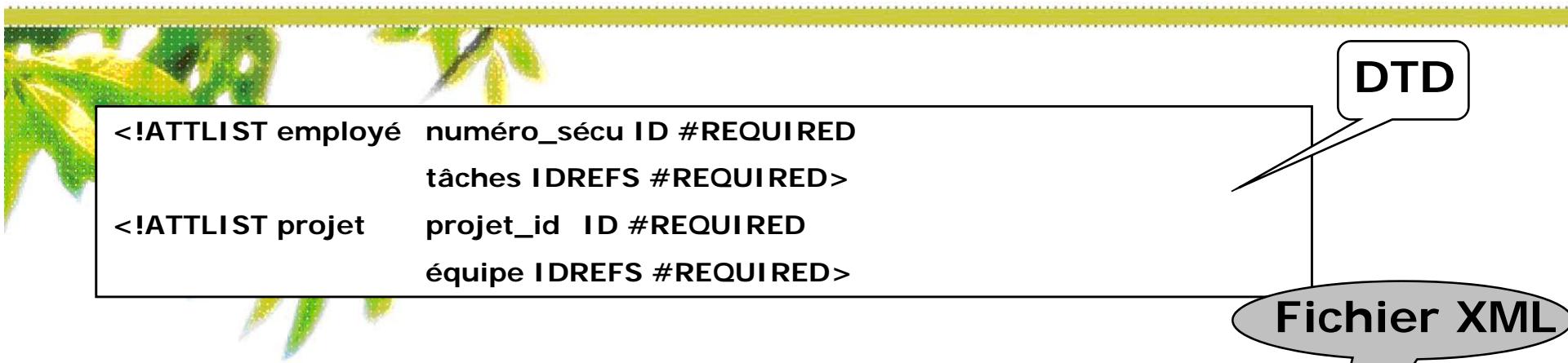
XML – Attribut IDREFS

- Référence à plusieurs autres éléments, l'attribut contient :

- ✓ Liste de noms XML séparés par des blancs
- ✓ Chacun d'eux est l'ID d'un élément du document
- ✓ Exemple :

Reprise exemple IDREF :

On remplace l'affectation des sous-éléments de l'élément employé avec un seul attribut tâches, idem pour les sous-éléments membre-équipe



DTD

```
<!ATTLIST employé numéro_sécu ID #REQUIRED  
          tâches IDREFS #REQUIRED>  
  
<!ATTLIST projet    projet_id  ID #REQUIRED  
          équipe IDREFS #REQUIRED>
```

Fichier XML

```
<projet projet_id="p1" équipe="ss123-45-6789 ss876-54-3210">  
  <but>Plan de développement stratégique</but>  
</projet>  
  
<projet projet_id="p2" équipe="ss123-45-6789 ss876-12-3456">  
  <but>Déployez Linux</but>  
</projet>  
  
<employé numéro_sécu="ss123-45-6789" tâches="p1 p2">  
  <nom>Fred Smith</nom>  
</employé>  
  
<employé numéro_sécu="ss876-54-3210" tâches="p1">  
  <nom>Jill Jones</nom>  
</employé>  
  
<employé numéro_sécu="ss876-12-3456" tâches="p2">  
  <nom>Sydney Lee</nom>  
</employé>
```



XML – Attribut ENTITY

- Contient le nom d'une entité déclarée (~ macro / raccourci), non analysée :

- ✓ Déclarée n'importe où dans la DTD
- ✓ Exemple :

```
<!ELEMENT bloc_de_texte (#PCDATA)>
<!ENTITY jpa "Je vous prie d'agréer mes cordiales salutations">
```

```
<?xml version="1.0" encoding="ISO-8859-1">
<!DOCTYPE bloc_de_texte SYSTEM "bloc_de_texte.dtd">
<bloc_de_texte>En l'attente, &jpa;</bloc_de_texte>
```



XML – Attribut ENTITY

Texte_courrier

- Permet l'imbrication :
- Entités imbriquées, Exemple :

```
<!ENTITY rs "Rodrigue et Chimène SARL" >
<!ENTITY c "info@cid.fr" >
<!ENTITY rsc "&rs;, &c;" >
<!ELEMENT texte_courriel (#PCDATA)>
```

```
<?xml version="1.0" encoding="ISO-8859-1"
?>
<!DOCTYPE texte_courrier SYSTEM
"texte_courriel.dtd">

<texte_courriel>

La firme &rs; est heureuse de vous
présenter ses produits les plus récents.
Veuillez envoyer votre commande à &rsc;.

</texte_courriel>
```



XML – Attribut ENTITIES

- Contient plusieurs entités déclarées, non analysées :

- ✓ Liste des noms des entités séparés par des blancs
- ✓ Les entités sont déclarées n'importe où dans la DTD
- ✓ Exemple :

Un élément voir_séquences pourrait avoir un attribut ENTITIES identifiant les fichiers JPEG à afficher dans l'ordre où ils ont à être montrés:

```
<!ATTLIST voir_séquences séquences ENTITIES #REQUIRED>
```

Si la DTD déclarait des noms d'entités non analysées appelées slide1, slide2, slide3, et ainsi de suite jusqu'à slide10, alors vous pourriez utiliser cet élément voir séquences pour inclure la présentation dans le document XML

```
<voir_séquences séquences="séquence1 séquence2 séquence3  
séquence4 séquence5 séquence6 séquence7 séquence8 séquence9  
séquence10" />
```



XML – Attribut NOTATION

- Contient le nom d'une notation déclarée dans la DTD

✓ Exemple :

Ces déclarations définissent quatre notations pour différents types d'image et indique alors que chaque élément image doit avoir un attribut type qui choisit exactement l'une d'elles:

```
<!NOTATION gif      SYSTEM "image/gif">
<!NOTATION tiff     SYSTEM "image/tiff">
<!NOTATION jpeg     SYSTEM "image/jpeg">
<!NOTATION png      SYSTEM "image/png">
<!ATTLIST image type NOTATION (gif | tiff | jpeg | png)
#REQUIRED>
```

Un type énumération ne peut pas spécifier image/png ou image/gif comme une valeur autorisée à cause du slash qui n'est pas un caractère autorisé dans les noms XML.



XML – Attribut par défaut

- Chaque déclaration ATTLIST inclut une déclaration par défaut pour cet attribut :
 - ✓ **#IMPLIED** : L'attribut est optionnel. Chaque instance de l'élément peut, ou ne pas, proposer une valeur pour l'attribut. Aucune valeur par défaut n'est fournie.
 - ✓ **#REQUIRED** : L'attribut est obligatoire. Chaque instance de l'élément doit proposer une valeur pour l'attribut. Aucun valeur par défaut n'est fournie.
 - ✓ **#FIXED** : La valeur de l'attribut est fixe et non modifiable. Cet attribut a une valeur spécifiée, que l'attribut soit ou ne soit pas explicitement noté sur l'instance particulière d'un élément.
=>**Littéral** : La valeur par défaut en tant que chaîne entre guillemets.



XML – Attribut par défaut

- Exemples :

Les éléments personne peuvent, mais ne sont pas obligés d'avoir les attributs naissance et mort

```
<!ATTLIST personne naissance CDATA #IMPLIED mort CDATA  
#IMPLIED>
```

Chaque élément cercle doit avoir des attributs centre_x, centre_y et radian

```
<!ATTLIST cercle centre_x NMTOKEN #REQUIRED  
centre_y NMTOKEN #REQUIRED  
radian NMTOKEN #REQUIRED>
```



XML – Attribut par défaut

- **Exemples :**

Chaque élément biographie a un attribut xmlns:xlink et que la valeur de l'attribut est `http://www.w3.org/1999/Xlink`, et ce même si la balise de début de l'élément ne comporte pas explicitement d'attribut xmlns:xlink

```
<!ATTLIST biographie xmlns:xlink CDATA #FIXED  
"http://www.w3.org/1999/xlink">
```

Chaque élément page web a un attribut protocole. Si un élément page_web particulier n'a pas d'attribut protocole explicite, alors il en a un avec la valeur http

```
<!ATTLIST page web protocole NMTOKEN "http">
```



XML – Exercice

Bibliographie.dtd

Bibliographie.xml

- A partir du fichier XML bibliographie, créer sa DTD externe, une bibliographie :

Contient des livres et des articles ; les informations nécessaires pour un livre sont :

- son titre général, avec un attribut optionnel sous-titre ;
- les noms des auteurs ;
- ses tomes (élément vide) ayant un attribut requis nb_pages, un attribut optionnel sous-titre et pour chaque tome, leur nombre de pages ;
- des informations générales sur son édition comme par exemple le nom de l'éditeur, le lieu d'édition, le lieu d'impression, son numéro ISBN ;
- les informations nécessaires pour un article sont : son titre ; les noms des auteurs ; ses références de publication :
 - le journal avec attribut nom du journal (valeur par défaut Feuille de chou), numéro des pages, année de publication (valeurs possibles 2000, 2001, 2002, avant2000, et numéro du journal)
- on réservera aussi un champ optionnel pour un avis personnel.

XML – Exercice

Bibliographie.xml

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE bibliographie SYSTEM "bibliographie.dtd" >
<bibliographie>
<livre>
    <titre soustitre="Ma vie">Live</titre>
    <auteur>Georges Dupond</auteur>
    <edition>
        <editeur>Folio</editeur>
        <lieu_edition>Montbeliard</lieu_edition>
        <lieu_impression>Belfort</lieu_impression>
        <isbn>2-033-444-025</isbn>
    </edition>
    <avis>BOF!</avis>
</livre>
<article>
    <titre>DOM SAX XML</titre>
    <auteur>G.Delrhue</auteur>
    <auteur>P.Anderson</auteur>
    <journal nom_journal="01 Informatique" annee="2002">
        <page>4 </page>
        <num_journal>1</num_journal>
    </journal>
</article>
<article>
    <titre>Java</titre>
    <auteur>Sun Microsystem</auteur>
    <journal annee="2002">
        <page>4</page>
        <num_journal>2</num_journal>
    </journal>
</article>
</bibliographie>
```



XML – Exercice

Bibliographie.dtd

```
<!ELEMENT bibliographie (livre|article)*>
<!ELEMENT livre (titre, auteur+, edition, avis?)>
  <!ATTLIST titre soustitre CDATA #IMPLIED>
<!ELEMENT titre (#PCDATA)>
<!ELEMENT auteur (#PCDATA)>
<!ELEMENT tome EMPTY>
  <!ATTLIST tome nb_pages CDATA #REQUIRED soustitre CDATA #IMPLIED>
<!ELEMENT edition (editeur, lieu_edition, lieu_impression, isbn)>
<!ELEMENT editeur (#PCDATA)>
<!ELEMENT lieu_edition (#PCDATA)>
<!ELEMENT lieu_impression (#PCDATA)>
<!ELEMENT isbn (#PCDATA)>
<!ELEMENT avis (#PCDATA)>
<!ELEMENT article (titre, auteur+, journal)>
<!ELEMENT journal (page, num_journal)>
  <!ATTLIST journal nom_journal CDATA "Feuille de Chou" >
  <!ATTLIST journal annee (2000 | 2001 | 2002 | Avant2000 | inconnue ) #REQUIRED >
<!ELEMENT page (#PCDATA)>
<!ELEMENT num_journal (#PCDATA)>
<!ELEMENT annee (#PCDATA)>
```



XML – Entité générale

- XML fournit cinq références d'entité :
 - ✓ < Le signe inférieur à, le chevron ouvrant (<).
 - ✓ & L'esperluette(&).
 - ✓ > Le signe supérieur à, le chevron fermant (>).
 - ✓ " Les guillemets ("), encore appelés double quotes.
 - ✓ ' L'apostrophe ou la simple quote (').

- La DTD peut en définir beaucoup d'autres :
 - ✓ Les références d'entité sont définies par une déclaration ENTITY dans la DTD.



XML – Entité générale / Exemple

- **Exemple :**

Cette déclaration d'entité définit &super; comme abréviation pour supercalifragilisticexpalidocious

```
<!ENTITY super "supercalifragilisticexpalidocious">
```



XML – Entité générale / Exemple

- **Exemple :**

Les entités peuvent contenir aussi bien des balises que du texte. Cette déclaration définit &footer; comme une abréviation pour le pied de page standard d'une page web qui est répété sur plusieurs pages

```
<!ENTITY footer '>
<hr size="1" noshade="true"/>
<font CLASS="footer">
    <a href="index.html">Accueil O'&Reilly </a> |
    <a href="sales/bookstores/">Librairies O'&Reilly </a> |
    <a href="order new/">Comment commander</a> |<br>
    <a ref="http://international.oreilly.com/">International</a> |
    <a href="oreilly/about.html">Informations</a> |
    <a href="affiliates.html">Sociétés affiliées</a>
</font>
<p>
<font CLASS="copy">
    Copyright 2000, O'&Reilly & Associates, Inc.<br/>
    <a href="mailto:webmaster@oreilly.com">webmaster@oreilly.com</a>
</font>
</p>
'>
```



XML – Entité générale externe

- Permet le stockage d'informations répétitives appelées dans les documents
- Un appel d'entité générale externe analysée est déclarée dans la DTD, en utilisant ENTITY :
- Au lieu de la substitution du texte, on utilise :
 - ✓ Le mot-clé SYSTEM
 - ✓ L'URI du texte de substitution
 - ✓ Exemple

```
<!ENTITY footer SYSTEM  
"http://www.oreill.com/boilerplate/footer.xml">
```

Ou URI relative :

```
<!ENTITY footer SYSTEM "/boilerplate/footer.xml">
```



XML – Entité paramètre

- Définissent des entités "raccourcies" réutilisables (~ macro)
- Permet le partage de listes d'attributs et modèle de contenu
- Exemple

Application XML gérant des listes de biens immobiliers nécessitant des éléments distincts pour des appartements, des sous-locations, des logements et des maisons à vendre.

```
<!ELEMENT appartement (adresse, surface, chambres, salle_de_bain, loyer)>
<!ELEMENT collocation (adresse, surface, chambres, salle_de_bain, loyer)>
<!ELEMENT poulailler (adresse, surface, chambres, salle_de_bain, prix)>
<!ELEMENT logement (adresse, surface, chambres, salle_de_bain, prix)>
<!ELEMENT maison (adresse, surface, chambres, salle_de_bain, prix)>
```



XML – Entité paramètre / syntaxe

- Définie comme une entité générale, ajout caractère %
- Positionné entre <!ENTITY et le nom de l'entité
- Exemple

```
<!ENTITY % contenu_résidentiel "adresse, surface, chambres, salle_de_bain">  
<!ENTITY % contenu_location "loyer">  
<!ENTITY % contenu_achat "prix">  
  
<!ELEMENT appartement (%contenu_résidentiel;, %contenu_location;)>  
<!ELEMENT collocation (%contenu_résidentiel;, %contenu_location;)>  
<!ELEMENT poulailler (%contenu_résidentiel;, %contenu_achat;)>  
<!ELEMENT logement (%contenu_résidentiel;, %contenu_achat;)>  
<!ELEMENT maison (%contenu_résidentiel;, %contenu_achat;)>
```



XML – Entité générale / Redéfinition

- Possibilité de redéfinir les entités paramètre
- Utilisé dans le cas DTD interne et externe
 - ✓ La DTD interne redéfinie les entités externes (si besoin)
 - ✓ Sous-ensemble interne de la DTD lu en premier (prioritaire)



Espaces de noms



XML – Espace de noms

- **Objectifs :**

- ✓ Distinguer les éléments et attributs issus de vocabulaires différents partageant les mêmes noms
- ✓ Création de document composites (langages de balises différents : XML, HTML, MATHML, CML ...)
- ✓ Grouper tous les éléments et attributs d'une même application XML :
 - Utilisation de préfixe pour chaque élément/attribut
 - Association d'URI à chaque préfixe



XML – Nécessité des espaces de noms

- **Documents de différentes applications, dont les éléments ont :**
 - ✓ Identification différente
 - ✓ Signification similaire
 - **Un Document comportant des objets :**
 - ✓ Ayant le même nom
 - ✓ Une signification différente
- ➔ **Lever l'ambiguité par l'utilisation de préfixes**



XML – Espace de noms / syntaxe

- Lève les ambiguïtés sur les éléments et attributs par :
 - ✓ Assignation des éléments et attributs à des URI
 - ✓ Tous les éléments d'une application → URI application
 - ✓ URI = espace de noms
 - ✓ Même URI et même noms → égaux
 - ✓ URI ou/et noms différents → différents
 - ✓ Plusieurs espaces de noms pour une application



XML – Espace de noms / Préfixe

- **Caractéristique des éléments appartenant à des espaces de noms :**

- ✓ Ajout d'un préfixe à l'élément
- ✓ Chaque préfixe est associé à un seul URI
- ✓ Les noms associé au même URI sont dans le même espace
- ✓ Syntaxe : **espace de nom:élément**
- ✓ Les espaces de noms peuvent être déclarés dans l'élément racine
- ✓ Attribut xmlns (xml name space)

```
<Espace_de_nom:Elément>donnée</Espace_de_nom:Elément>
```



XML – Espace de noms / Préfixe

- **Les préfixes :**

- ✓ Constitués de n'importe quel caractère d'un nom XML (sauf ":")
- ✓ S'il commence par "xml" ➔ réservés XML

- **Les parties locales :**

- ✓ Ne contient pas le caractère ':'

- **Caractère ':' :**

- ✓ Séparation préfixe / partie locale seulement



XML – Espace de noms / URI -Préfixe

- Chaque préfixe doit être associé à un URI :

- ✓ Exemple : XSLT 1.0 sont associées à l'URI :

http://www.w3.org/1999/XSL/Transform.

- ✓ Utilisation attribut xmlns au préfixe

```
<?xml version="1.0"?>
<report xmlns:H='http://www.w3.org/REC-html40'
         xmlns:M='http://www.w3.org/REC-MathML'>
    <chapter>
        ...
        <M:mfrac H:style="color:blue">
            ...
    </chapter>
</report>
```



XML – Espace de noms & DTD

- Indépendant des DTD :

- ✓ Utilisable dans documents valides ou invalides
- ✓ Ne modifie pas leur syntaxe
- ✓ Exemple :

Utilisation d'un élément dc:titre

<!ELEMENT dc:titre (#PCDATA)>



Schémas XML



XML – Schémas / Introduction

- **Carences laissées par les DTD, pour représenter :**
 - Les contraintes
 - Les structures de données
 - Structure de définition DTD lourde à implémenter
- **Les schémas :**
 - Prise en charge des types de données
 - Modèle de contenu plus exhaustif
 - Facilitent l'échange, la fusion et la réutilisation des données provenant d'une ou plusieurs sources
 - Permettent de créer des modèles de contenu composites
 - Base des échanges des Web Services



XML – Schémas / Définition

- **La définition d'un schéma se réalise par le biais de plusieurs éléments représentant :**
 - L'arborescence d'un document XML
 - Les informations d'un document XML
- **Les éléments et attributs :**
 - **Sont représentés spécifiquement**
 - Possibilité de regroupement
 - Possibilité de définir des nœuds indéfinis (schéma générique)
 - **Peuvent être un type simple / un type complexe**
 - **Contenu décrit précisément (ordre d'apparition, alternatives)**



XML – Schémas / Définition

- **Type simple :**

- L'élément ou l'attribut ne contient qu'une valeur simple
- Exemple :

```
<element>  
    Valeur de l'élément  
</element>
```

- Définit à l'aide d'éléments facettes (définition unique de l'ensemble des valeurs)

- **Type complexe :**

- L'élément peut posséder des éléments, des attributs et une valeur simple
- Exemple :

```
<element attrib="valeur">  
    Valeur de l'élément  
    <element_enfant_1/>  
    <element_enfant_N/>  
</element>
```



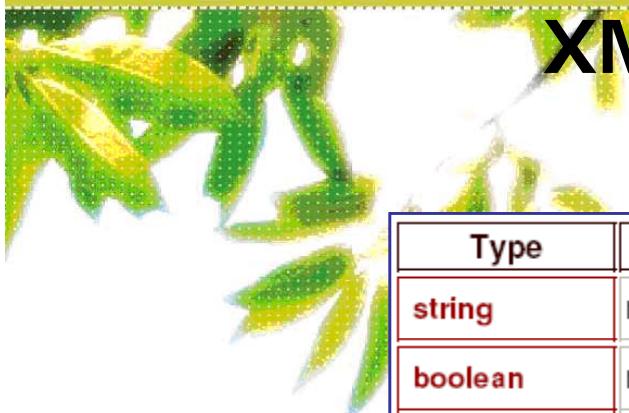
XML – Schémas / Facettes

- **Aspect de la définition d'une valeur simple**
- **Les valeurs peuvent être restreintes :**
 - par rapport à une liste de possibilités
 - À des limitations
 - À un modèle d'expression régulière
 - A une longueur, à un comportement
- **Certains éléments de schémas XML exprime :**
 - Des contraintes d'unicité et de référencement
 - Des contraintes d'identité (cf clé primaires, étrangères)
- **Les éléments d'annotation permettent :**
 - Insérer des commentaires
 - Insérer des informations destinées aux applications



XML – Schémas / types données

- **Type simple :**
 - Ne peut être qu'un type de données prédéfinies
 - Restriction possible par facette
- **Type de données représentées par 3 parties distinctes :**
 - Les espaces de valeur (ensemble de valeurs possibles)
 - *Exemple : { -2147483648 - 2147483647 }.*
 - Les espaces lexicaux (ensemble de littéraux valides)
 - Exemple : 9.9 ou 9.90 pour type float
 - Un ensemble de facettes (propriétés et restrictions)
- **Type de données prédéfinies en 2 ensembles distincts :**
 - Les primitifs
 - Les dérivés



XML – Schémas / types primitifs

Type	Description
string	représente une chaîne de caractères.
boolean	représente une valeur booléenne <i>true</i> ou <i>false</i> .
decimal	représente un nombre décimal
float	représente un nombre à virgule flottante.
double	représente un nombre réel double.
duration	représente une durée.
dateTime	représente une valeur date/heure.
time	représente une valeur horaire (format : hh:mm:ss.sss).
date	représente une date (format : CCYY-MM-DD).
gYearMonth	représente un mois et une année grégorienne (format : CCYY-MM).
gYear	représente une année (format : CCYY).
gMonthDay	représente le jour d'un mois (format : MM-DD).
gDay	représente le jour d'un mois (format : DD).
gMonth	représente le mois (format : MM).



XML – Schémas / types primitifs

Type	Description
hexBinary	représente un contenu binaire hexadécimal.
base64Binary	représente un contenu binaire de base 64.
anyURI	représente une adresse URI (ex.: http://www.site.com).
QName	représente un nom qualifié.
NOTATION	représente un nom qualifié.



XML – Schémas / types dérivés

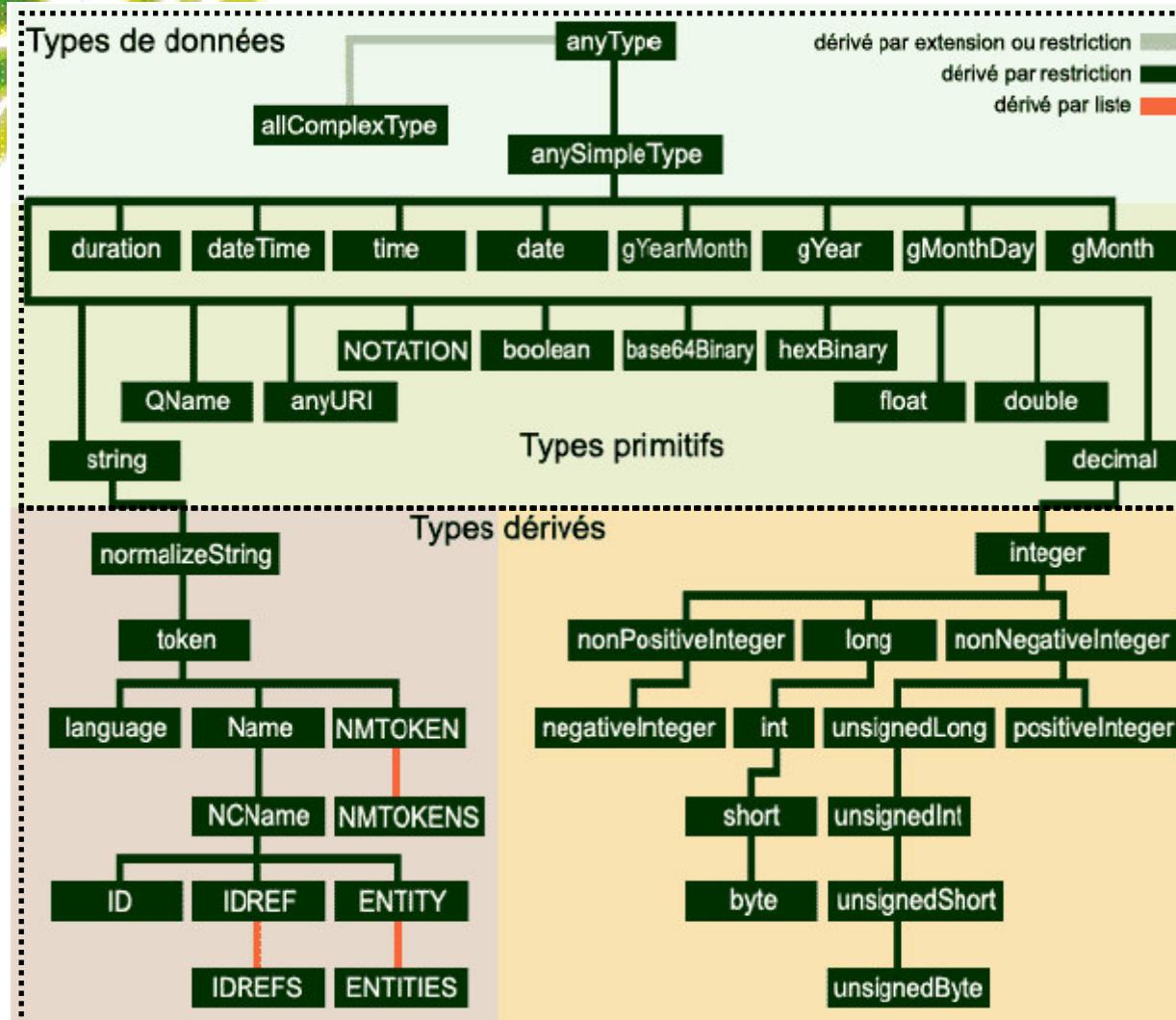
Type	Description
normalizedString	représente une chaîne de caractères dont les espaces blancs sont normalisés.
token	représente une chaîne de caractères sans espaces blancs.
language	représente un langage exprimé sous forme de mot clés répondant à la RFC 1766.
NMTOKEN	représente le type d'attribut NMTOKEN de XML 1.0.
NMTOKENS	représente le type d'attributs NMTOKENS de XML 1.0.
Name	représente un nom XML.
NCName	représente un nom non-implanté (<i>non-colonized</i>) dans .
id	représente le type d'attribut ID de XML 1.0.
IDREF	représente le type d'attribut IDREF de XML 1.0.
IDREFS	représente le type d'attribut IDREFS de XML 1.0.
ENTITY	représente le type d'attribut ENTITY de XML 1.0.
ENTITIES	représente le type d'attribut ENTITIES de XML 1.0.



XML – Schémas / types dérivés

Type	Description
integer	représente un nombre entier.
nonPositiveInteger	représente un nombre entier négatif incluant le zéro.
negativeInteger	représente un nombre entier négatif dont la valeur maximum est -1.
long	représente un nombre entier long dont l'intervalle est {-9223372036854775808 - 9223372036854775807}.
int	représente un nombre entier dont l'intervalle est {-2147483648 - 2147483647}.
short	représente un nombre entier court dont l'intervalle est {-32768 - 32767}.
byte	représente un entier dont l'intervalle est {-128 - 127}.
nonNegativeInteger	représente un nombre entier positif incluant le zéro.
unsignedLong	représente un nombre entier long non-signé dont l'intervalle est {0 - 18446744073709551615}.
unsignedInt	représente un nombre entier non-signé dont l'intervalle est {0 - 4294967295}.
unsignedShort	représente un nombre entier court non-signé dont l'intervalle est {0 - 65535}.
unsignedByte	représente un nombre entier non-signé dont l'intervalle est {0 - 255}.
positiveInteger	représente un nombre entier positif commençant à 1.

XML – Hiérarchie des types prédéfinis





XML – Schémas / Résumé

- **Alternative aux DTD (précision plus importante)**
- **Il décrit :**
 - Les éléments qui composent un document
 - Les attributs des éléments
 - La hiérarchie entre éléments
 - L'ordre des sous-éléments
 - Le nombre de sous-éléments
 - Les types d'éléments et attributs
 - Les valeurs par défaut
 - Le format ou la restriction des valeurs d'un élément ou attribut



XML – Schémas / Création d'un schéma

- **Méthodologie :**
 1. Entête du schéma
 2. Décrire structures de données des éléments simples et des attributs (utilisation des restrictions)
 3. Décrire les éléments composés (types complexes)



XML – Schémas / Composantes

- Définition de schéma
- Définition des éléments et attributs
 - Complexes
 - Simples
- Les restrictions
- Les contraintes
- Les annotations
- Des clefs et références



XML – Schémas / Appel

- **Appel du schéma du document XML**

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<librairie
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:SchemaLocation="group.xsd">
```

Espace de nom utilisé

Elément racine

Schéma du document



DEFINITION

Définitions de Schéma



XML – Elément schéma

- **<xsd:schema>..</xsd:schema>**
- Element premier du document schema XML
- Obligatoire
- Permet la définition des composants du document XML

DEFINITION

```
<xsd:schema  
attributeFormDefault = (qualified | unqualified) : unqualified  
blockDefault = (#all | Liste de (extension | restriction | substitution)) :  
elementFormDefault = (qualified | unqualified) : unqualified  
finalDefault = (#all | Liste de (extension | restriction)) : "  
id = ID  
targetNamespace = adresse_URI  
version = token  
xml:lang = language  
{tout attribut ayant un espace de noms  
différent de celui du schéma...}>  
Contenu : ((include | import | redefine | annotation)*,  
((simpleType | complexType | group | attributeGroup)  
| element | attribute | notation), annotation*)*)  
</xsd:schema>
```



XML – Elément schéma - Attributs

DEFINITION

Attributs	Description
attributeFormDefault	indique si les attributs XML doivent être qualifiés par un espace de noms.
blockDefault	empêche, par défaut, l'utilisation de types dérivés dans des éléments attendant le type de base.
elementFormDefault	indique si les éléments XML doivent être qualifiés par un espace de noms.
finalDefault	empêche, par défaut, la dérivation de type par restriction, extension ou les deux.
id	précise un identificateur unique pour le schéma.
targetNamespace	indique un espace de noms cible pour tout élément étranger au vocabulaire de schéma XML.
version	indique un numéro de version.
xml:lang	indique le langage dans lequel est conçu le document.

```
1 <?xml version="1.0" encoding="ISO-8859-1"?>
2
3 <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
4   elementFormDefault="qualified"
5   attributeFormDefault="unqualified"
6   lang="fr">
```



XML – Elément include

DEFINITION

- **<xsd:include>...</xsd:include>**
- Permet l'inclusion d'un schéma de même espace de nom dans un autre schéma
- Attributs : identifient et précisent l'adresse du schéma à inclure
- L'élément include ne peut être inclus que dans l'élément schéma

Attributs	Description
id	précise un identificateur unique pour l'élément.
schemaLocation	spécifie une adresse URI pointant vers un schéma XML.

```
<xsd:include  
    id = ID  
    schemaLocation = anyURI  
(tout attribut ayant un espace de noms  
différent de celui du schéma...)>  
Contenu : (annotation?)  
</xsd:include>
```

```
<xsd:schema  
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"  
    elementFormDefault="qualified"  
    attributeFormDefault="unqualified">  
  
<xsd:include schemaLocation="c:\inclusion.xsd"/>
```



XML – Elément import

DEFINITION

- **<xsd:import>...</xsd:import>**
- Permet d'importer un schéma d'espace de noms différent dans un autre schéma
 - Attributs : identifient et précisent l'adresse du schema à Inclure
 - L'élément import ne peut être inclus que dans l'élément schema

Attributs	Description
id	précise un identificateur unique pour l'élément.
namespace	spécifie l'espace de noms du schéma XML.
schemaLocation	spécifie une adresse URI pointant vers un schéma XML.

```
<xsd:import  
id = ID  
namespace = anyURI  
schemaLocation = anyURI  
(tout attribut ayant un espace de noms  
différent de celui du schéma...)>  
Contenu : (annotation?)  
</xsd:import>
```

```
<xsd:schema  
xmlns:xsd="http://www.w3.org/2001/XMLSchema"  
elementFormDefault="qualified"  
xmlns="http://www.site.com/schema"  
targetNamespace="http://www.site.com/schema"  
xmlns:site="http://www.site.com">  
<xsd:import  
schemaLocation="c:\import.xsd"  
namespace="http://www.site.com"/>
```



XML – Elément redefine

DEFINITION

- **<xsd:redefine>...</xsd:redefine>**
- Permet d'importer un schéma de même espace de noms et de redéfinir les déclarations dans un autre schéma
 - Attributs : identifient et précisent l'adresse du schéma à Inclure
 - L'élément import ne peut être inclus que dans l'élément schéma

Attributs	Description
id	précise un identificateur unique pour l'élément.
schemaLocation	spécifie une adresse URI pointant vers un schéma XML à rédéfinir.

```
<xsd:redefine  
    id = ID  
    schemaLocation = anyURI  
    {tout attribut ayant un espace de noms  
     différent de celui du schéma...}>  
    Contenu : (annotation |  
              (simpleType | complexType  
              | group | attributeGroup))*  
</xsd:redefine>
```

```
<xsd:schema  
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"  
    elementFormDefault="qualified">  
    <xsd:redefine schemaLocation="c:\redefine.xsd">
```



DEFINITION

Définition d'élément et attributs

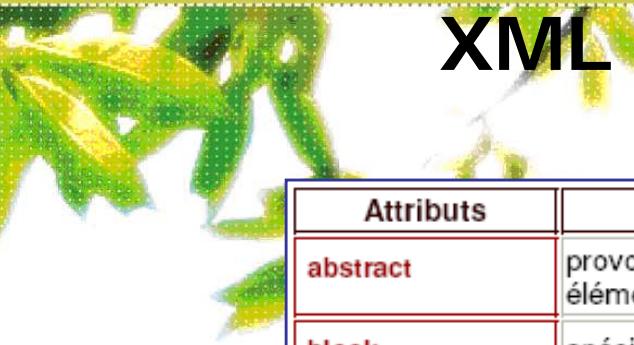


XML – Elément element

- **<xsd:element> ... </xsd:element>**
- Permet de représenter un élément XML dans une définition de schéma
- Ne peut être inclus que dans :
 - all
 - choice
 - schema
 - sequence

DEFINITION

```
<xsd:element  
abstract = boolean : false  
block = (#all | Liste de (substitution | extension | restriction))  
default = string  
final = (#all | Liste de (extension | restriction))  
fixed = string  
form = (qualified | unqualified)  
id = ID  
maxOccurs = (nonNegativeInteger | unbounded) : 1  
minOccurs = nonNegativeInteger : 1  
name = NCName  
nullable = boolean : false  
ref = QName  
substitutionGroup = QName  
type = QName  
{tout attribut ayant un espace de noms  
différent de celui du schéma...}>  
Contenu : (annotation?,  
          ((simpleType | complexType)?,  
           (unique | key | keyref)*))  
</xsd:element>
```



XML – Elément element / attributs

DEFINITION

Attributs	Description
abstract	provoque l'abstraction (<i>true</i>) de l'élément XML, devant être remplacé par un autre élément.
block	spécifie une valeur de blocage du type dans des éléments attendant le type de base.
default	précise une valeur par défaut pour l'élément.
final	empêche la dérivation de type par restriction, extension ou les deux.
fixed	empêche une dérivation par restriction du type de l'élément.
form	indique si l'élément XML doit être ou non qualifié par un espace de noms.
id	précise un identificateur unique pour l'élément.
maxOccurs	précise le nombre d'occurrences maximum de l'élément. Par défaut, ce nombre est égal à 1.
minOccurs	précise le nombre d'occurrences minimum de l'élément. Par défaut, ce nombre est égal à 1.
name	indique le nom de l'élément XML.
nillable	signifie qu'un élément peut être valide (<i>true</i>) lorsqu'il est nul, s'il est porteur d'un attribut qualifié d'espace de noms <i>xsd:nil</i> .
ref	spécifie une référence à un autre élément de schéma.
substitutionGroup	définit un élément pour lequel l'élément peut se substituer.
type	fournit le type de données accepté par l'élément.

XML – Elément element - exemple

DEFINITION

```
<?xml version="1.0"?>
<element_racine
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="c:\schema.xsd">
    <element_enfant>
        Chaîne de caractères
    </element_enfant>
</element_racine>
```

Document XML

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:element name="element_enfant" type="xsd:string"/>
    <xsd:element name="element_racine">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element ref="element_enfant"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
</xsd:schema>
```

Type simple

schéma XML : schema.xsd

L'élément element_enfant est décrit par le biais d'un type simple String

XML – Elément element - exemple

DEFINITION

```
<?xml version="1.0"?>
<element_racine>
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="c:\schema.xsd">
    <element_enfant>
        Chaîne de caractères
    </element_enfant>
</element_racine>
```

Document XML

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:element name="element_enfant" type="xsd:string"/>
    <xsd:element name="element_racine">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element ref="element_enfant"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
</xsd:schema>
```

Type complexe

schéma XML : schema.xsd

L'élément element_racine est un type complexe comprenant (en séquence) l'élément element_enfant

XML – Elément element - exemple

```
<?xml version="1.0" encoding="utf-8"?>
<album
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="CD.xsd">
  <titre>OK Computer</titre>
  <artiste>Radiohead</artiste>
  <datepub>1997-07-01</datepub>
  <pistes>
    <piste>Airbag</piste>
    <piste>Paranoid Android</piste>
    <piste>The Tourist</piste>
  </pistes>
</album>
```

Document XML

schéma XML

DEFINITION

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

  <xsd:complexType name="TypePiste">
    <xsd:sequence>
      <xsd:element name="piste" maxOccurs="12" type="xsd:string" />
    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="TypeAlbum">
    <xsd:sequence>
      <xsd:element name="titre" type="xsd:string"/>
      <xsd:element name="artiste" type="xsd:string"/>
      <xsd:element name="datepub" type="xsd:date"/>
      <xsd:element name="pistes" type="TypePiste"/>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:element name="album" type="TypeAlbum"/>
</xsd:schema>
```

restriction

Types de données xml

Type de données définie dans le schéma



XML – Elément element - exercice

Créer un schéma de validation du document XML suivant :

```
<?xml version="1.0" encoding="UTF-8"?>
<departement xsi:noNamespaceSchemaLocation="departement.xsd"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

    <employe>Georges Durand</employe>

</departement>
```

DEFINITION



XML – Elément element - exercice

DEFINITION

```
<?xml version="1.0" encoding="UTF-8"?>
<departement xsi:noNamespaceSchemaLocation="departement.xsd"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

  <employe>Georges Durand</employe>

</departement>
```

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

  <xsd:element name="employe" type="xsd:string"/>

  <xsd:element name="departement">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="employe"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

</xsd:schema>
```



XML – Elément group

- **<xsd:group>...</xsd:group>**
- Permet de définir un groupe d'éléments
- Ne peut être inclus que dans :
 - choice
 - complex Type
 - redefine
 - schema
 - sequence

```
<xsd:group
  id = ID
  maxOccurs = (nonNegativeInteger | unbounded) : 1
  minOccurs = nonNegativeInteger : 1
  name = NCName
  ref = QName
  {tout attribut ayant un espace de noms
   différent de celui du schéma...}>
  Contenu : (annotation? ,
             (all | choice | sequence)?)
</xsd:group>
```

DEFINITION

Attributs	Description
id	précise un identificateur unique pour le groupe.
maxOccurs	précise le nombre d'occurrences maximum du groupe. Par défaut, ce nombre est égal à 1.
minOccurs	précise le nombre d'occurrences minimum du groupe. Par défaut, ce nombre est égal à 1.
name	indique le nom du groupe.
ref	indique une référence à un groupe d'attributs.



XML – Elément attribute

DEFINITION

- **<xsd:attribute>...</xsd:attribute>**
- Permet de définir un attribut d'élément
- Ne peut être inclus que dans :
 - attributeGroup
 - complex Type
 - extension
 - schema

```
<xsd:attribute  
default = string  
fixed = string  
form = (qualified | unqualified)  
id = ID  
name = NCName  
ref = QName  
type = QName  
use = (optional | prohibited | required) : optional  
{tout attribut ayant un espace de noms  
différent de celui du schéma...}>  
Contenu : (annotation?, (simpleType?))  
</xsd:attribute>
```

Attributs	Description
précise une valeur par défaut pour l'attribut.	
fixed	empêche une dérivation par restriction du type de l'attribut.
form	indique si l'attribut XML doit être ou non qualifié par un espace de noms.
id	précise un identificateur unique pour l'attribut.
name	indique le nom de l'attribut XML.
ref	spécifie une référence à un autre élément de schéma.
type	fournit le type de données accepté par l'attribut.
use	indique comment l'attribut doit apparaître.

Exemple : <xsd:attribute name="maj" type="xsd:date"
use="optional" default="2003-10-11" />



XML – Elément attributeGroup

- **<xsd:attributeGroup>...</xsd:AttributeGroup>**
- Permet de regrouper plusieurs attributs d'éléments
- Ne peut être inclus que dans :
 - attributeGroup
 - complex Type
 - extension
 - redefine
 - schema

```
<xsd:attributeGroup  
id = ID  
name = NCName  
ref = QName  
{tout attribut ayant un espace de noms  
différent de celui du schéma...}>  
Contenu : (annotation?)  
</xsd:attributeGroup>
```

DEFINITION

Attributs	Description
id	précise un identificateur unique pour l'élément.
name	indique le nom du groupe.
ref	indique une référence à un groupe d'attributs.



XML – Elément complexType

DEFINITION

- **<xsd:complexType>...</xsd:complexType>**
- Permet de définir un type de données complexe
- Ne peut être inclus que dans :
 - element
 - redefine
 - schema

```
<xsd:complexType>
abstract = booléen : false
block = (#all | Liste de (extension | restriction))
final = (#all | Liste de (extension | restriction))
id = ID
mixed = booléen : false
name = NCName
{tout attribut ayant un espace de noms
différent de celui du schéma...}>
Contenu : (annotation?, (simpleContent | complexContent |
((group | all | choice | séquence)?,
((attribute | attributeGroup)*, anyAttribute?))))
</xsd:complexType>
```

Attributs	Description
abstract	provoque l'abstraction (<i>true</i>) de l'élément XML, devant être remplacé par un autre élément.
block	spécifie une valeur de blocage du type dans des éléments attendant le type de base.
default	précise une valeur par défaut pour l'élément.
final	empêche la dérivation de type par restriction, extension ou les deux.
id	précise un identificateur unique pour l'élément.
mixed	indique un contenu mixte (<i>true</i>) ou un contenu à base d'éléments seuls (<i>false</i>) par défaut.
name	indique le nom de l'élément XML.



XML – Elément complexType - Exemple

DEFINITION

```
<xsd:complexType>
  <xsd:sequence>
    <xsd:element name="nom" type="xsd:string" />
    <xsd:element name="prénom" type="xsd:string" />
    <xsd:element name="dateDeNaissance" type="xsd:date" />
    <xsd:element name="adresse" type="xsd:string" />
    <xsd:element name="adresseElectronique" type="xsd:string" />
    <xsd:element name="téléphone" type="numéroDeTéléphone" />
  </xsd:sequence>
</xsd:complexType>
```

DEFINITION

XML – exemple

Soit le document XML suivant :

```
<?xml version="1.0" encoding="UTF-8"?>
<librairie xsi:noNamespaceSchemaLocation="librairie.xsd" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

    <livre isbn="2212110472" categorie="XML">
        <titre>Services Web avec XML, SOAP, WSDL, UDDI, ebXML ...</titre>
        <auteur>Jean-Marie Chauvet</auteur>
        <editeur>Eyrolles</editeur>
    </livre>

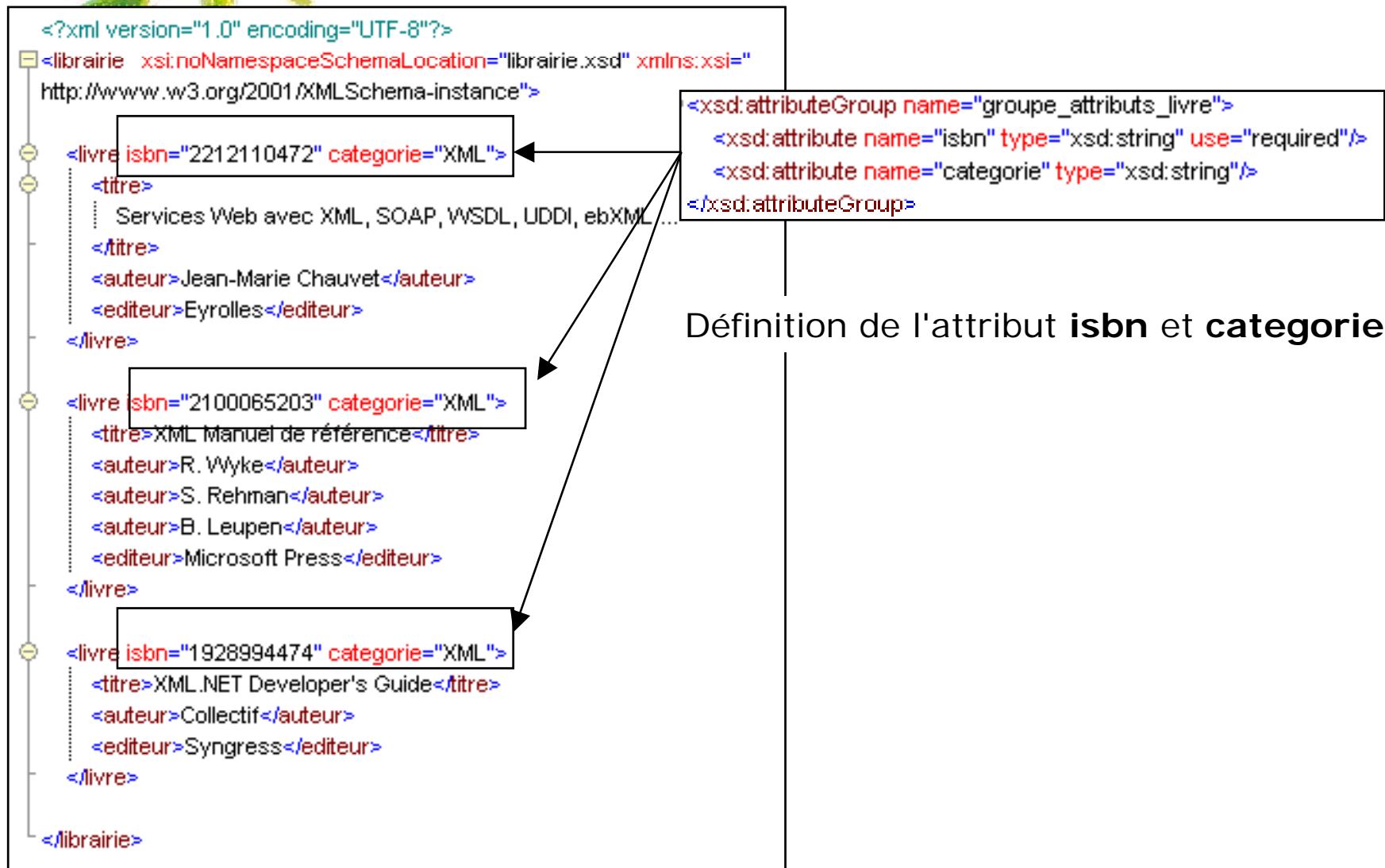
    <livre isbn="2100065203" categorie="XML">
        <titre>XML Manuel de référence</titre>
        <auteur>R. Wyke</auteur>
        <auteur>S. Rehman</auteur>
        <auteur>B. Leupen</auteur>
        <editeur>Microsoft Press</editeur>
    </livre>

    <livre isbn="1928994474" categorie="XML">
        <titre>XML.NET Developer's Guide</titre>
        <auteur>Collectif</auteur>
        <editeur>Syngress</editeur>
    </livre>

</librairie>
```

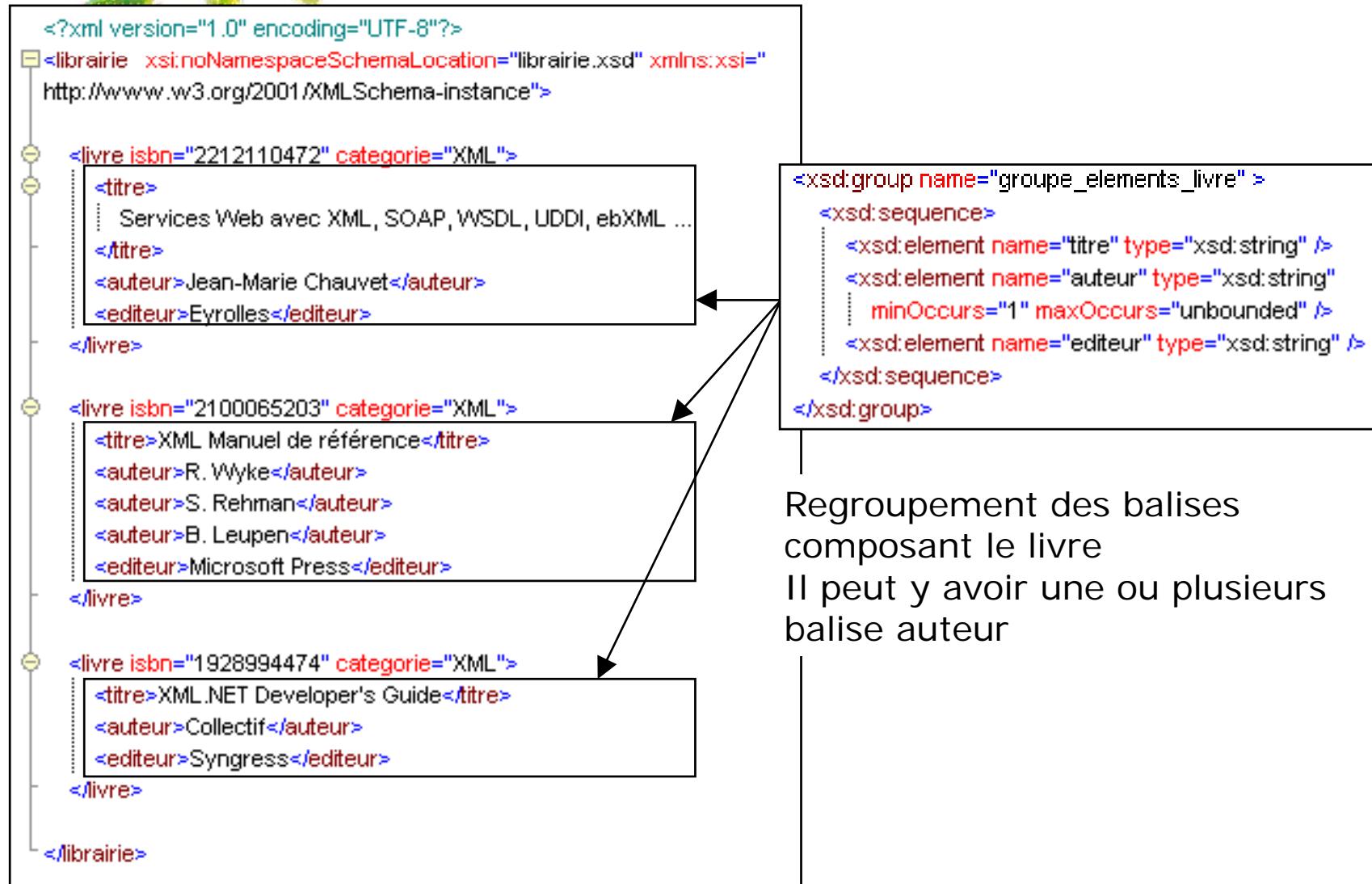
XML – exemple

1°) Regroupement des attributs : **attributeGroup : groupe_attributs_livre**



XML – exemple

2°) Regroupement des éléments : **group : groupe_elements_livre**

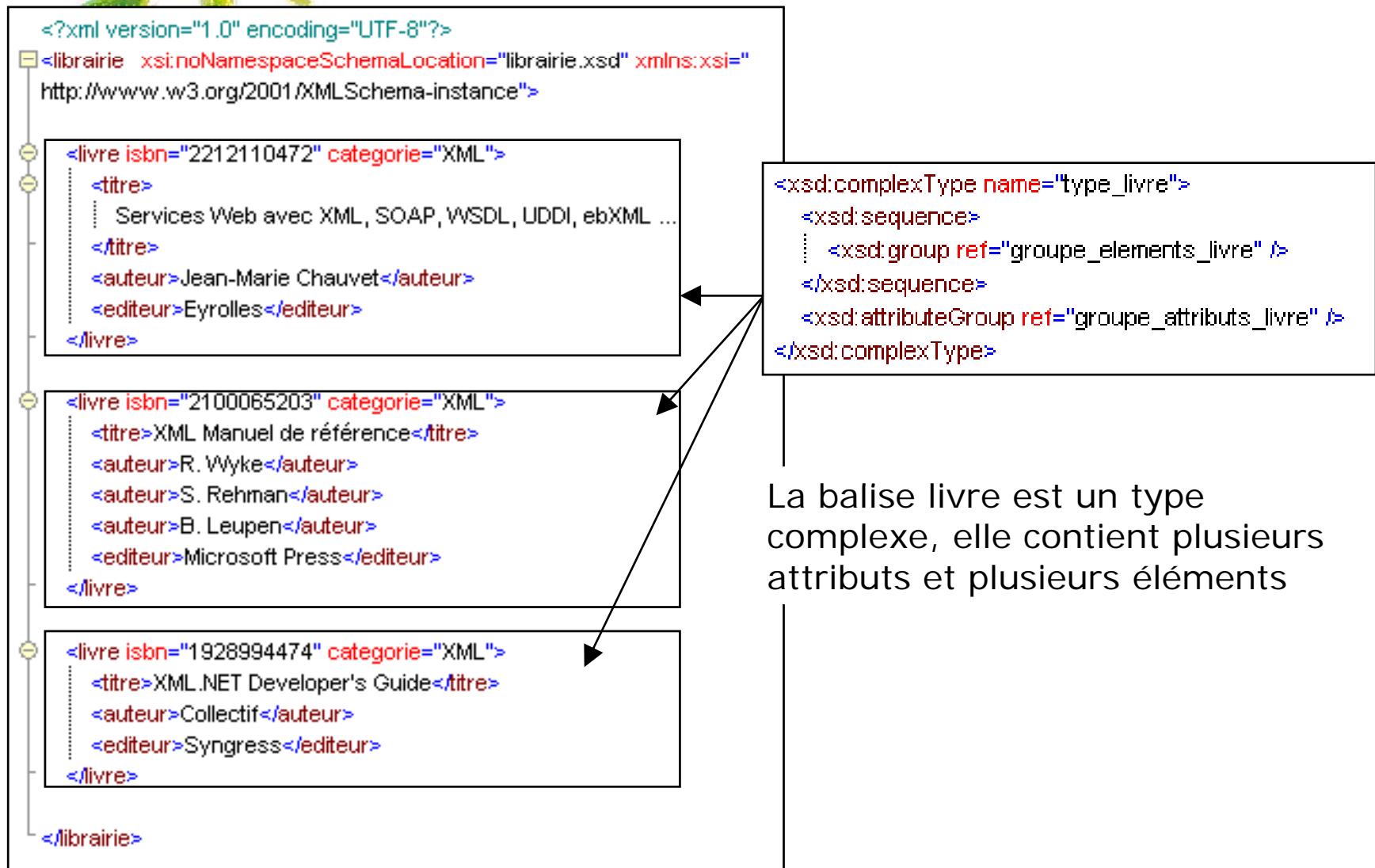


DEFINITION

XML – exemple

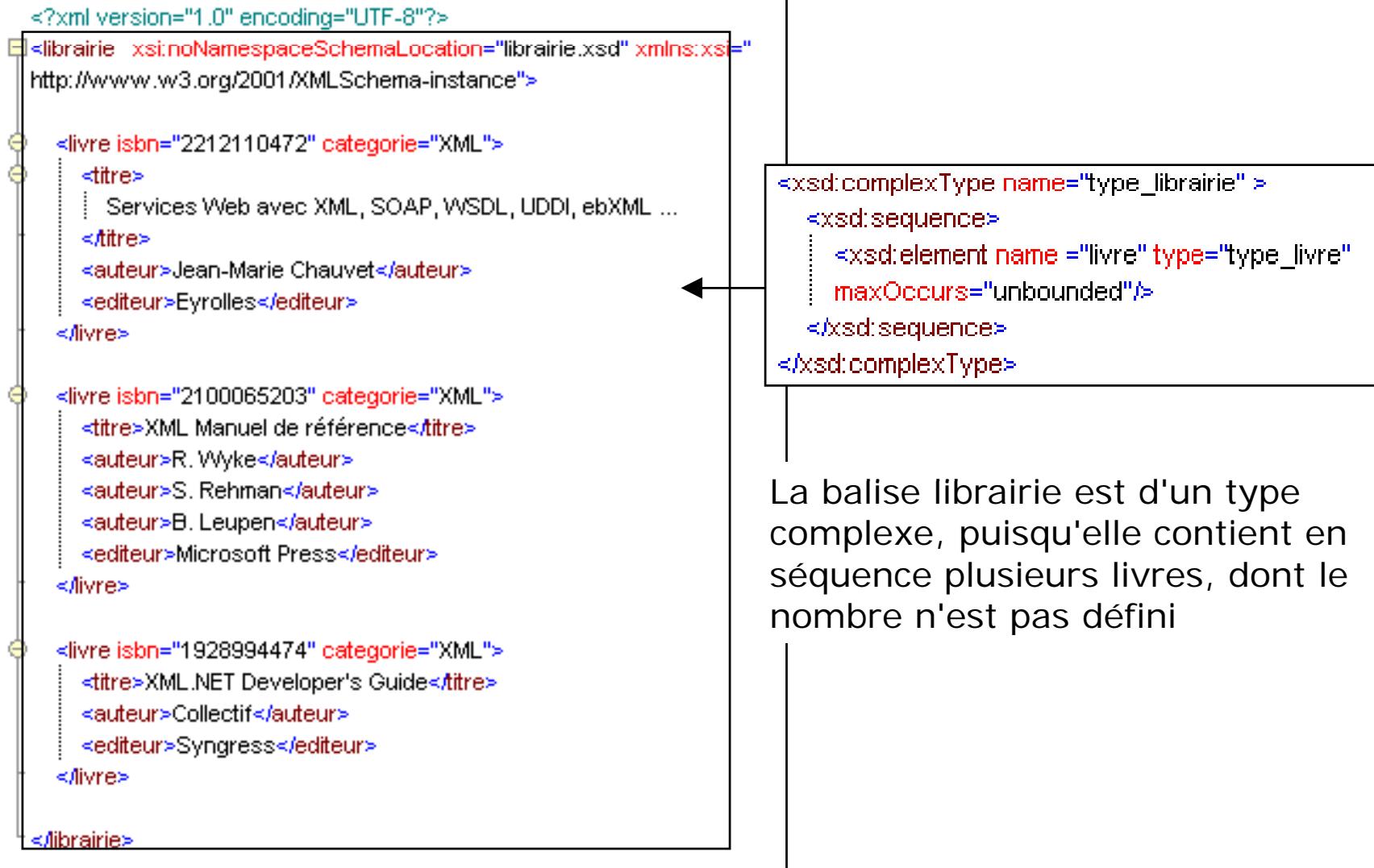
3°) Définition du type de l'élément livre : ComplexType : **type_livre**

DEFINITION



XML – exemple

4°) Définition du type de l'élément librairie : ComplexType : **type_librarie**



XML – exemple

5°) Définition de l'élément racine librairie de type : **type_librairie**



```
<?xml version="1.0" encoding="UTF-8"?>
<librairie xsi:noNamespaceSchemaLocation="librairie.xsd" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

    <livre isbn="2212110472" categorie="XML">
        <titre> Services Web avec XML, SOAP, WSDL, UDDI, ebXML ...
        </titre>
        <auteur>Jean-Marie Chauvet</auteur>
        <éditeur>Eyrolles</éditeur>
    </livre>

    <livre isbn="2100065203" categorie="XML">
        <titre> XML Manuel de référence</titre>
        <auteur>R. Wyke</auteur>
        <auteur>S. Rehman</auteur>
        <auteur>B. Leupen</auteur>
        <éditeur>Microsoft Press</éditeur>
    </livre>

    <livre isbn="1928994474" categorie="XML">
        <titre> XML.NET Developer's Guide</titre>
        <auteur>Collectif</auteur>
        <éditeur>Syngress</éditeur>
    </livre>

</librairie>
```

<xsd:element name="librairie" type="type_librairie"/>

Sans oublier de définir l'élément racine de type type_librairie



XML – Elément complexContent

- **<xsd:complexContent>...</xsd:complexContent>**
- Permet de définir un contenu complexe pour un élément XML
- Ne peut être inclus que dans :
 - complexType

```
<xsd:complexContent
    id = ID
    mixed = booléen
    {tout attribut ayant un espace de noms
        différent de celui du schéma...}>
    Contenu : (annotation?, (restriction | extension))
</xsd:complexContent>
```

DEFINITION

Attributs	Description
id	précise un identificateur unique pour l'élément.
mixed	indique un contenu mixte (<i>true</i>) ou un contenu à base d'éléments seuls (<i>false</i>) par défaut.



XML – Elément simpleType

- **<xsd:simpleType>...</xsd:simpleType>**
- Permet de définir un type de données simple pour des éléments XML
- Ne peut être inclus que dans :
 - attribute
 - element
 - list
 - redefine
 - restriction
 - schema
 - union

```
<xsd:simpleType  
final = (#all | (list | union | restriction))  
id = ID  
name = NCName  
{tout attribut ayant un espace de noms  
différent de celui du schéma...}>  
Contenu : (annotation?, (restriction | list | union))  
</xsd:simpleType>
```

DEFINITION

Attributs	Description
final	empêche la dérivation de type par restriction, extension ou les deux.
id	précise un identificateur unique pour l'élément.
name	indique le nom de l'élément XML.



XML – Elément simpleContent

- **<xsd:simpleContent>...</xsd:simpleContent>**
- Permet de créer un type de données complexe à partir d'un type de données simple
- Ne peut être inclus que dans :
 - complexType

DEFINITION

```
<xsd:simpleContent  
    id = ID  
    {tout attribut ayant un espace de noms  
        différent de celui du schéma...}>  
    Contenu : (annotation?, (restriction | extension))  
</xsd:simpleContent>
```

Attributs	Description
id	précise un identificateur unique pour l'élément.



XML – Elément extension

- **<xsd:extension>...</xsd:extension>**
- Permet d'étendre la définition d'un élément ou attribut XML à un autre type de données spécifié
- Ne peut être inclus que dans :
 - complexContent
 - simpleContent

```
<xsd:extension  
base = QName  
id = ID  
{tout attribut ayant un espace de noms  
différent de celui du schéma...}>  
Contenu : (annotation?,  
((group | all | choice | séquence)?,  
((attribute | attributeGroup)*, anyAttribute?)))  
</xsd:extension>
```

DEFINITION

Attributs	Description
base	indique un type de données de base.
id	précise un identificateur unique pour l'élément.

DEFINITION

XML – Exemple

Soit le document xml :

```
<?xml version="1.0" encoding="UTF-8"?>
<employees xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="employees.xsd">

    <employe>
        <prenom>Albert</prenom>
        <nom>Dupont</nom>
    </employe>

    <employe_full>
        <prenom>Georges</prenom>
        <nom>Durand</nom>
        <adresse>37 grande rue</adresse>
        <ville>Montbéliard</ville>
        <pays>France</pays>
    </employe_full>

</employees>
```

XML – Exemple

1°) Définition du type : **personneinfo**

The diagram illustrates the XML schema definition for the 'personneinfo' type. It shows an XML instance on the left and its corresponding XSD schema on the right.

XML Instance:

```
<?xml version="1.0" encoding="UTF-8"?>
<employes xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="employes.xsd">

    <employe>
        <prenom>Albert</prenom>
        <nom>Dupont</nom>
    </employe>

    <employe_full>
        <prenom>Georges</prenom>
        <nom>Durand</nom>
        <adresse>37 grande rue</adresse>
        <ville>Montbéliard</ville>
        <pays>France</pays>
    </employe_full>

</employes>
```

XSD Schema:

```
<xsd:complexType name="personneinfo">
    <xsd:sequence>
        <xsd:element name="prenom" type="xsd:string"/>
        <xsd:element name="nom" type="xsd:string"/>
    </xsd:sequence>
</xsd:complexType>
```

A callout arrow points from the 'nom' element in the XML instance to the 'xsd:element' declaration in the XSD schema, indicating that the XSD schema defines the type for the 'nom' element.

Description:

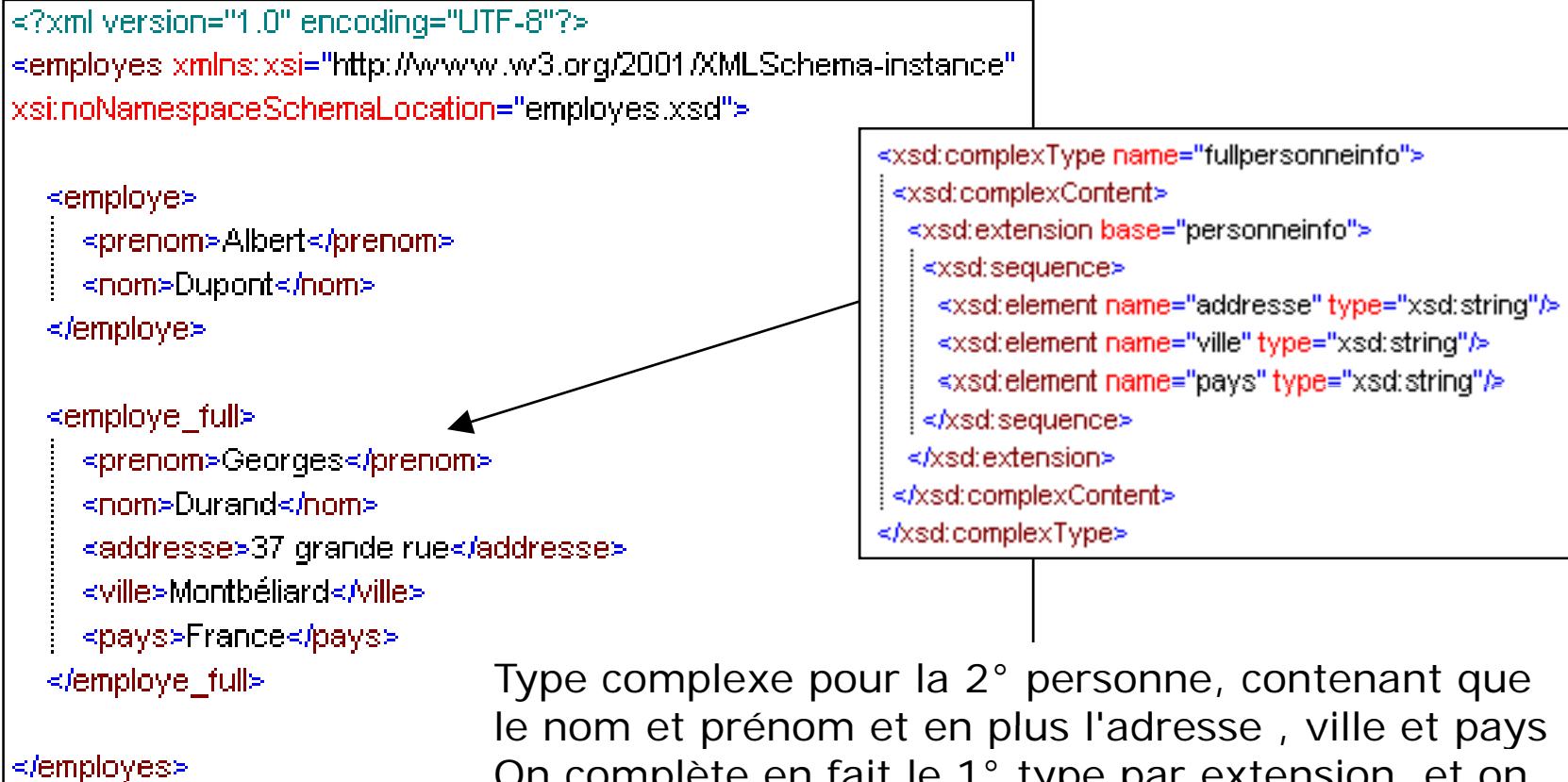
Type complexe pour la 1° personne, ne contenant que le nom et prénom

DEFINITION

XML – Exemple

DEFINITION

2°) Définition du type : **fullpersonneinfo**



Type complexe pour la 2° personne, contenant que le nom et prénom et en plus l'adresse , ville et pays
On complète en fait le 1° type par extension, et on ajoute les éléments supplémentaires
C'est un contenu complexe

XML – Exemple

DEFINITION

3°) Définition du type : **type_employe**

```
<?xml version="1.0" encoding="UTF-8"?>
<employees xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="employees.xsd">

    <employe>
        <prenom>Albert</prenom>
        <nom>Dupont</nom>
    </employe>

    <employe_full>
        <prenom>Georges</prenom>
        <nom>Durand</nom>
        <adresse>37 grande rue</adresse>
        <ville>Montbéliard</ville>
        <pays>France</pays>
    </employe_full>
</employees>
```

```
<xsd:complexType name="type_employe">
    <xsd:sequence>
        <xsd:element name="employe" type="personneinfo"
            minOccurs="0" maxOccurs="unbounded" />
        <xsd:element name="employe_full" type="fullpersonneinfo"
            minOccurs="0" maxOccurs="unbounded" />
    </xsd:sequence>
</xsd:complexType>

<xsd:element name="employees" type="type_employe" />
```

On définit enfin le type de l'élément **employees**, et l'élément racine



XML – Elément union

DEFINITION

- **<xsd:union>...</xsd:union>**
- Permet à un élément ou attribut XML, d'être une ou plusieurs instances d'un type de donnée formé par la réunion de plusieurs types atomiques ou listes
- Ne peut être inclus que dans :
 - simple Type

```
<xsd:union  
    id = ID  
    memberTypes = Liste de QName  
    {tout attribut ayant un espace de noms  
     différent de celui du schéma...}>  
    Contenu : (annotation?, (simpleType*))  
</xsd:union>
```

Attributs	Description
id	précise un identificateur unique pour l'élément.
memberTypes	spécifie une liste de noms de types de données séparés par un espace blanc.

XML – Elément union - Exemple

DEFINITION

```
<?xml version="1.0" encoding="UTF-8"?>
<notation xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="notation.xsd">

    <note_devoir>11</note_devoir>
    <note_devoir>AB</note_devoir>
    <note_devoir>12 B+</note_devoir>
    <note_devoir>14</note_devoir>
    <note_devoir>A 20</note_devoir>

</notation>
```

Union de type string et note_chiffree

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

    <xs:simpleType name="note_chiffree">
        <xs:restriction base="xs:integer">
            <xs:minInclusive value="0"/>
            <xs:maxInclusive value="20"/>
        </xs:restriction>
    </xs:simpleType>

    <xs:simpleType name="type_note">
        <xs:union memberTypes="xs:string note_chiffree"/>
    </xs:simpleType>

    <xs:complexType name="type_notation">
        <xs:sequence>
            <xs:element name="note_devoir" type="type_note"
maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>

    <xs:element name="notation" type="type_notation"/>

</xs:schema>
```



CONTRAINTE

Contraintes



XML – Elément any

- **<xsd:any>...</xsd:any>**
- Permet de définir n'importe quel élément
- Ne peut être inclus que dans :
 - choice
 - sequence

CONTRAINTE

```
<xsd:any  
id = ID  
maxOccurs = (nonNegativeInteger | unbounded) : 1  
minOccurs = nonNegativeInteger : 1  
namespace = (##any | ##other)  
| Liste de (anyURI | (##targetNamespace | ##local)) )  
: ##any  
processContents = (lax | skip | strict) : strict  
tout attribut ayant un espace de noms  
différent de celui du schéma...>  
Contenu : (annotation?)  
</xsd:any>
```

Attributs	Description
id	précise un identificateur unique pour l'élément.
maxOccurs	précise le nombre d'occurrences maximum de l'élément. Par défaut, ce nombre est égal à 1.
minOccurs	précise le nombre d'occurrences minimum de l'élément. Par défaut, ce nombre est égal à 1.
namespace	spécifie un ou plusieurs espaces de noms.
processContents	précise le type de processus de contenu.



XML – Elément any - exemple

CONTRAINTE

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<element_racine
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="c:.xsd">
<element_enfant attribut="valeur"/>
<element_enfant attribut="valeur"/>
<b><autre_element attribut="valeur"/></b>
</element_enfant>
<element_enfant attribut="valeur"/>
<b><un_autre_element/></b>
</element_enfant>
</element_racine>
```

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsd:schema
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified">
<xsd:complexType name="autre_elementType">
<xsd:attribute name="attribut" type="xs:string" use="required"/>
</xsd:complexType>
<xsd:complexType name="type_enfant">
<xsd:sequence>
<xsd:any minOccurs="0" processContents="lax"/>
</xsd:sequence>
<xsd:attribute name="attribut" type="xs:string" use="required"/>
</xsd:complexType>
<xsd:element name="element_racine">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="element_enfant"
    type="type_enfant"
    maxOccurs="unbounded"/>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:schema>
```



XML – Elément anyAttribute

- **<xsd:anyAttribute>...</xsd:anyAttribute>**
- Permet de définir n'importe quel attribut dans un schéma
- Ne peut être inclus que dans :
 - attributeGroup
 - complex Type
 - extension

```
<xsd:anyAttribute  
id = ID  
namespace = ((##any | ##other)  
| Liste de (anyURI  
| ##targetNamespace | ##local)) )  
:##any  
processContents = (lax | skip | strict) : strict  
{tout attribut ayant un espace de noms  
différent de celui du schéma...}>  
Contenu : (annotation?)  
</xsd:anyAttribute>
```

CONTRAINTE

Attributs	Description
id	précise un identificateur unique pour l'élément.
namespace	spécifie un ou plusieurs espaces de noms.
processContents	précise le type de processus de contenu.



XML – Elément anyAttribute

CONTRAINTE

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<element_racine
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="c:\schema.xsd">
    <element_enfant attribut="valeur" attribut_2="valeur"/>
    <element_enfant attribut="valeur"/>
</element_racine>
```

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsd:schema
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified">

    <xsd:complexType name="type_enfant">
        <xsd:attribute name="attribut" type="xs:string" use="required"/>
        <xsd:anyAttribute processContents="lax"/>
    </xsd:complexType>
    <xsd:element name="element_racine">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="element_enfant"
                    type="type_enfant"
                    maxOccurs="unbounded"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
</xsd:schema>
```

CONTRAINTE



XML – Elément all

- **<xsd:all>...</xsd:all>**
- Permet de spécifier pour un type complexe un à plusieurs éléments devant apparaître :
 - Une fois ou pas du tout
 - Dans une ordre quelconque
- Ne peut être inclus que dans :
 - complex Type
 - group

```
<xsd:all  
id = ID  
maxOccurs = 1 : 1  
minOccurs = (0 | 1) : 1  
{tout attribut ayant un espace de noms  
différent de celui du schéma...}>  
Contenu : (annotation?, element*)  
</xsd:all>
```

Attributs	Description
id	précise un identificateur unique pour l'élément.
maxOccurs	précise le nombre d'occurrences maximum de l'élément. Par défaut, ce nombre est égal à 1.
minOccurs	précise le nombre d'occurrences minimum de l'élément. Par défaut, ce nombre est égal à 1.

CONTRAINTE

XML – Elément all

```
<?xml version="1.0"?>
<element_racine
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="F:\Untitled2.xsd">
<element_enfant id="01">
<element_3>1000</element_3>
<element_2>100</element_2>
<element_1>10</element_1>
</element_enfant>
<element_enfant id="02">
<element_1>100</element_1>
<element_3>10000</element_3>
<element_2>1000</element_2>
</element_enfant>
</element_racine>
```

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsd:schema
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified">

<xsd:complexType name="element_enfantType">
<xsd:all minOccurs="0" maxOccurs="1">
    <xsd:element name="element_1" type="xsd:integer"/>
    <xsd:element name="element_2" type="xsd:integer"/>
    <xsd:element name="element_3" type="xsd:integer"/>
</xsd:all>
```



XML – Elément choice

- **<xsd:choice>...</xsd:choice>**
- Propose une structure de choix entre plusieurs éléments possibles
- Ne peut être inclus que dans :
 - choice
 - complex Type
 - group
 - sequence

```
<xsd:choice  
id = ID  
maxOccurs = (nonNegativeInteger | unbounded) : 1  
minOccurs = nonNegativeInteger : 1  
{tout attribut ayant un espace de noms  
différent de celui du schéma...}>  
Contenu : (annotation?, (element | group | choice | séquence | any)*)  
</xsd:choice>
```

CONTRAINTE

Attributs	Description
id	précise un identificateur unique pour l'élément.
maxOccurs	précise le nombre d'occurrences maximum de l'élément. Par défaut, ce nombre est égal à 1.
minOccurs	précise le nombre d'occurrences minimum de l'élément. Par défaut, ce nombre est égal à 1.



XML – Elément choice

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<element_racine
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="c:\schema.xsd">
    <element_enfant id="02">
        <element_choix_1>10</element_choix_1>
    </element_enfant>
    <element_enfant id="01">
        <element_choix_2>10</element_choix_2>
    </element_enfant>
</element_racine>
```

CONTRAINTE

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsd:schema
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified">
    <xsd:element name="element_choix_1" type="xsd:integer"/>
    <xsd:element name="element_choix_2" type="xsd:integer"/>

    <xsd:complexType name="element_enfantType">
        <xsd:choice>
            <xsd:element ref="element_choix_1"/>
            <xsd:element ref="element_choix_2"/>
        </xsd:choice>
        <xsd:attribute name="id" use="required">
            <xsd:simpleType>
                <xsd:restriction base="xsd:NMTOKEN">
                    <xsd:enumeration value="01"/>
                    <xsd:enumeration value="02"/>
                </xsd:restriction>
            </xsd:simpleType>
        </xsd:attribute>
    </xsd:complexType>
```



XML – Elément sequence

- **<xsd:sequence>...</xsd:sequence>**
- Permet dans un type de données complexe de définir un à plusieurs éléments devant apparaître dans un ordre prédéfini
- Ne peut être inclus que dans :
 - choice
 - complex Type
 - group
 - sequence

```
<xsd:sequence  
id = ID  
maxOccurs = (nonNegativeInteger | unbounded) : 1  
minOccurs = nonNegativeInteger : 1  
{tout attribut ayant un espace de noms  
différent de celui du schéma...}>  
Contenu : (annotation?,  
          (element | group | choice | séquence | any)*)  
</xsd:sequence>
```

CONTRAINTE

Attributs	Description
id	précise un identificateur unique pour l'élément.
maxOccurs	précise le nombre d'occurrences maximum de l'élément. Par défaut, ce nombre est égal à 1.
minOccurs	précise le nombre d'occurrences minimum de l'élément. Par défaut, ce nombre est égal à 1.



XML – Elément sequence

CONTRAINTE

```
<?xml version="1.0"?>
<element_racine>
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="c:\schema.xsd">
  <element_enfant id="01">
    <element_1>10</element_1>
    <element_2>1000000</element_2>
    <element_3>1</element_3>
  </element_enfant>
  <element_enfant id="02">
    <element_1>100</element_1>
    <element_2>100000</element_2>
    <element_3>10000</element_3>
  </element_enfant>
</element_racine>
```

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsd:schema
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">
  <xsd:element name="element_1" type="xsd:integer"/>
  <xsd:element name="element_2" type="xsd:integer"/>
  <xsd:element name="element_3" type="xsd:integer"/>

  <xsd:complexType name="element_enfantType">
    <xsd:sequence>
      <xsd:element ref="element_1"/>
      <xsd:element ref="element_2"/>
      <xsd:element ref="element_3"/>
    </xsd:sequence>
    <xsd:attribute name="id" use="required">
      <xsd:simpleType>
        <xsd:restriction base="xsd:NMTOKEN">
          <xsd:enumeration value="01"/>
          <xsd:enumeration value="02"/>
        </xsd:restriction>
      </xsd:simpleType>
    </xsd:attribute>
  </xsd:complexType>
```

XML – exercice

client_xsd.xml

Créer un schéma XML permettant la validation de ce fichier

CONTRAINTE

```
<?xml version="1.0" encoding="UTF-8"?>
<client xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="client.xsd">
    <nom_complet>
        <nom>Fino</nom>
        <prenom>Jean-Marie</prenom>
    </nom_complet>
    <adresse>
        <num_rue>24</num_rue>
        <rue>Grand chemin</rue>
        <ville>Montbéliard</ville>
    </adresse>
    <contact>
        <telephone>03-81-90-99-00</telephone>
        <fax>03-81-90-99-01</fax>
        <email>jmfino@divae.fr</email>
    </contact>
</client>
```

CONTRAINTE

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:complexType name="typeNomComplet">
        <xsd:sequence>
            <xsd:element name="nom" type="xsd:string"/>
            <xsd:element name="prenom" type="xsd:string"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="typeAdresse">
        <xsd:sequence>
            <xsd:element name="num_rue" type="xsd:int"/>
            <xsd:element name="rue" type="xsd:string"/>
            <xsd:element name="ville" type="xsd:string"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="typeContact">
        <xsd:sequence>
            <xsd:element name="telephone" type="xsd:string"/>
            <xsd:element name="fax" type="xsd:string"/>
            <xsd:element name="email" type="xsd:string"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="typeClient">
        <xsd:sequence>
            <xsd:element name="nom_complet" type="typeNomComplet"/>
            <xsd:element name="adresse" type="typeAdresse"/>
            <xsd:element name="contact" type="typeContact"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:element name="client" type="typeClient"/>
</xsd:schema>
```

XML – exercice

client.xsd



XML – exercice

agents.xml

Créer un schéma XML permettant la validation de ce fichier

CONTRAINTE

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<personnes xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="agents.xsd">
    <personne>
        <nom>Dupond</nom>
        <service>Achats</service>
    </personne>
    <personne>
        <nom>Durand</nom>
        <service>Achats</service>
    </personne>
    <personne>
        <nom>Dupuis</nom>
        <service>Courrier</service>
    </personne>
</personnes>
```

XML – Correction

agents.xsd

CONTRAINTE

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

    <!-- Définition des types simples -->
    <xsd:element name="nom" type="xsd:string"/>
    <xsd:element name="service" type="xsd:string"/>

    <!-- Définition des types complexes -->
    <!-- Définition élément personne -->
    <xsd:element name="personne">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element ref="nom"/>
                <xsd:element ref="service"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>

    <!-- Définition élément personnes (racine) -->
    <xsd:element name="personnes">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element ref="personne" maxOccurs="unbounded"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
</xsd:schema>
```



XML – Elément list

- **<xsd:list>...</xsd:list>**
- Permet de créer de nouveaux types de listes par dérivation de types de données atomiques existants
- Ne peut être inclus que dans :
 - simple Type

```
<xsd:list  
id = ID  
itemType = QName  
{tout attribut ayant un espace de noms  
différent de celui du schéma...}>  
Contenu : (annotation?, (simpleType?))  
</xsd:list>
```

CONTRAINTE

Attributs	Description
id	précise un identificateur unique pour l'élément.
itemType	spécifie le nom d'un type de données existants.

XML – Elément list

CONTRAINTE

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<element_racine
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="c:\schema.xsd">
    <note_devoir>
        15 16 18 19 18 19
    </note_devoir>
    <note_devoir>
        9 12 11 14 8 15
    </note_devoir>
</element_racine>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified"
    attributeFormDefault="unqualified">
    <xsd:simpleType name="note_chiffree">
        <xsd:restriction base="xsd:integer">
            <xsd:minInclusive value="0"/>
            <xsd:maxInclusive value="20"/>
        </xsd:restriction>
    </xsd:simpleType>

    <xsd:simpleType name="type_note">
        <xsd:list itemType="note_chiffree">
    </xsd:simpleType>

    <xsd:complexType name="type_element_racine">
        <xsd:sequence>
            <xsd:element name="note_devoir"
                type="type_note"
                maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>

    <xsd:element name="element_racine"
        type="type_element_racine"/>
</xsd:schema>
```



XML – Elément unique

- **<xsd:unique>...</xsd:unique>**
- Permet de définir une contrainte d'unicité sur un nœud XML
- Ne peut être inclus que dans :
 - element

CONTRAINTE

Attributs	Description
id	précise un identificateur unique pour l'élément.
name	spécifie un nom pour l'élément.

```
<xsd:unique  
    id = ID  
    name = NCName  
    {tout attribut ayant un espace de noms  
     différent de celui du schéma...}>  
    Contenu : (annotation?, (selector, field+))  
</xsd:unique>
```

CONTRAINTE

XML – Elément unique

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<site:annuaire
    xmlns:site="http://www.site.com"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.site.com c:\schema.xsd">
<page numero="1" langue="US">
<titre>XML Schema</titre>
<lien>http://www.w3.org/XML/Schema/</lien>
<commentaire>
    XML Schemas express shared vocabularies and allow machines
    ...
</commentaire>
<cle_site>1</cle_site>
</page>
<page numero="2" langue="FR">
<titre>XML Schema tome 0 : Introduction</titre>
<lien>http://xmlfr.org/w3c/TR/xmlschema-0/</lien>
<commentaire>
    Le tome 0 de la spécification XML Schema n'est qu'une introduction

```

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsd:schema
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns="http://www.site.com"
    targetNamespace="http://www.site.com">
<xsd:complexType name="type_page">
<xsd:sequence>
    <xsd:element name="titre" type="xsd:string"/>
    <xsd:element name="lien" type="xsd:string"/>
    <xsd:element name="commentaire" type="xsd:string"/>
    <xsd:element name="cle_site"
        type="xsd:positiveInteger"/>
</xsd:sequence>
<xsd:attribute name="numero"
    type="xsd:positiveInteger"
    use="required"/>
<xsd:attribute name="langue"
    type="xsd:string"
    use="required"/>
</xsd:complexType>

<xsd:complexType name="type_site">
<xsd:sequence>
    <xsd:element name="nom" type="xsd:string"/>
    <xsd:element name="url" type="xsd:string"/>
</xsd:sequence>
<xsd:attribute name="numero"
    type="xsd:positiveInteger"
    use="required"/>
</xsd:complexType>

<xsd:complexType name="type_annuaire">
<xsd:sequence>
    <xsd:element name="page"
        maxOccurs="unbounded"
        type="type_page">
        <xsd:unique name="cle_page">
            <xsd:selector xpath=".//page"/>
            <xsd:field xpath=".//*[@numero]"/>
        </xsd:unique>
    </xsd:element>
```



RESTRICTION

Restrictions (facettes)



XML – Elément restriction

RESTRICTION

- **<xsd:restriction>...</xsd:restriction>**
- Permet de restreindre les données permises dans un élément ou attribut XML
- Ne peut être inclus que dans :
 - complexContent
 - simpleContent
 - simpleType

```
<xsd:restriction  
base = QName  
id = ID  
{tout attribut ayant un espace de noms  
différent de celui du schéma...}>  
Contenu : (annotation?,  
          (group | all | choice | séquence)?,  
          ((attribute | attributeGroup)*, anyAttribute?))  
</xsd:restriction>
```

Attributs	Description
base	indique un type de données de base.
id	précise un identificateur unique pour l'élément.



XML – Elément restriction

RESTRICTION

```
<?xml version="1.0"?>
<element_racine
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="c:\schema.xsd">
    <nombre>51</nombre>
    <nombre>90</nombre>
    <nombre>80</nombre>
    <nombre>99</nombre>
</element_racine>
```

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsd:schema
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified">
    <xsd:simpleType name="entre50et100">
        <xsd:restriction base="xsd:nonNegativeInteger">
            <xsd:minExclusive value="50"/>
            <xsd:maxExclusive value="100"/>
        </xsd:restriction>
    </xsd:simpleType>

    <xsd:element name="nombre" type="entre50et100"/>

    <xsd:element name="element_racine">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element ref="nombre" maxOccurs="unbounded"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
</xsd:schema>
```



XML – Elément minInclusive

RESTRICTION

- **<xsd:minInclusive>...</xsd:minInclusive>**
- Permet de définir une valeur minimum inclusive pour un élément ou attribut XML
- Ne peut être inclus que dans :
 - restriction

```
<xsd:minInclusive  
fixed = booléen : false  
id = ID  
value = simpleType  
{tout attribut ayant un espace de noms  
différent de celui du schéma...}>  
Contenu : (annotation?)  
</xsd:minInclusive>
```

Attributs	Description
fixed	permet de fixer la valeur de l'élément.
id	précise un identificateur unique pour l'élément.
value	spécifie une valeur de type simple (simpleType).

XML – Elément minInclusive



RESTRICTION

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<element_racine
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="c:\schema.xsd">
    <valeur attribut="200">0</valeur>
    <valeur attribut="170">19</valeur>
    <valeur attribut="100">100</valeur>
</element_racine>
```

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsd:schema
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified">
    <xsd:element name="element_racine">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="valeur"
                    type="valeurType"
                    maxOccurs="unbounded"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>

    <xsd:complexType name="valeurType">
        <xsd:simpleContent>
            <xsd:extension base="valeurSimpleType">
                <xsd:attribute name="attribut" use="required">
                    <xsd:simpleType>
                        <xsd:restriction base="xsd:nonNegativeInteger">
                            <xsd:minInclusive value="100"/>
                            <xsd:maxInclusive value="200"/>
                        </xsd:restriction>
                    </xsd:simpleType>
                </xsd:attribute>
            </xsd:extension>
        </xsd:simpleContent>
    </xsd:complexType>

    <xsd:simpleType name="valeurSimpleType">
        <xsd:restriction base="xsd:nonNegativeInteger">
            <xsd:minInclusive value="0"/>
            <xsd:maxInclusive value="100"/>
        </xsd:restriction>
    </xsd:simpleType>
</xsd:schema>
```



XML – Elément maxInclusive

RESTRICTION

- **<xsd:maxInclusive>...</xsd:maxInclusive>**
- Permet de définir une valeur maximum inclusive pour un élément ou attribut XML
- Ne peut être inclus que dans :
 - restriction

```
<xsd:maxInclusive  
fixed = booléen : false  
id = ID  
value = simpleType  
{tout attribut ayant un espace de noms  
différent de celui du schéma...}>  
Contenu : (annotation?)  
</xsd:maxInclusive>
```

Attributs	Description
fixed	permet de fixer la valeur de l'élément.
id	précise un identificateur unique pour l'élément.
value	spécifie une valeur de type simple (simpleType).



XML – Elément minExclusive

RESTRICTION

- **<xsd:minExclusive>...</xsd:minExclusive>**
- Permet de définir une valeur minimum exclusive pour un attribut ou élément XML
- Ne peut être inclus que dans :
 - restriction

```
<xsd:minExclusive  
fixed = booléen : false  
id = ID  
value = simpleType  
{tout attribut ayant un espace de noms  
différent de celui du schéma...}>  
Contenu : (annotation?)  
</xsd:minExclusive>
```

Attributs	Description
fixed	permet de fixer la valeur de l'élément.
id	précise un identificateur unique pour l'élément.
value	spécifie une valeur de type simple (simpleType).



XML – Elément maxExclusive

RESTRICTION

- **<xsd:maxExclusive>...</xsd:maxExclusive>**
- Permet de définir une valeur maximum exclusive pour un élément ou attribut XML
- Ne peut être inclus que dans :
 - restriction

```
<xsd:maxExclusive  
fixed = booléen : false  
id = ID  
value = simpleType  
{tout attribut ayant un espace de noms  
différent de celui du schéma...}>  
Contenu : (annotation?)  
</xsd:maxExclusive>
```

Attributs	Description
fixed	permet de fixer la valeur de l'élément.
id	précise un identificateur unique pour l'élément.
value	spécifie une valeur de type simple (simpleType).



XML – Elément minLength

RESTRICTION

- **<xsd:minLength>...</xsd:minLength>**
- Permet de définir une longueur minimum pour un élément ou attribut XML
- Ne peut être inclus que dans :
 - restriction

Attributs	Description
fixed	permet de fixer la valeur de l'élément.
id	précise un identificateur unique pour l'élément.
value	spécifie une longueur maximum.

```
<xsd:minLength  
fixed = booléen : false  
id = ID  
value = nonNegativeInteger  
{tout attribut ayant un espace de noms  
différent de celui du schéma...}>  
Contenu : (annotation?)  
</xsd:minLength>
```



XML – Elément minLength

RESTRICTION

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<element_racine
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="c:\schema.xsd">
<adresse ref_site="1201203609">
http://xmlfr.org/documentation/tutoriels/001219-0001#maxlength
</adresse>
<adresse ref_site="3269135683">
http://www.w3.org/TR/2001/REC-xmlschema-2-20010502#rf-maxlength
</adresse>
<adresse ref_site="2180877944">
http://www.xml.com/pub/a/2000/11/29/schemas/part1.html?page=8
</adresse>
</element_racine>
```

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsd:schema
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified">
<xsd:simpleType name="adresse_uri">
<xsd:restriction base="xsd:anyURI">
    <xsd:minLength value="15"/>
    <xsd:maxLength value="255"/>
</xsd:restriction>
</xsd:simpleType>

<xsd:complexType name="adresseType">
<xsd:simpleContent>
    <xsd:extension base="adresse_uri">
        <xsd:attribute name="ref_site" use="required">
            <xsd:simpleType>
                <xsd:restriction base="xsd:string">
                    <xsd:length value="10" fixed="true"/>
                </xsd:restriction>
            </xsd:simpleType>
        </xsd:attribute>
    </xsd:extension>
</xsd:simpleContent>
</xsd:complexType>

<xsd:element name="element_racine">
<xsd:complexType>
    <xsd:sequence>
        <xsd:element name="adresse"
            type="adresseType"
            maxOccurs="unbounded"/>
    </xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:schema>
```



XML – Elément maxLength

RESTRICTION

- **<xsd:maxLength>...</xsd:maxLength>**
- Permet de définir une longueur maximum pour un élément ou attribut XML
- Ne peut être inclus que dans :
 - restriction

Attributs	Description
fixed	permet de fixer la valeur de l'élément.
id	précise un identificateur unique pour l'élément.
value	spécifie une longueur maximum.

```
<xsd:maxLength  
fixed = booléen : false  
id = ID  
value = nonNegativeInteger  
(tout attribut ayant un espace de noms  
différent de celui du schéma...)>  
Contenu : (annotation?)  
</xsd:maxLength>
```



XML – Elément length

RESTRICTION

- **<xsd:length>...</xsd:length>**
- Permet de définir une longueur pour un élément ou attribut XML
- Ne peut être inclus que dans :
 - restriction

Attributs	Description
fixed	permet de fixer la valeur de l'élément.
id	précise un identificateur unique pour l'élément.
value	spécifie une longueur en caractères ou en octets.

```
<xsd:length  
fixed = booléen : false  
id = ID  
value = nonNegativeInteger  
{tout attribut ayant un espace de noms  
différent de celui du schéma...}>  
Contenu : (annotation?)  
</xsd:length>
```



XML – Elément length

RESTRICTION

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<element_racine>
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="c:\schema.xsd">
        <appreciation ref_eleve="TA120360">
            Elève très studieux méritant d'entrée
            en faculté de Lettres
        </appreciation>
        <appreciation ref_eleve="TB213568">
            Bon élève devant être plus attentionné
            pour obtenir de bien meilleur résultat
        </appreciation>
        <appreciation ref_eleve="SA218877">
            Très bon élève méritant de passer
            en terminal scientifique
        </appreciation>
    </element_racine>
```

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsschema>
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified">
        <xselement name="element_racine">
            <xsccomplexType>
                <xsssequence>
                    <xselement name="appreciation"
                        type="type_appreciation"
                        maxOccurs="unbounded"/>
                </xsssequence>
            </xsccomplexType>
        </xselement>

        <xsccomplexType name="type_appreciation">
            <xssimpleContent>
                <xsextension base="xs:string">
                    <xssattribute name="ref_eleve" use="required">
                        <xssimpleType>
                            <xssrestriction base="xs:string">
                                <xsslength value="8" fixed="true"/>
                            </xssrestriction>
                        </xssimpleType>
                    </xssattribute>
                </xsextension>
            </xssimpleContent>
        </xsccomplexType>
    </xsschema>
```



XML – Schéma - Exercice

1 – Ecrire le schéma du document suivant – hello.xml – hello.xsd :

Le message ne doit pas dépasser 32 caractères

RESTRICTION

```
<?xml version="1.0" encoding="UTF-8"?>
<hello xsi:noNamespaceSchemaLocation="hello.xsd" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    Hello World
</hello>
```



XML – Schéma - Exercice

```
<?xml version="1.0" encoding="UTF-8"?>
<hello xsi:noNamespaceSchemaLocation="hello.xsd" xmlns:xsi="
http://www.w3.org/2001/XMLSchema-instance">
    Hello World
</hello>
```

RESTRICTION

```
<?xml version="1.0" encoding="UTF-8"?>
<xss:schema xmlns:xss="http://www.w3.org/2001/XMLSchema">
    <!-- Définition de la chaîne de 32 caractères -->
    <xss:simpleType name="Texte32">
        <xss:restriction base="xss:string">
            <xss:maxLength value="32" />
        </xss:restriction>
    </xss:simpleType>

    <!-- Élément racine -->
    <xss:element name="hello" type="Texte32" />
</xss:schema>
```



XML – Elément enumeration

RESTRICTION

- **<xsd:enumeration>...</xsd:enumeration>**
- Permet de contraindre la valeur d'un élément ou d'un attribut à une seule valeur possible
- Ne peut être inclus que dans :
 - restriction

```
<xsd:enumeration  
id = ID  
value = simpleType  
{tout attribut ayant un espace de noms  
différent de celui du schéma...}>  
Contenu : (annotation?)  
</xsd:enumeration>
```

Attributs	Description
id	précise un identificateur unique pour l'élément.
value	spécifie une valeur de type simple (simpleType).



XML – Elément enumeration

RESTRICTION

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<element_racine
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="c:schema.xsd">
<element_enfant attribut="Veuf">
Monsieur
</element_enfant>
<element_enfant attribut="Mariée">
Madame
</element_enfant>
<element_enfant attribut="Célibataire">
Mademoiselle
</element_enfant>
<element_enfant attribut="Concubin">
Monsieur
</element_enfant>
</element_racine>
```

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsd:schema
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified">
<xsd:complexType name="type_enfant">
<xsd:simpleContent>
    <xsd:extension base="valeurs_enfant">
        <xsd:attribute name="attribut" use="required">
            <xsd:simpleType>
                <xsd:restriction base="xsd:NMTOKEN">
                    <xsd:enumeration value="Célibataire"/>
                    <xsd:enumeration value="Concubin"/>
                    <xsd:enumeration value="Concupine"/>
                    <xsd:enumeration value="Marié"/>
                    <xsd:enumeration value="Mariée"/>
                    <xsd:enumeration value="Veuf"/>
                    <xsd:enumeration value="Veuve"/>
                </xsd:restriction>
            </xsd:simpleType>
        </xsd:attribute>
    </xsd:extension>
</xsd:simpleContent>
</xsd:complexType>

<xsd:simpleType name="valeurs_enfant">
<xsd:restriction base="xsd:string">
    <xsd:enumeration value="Madame"/>
    <xsd:enumeration value="Mademoiselle"/>
    <xsd:enumeration value="Monsieur"/>
</xsd:restriction>
</xsd:simpleType>

<xsd:element name="element_racine">
<xsd:complexType>
    <xsd:sequence>
        <xsd:element name="element_enfant"
            type="type_enfant"
            maxOccurs="unbounded"/>
    </xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:schema>
```



XML – Elément pattern

RESTRICTION

- **<xsd:pattern>...</xsd:pattern>**
- Permet de créer un modèle pour la valeur d'un élément ou attribut XML via une expression régulière
- Ne peut être inclus que dans :
 - restriction

```
<xsd:pattern  
id = ID  
value = simpleType  
{tout attribut ayant un espace de noms  
différent de celui du schéma...}>  
Contenu : (annotation?)  
</xsd:pattern>
```

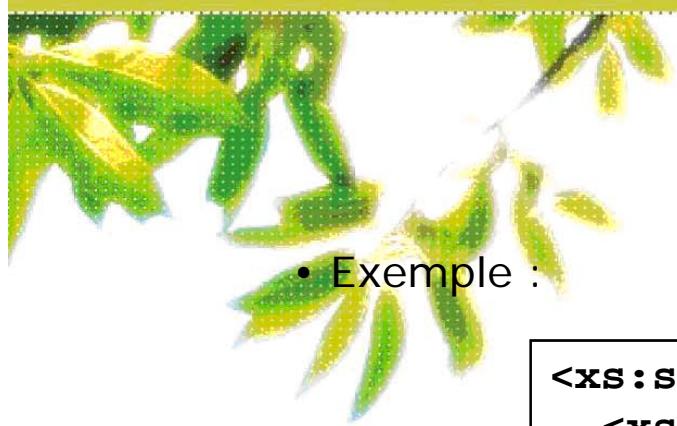
Attributs	Description
id	précise un identificateur unique pour l'élément.
value	spécifie une valeur de type simple (simpleType).



XML – Elément pattern

RESTRICTION

- La facette pattern agit également sur les chaînes de caractères en vérifiant que la valeur est compatible avec une expression régulière.
- Plusieurs facettes peuvent être utilisées et, dans ce cas, il suffira que la valeur concorde avec au moins l'une des expressions régulières définies.
- Les expressions régulières (concept lié au pattern matching) représentent un langage pour décrire des mots (motifs) d'un point de vue syntaxique.
- Elles sont présentes dans la plupart des langages de programmation (Java, Perl...).



XML – Elément pattern

- Exemple :

```
<xs:simpleType nom="monEntier">
  <xss:restriction base="xs:int">
    <xs:pattern value="10"/>
    <xs:pattern value="30"/>
  </xs:restriction>
</xs:simpleType>
```

RESTRICTION

Dans cet exemple, nous autorisons pour le type monEntier, ou bien l'entier 10 ou bien l'entier 30



XML – Elément pattern

RESTRICTION

- Les caractères {}, *, + et ? vont servir à exprimer des quantités de caractères :
 - {m,n} signifie de m à n ;
 - * signifie 0 ou plus ;
 - + signifie au moins 1 ;
 - ? est l'option (0 ou 1).
- Exemple :

```
<xs:simpleType>
  <xs:restriction base="xs:int">
    <xs:pattern value="0{1,2}10?"/>
  </xs:restriction>
</xs:simpleType>
```

Dans cet exemple, nous autorisons les entiers 01, 001, 010 et 0010



XML – Elément pattern

RESTRICTION

- Le langage des expressions régulières comprend également des classes de caractères :
 - \w : n'importe quel caractère
 - \s : blanc ;
 - \S : tout sauf un blanc ;
 - \d : un chiffre ;
 - \D : tout sauf un chiffre ;
 - \w : caractère alphanumérique plus "–" ;
 - \W : tout sauf un caractère alphanumérique plus "–".
- Exemple :

```
<xs:pattern value="\w\d+\w"/>
```
- Cet exemple limite une valeur à un nombre alphanumérique de taille quelconque entouré de 2 caractères mot (A123b, r1t...).



XML – Elément pattern

RESTRICTION

- Il est également possible de spécifier des plages de caractères avec les caractères crochets.
- Exemples :
 - [0-9] : un chiffre
 - [a-z] : une lettre minuscule
 - [a-zA-Z] : une lettre minuscule ou majuscule
 - [^0] : tout sauf 0
 - [0-9E] : un chiffre ou E
- Exemple :

```
<xs:pattern value="[0-9]+[a-z]" />
```
- Ce dernier pattern comprend au moins un chiffre suivi d'un caractère alphabétique en minuscule (1a, 123b, 99z...).



XML – Elément pattern

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<element_racine>
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="c:\schema.xsd">
        <element_enfant>webmaster@laltruiste.com</element_enfant>
        <element_enfant>paris.claude@free.fr </element_enfant>
        <element_enfant>aurore_f@lycos.com</element_enfant>
        <element_enfant>spiderman@web.com</element_enfant>
    </element_racine>
```

RESTRICTION

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsd:schema>
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified">
        <xsd:element name="element_enfant">
            <xsd:simpleType>
                <xsd:restriction base="xsd:string">
                    <b><xsd:pattern value="(.+)@(.+)"></b>
                </xsd:restriction>
            </xsd:simpleType>
        </xsd:element>
        <xsd:element name="element_racine">
            <xsd:complexType>
                <xsd:sequence>
                    <xsd:element ref="element_enfant" maxOccurs="unbounded"/>
                </xsd:sequence>
            </xsd:complexType>
        </xsd:element>
    </xsd:schema>
```

XML – Exercice

RESTRICTION

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<achat commande="CDE-123456"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation="achat.xsd">
    <client>John Smith</client>
    <email>jsmith@gmail.com</email>
    <livraison>
      <nom>Ola Nordmann</nom>
      <adresse>Langgt 23</adresse>
      <ville>4000 Stavanger</ville>
      <pays>Norway</pays>
    </livraison>
    <article>
      <titre>Empire Burlesque</titre>
      <note>Special Edition</note>
      <quantite>1</quantite>
      <prix>10.90</prix>
    </article>
    <article>
      <titre>Hide your heart</titre>
      <quantite>1</quantite>
      <prix>9.90</prix>
    </article>
  </achat>
```

achat.xml

Ecrire le schéma du fichier achat.xml, en considérant :

- Le n° de commande doit contenir 6 chiffres
- Veillez à ce que l'email contienne le symbole @

XML – Exercice

achat.xsd

RESTRICTION

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <!-- definition of simple elements -->
    <xs:element name="client" type="xs:string"/>
    <xs:element name="nom" type="xs:string"/>
    <xs:element name="adresse" type="xs:string"/>
    <xs:element name="ville" type="xs:string"/>
    <xs:element name="pays" type="xs:string"/>
    <xs:element name="titre" type="xs:string"/>
    <xs:element name="note" type="xs:string"/>
    <xs:element name="quantite" type="xs:positiveInteger"/>
    <xs:element name="prix" type="xs:decimal"/>
    <!-- definition of attributes -->
    <xs:attribute name="commande" type="type_cde"/>
    <xs:element name="email">
        <xs:simpleType>
            <xs:restriction base="xs:string">
                <xs:pattern value="(.)+@(.)+."/>
            </xs:restriction>
        </xs:simpleType>
    </xs:element>
    <xs:simpleType name="type_cde">
        <xs:restriction base="xs:string">
            <xs:pattern value="[0-9]{6}">
        </xs:restriction>
    </xs:simpleType>

```

```
<!-- definition of complex elements -->
<xs:element name="livraison">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="nom"/>
            <xs:element ref="adresse"/>
            <xs:element ref="ville"/>
            <xs:element ref="pays"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
<xs:element name="article">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="titre"/>
            <xs:element ref="note" minOccurs="0"/>
            <xs:element ref="quantite"/>
            <xs:element ref="prix"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
<xs:element name="achat">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="client"/>
            <xs:element ref="email"/>
            <xs:element ref="livraison"/>
            <xs:element ref="article" maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:attribute ref="commande" use="required"/>
    </xs:complexType>
</xs:element>
</xs:schema>
```



XML – Elément totalDigits

RESTRICTION

- **<xsd:totalDigits>...</xsd:totalDigits>**
- Permet de définir le nombre total de chiffres dans un élément ou attribut XML
- Ne peut être inclus que dans :
 - restriction

```
<xsd:totalDigits  
fixed = boolean : false  
id = ID  
value = positiveInteger  
{tout attribut ayant un espace de noms  
différent de celui du schéma...}>  
Contenu : (annotation?)  
</xsd:totalDigits>
```

Attributs	Description
fixed	permet de fixer la valeur de l'élément.
id	précise un identificateur unique pour l'élément.
value	spécifie un nombre total de chiffres.



XML – Elément totalDigits

RESTRICTION

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<produit
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="c:\schema.xsd">
    <prix>10.25</prix>
    <prix>1010.55</prix>
    <prix>99.50</prix>
    <prix>23.45</prix>
    <prix>250.00</prix>
    <prix>9999.99</prix>
</produit>
```

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsd:schema
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified">
    <xsd:simpleType name="type_prix">
        <xsd:restriction base="xsd:float">
            <xsd:totalDigits value="6">
                <xsd:fractionDigits value="2"/>
            </xsd:restriction>
        </xsd:simpleType>
        <xsd:element name="produit">
            <xsd:complexType>
                <xsd:sequence>
                    <xsd:element name="prix" type="type_prix" maxOccurs="unbounded"/>
                </xsd:sequence>
            </xsd:complexType>
        </xsd:element>
    </xsd:schema>
```



XML – Elément fractionDigit

RESTRICTION

- **<xsd:fractionDigits>...</xsd:fractionDigits>**
- Permet de définir le nombre total de chiffres dans la partie fractionnaire d'un nombre à virgule flottante
- Ne peut être inclus que dans :
 - restriction

```
<xsd:fractionDigits  
fixed = boolean : false  
id = ID  
value = nonNegativeInteger  
{tout attribut ayant un espace de noms  
différent de celui du schéma...}>  
Contenu : (annotation?)  
</xsd:fractionDigits>
```

Attributs	Description
fixed	permet de fixer la valeur de l'élément.
id	précise un identificateur unique pour l'élément.
value	spécifie une longueur en caractères ou en octets.



XML – Elément fractionDigit

RESTRICTION

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<produit
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="c:\schema.xsd">
    <prix>10.25</prix>
    <prix>1010.55</prix>
    <prix>99.50</prix>
    <prix>23.45</prix>
    <prix>250.00</prix>
    <prix>9999.99</prix>
</produit>
```

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsd:schema
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified">
    <xsd:simpleType name="type_prix">
        <xsd:restriction base="xsd:float">
            <xsd:totalDigits value="6"/>
            <xsd:fractionDigits value="2">
        </xsd:restriction>
    </xsd:simpleType>
    <xsd:element name="produit">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="prix" type="type_prix" maxOccurs="unbounded"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
</xsd:schema>
```



XML – Elément whiteSpace

RESTRICTION

- **<xsd:whiteSpace>...</xsd:whiteSpace>**
- Permet de définir un comportement à adopter vis à vis des espaces blancs dans une valeur type chaîne de caractères
- Ne peut être inclus que dans :
 - restriction

```
<xsd:whiteSpace  
fixed = booléen : false  
id = ID  
value = (collapse | preserve | replace)  
(tout attribut ayant un espace de noms  
différent de celui du schéma...)>  
Contenu : (annotation?)  
</xsd:whiteSpace>
```

Attributs	Description
fixed	permet de fixer la valeur de l'élément.
id	précise un identificateur unique pour l'élément.
value	spécifie un comportement à appliquer aux espaces blancs dans une chaîne de caractères.



XML – Elément whitespace

RESTRICTION

```
<?xml version="1.0" encoding="iso-8859-1"?>
<poesie
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="c:\schema.xsd">
    <titre>Locution des pierrots</titre>
    <texte>Je ne suis qu'un viveur lunaire
    Qui fait des ronds dans le bassin
    Et cela, sans autre dessein
    Que de devenir légendaire.

    Retroussant d'un air de défin
    Mes manches de Mandarin pâle,
    J'arrondis ma bouche et - j'exhale
    Des conseils doux de Crucifix

    Ah! oui, devenir légendaire,
    Au seuil des siècles charlatans !
    Mais où sont les Lunes d'antan ?
    Et que Dieu n'est-il à refaire ?</texte>
    <auteur>Jules Laforgue</auteur>
</poesie>
```

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsd:schema
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified">
    <xsd:simpleType name="type_texte">
        <xsd:restriction base="xsd:string">
            <xsd:whiteSpace value="preserve"/>
        </xsd:restriction>
    </xsd:simpleType>

    <xsd:element name="poesie">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="titre" type="xsd:string"/>
                <xsd:element name="texte" type="type_texte"/>
                <xsd:element name="auteur" type="xsd:string"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
</xsd:schema>
```



ANNOTATION

Annotations



XML – Elément annotation

- **<xsd:annotation>...</xsd:annotation>**
- Permet une structure permettant de fournir des informations applicatives ou destinées à un utilisateur dans un schéma
- Ne peut être inclus que dans :

ANNOTATION

- all
- any
- anyAttribute
- attribute
- attributeGroup
- choice
- complexContent
- complexType
- element
- enumeration
- extension
- field
- fractionDigits
- group
- import
- include
- key
- keyref
- length
- list
- maxExclusive
- maxInclusive
- maxLength
- minExclusive
- minInclusive
- minLength
- pattern
- redefine
- restriction
- schema
- selector
- sequence
- simpleContent
- simpleType
- totalDigits
- union
- unique
- whiteSpace

```
<xsd:annotation>
  id = ID
  {tout attribut ayant un espace de noms
   différent de celui du schéma...}
  Contenu : (appinfo | documentation)*
</xsd:annotation>
```

Attributs	Description
id	précise un identificateur unique pour l'élément.



XML – Elément annotation

ANNOTATION

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<element_racine
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="c:\schema.xsd">
    <code_service>K512</code_service>
    <code_service>R256</code_service>
</element_racine>
```

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsd:schema
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified">
    <xsd:annotation>
        <xsd:documentation>
            Liste des codes de services établie le 12/04/2002
        </xsd:documentation>
    </xsd:annotation>
    <xsd:element name="code_service">
        <xsd:simpleType>
            <xsd:restriction base="xsd:string">
                <xsd:enumeration value="K512"/>
                <xsd:enumeration value="R256"/>
                <xsd:enumeration value="A002"/>
            </xsd:restriction>
        </xsd:simpleType>
    </xsd:element>
    <xsd:element name="element_racine">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element ref="code_service" maxOccurs="unbounded"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
</xsd:schema>
```



XML – Elément appinfo

- **<xsd:appinfo>...</xsd:appinfo>**
- Permet de spécifier des informations destinées à être utilisées par une application
- Ne peut être inclus que dans :
 - annotation

ANNOTATION

Attributs	Description
source	spécifie une adresse URI pointant vers une information.

```
<xsd:appinfo  
source = anyURI>  
Contenu : ({any})*  
</xsd:appinfo>
```

ANNOTATION

```
<?xml version="1.0"?>
<element_racine xmlns="http://www.demo.org"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.demo.org/schema.xsd">
    <a>10</a>
    <b>20</b>
</element_racine>
```

XML – Elément appinfo

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsd:schema
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://www.demo.org"
    xmlns="http://www.demo.org"
    xmlns:sch="http://www.ascc.net/xml/schematron"
    elementFormDefault="qualified">
    <xsd:annotation>
        <xsd:appinfo>
            <sch:title>Schematron Validation</sch:title>
            <sch:ns prefix="d" uri="http://www.demo.org"/>
        </xsd:appinfo>
    </xsd:annotation>
    <xsd:element name="element_racine">
        <xsd:annotation>
            <xsd:appinfo>
                <sch:pattern name="Vérifie que A est plus grand que B">
                    <sch:rule context="d:element_racine">
                        <sch:assert test="d:A > d:B" diagnostics="moinsGrandQue">
                            A doit être plus grand que B.
                        </sch:assert>
                    </sch:rule>
                </sch:pattern>
                <sch:diagnostics>
                    <sch:diagnostic id="moinsGrandQue">
                        Erreur! A est moins grand que B
                        A = <sch:value-of select="d:A"/>
                        B = <sch:value-of select="d:B"/>
                    </sch:diagnostic>
                </sch:diagnostics>
            </xsd:appinfo>
        </xsd:annotation>
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="A" type="xsd:integer"/>
                <xsd:element name="B" type="xsd:integer"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
</xsd:schema>
```



XML – Elément documentation

ANNOTATION

- **<xsd:documentation>...</xsd:documentation>**
- Permet de spécifier des informations à propos du schéma destinées à l'utilisateur
- Ne peut être inclus que dans :
 - annotation

```
<xsd:documentation  
source = anyURI  
xml:lang = language>  
Contenu : ((any))  
</xsd:documentation>
```

Attributs	Description
source	spécifie une adresse URI pointant vers une information.
xml:lang	précise le langage dans lequel est écrit la documentation.



XML – Elément documentation

ANNOTATION

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<element_racine
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="c:\schema.xsd">
    <code_service>K512</code_service>
    <code_service>R256</code_service>
</element_racine>
```

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsd:schema
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified">
    <xsd:element name="code_service">
        <xsd:simpleType>
            <xsd:annotation>
                <xsd:documentation>
Liste des codes de services établie le 12/04/2002
                </xsd:documentation>
            </xsd:annotation>
            <xsd:restriction base="xsd:string">
                <xsd:enumeration value="K512"/>
                <xsd:enumeration value="R256"/>
                <xsd:enumeration value="A002"/>
            </xsd:restriction>
        </xsd:simpleType>
    </xsd:element>
    <xsd:element name="element_racine">
        <xsd:complexType>
            <xsd:annotation>
                <xsd:documentation>
L'élément code_service est limité à 50 occurrences par page
                </xsd:documentation>
            </xsd:annotation>
            <xsd:sequence>
                <xsd:element ref="code_service" maxOccurs="50"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
</xsd:schema>
```



CLEF - REFERENCE

Clefs et références



XML – Elément key

CLEF - REFERENCE

- **<xsd:key>...</xsd:key>**
- Permet de définir un élément clef dans une structure XML
- Ne peut être inclus que dans :
 - element

```
<xsd:key  
id = ID  
name = NCName  
{tout attribut ayant un espace de noms  
différent de celui du schéma...}>  
Contenu : (annotation?, (selector, field+))  
</xsd:key>
```

Attributs	Description
id	précise un identificateur unique pour l'élément.
name	spécifie un nom pour l'élément.



XML – Elément keyref

CLEF - REFERENCE

- **<xsd:keyref>...</xsd:keyref>**
- Permet de créer un référence a une clef existante dans un schéma XML
- Ne peut être inclus que dans :
 - element

Attributs	Description
id	précise un identificateur unique pour l'élément.
name	spécifie un nom pour l'élément.
refer	se réfère à un élément <i>key</i> existant.

```
<xsd:keyref  
id = ID  
name = NCName  
refer = QName  
{tout attribut ayant un espace de noms  
différent de celui du schéma...}>  
Contenu : (annotation?, (selector, field+))  
</xsd:keyref>
```



XML – Elément selector

CLEF - REFERENCE

- **<xsd:selector>...</xsd:selector>**
- Permet de définir une expression XPath chargée de sélectionner un élément XML pour lui appliquer une clef
- Ne peut être inclus que dans :
 - key
 - keyref
 - unique

```
<xsd:selector  
id = ID  
xpath = {sous-ensemble d'expressions XPath}  
{tout attribut ayant un espace de noms  
différent de celui du schéma...}>  
Contenu : (annotation?)  
</xsd:selector>
```

Attributs	Description
id	précise un identificateur unique pour l'élément.
xpath	spécifie un sous-ensemble d'expressions XPath.



XML – Elément field

CLEF - REFERENCE

- **<xsd:field>...</xsd:field>**
- Permet de sélectionner par le biais d'une expression XPath des éléments ou attributs destinés à être utilisés comme clef
- Ne peut être inclus que dans :
 - key
 - keyref
 - unique

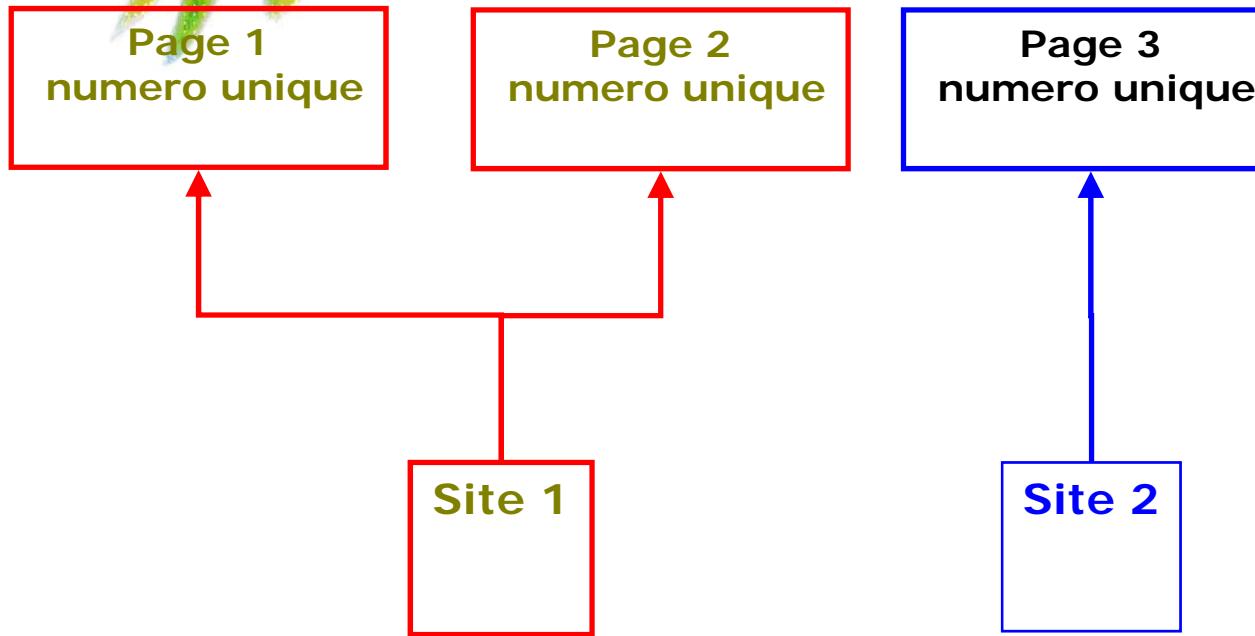
```
<xsd:field  
id = ID  
xpath = {sous-ensemble d'expressions XPath}  
{tout attribut ayant un espace de noms  
différent de celui du schéma...}>  
Contenu : (annotation?)  
</xsd:field>
```

Attributs	Description
id	précise un identificateur unique pour l'élément.
xpath	spécifie un sous-ensemble d'expressions XPath.

XML – Exemple



L'attribut numero des pages est unique



CLEF - REFERENCE

L'attribut numero des sites est une clef

CLEF - REFERENCE

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<annuaire xsi:noNamespaceSchemaLocation="annuaire.xsd"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <page numero="1" langue="US">
    <titre>XML Schema</titre>
    <lien>http://www.w3.org/XML/Schema/</lien>
    <cle_site>1</cle_site>
  </page>

  <page numero="2" langue="FR">
    <titre>XML Schema tome 0 : Introduction</titre>
    <lien>http://xmlfr.org/w3c/TR/xmlschema-0/</lien>
    <cle_site>1</cle_site>
  </page>

  <page numero="3" langue="FR">
    <titre>XML Schema tome 1 : Structures</titre>
    <lien>http://xmlfr.org/w3c/TR/xmlschema-1/</lien>
    <cle_site>2</cle_site>
  </page>

  <site numero="1">
    <nom>W3C</nom>
    <url>http://www.w3.org/</url>
  </site>

  <site numero="2">
    <nom>XMLfr</nom>
    <url>http://www.xmlfr.org/</url>
  </site>
</annuaire>
```

XML – Exemple

Clef du site de référence
numero de l'élément site

XML – Exemple

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<annuaire xsi:noNamespaceSchemaLocation="annuaire.xsd"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <page numero="1" langue="US">
    <titre>XML Schema</titre>
    <lien>http://www.w3.org/XML/Schema/</lien>
    <cle_site>1</cle_site>
  </page>

  <page numero="2" langue="FR">
    <titre>XML Schema tome 0 : Introduction</titre>
    <lien>http://xmlfr.org/w3c/TR/xmlschema-0/</lien>
    <cle_site>1</cle_site>
  </page>

  <page numero="3" langue="FR">
    <titre>XML Schema tome 1 : Structures</titre>
    <lien>http://xmlfr.org/w3c/TR/xmlschema-1/</lien>
    <cle_site>2</cle_site>
  </page>

  <site numero="1">
    <nom>W3C</nom>
    <url>http://www.w3.org/</url>
  </site>

  <site numero="2">
    <nom>XMLfr</nom>
    <url>http://www.xmlfr.org/</url>
  </site>
</annuaire>
```

```
<xsd:complexType name="type_page">
  <xsd:sequence>
    <xsd:element name="titre" type="xsd:string"/>
    <xsd:element name="lien" type="xsd:string"/>
    <xsd:element name="cle_site" type="xsd:positiveInteger"/>
  </xsd:sequence>
  <xsd:attribute name="numero" type="xsd:positiveInteger" use="required"/>
  <xsd:attribute name="langue" type="xsd:string" use="required"/>
</xsd:complexType>
```

Définition du type pour la page

L'élément **cle_site** est déclaré simplement en positiveinteger

L'attribut **numero** est simplement obligatoire

XML – Exemple

CLEF - REFERENCE

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<annuaire xsi:noNamespaceSchemaLocation="annuaire.xsd"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <page numero="1" langue="US">
    <titre>XML Schema</titre>
    <lien>http://www.w3.org/XML/Schema/</lien>
    <cle_site>1</cle_site>
  </page>

  <page numero="2" langue="FR">
    <titre>XML Schema tome 0 : Introduction</titre>
    <lien>http://xmlfr.org/w3c/TR/xmlschema-0/</lien>
    <cle_site>1</cle_site>
  </page>

  <page numero="3" langue="FR">
    <titre>XML Schema tome 1 : Structures</titre>
    <lien>http://xmlfr.org/w3c/TR/xmlschema-1/</lien>
    <cle_site>2</cle_site>
  </page>

  <site numero="1">
    <nom>W3C</nom>
    <url>http://www.w3.org/</url>
  </site>

  <site numero="2">
    <nom>XMLfr</nom>
    <url>http://www.xmlfr.org/</url>
  </site>
</annuaire>
```

```
<xsd:complexType name="type_site">
  <xsd:sequence>
    <xsd:element name="nom" type="xsd:string"/>
    <xsd:element name="url" type="xsd:string"/>
  </xsd:sequence>
  <xsd:attribute name="numero" type="xsd:positiveInteger" use="required"/>
</xsd:complexType>
```

Définition du type pour le site

L'attribut **numero** est simplement obligatoire

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<annuaire xsi:noNamespaceSchemaLocation="annuaire.xsd"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <page numero="1" langue="US">
    <titre>XML Schema</titre>
    <lien>http://www.w3.org/XML/Schema/</lien>
    <cle_site>1</cle_site>
  </page>

  <page numero="2" langue="FR">
    <titre>XML Schema tome 0 : Introduction</titre>
    <lien>http://xmlfr.org/w3c/TR/xmlschema-0/</lien>
    <cle_site>1</cle_site>
  </page>

  <page numero="3" langue="FR">
    <titre>XML Schema tome 1 : Structures</titre>
    <lien>http://xmlfr.org/w3c/TR/xmlschema-1/</lien>
    <cle_site>2</cle_site>
  </page>

  <site numero="1">
    <nom>W3C</nom>
    <url>http://www.w3.org/</url>
  </site>

  <site numero="2">
    <nom>XMLfr</nom>
    <url>http://www.xmlfr.org/</url>
  </site>
</annuaire>
```

XML – Exemple

Définition du type pour l'annuaire

```
<xsd:complexType name="type_annuaire">
  <xsd:sequence>
    <xsd:element name="page">
    <xsd:element name="site">
  </xsd:sequence>
</xsd:complexType>
```

On déclare en séquence dans un type complexe les éléments **page** et **site**

XML – Exemple

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<annuaire xsi:noNamespaceSchemaLocation="annuaire.xsd"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <page numero="1" langue="US">
    <titre>XML Schema</titre>
    <lien>http://www.w3.org/XML/Schema/</lien>
    <cle_site>1</cle_site>
  </page>

  <page numero="2" langue="FR">
    <titre>XML Schema tome 0 : Introduction</titre>
    <lien>http://xmlfr.org/w3c/TR/xmlschema-0/</lien>
    <cle_site>1</cle_site>
  </page>

  <page numero="3" langue="FR">
    <titre>XML Schema tome 1 : Structures</titre>
    <lien>http://xmlfr.org/w3c/TR/xmlschema-1/</lien>
    <cle_site>2</cle_site>
  </page>

  <site numero="1">
    <nom>W3C</nom>
    <url>http://www.w3.org/</url>
  </site>

  <site numero="2">
    <nom>XMLfr</nom>
    <url>http://www.xmlfr.org/</url>
  </site>
</annuaire>
```

Définition du type pour l'annuaire
L'élément de **type_page**

```
<xsd:element name="page" maxOccurs="unbounded" type="type_page">
  <xsd:unique name="cle_page">
    <xsd:selector xpath=".//page"/>
    <xsd:field xpath="./@numero"/>
  </xsd:unique>
</xsd:element>
```

On déclare un contrainte d'unicité :
clef_page, permettant de préciser :

- Qu'elle est unique (xsd:unique)
- Qu'elle provient de la balise page située à un nœud en dessous de l'élément racine (.//page)
- Qu'elle concerne l'attribut numero (./@numero)

CLEF - REFERENCE

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<annuaire xsi:noNamespaceSchemaLocation="annuaire.xsd"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <page numero="1" langue="US">
    <titre>XML Schema</titre>
    <lien>http://www.w3.org/XML/Schema/</lien>
    <cle_site>1</cle_site>
  </page>

  <page numero="2" langue="FR">
    <titre>XML Schema tome 0 : Introduction</titre>
    <lien>http://xmlfr.org/w3c/TR/xmlschema-0/</lien>
    <cle_site>1</cle_site>
  </page>

  <page numero="3" langue="FR">
    <titre>XML Schema tome 1 : Structures</titre>
    <lien>http://xmlfr.org/w3c/TR/xmlschema-1/</lien>
    <cle_site>2</cle_site>
  </page>

  <site numero="1">
    <nomp>W3C</nomp>
    <url>http://www.w3.org/</url>
  </site>

  <site numero="2">
    <nomp>XMLfr</nomp>
    <url>http://www.xmlfr.org/</url>
  </site>
</annuaire>
```

XML – Exemple

Définition du type pour l'annuaire
L'élément de **type_site**

```
<xsd:element name="site" maxOccurs="unbounded" type="type_site">
  <xsd:key name="num_site">
    <xsd:selector xpath=".//site"/>
    <xsd:field xpath="./@numero"/>
  </xsd:key>
  <xsd:keyref name="cle_site" refer="num_site">
    <xsd:selector xpath=".//page"/>
    <xsd:field xpath=".//cle_site"/>
  </xsd:keyref>
</xsd:element>
```

On déclare une clef : num_site,
permettant de préciser :

- Que c'est une clef (xsd:key)
- Qu'elle provient de la balise site située à un nœud en dessous de l'élément racine (.//site)
- Qu'elle concerne l'attribut numero (.//@numero)

```

<?xml version="1.0" encoding="ISO-8859-1"?>
<annuaire xsi:noNamespaceSchemaLocation="annuaire.xsd"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <page numero="1" langue="US">
    <titre>XML Schema</titre>
    <lien>http://www.w3.org/XML/Schema/</lien>
    <cle_site>1</cle_site>
  </page>

  <page numero="2" langue="FR">
    <titre>XML Schema tome 0 : Introduction</titre>
    <lien>http://xmlfr.org/w3c/TR/xmlschema-0/</lien>
    <cle_site>1</cle_site>
  </page>

  <page numero="3" langue="FR">
    <titre>XML Schema tome 1 : Structures</titre>
    <lien>http://xmlfr.org/w3c/TR/xmlschema-1/</lien>
    <cle_site>2</cle_site>
  </page>

  <site numero="1">
    <nomp>W3C</nomp>
    <url>http://www.w3.org/</url>
  </site>

  <site numero="2">
    <nomp>XMLfr</nomp>
    <url>http://www.xmlfr.org/</url>
  </site>
</annuaire>

```

XML – Exemple

Définition du type pour l'annuaire
L'élément de **type_site**

```

<xsd:element name="site" maxOccurs="unbounded" type="type_site">
  <xsd:key name="num_site">
    <xsd:selector xpath=".//site"/>
    <xsd:field xpath="./@numero"/>
  </xsd:key>
  <xsd:keyref name="cle_site" refer="num_site">
    <xsd:selector xpath=".//page"/>
    <xsd:field xpath=".//cle_site"/>
  </xsd:keyref>
</xsd:element>

```

On déclare la référence pour les autre éléments à la clef : cle_site, permettant de préciser :

- Que c'est une référence (xsd:keyref) à la clef num_site
- Qu'elle concerne la balise page (.//page)
- Que c'est une balise qui se nomme cle_site (.//cle_site)

XML – Exercice

- Créer le schéma correspondant au fichier lesProjets.xml

Projet

Projet_id = "p1"

But : Plan de développement stratégique

Projet

Projet_id = "p2"

But : Déployez Linux

CLEF - REFEREE

Employé

numéro_secu : "ss123-45-6789"
nom : "Fred Smith"

Employé

numéro_secu : "ss876-54-3210"
nom : "Jill Jones"

Employé

numéro_secu : "ss876-12-3456"
nom : "Sidney Lee"

```
<lesProjets>
```

```
<projet projet_id="p1">
```

```
<but>Plan de développement stratégique</but>
```

```
<membre équipe employé="ss123-45-6789"/>
```

```
<membre équipe employé="ss876-54-3210"/></projet>
```

```
<projet projet_id="p2">
```

```
<but>Déployez Linux</but>
```

```
<membre équipe employé="ss123-45-6789"/>
```

```
<membre équipe employé="ss876-12-3456"/></projet>
```

```
<employé numéro_sécu="ss123-45-6789">
```

```
<nom>Fred Smith</nom>
```

```
<tâches projet_id="p1"/>
```

```
<tâches projet_id="p2"/></employé>
```

```
<employé numéro_sécu="ss876-54-3210">
```

```
<nom>Jill Jones</nom>
```

```
<tâches projet_id="p1"/></employé>
```

```
<employé numéro_sécu="ss876-12-3456">
```

```
<nom>Sydney Lee</nom>
```

```
<tâches projet_id="p2"/></employé>
```

```
</lesProjets>
```

On considère le numéro_secu unique pour l'employé
Et projet_id comme clef pour le projet

XML – Exercice

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<lesProjets>
  <projet projet_id="p1">
    <but>Plan de développement stratégique</but>
    <membre équipe employé="ss123-45-6789">
      <membre équipe employé="ss876-54-3210">
    </membre équipe>
  </projet>
  <projet projet_id="p2">
    <but>Déployez Linux</but>
    <membre équipe employé="ss123-45-6789">
      <membre équipe employé="ss876-12-3456">
    </membre équipe>
  </projet>
  <employé numéro_sécu="ss123-45-6789">
    <nom>Fred Smith</nom>
    <tâches projet_id="p1"/>
    <tâches projet_id="p2"/>
  </employé>
  <employé numéro_sécu="ss876-54-3210">
    <nom>Jill Jones</nom>
    <tâches projet_id="p1"/>
  </employé>
  <employé numéro_sécu="ss876-12-3456">
    <nom>Sydney Lee</nom>
    <tâches projet_id="p2"/>
  </employé>
</lesProjets>
```

Définition du **type_projet** pour les projets

```
<xsd:complexType name="type_projet">
  <xsd:sequence>
    <xsd:element name="but" type="xsd:string"/>
    <xsd:element name="membre équipe" type="type_membre équipe">
      maxOccurs="unbounded"
    </xsd:element>
  </xsd:sequence>
  <xsd:attribute name="projet_id" type="xsd:string" use="required"/>
</xsd:complexType>
```

Définition du **type_membre équipe** pour membre_équipe des projets

```
<xsd:complexType name="type_membre équipe">
  <xsd:attribute name="employé" type="xsd:string" use="required"/>
</xsd:complexType>
```

XML – Exercice

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<lesProjets>
    <projet projet_id="p1">
        <but>Plan de développement stratégique</but>
        <membre équipe employé="ss123-45-6789"/>
        <membre équipe employé="ss876-54-3210"/>
    </projet>
    <projet projet_id="p2">
        <but>Déployez Linux</but>
        <membre équipe employé="ss123-45-6789"/>
        <membre équipe employé="ss876-12-3456"/>
    </projet>
    <employé numéro_sécu="ss123-45-6789">
        <nom>Fred Smith</nom>
        <tâches projet_id="p1"/>
        <tâches projet_id="p2"/>
    </employé>
    <employé numéro_sécu="ss876-54-3210">
        <nom>Jill Jones</nom>
        <tâches projet_id="p1"/>
    </employé>
    <employé numéro_sécu="ss876-12-3456">
        <nom>Sydney Lee</nom>
        <tâches projet_id="p2"/>
    </employé>
</lesProjets>
```

Définition du **type_employé** pour les employés

```
<xsd:complexType name="type_employé">
    <xsd:sequence>
        <xsd:element name="nom" type="xsd:string"/>
        <xsd:element name="tâches" type="type_tâche" maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:attribute name="numéro_sécu" type="xsd:string" use="required"/>
</xsd:complexType>
```

Définition du **type_tâche** pour les tâches des employés

```
<xsd:complexType name="type_tâche">
    <xsd:attribute name="projet_id" type="xsd:string" use="required"/>
</xsd:complexType>
```

XML – Exercice

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<lesProjets>
    <projet projet_id="p1">
        <but>Plan de développement stratégique</but>
        <membre équipe employé="ss123-45-6789"/>
        <membre équipe employé="ss876-54-3210"/>
    </projet>
    <projet projet_id="p2">
        <but>Déployez Linux</but>
        <membre équipe employé="ss123-45-6789"/>
        <membre équipe employé="ss876-12-3456"/>
    </projet>
    <employé numéro_sécu="ss123-45-6789">
        <nom>Fred Smith</nom>
        <tâches projet_id="p1"/>
        <tâches projet_id="p2"/>
    </employé>
    <employé numéro_sécu="ss876-54-3210">
        <nom>Jill Jones</nom>
        <tâches projet_id="p1"/>
    </employé>
    <employé numéro_sécu="ss876-12-3456">
        <nom>Sydney Lee</nom>
        <tâches projet_id="p2"/>
    </employé>
</lesProjets>
```

Définition du **type_lesProjets** pour les projets, élément lesProjets

```
<xsd:complexType name="type_lesProjets">
    <xsd:sequence>

        <xsd:element name="projet" type="type_projet" maxOccurs="unbounded">
            <xsd:key name="cle_projet">
                <xsd:selector xpath=".//projet"/>
                <xsd:field xpath=".//*[@projet_id]"/>
            </xsd:key>

            <xsd:keyref name="tâches" refer="cle_projet">
                <xsd:selector xpath=".//employé/tâches"/>
                <xsd:field xpath=".//*[@projet_id]"/>
            </xsd:keyref>

        </xsd:element>

        <xsd:element name="employé" type="type_employé" maxOccurs="unbounded">
            <xsd:unique name="clef_secu">
                <xsd:selector xpath=".//employé"/>
                <xsd:field xpath=".//*[@numéro_sécu]"/>
            </xsd:unique>

        </xsd:element>

    </xsd:sequence>
</xsd:complexType>
```

Elément racine :

```
<xsd:element name="lesProjets" type="type_lesProjets"/>
```

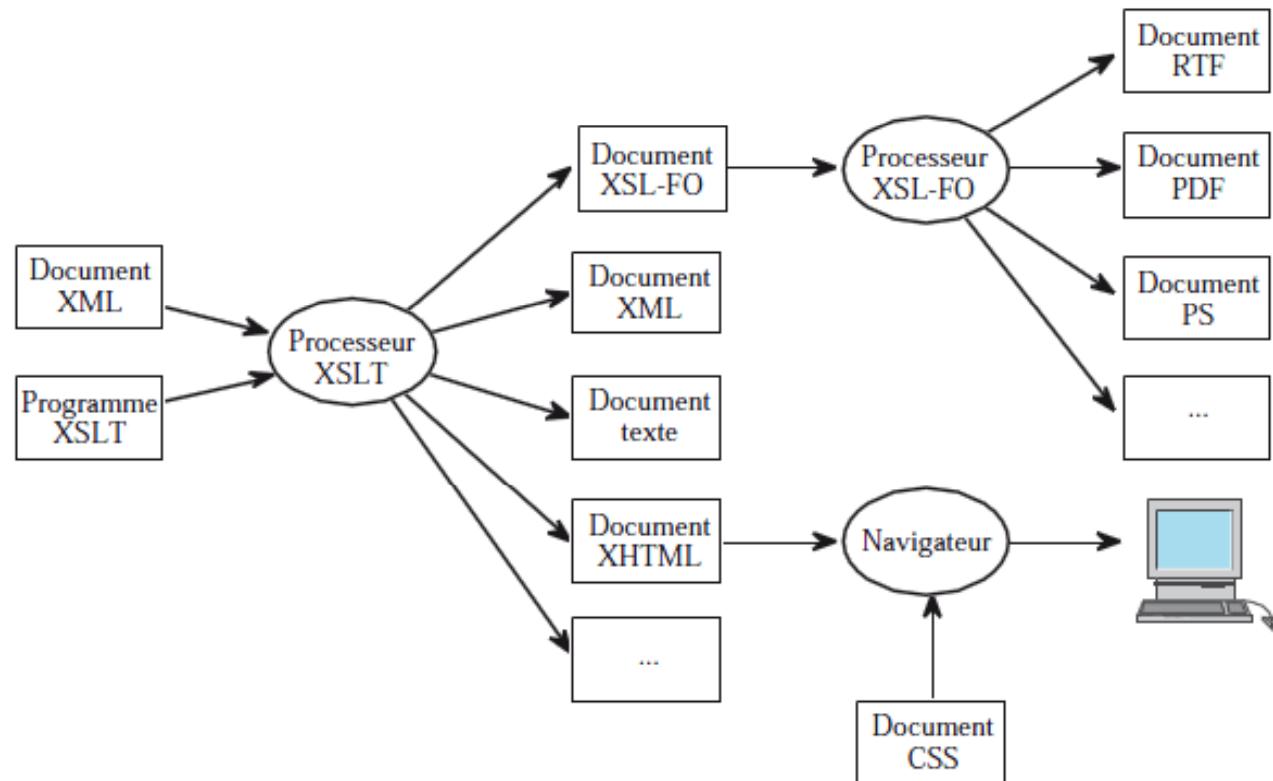


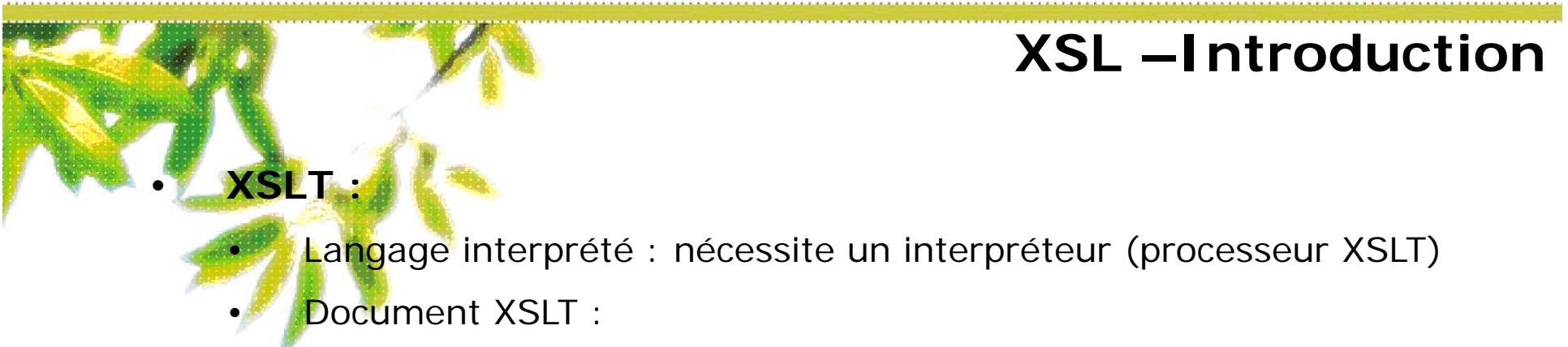
XSL
eXtensible Stylesheet Language

XSLT
XSL Transformation

XSL – Introduction

- XSL est divisé en 2 parties :
 - **XSLT – XSL Transformation** d'un arbre XML en un autre arbre XML
 - **XSL-FO – XSL FOrmating Objects** : langage de type XML utilisé pour la description de pages '*imprimables*' en haute typographie





XSL –Introduction

- **XSLT :**
 - Langage interprété : nécessite un interpréteur (processeur XSLT)
 - Document XSLT :
 - Contient des règles de modèle
 - Chaque règle possède :
 - Un motif
 - Un modèle
- **Processseur XSLT :**
 - Compare les éléments d'un document XML en entrée avec les règles d'une feuille de style
 - Si correspondance => Écriture dans la destination
 - A la fin du processus : sérialisation dans un document en sortie (XML ou autre)



XSLT - Exemple

8.1.xml

- Pas de DTD ou schéma, XSLT travaille aussi bien avec des documents valides qu'invalides

```
<?xml version="1.0" ?>
<people>
  <person born="1912" died="1954">
    <name>
      <first_name>Alan</first_name>
      <last_name>Turing</last_name>
    </name>
    <profession>computer scientist</profession>
    <profession>mathematician</profession>
    <profession>cryptographer</profession>
  </person>

  <person born="1918" died="1988">
    <name>
      <first_name>Richard</first_name>
      <middle_initial>P</middle_initial>
      <last_name>Feynman</last_name>
    </name>
    <profession>physicist</profession>
    <hobby>Playing the bongoes</hobby>
  </person>
</people>
```



XSLT – xsl:stylesheet et xsl:transformation

8.0.xsl

- **Feuille de style XSLT = document XML**
 - Peut avoir une DTD (la plupart n'en ont pas)
 - Élément racine : **stylesheet** ou **transform** (synonymes)
 - Espace de nom : <http://www.w3.org/1999/XSL/Transform>
 - Rattaché au prefixe xsl
 - **xsl:transform** ou **xsl:stylesheet**
- **Exemple : feuille de style XSLT minimale :**

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
                  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

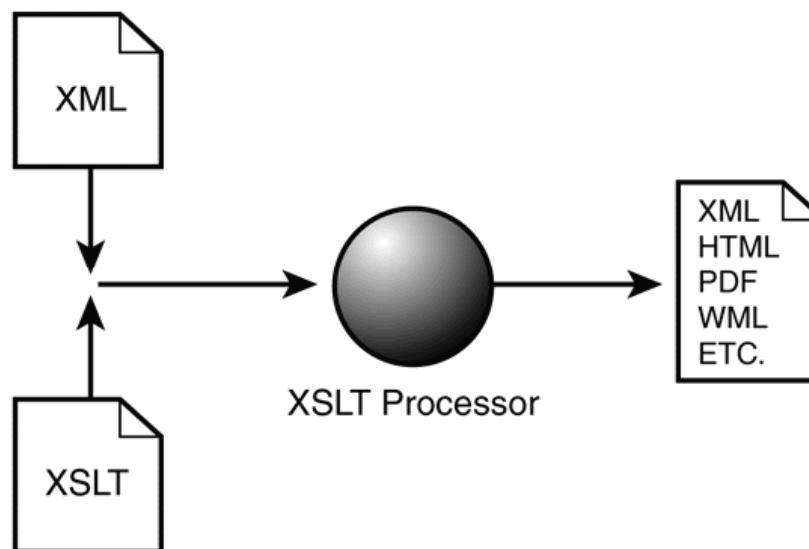
</xsl:stylesheet>
```

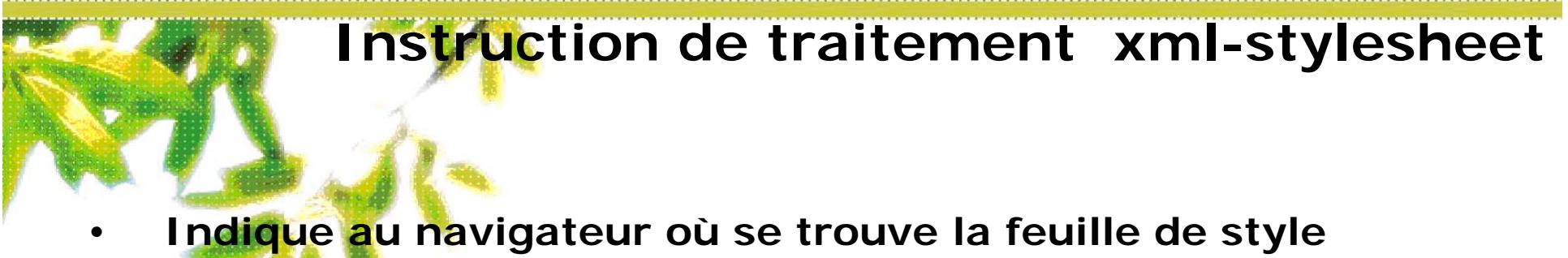


```
<?xml version="1.0"?>AlanTuringcomputer scientistmathematiciancryptographer
RichardPFeynmanphysicistPlaying the bongoes
```

XSLT – Processeur XSLT

- **Processeur XSLT :**
 - Composant logiciel permettant de lire :
 - Un document XML
 - Une feuille de style XSLT
 - Et de convertir en sortie :
 - Un document suivant les instructions spécifiées par la feuille de style





Instruction de traitement xmlstylesheet

- **Indique au navigateur où se trouve la feuille de style**
 - Si la feuille de style est de type XSLT, l'attribut type doit être renseigné (application-xml)
 - Exemple :

```
<?xml version="1.0" ?>
<?xml-stylesheet type="application/xml" href="http://www.divae.fr/styles/personne.xsl" ?>
<personnes>
  ...
```

- Internet Explorer utilise type="text/xsl" pour les feuilles de styles XSLT



Modèles et règles de modèles

- **Contrôle de la sortie créée : ajout de règles de modèles à la feuille de style XSLT**
 - Chaque modèle est représenté par un élément **xsl:template**
 - chaque élément a un attribut **match**, dont la valeur est un motif XPath
- Exemple :

```
<xsl:template match="personne">Une personne</xsl:template>
```

A chaque élément personne, le processeur de feuille de style doit émettre le texte "*Une personne*"



Modèles et règles de modèles - exemple

8.1.xsl

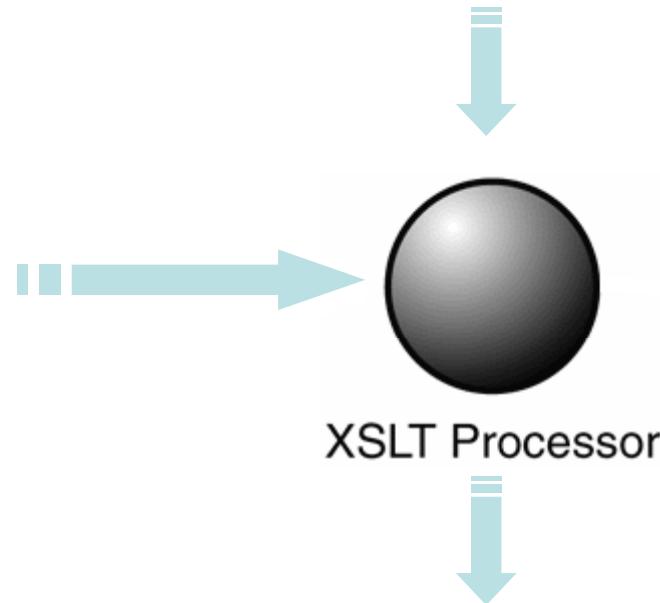
8-1.xml

```
<?xml version="1.0" ?>
<people>
  <person born="1912" died="1954">
    <name>
      <first_name>Alan</first_name>
      <last_name>Turing</last_name>
    </name>
    <profession>computer scientist</profession>
    <profession>mathematician</profession>
    <profession>cryptographer</profession>
  </person>

  <person born="1918" died="1988">
    <name>
      <first_name>Richard</first_name>
      <middle_initial>P</middle_initial>
      <last_name>Feynman</last_name>
    </name>
    <profession>physicist</profession>
    <hobby>Playing the bongoes</hobby>
  </person>
</people>
```

8-1.xsl

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="person">Une personne </xsl:template>
</xsl:stylesheet>
```



XSLT Processor

```
<?xml version="1.0" ?>Une personne Une personne
```



Valeur d'un élément : xsl:value-of

8-2.xsl

- Calcule la valeur textuelle d'un élément particulier en entrée, puis l'insère dans le résultat
 - Un attribut **select** contenant une **expression XPath** identifie l'élément dont la valeur est calculée
- Exemple :

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:template match="person">
        <p>
            <xsl:value-of select="name"/>
        </p>
    </xsl:template>
</xsl:stylesheet>
```



```
<?xml version="1.0"?><p> Alan Turing </p><p> Richard P Feynman </p>
```

Application de modèles : **xsl:apply-templates**

- Par défaut, un processeur XSLT lit un document séquentiellement de haut en bas, dans l'ordre d'apparition des éléments
 - Un modèle permet de changer l'ordre de parcours
 - L'élément **xsl:apply-templates**, permet le choix du traitement
 - Son attribut **select** contient une **expression XPath** spécifiant les nœuds à traiter lors de la transformation
- Exemple :
- On veut n'afficher que les noms des personnes, avec le nom de famille en premier, quelque soit l'ordre en entrée

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

    <xsl:template match="name">
        <xsl:value-of select="first_name"/>
        <xsl:value-of select="last_name"/>
    </xsl:template>

</xsl:stylesheet>
```

Application de modèles : xsl:apply-templates

- Feuille de style (à priori) :

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

    <xsl:template match="name">
        <xsl:value-of select="first_name"/>
        <xsl:value-of select="last_name"/>
    </xsl:template>

</xsl:stylesheet>
```

- Non suffisante, il faut indiquer, il faut aussi définir le modèle correspondant à l'élément racine, pour afficher les éléments souhaités et pas les autres
- Soit, définir :

```
<xsl:template match="person">
    <xsl:apply-templates select="name"/>
</xsl:template>
```

Application de modèles : xsl:apply-templates

8-3.xsl

- Feuille de style finale :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

    <xsl:template match="name">
        <xsl:value-of select="last_name"/>,
        <xsl:value-of select="first_name"/>
    </xsl:template>

    <xsl:template match="person">
        <xsl:apply-templates select="name"/>
    </xsl:template>

</xsl:stylesheet>
```

Affichage du nom et prénom

Application du template sur la balise person et sélection de la balise name contenue



```
<?xml version="1.0" ?>Turing,
:
AlanFeynman,
Richard
```



Génération HTML

- Veiller à un document bien formé :

8-4.xsl

8.1.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:template match="name">
    <xsl:value-of select="last_name"/>,
    <xsl:value-of select="first_name"/>
  </xsl:template>

  <xsl:template match="person">
<html>
  <head>
    <title>Scientifiques célèbres</title>
  </head>
  <body>
    <xsl:apply-templates select="name"/>
  </body>
</html>
</xsl:template>

</xsl:stylesheet>
```



```
<html>
<head>
<META http-equiv="Content-Type" content="text/html">
<title>Scientifiques célèbres</title></head>
<body>Turing,
      Alan</body>
</html>
<html>
<head>
<META http-equiv="Content-Type" content="text/html">
<title>Scientifiques célèbres</title></head>
<body>Feynman,
      Richard</body>
</html>
```

Le processeur génère autant de balise **<body>** que d'éléments personnes trouvés

Génération HTML

8-5.xsl

8.1.xml

- Générer un seul document bien formé :
 - Dans le cas d'application de modèles à tout type de sous éléments, plutôt qu'à un élément, on définit l'élément **racine** en **omettant l'attribut select**, pour éviter la répétition

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:template match="people">
    <html>
      <head>
        <title>Scientifiques célèbres</title>
      </head>
      <body>
        <xsl:apply-templates/>
      </body>
    </html>
  </xsl:template>

  <xsl:template match="name">
    <xsl:value-of select="last_name"/>,
    <xsl:value-of select="first_name"/>
  </xsl:template>

  <xsl:template match="person">
    <xsl:apply-templates select="name"/>,
  </xsl:template>

</xsl:stylesheet>
```

The diagram illustrates the transformation process. On the left, a red-bordered box contains the XSLT code. A teal arrow points from this box to a second box on the right, which contains the resulting HTML structure. The HTML output is a well-formed document with an `<html>` root element, followed by a `<head>` section containing a `<title>Scientifiques célèbres</title>` and a `<META http-equiv="Content-Type" content="text/html">`. The `<body>` section contains two `<xsl:value-of>` elements: one for the last name ('Turing') and one for the first name ('Alan').

```
<html>
  <head>
    <META http-equiv="Content-Type" content="text/html">
    <title>Scientifiques célèbres</title>
  </head>
  <body>
    Turing,
    Alan,
  </body>
</html>
```

Génération HTML - Exercice

A partir du fichier librairie.xml, afficher sous forme html, pour les livres,
les titres et auteurs

Librairie.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<librairie xsi:noNamespaceSchemaLocation="librairie.xsd"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

  <livre isbn="2212110472" categorie="XML">
    <titre>Services Web avec XML, SOAP, WSDL, UDDI, ebXML ...
    </titre>
    <auteur>Jean-Marie Chauvet</auteur>
    <éditeur>Eyrolles</éditeur>
  </livre>

  <livre isbn="2100065203" categorie="XML">
    <titre>XML Manuel de référence</titre>
    <auteur>R. Wyke</auteur>
    <auteur>S. Rehman</auteur>
    <auteur>B. Leupen</auteur>
    <éditeur>Microsoft Press</éditeur>
  </livre>

  <livre isbn="1928994474" categorie="XML">
    <titre>XML.NET Developer's Guide</titre>
    <auteur>Collectif</auteur>
    <éditeur>Syngress</éditeur>
  </livre>

</librairie>
```

Génération HTML - Exercice

A partir du fichier librairie.xml, afficher sous forme html, pour les livres, les titres et auteurs

```
<?xml version="1.0" encoding="UTF-8"?>
<librairie xsi:noNamespaceSchemaLocation="librairie.xsd"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

    <livre isbn="2212110472" categorie="XML">
        <titre>
            Services Web avec XML, SOAP, WSDL, UDDI, ebXML ...
        </titre>
        <auteur>Jean-Marie Chauvet</auteur>
        <editeur>Eyrolles</editeur>
    </livre>

    <livre isbn="2100065203" categorie="XML">
        <titre>XML Manuel de référence</titre>
        <auteur>R. Wyke</auteur>
        <auteur>S. Rehman</auteur>
        <auteur>B. Leupen</auteur>
        <editeur>Microsoft Press</editeur>
    </livre>

    <livre isbn="1928994474" categorie="XML">
        <titre>XML.NET Developer's Guide</titre>
        <auteur>Collectif</auteur>
        <editeur>Syngress</editeur>
    </livre>

</librairie>
```

Librairie.xml

Librairie_O1.xsl

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

    <xsl:template match="librairie">
        <html>
            <head>
                <title>Les livres de la librairie</title>
            </head>
            <body>
                <xsl:apply-templates />
            </body>
        </html>
    </xsl:template>

    <xsl:template match="livre">
        <ul>
            <li><xsl:value-of select="titre"/></li>
            <li><xsl:value-of select="auteur"/></li>
            <li><xsl:value-of select="editeur"/></li>
        </ul>
    </xsl:template>
</xsl:stylesheet>
```



XPath



Langage XPath

- **Problème principal** : Définir à quel endroit du document se trouve la données à extraire
 - **XPath** : Référencer des ensembles d'éléments, d'attributs, de texte ... figurant dans un document XML
 - Langage d'expressions
 - Intervient dans d'autres langages :
 - XPointer
 - XQuery
 - Utilise un modèle de représentation arborescente d'un document XML



XPath – Présentation

- XPath est un langage d'interrogation des documents XML.
- Il permet de sélectionner certaines parties d'un document XML :
 - des sous-arbres,
 - des noeuds,
 - des attributs...
- Il permet de naviguer dans l'arbre du document :
 - à partir d'un noeud origine,
 - vers un ou plusieurs noeuds destinations,
 - en sélectionnant des chemins dans l'arbre.



XPath – Syntaxe générale

- L'endroit d'où l'on part se nomme le **noeud courant**.
 - En 1[°] lecture on peut imaginer qu'il s'agit de la racine du document, mais n'importe quel noeud peut jouer ce rôle.
- Ensuite, il y a trois autres types d'éléments :
 - **un axe** :
 - la direction dans laquelle on se dirige à partir du noeud courant (vers le père, vers les fils, vers les frères de gauche...) ;
 - **un filtre** :
 - le type de noeuds qui nous intéresse dans l'axe choisi (des noeuds quelconques, des éléments quelconques ou un élément précis, des commentaires...) ;
 - **un prédicat** (optionnel) :
 - des conditions supplémentaires pour sélectionner des noeuds parmi ceux retenus par le filtre dans l'axe.



XPath – Syntaxe générale

- A eux trois, ils constituent **une étape**.
- Le principe de la syntaxe XPath est semblable à celui de l'adressage du système de fichiers
 - c'est l'enchaînement de plusieurs étapes, séparées par des "/".
- Exemple :
 - [/]etape1/etape2/.../etapeX
- Si le chemin commence par un "/", il s'agit d'un chemin absolu,
 - C'est-à-dire prenant son origine à la racine du document et non pas au noeud courant.
- Il est possible de faire une disjonction de requêtes XPath avec le signe "|" :
 - On obtient alors l'union des deux ensembles de noeuds correspondants.
- Chaque étape renvoie un ensemble de noeuds et pour chacun d'entre eux, on applique les étapes suivantes.



XPath – Syntaxe générale

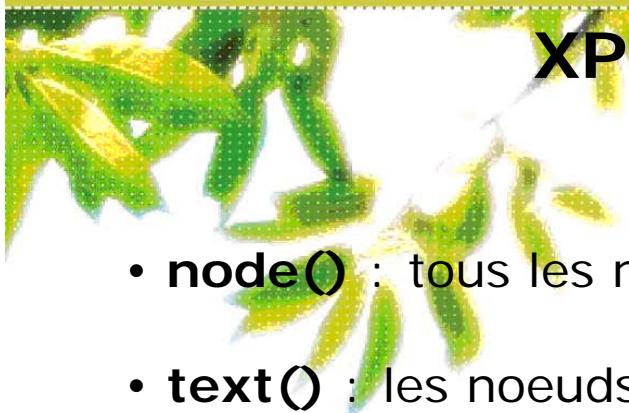
- Une étape a la forme :
`axe::filtre[predicat1][predicat2]...`
- L'**axe** définit le **sens du parcours** des noeuds ;
- Le **filtre** indique le **type des noeuds qui seront retenus** dans l'évaluation de l'expression
- Le (ou les) **prédictat(s)** exprime(nt) des **propriétés** que doivent



XPath – Syntaxe générale – Les axes

- 8 axes :

- **self** : le noeud courant lui-même ;
- **child** : les enfants du noeud courant ;
- **descendant, descendant-or-self** : tous les descendants du noeud courant ;
- **parent** : le père du noeud courant ;
- **ancestor, ancestor-or-self** : les ancêtres du nœud courant ;
- **attribute** : les attributs du noeud courant ;
- **preceding, following** : les noeuds, précédents ou suivants du noeud courant, dans l'ordre de lecture du document ;
- **preceding-sibling, following-sibling** : les frères, précédent ou suivant, le noeud courant ;



XPath – Syntaxe générale – Les filtres

- **node()** : tous les noeuds ;
- **text()** : les noeuds textuels ;
- ***** : tous les éléments ;
- **nom** : les éléments portant ce nom ;
- **comment()** : les noeuds commentaires ;
- **processing-instruction('cible')** : les nœuds instructions, seulement les instructions cibles si cet argument est fourni.



XPath – Syntaxe générale – Les prédictats

- Ils prennent la forme de tests que les noeuds sélectionnés devront vérifier. Ces tests peuvent impliquer des fonctions ou de nouveaux chemins XPath.

```
child::tata[position()=2]  
child::tata[@ref='id125']
```

- Une expression entre crochets permet de spécifier plus précisément un élément.

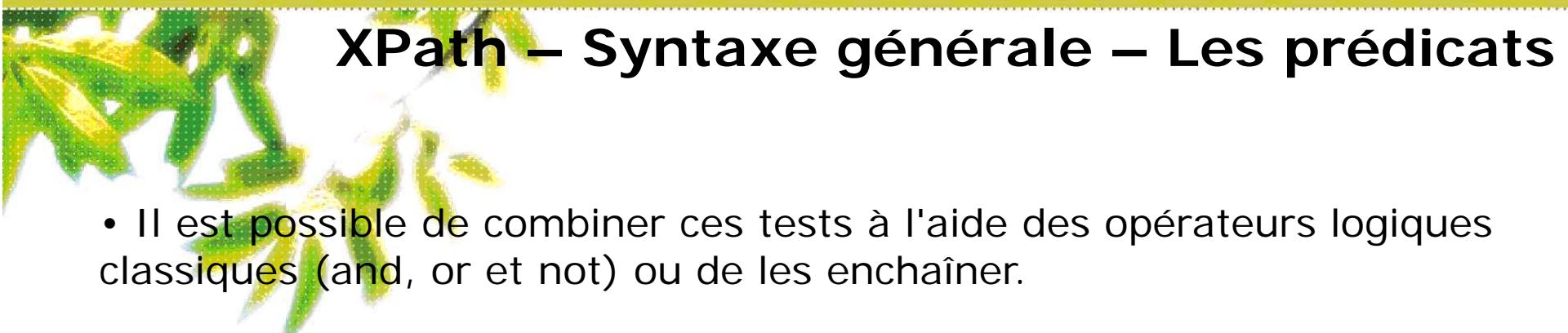
```
/bibliographie/livre[1]
```

- sélectionne le premier **élément livre** de **bibliographie**.

- Un nombre entre crochets donne la position d'un élément dans le jeu sélectionné.

```
/bibliographie/livre[last()]
```

- sélectionne le dernier élément livre de bibliographie.



XPath – Syntaxe générale – Les prédictats

- Il est possible de combiner ces tests à l'aide des opérateurs logiques classiques (and, or et not) ou de les enchaîner.

```
child::tata[@ref='id125' or @ref='id47']
```

```
child::tata[contains(text(),'coucou') and position()=2]
```

```
child::tata[contains(text(),'coucou')][position()=2]
```

- Les deux dernières requêtes ne sont pas équivalentes :
 - la première renvoie le deuxième fils tata si celui-ci contient le texte coucou ;
 - la seconde sélectionne tous les fils tata qui contiennent le texte coucou et parmi ceux-ci renvoie le deuxième.



XPath – Syntaxe générale – Les fonctions

- Positions relatives et information locale :
 - **position()** : position dans le contexte ;
 - **name()** : retourne le nom de l'élément ;
 - **count()** : compte le nombre d'éléments sélectionnés ;
 - **last()** : indicateur de dernière position.
- Fonctions booléennes : **and, or, not** ;
- Opérateurs : **mod, >, <=,...**
- Fonctions de chaîne : **contains, substringbefore, stringlength,...**
- Fonctions d'environnement : **normalize-space**.
 - Elle supprime les espaces de début et de fin et remplace les séquences d'espaces blancs par un seul espace.



XPath – Etapes de localisation

personnes.xml

personnes_01.xsl

- Sélection d'attributs : @nom_de_l_attribut :

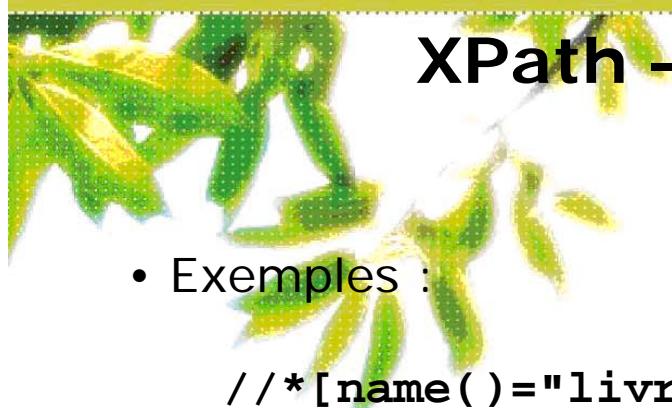
```
<?xml version="1.0" encoding="ISO-8859-1"?>
  <xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

    <xsl:template match="/">
      <html>
        <xsl:apply-templates select="personnes"/>
      </html>
    </xsl:template>

    <xsl:template match="personnes">
      <table>
        <xsl:apply-templates select="personne"/>
      </table>
    </xsl:template>

    <xsl:template match="personne">
      <tr>
        <td><xsl:value-of select="nom"/></td>
        <td><xsl:value-of select="@naissance"/></td>
        <td><xsl:value-of select="@mort"/></td>
      </tr>
    </xsl:template>

  </xsl:stylesheet>
```



XPath – Syntaxe générale – Les fonctions

- Exemples :

```
//*[name()="livre"]
```

- éléments livre

```
//*[count(livre)="2"]
```

- éléments ayant 2 enfants livre

```
//*[count(*)="2"]
```

- éléments ayant deux enfants

```
//*[stringlength(name())="3"]
```

- retourne tous les éléments dont le nom a trois caractères.



XPath – Syntaxe abrégée

- Il existe une notation abrégée permettant de naviguer dans l'arborescence d'un document :
 - **element** : sélectionne tous les éléments élément fils du noeud courant (`child ::element`).
 - ***** : sélectionne tous les éléments fils du noeud courant.
 - **/** : représente l'élément racine.
 - **//** : représente n'importe quel descendant de l'élément racine, donc tous les éléments (`descendant-or-self ::node()`).
 - **.** : représente l'élément courant (`self ::node()`).
 - **..** : permet de remonter d'un niveau dans l'arborescence du document par rapport à l'élément courant (`parent ::node()`).



XPath –Syntaxe abrégée

- **/element** : sélectionne tous les éléments element sous l'élément racine () .
- **./element** : sélectionne tous les éléments element sous l'élément courant (following ::element).
- **../element** : sélectionne tous les éléments element sous l'élément parent du noeud courant (preceding ::element).
- **//element** : sélectionne tous les éléments element descendants du noeud courant à quelque niveau de profondeur que ce soit.
- **@attribut** : sélectionne tous les attributs attribut du noeud courant (attribute ::attribut).
- | : correspond à un OU (OR).



XPath –Syntaxe abrégée - Exemples

- Exemples d'expressions équivalentes :

`parent::personnel`

`../personnel`

`//toto`

`/descendant-or-self::node() /child::toto`

`toto[2]`

`child::toto[position() = 2]`

XPath –Syntaxe abrégée - Exemple

```
<bibliothèque>
  <livre année="2007">
    <titre>XPath par la pratique</titre>
    <auteur>Thomas Frenz</auteur>
    <éditeur>
      <nom>Paris Editions</nom>
      <ville>Paris</ville>
    </éditeur>
    <prix>15 euros</prix>
  </livre>
</bibliothèque>
```

//éditeur/parent ::* renvoie le parent du nœud éditeur, c'est à dire livre.

/livre/éditeur/descendant ::* renvoie les descendants de éditeur, c'est à dire nom et ville.

//éditeur/following-sibling ::* renvoie le noeud prix.

//éditeur/preceding-sibling ::* renvoie les nœuds auteur, titre.

//auteur/following ::* renvoie les noeuds éditeur, nom, ville, prix.

//auteur/preceding ::* renvoie le noeud titre.



XPath- Modèles implicites

- Il existe 7 sortes de nœuds dans un document XML :
 1. **root** : Le nœud **racine de l'arbre XML**
 2. **element** : Le type d'un nœud **élément XML**
 3. **attribute** : Le type d'un nœud **attribut d'un élément XML**
 4. **text** : Le type d'un nœud faisant partie d'un élément
 5. **comment** : Les nœuds de **commentaires**
 6. **processing-instruction** : Les nœuds **d'instructions de traitement**
 7. **namespace** : Les **espaces de noms**
- Pour chacun, XSLT fournit une règle prédéfinie par défaut
 - Si l'utilisateur n'a pas défini d'instruction spécifique
 - Ces règles modèles déterminent quels nœuds seront activés et quand



XPath – for each

catalog_01.xsl

- L'élément XSL <xsl:for-each> peut être utilisée pour sélectionner chaque élément XML d'un ensemble de noeuds spécifiés :
- Exemple :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:template match="/">
    <html>
      <body>
        <h2>My CD Collection</h2>
        <table border="1">
          <tr bgcolor="#9acd32">
            <th>Title</th>
            <th>Artist</th>
          </tr>
          <xsl:for-each select="catalog/cd">
            <tr>
              <td><xsl:value-of select="title"/></td>
              <td><xsl:value-of select="artist"/></td>
            </tr>
          </xsl:for-each>
        </table>
      </body>
    </html>
  </xsl:template>

</xsl:stylesheet>
```

XPath – for each - Exemple

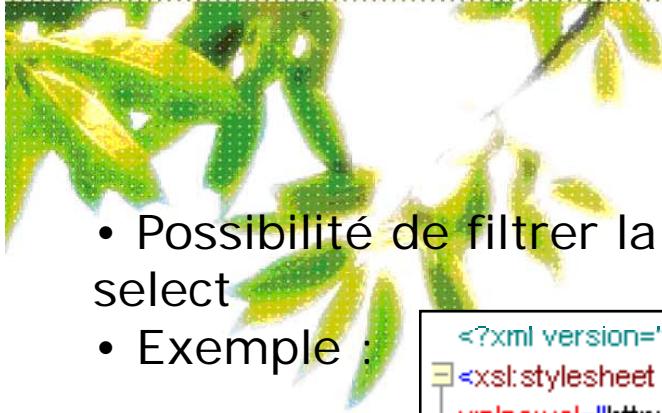
```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:template match="/">
    <html>
      <body>
        <h2>Les personnes</h2>
          <!-- On se positionne sur l'element nom -->
          <xsl:for-each select="personnes/personne/nom">
            <ul>
              <xsl:value-of select="nom_famille" />,
              <xsl:value-of select="prénom" /> :
              <!-- On remonte d'un niveau pour se positionner sur les professions -->
              <xsl:for-each select=".//profession">
                <li><xsl:value-of select="text()" /></li>
              </xsl:for-each>
            </ul>
          </xsl:for-each>
        </body>
      </html>
    </xsl:template>

  </xsl:stylesheet>
```

personnes.xml

personnes_03.xsl



XPath – for each - Filtrage

catalog_02.xsl

- Possibilité de filtrer la sortie en ajoutant un critère pour l'attribut select
- Exemple :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:template match="/">
    <html>
      <body>
        <h2>My CD Collection</h2>
        <table border="1">
          <tr bgcolor="#9acd32">
            <th>Title</th>
            <th>Artist</th>
          </tr>
          <xsl:for-each select="catalog/cd[artist='Bob Dylan']">
            <tr>
              <td><xsl:value-of select="title"/></td>
              <td><xsl:value-of select="artist"/></td>
            </tr>
          </xsl:for-each>
        </table>
      </body>
    </html>
  </xsl:template>

</xsl:stylesheet>
```



XPath – sort

catalog_03.xsl

- Pour trier le résultat, il suffit d'ajouter un élément `<xsl:sort>` dans l'élément `<xsl:for-each>`
- Exemple :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

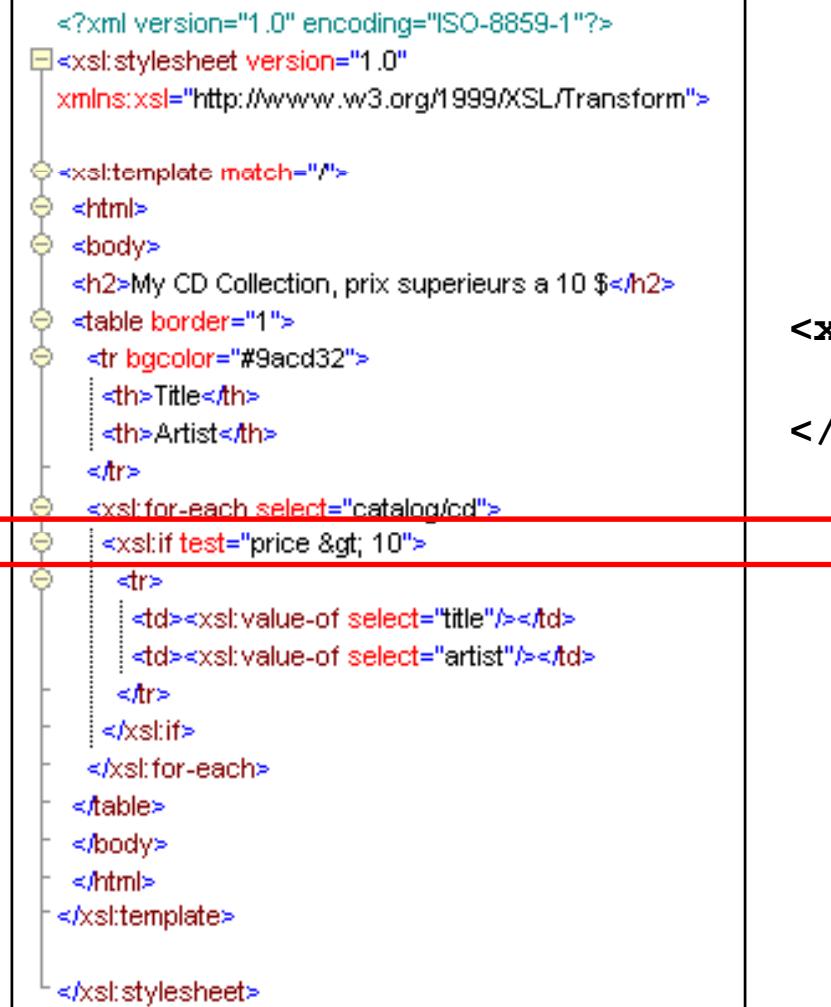
  <xsl:template match="/">
    <html>
      <body>
        <h2>My CD Collection</h2>
        <table border="1">
          <tr bgcolor="#9acd32">
            <th>Title</th>
            <th>Artist</th>
          </tr>
          <xsl:for-each select="catalog/cd">
            <xsl:sort select="artist"/>
            <tr>
              <td><xsl:value-of select="title"/></td>
              <td><xsl:value-of select="artist"/></td>
            </tr>
          </xsl:for-each>
        </table>
      </body>
    </html>
  </xsl:template>

</xsl:stylesheet>
```

XPath – if

catalog_04.xsl

- Pour intégrer un test conditionnel, intégrer l'élément <xsl:if>
- Exemple :



```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:template match="/">
    <html>
      <body>
        <h2>My CD Collection, prix superieurs a 10 $</h2>
        <table border="1">
          <tr bgcolor="#9acd32">
            <th>Title</th>
            <th>Artist</th>
          </tr>
          <xsl:for-each select="catalog/cd">
            <xsl:if test="price > 10">
              <tr>
                <td><xsl:value-of select="title"/></td>
                <td><xsl:value-of select="artist"/></td>
              </tr>
            </xsl:if>
          </xsl:for-each>
        </table>
      </body>
    </html>
  </xsl:template>

</xsl:stylesheet>
```

```
<xsl:if test="expression">
  ...Si l'expression est vraie...
</xsl:if>
```

XPath – choose

catalog_05.xsl

- Pour intégrer un test conditionnel multiple, intégrer l'élément `<xsl:choose>`

- **Exemple :**

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
  <html>
    <body>
      <h2>My CD Collection</h2>
      <table border="1">
        <tr bgcolor="#9acd32">
          <th>Title</th>
          <th>Artist</th>
        </tr>
        <xsl:for-each select="catalog/cd">
          <tr>
            <td><xsl:value-of select="title"/></td>
            <xsl:choose>
              <xsl:when test="price > 10">
                <td bgcolor="#ff0000">
                  <xsl:value-of select="artist"/>
                </td>
              </xsl:when>
              <xsl:otherwise>
                <td><xsl:value-of select="artist"/></td>
              </xsl:otherwise>
            </xsl:choose>
          </tr>
        </xsl:for-each>
      </table> </body> </html>
    </xsl:template>
</xsl:stylesheet>
```

```
<xsl:choose>
  <xsl:when test="expression">
    ... Si expression vraie ...
  </xsl:when>
  <xsl:otherwise>
    ... Sinon ....
  </xsl:otherwise>
</xsl:choose>
```



XPath – Exercice

lesGaulois.xml

lesGaulois_01.xsl

- A partir du fichier villageois, afficher les informations concernant Astérix, les images sont dans le répertoire imagesGaulois

Les Gaulois



Nom : Asterix
ADRESSE : cote carrières
SPECIALITE : Guerrier

XPath – Exercice

lesGaulois.xml

lesGaulois_01.xsl

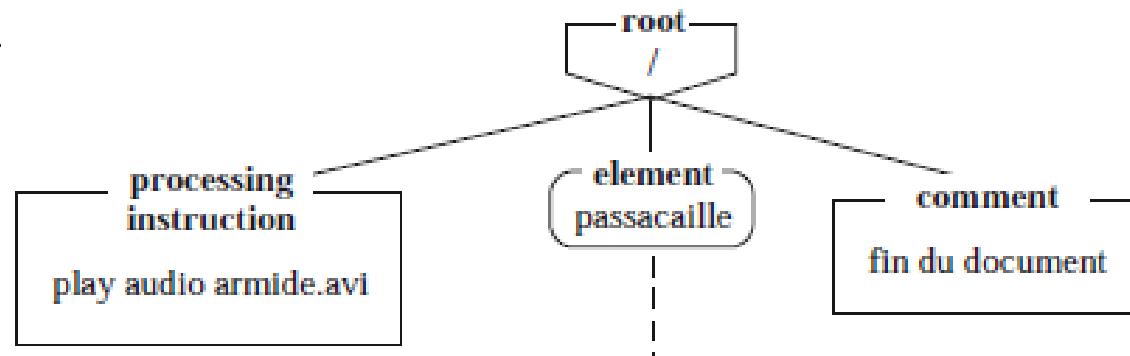
```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <html>
      <head>
        <title>Fiche Villageois</title>
      </head>
      <body>
        <h2>Les Gaulois</h2>
        <xsl:for-each select="lesgaulois/villageois">
          <xsl:if test="NOM='Asterix'">
            <div style="float:left">
              
            </div>
            <div style="float:left;font-family:verdana;font-weigth:bold;font-size:12px;margin-left:10px;">
              Nom :
            </div>
            <xsl:value-of select="NOM" />
            <br />
            <div style="float:left;font-family:verdana;font-weigth:bold;font-size:12px;margin-left:10px;">
              ADRESSE :
            </div>
            <xsl:value-of select="ADRESSE" />
            <br />
            <div style="float:left;font-family:verdana;font-weigth:bold;font-size:12px;margin-left:10px;">
              SPECIALITE :
            </div>
            <xsl:value-of select="NOM_SPECIALITE" />
          </xsl:if>
        </xsl:for-each>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

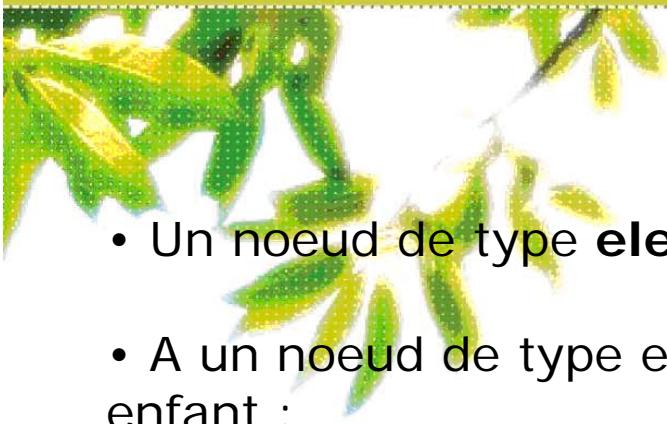
XPath

Nœud de type root

- Un seul noeud peut être de type **root**.
- A la racine sont attachés, dans un lien parent-enfant :
 - L'élément racine du document XML proprement dit (après le prologue),
 - Les instructions de traitement
 - Les commentaires qui interviennent dans le prologue et après la fin de l'élément racine du document XML
- Exemple

```
<?xml version='1.0' encoding='ISO-8859-1' standalone='no' ?>
<!DOCTYPE passacaille SYSTEM "Danse.dtd" >
<?play audio armide.avi?>
<passacaille>
  ...
</passacaille>
<!-- fin du document -->
```



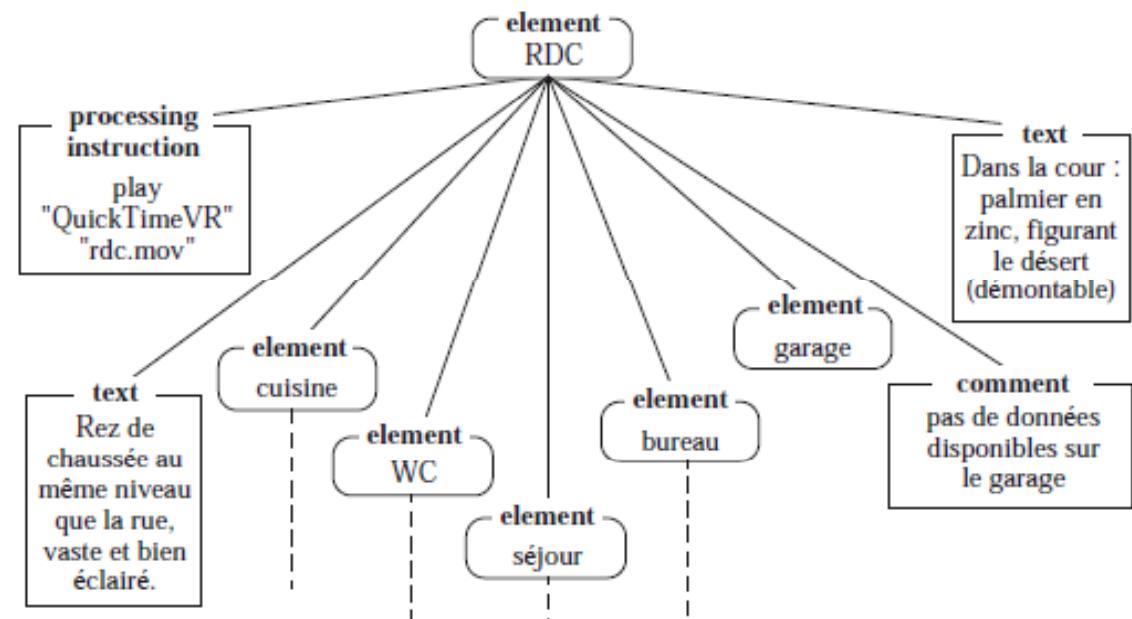


Nœud de type element

- Un noeud de type **element** pour **chaque balise** du document
- A un noeud de type element sont attachés, dans un lien parent-enfant :
 - Les noeuds de type element, enfants directs de l'élément considéré
 - Les noeuds de type :
 - processing instruction,
 - comment,
 - et text
 - qui font partie du contenu de l'élément considéré

Nœud de type element - Exemple

```
<RDC>
  <?play "QuicktimeVR" "rdc.mov" ?>
  Rez de chaussée au même niveau que la rue, vaste et bien éclairé.
  <cuisine>Evier inox. Mobilier encastré.</cuisine>
  <WC>Lavabo. Cumulus 200L.</WC>
  <séjour>
    Cheminée en pierre. Poutres au plafond. Carrelage terre cuite.
    Grande baie vitrée.
  </séjour>
  <bureau>Bibliothèque encastrée.</bureau>
  <garage/>
  <!-- pas de données disponibles sur le garage -->
  Dans la cour : palmier en zinc, figurant le désert(démontable).
</RDC>
```



Nœud de type element - Exemple

- Valeur textuelle d'un noeud de type element : concaténation des valeurs textuelles de tous ses descendants de type text (pas uniquement les enfants directs, mais seulement les noeuds text) pris dans l'ordre de lecture du document.
- Exemple : Valeur textuelle de <RDC>

```
<RDC>
  <?play "QuicktimeVR" "rdc.mov" ?>
  Rez de chaussée au même niveau que la rue, vaste et bien
  éclairé.
  <cuisine>Evier inox. Mobilier encastré.</cuisine>
  <WC>Lavabo. Cumulus 200L.</WC>
  <séjour>
    Cheminée en pierre. Poutres au plafond. Carrelage terre
    cuite. Grande baie vitrée.
  </séjour>
  <bureau>Bibliothèque encastrée.</bureau>
  <garage/>
  <!-- pas de données disponibles sur le garage
  Dans la cour : palmier en zinc, figurant le
  désert(démontable).
</RDC>
```

Rez de chaussée au même niveau que la rue,
vaste et bien éclairé.

Evier inox. Mobilier encastré.

Lavabo. Cumulus 200L.

Cheminée en pierre. Poutres au plafond.
Carrelage terre cuite. Grande baie vitrée.

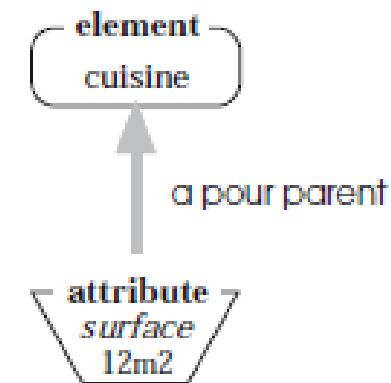
Bibliothèque encastrée.

Dans la cour : palmier en zinc, figurant
le désert (démontable).

Nœud de type attribute

- Les noeuds de type attribute sont attachés au noeud élément considéré par un lien spécial : **L'élément est parent des noeuds attributs**
- **Mais l'ensemble des enfants d'un élément ne contient pas les attributs**
- Exemple :

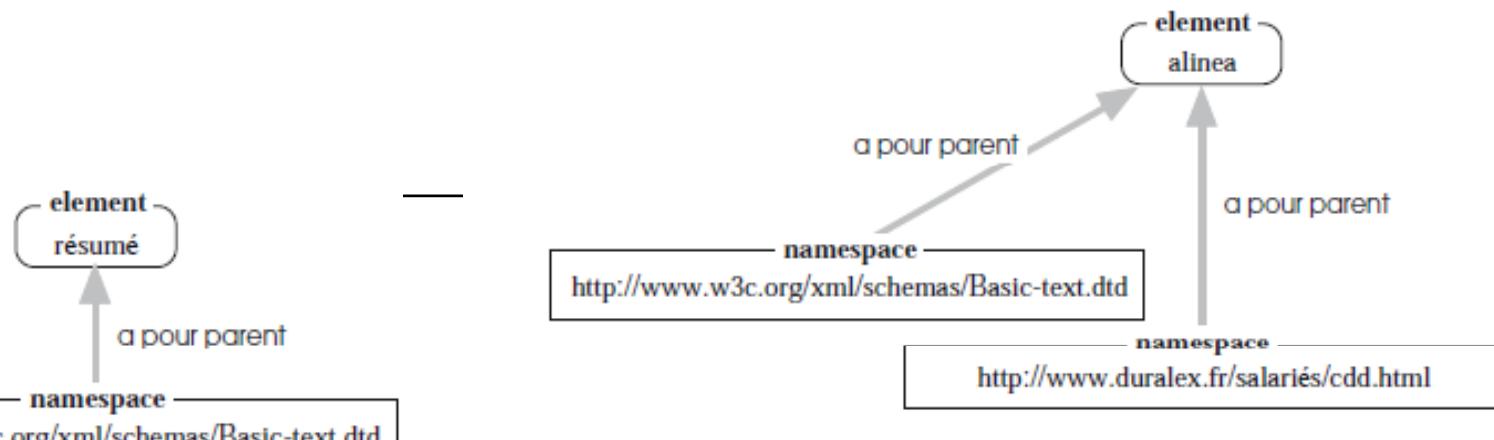
```
<cuisine surface='12m2'>  
  ...  
</cuisine>
```



Nœud de type namespace

- Un noeud de type namespace est créé, pour chaque élément, et pour chaque domaine nominal visible.
- Comme pour un noeud de type attribute, l'élément est le parent du noeud de type namespace
- Mais pourtant ne le possède pas en tant qu'enfant
- Exemple :

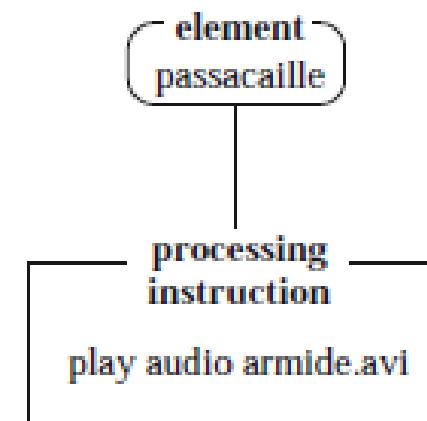
```
<résumé xmlns="http://www.w3c.org/xml/schemas/Basic-text.dtd" >
  ...
  <jur:alinea numero="12.2.3.5" xmlns:jur="http://www.duralex.fr/salariés/cdd.html">
    <texte>
      <alinea>
        ...
        <alinea>
          </texte>
        </jur:alinea>
        ...
    </résumé>
```



Nœud de type processing instruction

- Un noeud de type processing-instruction est créé pour chaque instruction de traitement,
 - sauf si elle intervient à l'intérieur de la partie DTD du document.
 - Un noeud de type processing-instruction n'a pas d'enfant
- Exemple

```
<passacaille>
  <?play audio armide.avi?>
  ...
</passacaille>
```

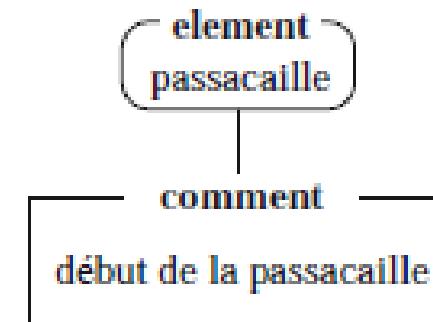




Nœud de type comment

- Un noeud de type comment est créé pour chaque commentaire,
 - sauf s'il intervient à l'intérieur de la partie DTD du document.
 - Un noeud de type comment n'a pas d'enfant
- Exemple :

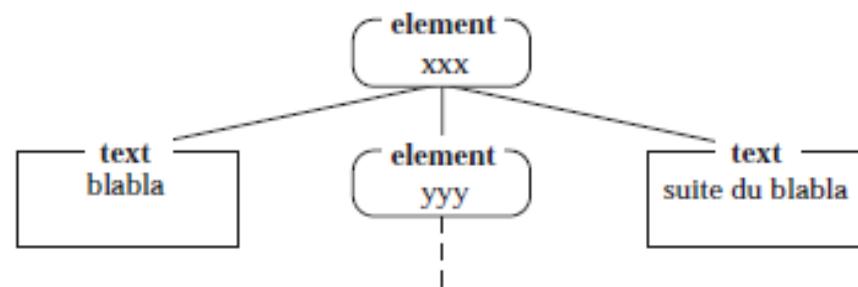
```
<passacaille>
  <!-- début de la passacaille -->
  ...
</passacaille>
```



Nœud de type text

- Chaque noeud de type element peut avoir des noeuds enfants de type text.
- Un noeud de type text n'a pas d'enfant
- Exemple :

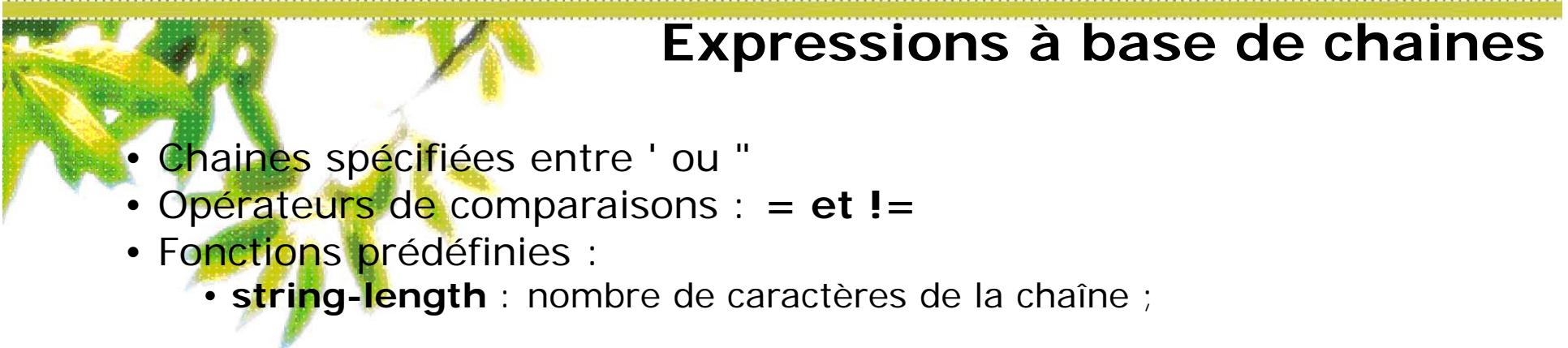
```
<xxx>
  blabla
  <yyy> ... </yyy>
  suite du blabla
</xxx>
```





Expressions numériques

- Manipulent des nombres (objet de type number)
- Opérateurs de comparaisons : = <= >= !=
- Opérateurs : + - * **div mod**
- 3 fonctions mathématiques :
 - **floor** (plus grand entier inférieur),
 - **ceiling** (plus petit entier supérieur)
 - **round** (plus proche entier)
- Exemples :
 - $\$b * 100 + \$d - 4800 + \text{floor}(\$m \text{ div } 10)$
 - $\$J + 31741 - (\$J \text{ mod } 7) \text{ mod } 146097 \text{ mod } 36524 \text{ mod } 1461$
 - $((\$d4 - \$L) \text{ mod } 365) + \$L$



Expressions à base de chaînes

- Chaînes spécifiées entre ' ou "
- Opérateurs de comparaisons : = et !=
- Fonctions prédéfinies :
 - **string-length** : nombre de caractères de la chaîne ;
 - **concat** : concaténer deux chaînes en une seule
 - **normalize-space** : supprime tous les espaces blancs en tête, et en fin, remplace toute autre séquence interne d'espaces blancs par un seul
 - **translate** : remplace certains caractères par d'autres
 - **substring** : extrait une sous-chaîne
 - **contains** : teste la présence d'une sous-chaîne dans une chaîne
 - **starts-with** : teste la présence d'une sous-chaîne au début de chaîne
 - **substring-after** : recherche une sous-chaîne et extrait la sous-chaîne située après
 - **substring-before** : recherche une sous-chaîne et extrait la sous-chaîne située avant



Chemin de localisation

- **Location path :**
- Suite d'étapes de localisation (location step) séparées par "/"
- Le "://" initial indique un chemin absolu, s'il n'existe pas => chemin relatif
- Le noeud de contexte est indispensable pour l'évaluation d'un chemin de localisation
- Exemples :

```
<AAA>
  <BBB/>
  <CCC/>
  <BBB/>
  <BBB/>
  <DDD>
    <BBB/>
  </DDD>
  <CCC/>
</AAA>
```

/AAA : Sélectionne l'élément racine AAA

/AAA/CCC : Sélectionne tous les enfants CCC de AAA

/AAA/DDD/BBB : Sélectionne tous les enfants BBB de DDD, enfant de AAA



Chemin de localisation - *

8.10.xml – 8.10.xsl

```
<AAA>
  <XXX>
    <DDD>
      <BBB/>
      <BBB/>
      <EEE/>
      <FFF/>
    </DDD>
  </XXX>
<CCC>
  <DDD>
    <BBB/>
    <BBB/>
    <EEE/>
    <FFF/>
  </DDD>
</CCC>
<CCC>
  <BBB>
    <BBB>
      <BBB/>
    </BBB>
  </BBB>
</CCC>
</AAA>
```

/AAA/CCC/DDD/* : Sélectionne tous les éléments inclus dans les éléments /AAA/CCC/DDD

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version = '1.0'
  xmlns:xsl='http://www.w3.org/1999/XSL/Transform'>

  <xsl:template match="AAA">
    <html>
      <head>
        <title>XSL</title>
      </head>
      <body>
        <xsl:apply-templates />
      </body>
    </html>
  </xsl:template>

  <xsl:template match=" /AAA/CCC/DDD/* ">
    <div style="color:red">
      <xsl:value-of select="name()" />
    </div>
  </xsl:template>

</xsl:stylesheet>
```

Chemin de localisation - * - Exercice

librairie.xml – librairie.xsl

- A partir du fichier librairie.xml, afficher sous forme html les noms des balises et les textes des balises contenus dans la balise livre

```
<?xml version="1.0" encoding="UTF-8"?>
<librairie xsi:noNamespaceSchemaLocation="librairie.xsd"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

  <livre isbn="2212110472" categorie="XML">
    <titre>
      Services Web avec XML, SOAP, WSDL, UDDI, ebXML ...
    </titre>
    <auteur>Jean-Marie Chauvet</auteur>
    <éditeur>Eyrolles</éditeur>
  </livre>

  <livre isbn="2100065203" categorie="XML">
    <titre>XML Manuel de référence</titre>
    <auteur>R. Wyke</auteur>
    <auteur>S. Rehman</auteur>
    <auteur>B. Leupen</auteur>
    <éditeur>Microsoft Press</éditeur>
  </livre>

  <livre isbn="1928994474" categorie="XML">
    <titre>XML.NET Developer's Guide</titre>
    <auteur>Collectif</auteur>
    <éditeur>Syngress</éditeur>
  </livre>

</librairie>
```

Chemin de localisation - * - Exercice

librairie.xml – librairie.xsl

- A partir du fichier librairie.xml, afficher sous forme html les noms des balises et les textes des balises contenus dans la balise livre

```
<?xml version="1.0" encoding="UTF-8"?>
<librairie xsi:noNamespaceSchemaLocation="librairie.xsd"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

  <livre isbn="2212110472" categorie="XML">
    <titre>
      Services Web avec XML, SOAP, WSDL, UDDI, ebXML ...
    </titre>
    <auteur>Jean-Marie Chauvet</auteur>
    <éditeur>Eyrolles</éditeur>
  </livre>

  <livre isbn="2100065203" categorie="XML">
    <titre>XML Manuel de référence</titre>
    <auteur>R. Wyke</auteur>
    <auteur>S. Rehman</auteur>
    <auteur>B. Leupen</auteur>
    <éditeur>Microsoft Press</éditeur>
  </livre>

  <livre isbn="1928994474" categorie="XML">
    <titre>XML.NET Developer's Guide</titre>
    <auteur>Collectif</auteur>
    <éditeur>Syngress</éditeur>
  </livre>

</librairie>
```

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version = '1.0'
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:template match="librairie">
    <html>
      <head>
        <title>Librairie</title>
      </head>
      <body>
        <ul>LesLivres
          <xsl:apply-templates />
        </ul>
      </body>
    </html>
  </xsl:template>

  <xsl:template match=" librairie/livre/*">
    <li>
      <span style="font-weight:bold;color:blue;">
        <xsl:value-of select="name()"/>
      </span>
      <xsl:value-of select="text()"/>
    </li>
  </xsl:template>

</xsl:stylesheet>
```



Chemin de localisation - *

8.11.xml – 8.11.xsl

```
<AAA>
  <XXX>
    <DDD>
      <BBB/>
      <BBB/>
      <EEE/>
      <FFF/>
    </DDD>
  </XXX>
  <CCC>
    <DDD>
      <BBB/>
      <BBB/>
      <EEE/>
      <FFF/>
    </DDD>
  </CCC>
  <CCC>
    <BBB>
      <BBB>
        <BBB/>
        <BBB>
      </BBB>
    </CCC>
  </AAA>
```

`/*/*/*/BBB` : Sélectionne tous les éléments BBB qui ont trois ancêtres

```
<xsl:template match="/*/*/*/BBB">
  <div style="color:red">
    | <xsl:value-of select="name()"/>
  </div>
</xsl:template>
```



Chemin de localisation - crochets

- Une expression entre crochets peut spécifier plus précisément un élément.
- Un nombre entre crochets donne la position d'un élément dans le jeu sélectionné.
- La fonction last sélectionne le dernier élément du jeu

```
<AAA>
  <BBB/>
  <BBB/>
  <BBB/>
  <BBB/>
</AAA>
```

/AAA/BBB[1] : Sélectionne le premier élément BBB, fils de l'élément racine AAA

```
<xsl:template match="/AAA/BBB[1]">
  <div style="color:red">
    | <xsl:value-of select="name()"/>
  </div>
</xsl:template>
```

```
<AAA>
  <BBB/>
  <BBB/>
  <BBB/>
  <BBB/>
</AAA>
```

/AAA/BBB[last()] : Sélectionne le dernier élément BBB, fils de l'élément racine AAA

```
<xsl:template match="/AAA/BBB[last()]">
  <div style="color:red">
    | <xsl:value-of select="name()"/>
  </div>
</xsl:template>
```



Chemin de localisation - attributs

- Les attributs sont spécifiés par le préfixe @

```
<AAA>
  <BBB id = "b1"/>
  <BBB id = "b2"/>
  <BBB name = "bbb"/>
  <BBB/>
</AAA>
```

//BBB[@id] : Sélectionne tous BBB qui ont un attribut id

```
<AAA>
  <BBB id = "b1"/>
  <BBB id = "b2"/>
  <BBB name = "bbb"/>
  <BBB/>
</AAA>
```

//BBB[@name] : Sélectionne tous BBB qui ont un attribut name

Chemin de localisation - attributs

```
<AAA>
  <BBB id = "b1"/>
  <BBB id = "b2"/>
  <BBB name = "bbb"/>
  <BBB/>
</AAA>
```

//BBB[@*]

Sélectionne tous BBB qui ont un attribut

```
<AAA>
  <BBB id = "b1"/>
  <BBB id = "b2"/>
  <BBB name = "bbb"/>
  <BBB/>
</AAA>
```

//BBB[not(@*)]

Sélectionne tous BBB qui n'ont pas d'attribut

```
<AAA>
  <BBB id = "b1"/>
  <BBB name = "bbb "/>
  <BBB name = "bbb"/>
</AAA>
```

//BBB[@id='b1']

Sélectionne tous les éléments BBB ayant un attribut id dont la valeur est b1

```
<AAA>
  <BBB id = "b1"/>
  <BBB name = "bbb "/>
  <BBB name = "bbb"/>
</AAA>
```

//BBB[@name='bbb']

Sélectionne tous les éléments BBB ayant un attribut name dont la valeur est bbb

```
<AAA>
  <BBB id = "b1"/>
  <BBB name = "bbb "/>
  <BBB name = "bbb"/>
</AAA>
```

//BBB[normalize-space(@name)='bbb']

Sélectionne tous les éléments BBB ayant un attribut name dont la valeur est bbb.

Les espaces de début et de fin sont supprimés avant la comparaison



Chemin de localisation – count()

- La fonction **count()** compte le nombre d'éléments sélectionnés.

```
<AAA>
  <CCC>
    <BBB/>
    <BBB/>
    <BBB/>
  </CCC>
  <DDD>
    <BBB/>
    <BBB/>
  </DDD>
  <EEE>
    <CCC/>
    <DDD/>
  </EEE>
</AAA>
```

```
<AAA>
  <CCC>
    <BBB/>
    <BBB/>
    <BBB/>
  </CCC>
  <DDD>
    <BBB/>
    <BBB/>
  </DDD>
  <EEE>
    <CCC/>
    <DDD/>
  </EEE>
</AAA>
```

```
<AAA>
  <CCC>
    <BBB/>
    <BBB/>
    <BBB/>
  </CCC>
  <DDD>
    <BBB/>
    <BBB/>
  </DDD>
  <EEE>
    <CCC/>
    <DDD/>
  </EEE>
</AAA>
```

//*[count(BBB)=2]
Sélectionne les éléments
ayant deux enfants BBB

//*[count(*)=2]
Sélectionne les
éléments ayant deux
enfants

//*[count(*)=3]
Sélectionne les
éléments ayant trois
enfants



Chemin de localisation – name() ...

- La fonction **name()** retourne le nom de l'élément,
- La fonction **start-with()** retourne vrai si la chaîne du premier argument commence par celle du deuxième
- La fonction **contains()** retourne vrai si la chaîne du premier argument contient celle du deuxième

```
<AAA>
  <BCC>
    <BBB/>
    <BBB/>
    <BBB/>
  </BCC>
  <DDB>
    <BBB/>
    <BBB/>
  </DDB>
  <BEC>
    <CCC/>
    <DBD/>
  </BEC>
</AAA>
```

```
<AAA>
  <BCC>
    <BBB/>
    <BBB/>
    <BBB/>
  </BCC>
  <DDB>
    <BBB/>
    <BBB/>
  </DDB>
  <BEC>
    <CCC/>
    <DBD/>
  </BEC>
</AAA>
```

```
<AAA>
  <BCC>
    <BBB/>
    <BBB/>
    <BBB/>
  </BCC>
  <DDB>
    <BBB/>
    <BBB/>
  </DDB>
  <BEC>
    <CCC/>
    <DBD/>
  </BEC>
</AAA>
```

`//*[name()='BBB']`

`//*[starts-with(name(),'B')]`

`//*[contains(name(),'C')]`



Chemin de localisation – string-length()

- La fonction **string-length()** retourne le nombre de caractères dans une chaîne.
- Vous devez utiliser **<** comme substitutif de **<**
- et **>** comme substitutif de **>**

```
//*[string-length(name()) < 3]
```

```
<AAA>
<Q/>
<SSSS/>
<BB/>
<CCC/>
<DDDDDDDD/>
<EEEE/>
</AAA>
```

```
<AAA>
<Q/>
<SSSS/>
<BB/>
<CCC/>
<DDDDDDDD/>
<EEEE/>
</AAA>
```

```
<AAA>
<Q/>
<SSSS/>
<BB/>
<CCC/>
<DDDDDDDD/>
<EEEE/>
</AAA>
```

```
//*[string-length(name()) = 3]
```

```
//*[string-length(name()) > 3]
```



Chemin de localisation – |

- Plusieurs chemins peuvent être combinés avec le séparateur |

```
<AAA>
  <BBB/>
  <CCC/>
  <DDD>
    <CCC/>
  </DDD>
  <EEE/>
</AAA>
```

//CCC | //BBB :
Sélectionne tous les éléments CCC et BBB

```
<AAA>
  <BBB/>
  <CCC/>
  <DDD>
    <CCC/>
  </DDD>
  <EEE/>
</AAA>
```

/AAA/EEE | //BBB
Sélectionne tous les éléments BBB et EEE qui sont enfants de l'élément racine AAA

```
<AAA>
  <BBB/>
  <CCC/>
  <DDD>
    <CCC/>
  </DDD>
  <EEE/>
</AAA>
```

/AAA/EEE |
//DDD/CCC | /AAA |
//BBB
Le nombre de combinaison n'est pas restreinte



Chemin de localisation – Axe enfant

- L'axe enfant contient les enfants du noeud contextuel.
- L'axe enfant est celui par défaut et il peut être omis

```
<AAA>  
<BBB/>  
<CCC/>  
</AAA>
```

/AAA

Equivalent à /child::AAA

```
<AAA>  
<BBB/>  
<CCC/>  
</AAA>
```

/AAA/BBB

Equivalent à
/child::AAA/child::BBB

```
<AAA>  
<BBB/>  
<CCC/>  
</AAA>
```

/child::AAA/BBB

Les deux possibilités
peuvent être
combinées

Chemin de localisation – Axe descendant

- l'axe descendant (descendant) contient les descendants du noeud contextuel;
- un descendant est un enfant ou un petit enfant etc...
- Aussi, l'axe descendant ne contient jamais de noeud de type attribut ou des noms d'espace.

```
<AAA>
  <BBB>
    <DDD>
      <CCC>
        <DDD/>
        <EEE/>
      </CCC>
    </DDD>
  </BBB>
  <CCC>
    <DDD>
      <EEE>
        <DDD>
          <FFF/>
        </DDD>
      </EEE>
    </DDD>
  </CCC>
</AAA>
```

/AAA/BBB/descendant::*
Sélectionne tous les descendants de /AAA/BBB

```
<AAA>
  <BBB>
    <DDD>
      <CCC>
        <DDD/>
        <EEE/>
      </CCC>
    </DDD>
  </BBB>
  <CCC>
    <DDD>
      <EEE>
        <DDD>
          <FFF/>
        </DDD>
      </EEE>
    </DDD>
  </CCC>
</AAA>
```

//CCC/descendant::*
Sélectionne tous les éléments qui ont CCC comme ancêtre



Chemin de localisation – Axe parent

- L'axe "parent" contient le parent du noeud contextuel s'il en a un

```
<AAA>
  <BBB>
    <DDD>
      <CCC>
        <DDD/>
        <EEE/>
      </CCC>
    </DDD>
  </BBB>
  <CCC>
    <DDD>
      <EEE>
        <DDD>
          <FFF/>
        </DDD>
      </EEE>
    </DDD>
  </CCC>
</AAA>
```

//DDD/parent::*

Sélectionne tous les parents de l'élément DDD

Chemin de localisation – Axe ancestor

- L'axe ancêtre (ancestor) contient les ancêtres du noeud contextuel; cela comprend son parent et les parents des parents etc...
- Aussi, cet axe contient toujours le noeud racine, sauf si le noeud contextuel est lui-même la racine.

```
<AAA>
  <BBB>
    <DDD>
      <CCC>
        <DDD/>
        <EEE/>
      </CCC>
    </DDD>
  </BBB>
  <CCC>
    <DDD>
      <EEE>
        <DDD>
          <FFF/>
        </DDD>
      </EEE>
    </DDD>
  </CCC>
</AAA>
```

/AAA/BBB/DDD/CCC/EEE/ancestor::*
Sélectionne tous les éléments donnés dans ce chemin absolu

```
<AAA>
  <BBB>
    <DDD>
      <CCC>
        <DDD/>
        <EEE/>
      </CCC>
    </DDD>
  </BBB>
  <CCC>
    <DDD>
      <EEE>
        <DDD>
          <FFF/>
        </DDD>
      </EEE>
    </DDD>
  </CCC>
</AAA>
```

//FFF/ancestor::*
Sélectionne tous les ancêtres de l'élément FFF



Chemin de localisation – Axe follow-sibling

- l'axe 'following-sibling' contient tous les noeuds frères qui suivent le noeud contextuel.

```
<AAA>
  <BBB>
    <CCC/>
    <DDD/>
  </BBB>
  <XXX>
    <DDD>
      <EEE/>
      <DDD/>
      <CCC/>
      <FFF/>
      <FFF>
        <GGG/>
      </FFF>
    </DDD>
  </XXX>
  <CCC>
    <DDD/>
  </CCC>
</AAA>
```

```
<AAA>
  <BBB>
    <CCC/>
    <DDD/>
  </BBB>
  <XXX>
    <DDD>
      <EEE/>
      <DDD/>
      <CCC/>
      <FFF/>
      <FFF>
        <GGG/>
      </FFF>
    </DDD>
  </XXX>
  <CCC>
    <DDD/>
  </CCC>
</AAA>
```

/AAA/BBB/following-sibling::*

//CCC/following-sibling::*

Chemin de localisation – Axe preceding-sibling

- L'axe 'preceding-sibling' contient tous les frères prédecesseurs du noeud contextuel; si le noeud contextuel est un attribut ou un espace de noms, la cible précédente est vide.

```
<AAA>
  <BBB>
    <CCC/>
    <ZZZ>
      <DDD/>
    </ZZZ>
  </BBB>
  <XXX>
    <DDD>
      <EEE/>
      <DDD/>
      <CCC/>
      <FFF/>
      <FFF>
        <GGG/>
      </FFF>
    </DDD>
  </XXX>
  <CCC>
    <DDD/>
  </CCC>
</AAA>
```

/AAA/XXX/preceding::*

XPath

```
<AAA>
  <BBB>
    <CCC/>
    <ZZZ>
      <DDD/>
    </ZZZ>
  </BBB>
  <XXX>
    <DDD>
      <EEE/>
      <DDD/>
      <CCC/>
      <FFF/>
      <FFF>
        <GGG/>
      </FFF>
    </DDD>
  </XXX>
  <CCC>
    <DDD/>
  </CCC>
</AAA>
```

//GGG/preceding::*

Chemin de localisation – Axe following

- L'axe suivant (following) tous les noeuds du même document que le noeud contextuel qui sont après le noeud contextuel dans l'ordre du document, à l'exclusion de tout descendant, des attributs et des espaces de noms.

```
<AAA>
  <BBB>
    <CCC/>
    <ZZZ>
      <DDD/>
      <DDD>
        <EEE/>
      </DDD>
    </ZZZ>
    <FFF>
      <GGG/>
    </FFF>
  </BBB>
  <XXX>
    <DDD>
      <EEE/>
      <DDD/>
      <CCC/>
      <FFF/>
      <FFF>
        <GGG/>
      </FFF>
    </DDD>
  </XXX>
  <CCC>
    <DDD/>
  </CCC>
</AAA>
```

XPath

/AAA/XXX/following::*

```
<AAA>
  <BBB>
    <CCC/>
    <ZZZ>
      <DDD/>
      <DDD>
        <EEE/>
      </DDD>
    </ZZZ>
    <FFF>
      <GGG/>
    </FFF>
  </BBB>
  <XXX>
    <DDD>
      <EEE/>
      <DDD/>
      <CCC/>
      <FFF/>
      <FFF>
        <GGG/>
      </FFF>
    </DDD>
  </XXX>
  <CCC>
    <DDD/>
  </CCC>
</AAA>
```

//ZZZ/following::*

Chemin de localisation – Axe preceding-sibling

- l'axe cible précédente (preceding-sibling) contient tous les prédecesseurs du noeud contextuel; si le noeud contextuel est un attribut ou un espace de noms, la cible précédente est vide.

```
<AAA>
  <BBB>
    <CCC/>
    <ZZZ>
      <DDD/>
      </ZZZ>
    </BBB>
    <XXX>
      <DDD>
        <EEE/>
        <DDD/>
        <CCC/>
        <FFF/>
        <FFF>
          <GGG/>
        </FFF>
      </DDD>
    </XXX>
    <CCC>
      <DDD/>
    </CCC>
  </AAA>
```

```
<AAA>
  <BBB>
    <CCC/>
    <ZZZ>
      <DDD/>
      </ZZZ>
    </BBB>
    <XXX>
      <DDD>
        <EEE/>
        <DDD/>
        <CCC/>
        <FFF/>
        <FFF>
          <GGG/>
        </FFF>
      </DDD>
    </XXX>
    <CCC>
      <DDD/>
    </CCC>
  </AAA>
```

/AAA/XXX/preceding::*

//GGG/preceding::*

XPath

Chemin de localisation – Axe descendant-of-self

- L'axe "descendant-or-self" contient le noeud contextuel et ses descendants

```
<AAA>
  <BBB>
    <CCC/>
    <ZZZ>
      <DDD/>
    </ZZZ>
  </BBB>
  <XXX>
    <DDD>
      <EEE/>
      <DDD/>
      <CCC/>
      <FFF/>
      <FFF>
        <GGG/>
      </FFF>
    </DDD>
  </XXX>
  <CCC>
    <DDD/>
  </CCC>
</AAA>
```

```
<AAA>
  <BBB>
    <CCC/>
    <ZZZ>
      <DDD/>
    </ZZZ>
  </BBB>
  <XXX>
    <DDD>
      <EEE/>
      <DDD/>
      <CCC/>
      <FFF/>
      <FFF>
        <GGG/>
      </FFF>
    </DDD>
  </XXX>
  <CCC>
    <DDD/>
  </CCC>
</AAA>
```

/AAA/XXX/descendant-or-self::*

//CCC/descendant-or-self::*

Chemin de localisation – Axe ancestor-or-self

- L'axe ancestor-or-self contient le noeud contextuel et ses ancêtres; ainsi l'axe ancestor-or-self contient toujours le noeud racine

```
<AAA>
  <BBB>
    <CCC/>
    <ZZZ>
      <DDD/>
    </ZZZ>
  </BBB>
  <XXX>
    <DDD>
      <EEE/>
      <DDD/>
      <CCC/>
      <FFF/>
      <FFF>
        <GGG/>
      </FFF>
    </DDD>
  </XXX>
  <CCC>
    <DDD/>
  </CCC>
</AAA>
```

```
<AAA>
  <BBB>
    <CCC/>
    <ZZZ>
      <DDD/>
    </ZZZ>
  </BBB>
  <XXX>
    <DDD>
      <EEE/>
      <DDD/>
      <CCC/>
      <FFF/>
      <FFF>
        <GGG/>
      </FFF>
    </DDD>
  </XXX>
  <CCC>
    <DDD/>
  </CCC>
</AAA>
```

/AAA/XXX/DDD/EEE/ancestor-or-self::*

//GGG/ancestor-or-self::*

XPath



Modèle implicite nœuds de texte et attributs

- **Règle la plus basique :**

- Copie la valeur des nœuds de texte et d'attribut dans le document en sortie

Tous les nœuds de texte

```
<xsl:template match="text() | @*">
    <xsl:apply-templates select=". "/>
</xsl:template>
```

Tous les nœuds d'attribut

Pour un nœud de texte : texte du nœud
Pour un nœud attribut : valeur de l'attribut

Modèle implicite nœuds de texte et attributs

[8-1_05.xls](#)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:template match="people">
    <html>
      <head>
        <title>Scientifiques célèbres</title>
      </head>
      <body>
        <dl>
          <xsl:apply-templates/>
        </dl>
      </body>
    </html>
  </xsl:template>

  <xsl:template match="person">
    <dt>
      <xsl:apply-templates select="name"/>
    </dt>
    <dd>
      <ul>
        <li>Né : <xsl:apply-templates select="@born" /></li>
        <li>Mort : <xsl:apply-templates select="@died" /></li>
      </ul>
    </dd>
  </xsl:template>
</xsl:stylesheet>
```



```
<html>
  <head>
    <META http-equiv="Content-Type" content="text/html">
    <title>Scientifiques célèbres</title>
  </head>
  <body>
    <dl>
      <dt>AlanTuring</dt>
      <dd>
        <ul>
          <li>Né : 1912</li>
          <li>Mort : 1954</li>
        </ul>
      </dd>
      <dt>RichardPFeynman</dt>
      <dd>
        <ul>
          <li>Né : 1918</li>
          <li>Mort : 1988</li>
        </ul>
      </dd>
    </dl>
  </body>
</html>
```



Modèle implicite éléments et nœud racine

- Règle la plus importante, garantit que les sous éléments seront traités :

```
|    Joker XPath générique      | Correspond au nœud racine  
|    <xsl:template match="*|/*>|  
|        <xsl:apply-templates/>|  
|    </xsl:template>|
```

Le modèle est appliqué à tous les nœuds (sauf si autre modèle défini), à l'exception :

- Des nœuds d'attribut
- Des nœuds d'espaces de nom



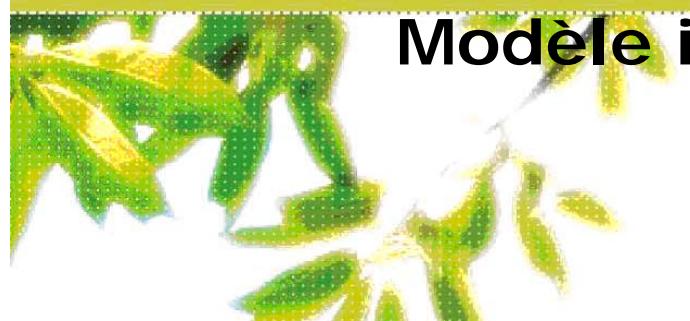
Modèle implicite commentaires et instructions

- Règle :

Instructions de traitement ou commentaires

```
<xsl:template match="processing-instruction() | comment()" />
```

Aucune sortie dans le document résultat, sauf si règle spécifique



Modèle implicite nœuds d'espace de noms

- **Règle implicite, non modifiable, car on ne peut l'écrire dans une feuille de style :**
 - Pas d'expression XPath correspondant aux nœuds d'espaces de noms

Génération HTML - Exercice

- Créer un fichier agent.xsl permettant de générer un fichier HTML donnant la liste des agents

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<personnes xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="agents.xsd">
  <personne>
    <nom>Dupond</nom>
    <service>Achats</service>
  </personne>
  <personne>
    <nom>Durand</nom>
    <service>Achats</service>
  </personne>
  <personne>
    <nom>Dupuis</nom>
    <service>Courrier</service>
  </personne>
</personnes>
```

Génération HTML - Exercice

[lsagents.xsl](#)

```
<?xml version="1.0" ?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="html" encoding="ISO-8859-1" />

  <xsl:template match="/">
    <html>
      <head>
        <title>
          Liste des personnes
        </title>
      </head>
      <body>
        <h1>
          Liste des personnes
        </h1>
        <blockquote>
          <xsl:apply-templates select="personne/nom" />
        </blockquote>
      </body>
    </html>
  </xsl:template>

  <xsl:template match="personne/nom">
    <p>
      <xsl:value-of select="." />
    </p>
  </xsl:template>

</xsl:stylesheet>
```



XQuery



XQUERY – Introduction

- XQuery est un langage de requête informatique permettant :
 - D'extraire des informations d'un document XML,
 - ou d'une collection de documents XML,
 - D'effectuer des calculs complexes à partir des informations extraites et de reconstruire de nouveaux documents ou fragments XML
- XQuery est une spécification du W3C
 - Version 1.0 finale : Janvier 2007 (8 ans d'élaboration)
 - Développé conjointement avec XSLT 2
 - avec lequel il partage le sous-ensemble XPath 2.
 - XQuery joue par rapport aux données XML un rôle similaire à celui du langage SQL vis-à-vis des données relationnelles (analogies entre ces deux langages).



XQUERY – Syntaxe

- Deux syntaxes distinctes pour XQuery :
- La syntaxe "naturelle" non-XML dite aussi **FLWOR** (prononcer flower), dont le nom vient des cinq instructions principales qui la composent (*for*, *let*, *where*, *order by* et *return*)
- La syntaxe **XQueryX** (pour « XML Syntax for XQuery »), dans laquelle une requête est un document XML.
 - De ce fait, elle est beaucoup plus verbeuse et moins lisible que la précédente et est destinée à des manipulations formelles par des programmes (éventuellement eux-mêmes écrits en XQuery).

XQUERY – Exemple

```
<employees>
  <employe>
    <nom>Dupond</nom>
    <prenom>Albert</prenom>
    <date_naissance>23/09/1958</date_naissance>
  </employe>
  <employe>
    <nom>Dupont</nom>
    <prenom>Alphonse</prenom>
    <date_naissance>23/12/1975</date_naissance>
  </employe>
  <employe>
    <nom>Dupont</nom>
    <prenom>Isabelle</prenom>
    <date_naissance>12/03/1967</date_naissance>
  </employe>
</employees>
```

```
for $b in //employe
where $b/nom = "Dupont"
return
<Dupont>{
  $b/prenom,
  $b/date_naissance
}</Dupont>
```

The diagram illustrates the XQuery query process. It starts with an XML document on the left, which is transformed by a large downward-pointing arrow into an XQuery query on the right. This XQuery query then produces an output XML document at the bottom.

```
<Dupont><prenom>Alphonse</prenom><date_naissance>23/12/1975</date_naissance></Dupont><Dupont><prenom>Isabelle</prenom><date_naissance>12/03/1967</date_naissance></Dupont>
```



XQUERY – Composante du langage

- XQuery est un langage spécifié de façon modulaire: possibilité d'ajout de modules optionnels.
- Le langage minimal se base sur la norme XPath 2 (qui spécifie le langage de requête XML proprement dit), augmentée par les principales fonctionnalités suivantes :
- L'expression FLWOR (For Let Where Order by Return), une puissante instruction de boucle, avec de nombreuses fonctionnalités, qui est assez similaire au SELECT de SQL.
- Grâce au **where**, il est possible d'écrire des jointures internes ou externes. XQuery version 1.1 ajoute le **group by**, et le "fenêtrage" (possibilité de découper la séquence d'entrée selon des conditions booléennes).
- Il existe d'autres constructions telles que **if** et **typeswitch** qui peuvent se composer avec le FLWOR.



XQUERY – Expressions de chemin

En fait pour extraire des informations d'un document XML il convient de toujours respecter :

Le chargement du document XML interrogé

La navigation à l'intérieur de ce document (XPath)

Le chargement d'un document XML se fera en utilisant une fonction d'ouverture

Syntaxe : **doc("chemin et nom du document_xml interrogé")**

Remarque : afin de ce simplifier la tâche tous les fichiers Xquery (.xq) se trouveront au même niveau que notre document XML

b) Le langage XQuery **utilise les expressions XPath** afin de naviguer dans le document XML interrogé. Il faut donc se positionner sur les éléments concernés par votre requête.



XQUERY – Expressions de chemin

Selector	Selected nodes
/	Document root
//	Any sub-path
*	Any element
name	Element of tag name
@*	Any attribute
@name	Attribute of name name
text()	Any text node
processing-instruction('name')	Processing instruction of given name
comment()	Any comment node
node()	Any node
id('value')	Element of id value



XQUERY – Exemple

Liste des noms des clients ([ex1.xq](#) et [ex1bis.xq](#))

requête

```
doc("commande.xml")/boutique/commande/client
```

réponse

```
<client origine="entreprise">Bernard</client>
<client origine="particulier">Jean-Marie</client>
<client origine="entreprise">Bernard</client>
<client origine="entreprise">Bernard</client>
<client origine="particulier">Bea </client>
```

OU

requête

Autre solution

```
doc("commande.xml")//commande/client
```

réponse

```
<client origine="entreprise">Bernard</client>
<client origine="particulier">Jean-Marie</client>
<client origine="entreprise">Bernard</client>
<client origine="entreprise">Bernard</client>
<client origine="particulier">Bea </client>
```



XQUERY – Exemple

: On désire la liste des numéros de commande et des noms des clients ([ex2.xq](#))

requête

```
doc("commande.xml")//commande/(numerodecommande,client)
```

réponse

```
<numerodecommande>1</numerodecommande>
<numerodecommande>2</numerodecommande>
<numerodecommande>3</numerodecommande>
<numerodecommande>4</numerodecommande>
<numerodecommande>5</numerodecommande>
<client origine="entreprise">Bernard</client>
<client origine="particulier">Jean-Marie</client>
<client origine="entreprise">Bernard</client>
<client origine="entreprise">Bernard</client>
<client origine="particulier">Bea</client>
```



XQUERY – Exercice

Effectuer une requête pour ramener les gaulois ([lesGaulois_01.xq](#))

- Leur nom
- Leur specialite
- Le nb de casques
- Le nom de la bataille



XQUERY – Exercice

Effectuer une requête pour ramener les gaulois ([leGaulois_01.xq](#))

- Leur nom
- Leur specialite
- Le nb de casques
- Le nom de la bataille

requête

```
xquery version "1.0";  
//lesgaulois/villageois/(NOM, NOM_SPECIALITE, QTE, NOM_BATAILLE)
```



XQUERY – Expressions XPath

- Expressions XPath :

- Obtention de la date de commande de la 1° commande (ex3.xq) :

```
doc( "commande.xml" )//commande[1]/datecommande
```

- Obtenir les commandes, dont la dte de commande est le 17/01/2007

(ex4.xq) :

```
doc( "commande.xml" )/boutique/commande[datecommande="17/01/2007"]
```

- Obtenir les commandes pour le client Bernars dont le N° de commande est supérieur à 2 (ex5.xq):

```
doc( "commande.xml" )//commande[client = "Bernard" and  
numerodecommande>2 ]
```

- Obtenir la dernière commande (ex6.xq) :

```
doc( "commande.xml" )//commande[last()]
```

- Obtenir toutes les informations de la 1° commande , sauf ce qui concerne la balise produit (ex7.xq) :

```
doc( "commande.xml" )//commande[1]/* except produit)
```



XQUERY – Expressions XPath - Exercices

- Expressions XPath :

- Obtenir le nom de la bataille , la quantité de casques et le nom du casque pour le 12° gaulois – lesGaulois_02.xq :
- Obtenir les noms des gaulois ayant ramenés plus de 2 casques – lesGaulois_03.xq :
- Obtenir le nom et specialite des gaulois ayant participé à la bataille de Babaorum – lesGaulois_04.xq :



XQUERY – Expressions XPath - Exercices

- Expressions XPath :

- Obtenir le nom de la bataille , la quantité de casques et le nom du casque pour le 12° gaulois – lesGaulois_02.xq :

```
//lesgaulois/villageois[12]/(NOM_BATAILLE, QTE, NOM_CASQUE)
```

- Obtenir les noms des gaulois ayant ramenés plus de 2 casques – lesGaulois_03.xq :

```
//lesgaulois/villageois[../QTE >2]/(NOM)  
//lesgaulois/villageois/(NOM)[../QTE >2]
```

- Obtenir le nom et specialite des gaulois ayant participé à la bataille de Babaorum – lesGaulois_04.xq

```
//lesgaulois/villageois/(NOM, NOM_SPECIALITE)[../NOM_BATAILLE =  
'Babaorum']
```

```
//lesgaulois/villageois )[NOM_BATAILLE = 'Babaorum'] /(NOM,  
NOM_SPECIALITE
```



XQUERY – Expressions XPath

- Expressions XPath :
 - Afficher un message – ex8.xq :

```
"Le client de la commande 5 est : ",  
doc("commande.xml")//commande[5]/client/text()
```
 - Tester un attribut – ex9.xq :

```
doc("commande.xml")//commande/client[@origine="particulier"]
```



XQUERY – Expressions XPath - Exercices

- Expressions XPath :

- Afficher un message – permettant de formaliser les informations concernant Astérix – lesGaulois_05.xq

Asterix est un Guerrier il a participé à la bataille de Gladiateurs



XQUERY – Expressions XPath - Exercices

- Expressions XPath :

- Afficher un message – permettant de formaliser les informations concernant Astérix – lesGaulois_05.xq

```
xquery version "1.0";
//lesgaulois/villageois[NOM = 'Asterix']/NOM/text() ,
" est un ", //lesgaulois/villageois[NOM =
'Asterix']/NOM_SPECIALITE/text() ,
" il a participe a la bataille de ", //lesgaulois/villageois
[NOM = 'Asterix']/NOM_BATAILLE/text()
```

XQUERY – Expression de comparaison

Comparaison générale	=, !=, <, >, <=, >=
Comparaison d'ordre	>>, <<
Comparaison logique	and, or
Comparaison de valeurs	eq, ne, it, le, gt, ge
Comparaison de nœuds	is, isnot, except
Comparaison de position	last(), position()

Syntaxe : doc("document_xml")/root-element/element[element expression valeur]

Pour se positionner sur un élément précis :

`doc("document_xml")/root-element/element[valeur]/element`

XQUERY – Constructeurs d'éléments

- Ils sont utilisés dans une requête XQUERY afin de construire d'autres nœuds.
 - Les accolades { } permettent de placer une expression au sein d'un constructeur
- Exemple 10 : Nom du client n° 2 apparaissant dans un nœud nommé <nomduclient> concernant la commande n° 2 (ex10.xq) .

requête	Réponse
<pre><nomduclient> { doc("commande.xml")//commande[2]/client } </nomduclient></pre>	<pre><nomduclient> <client origine="particulier"> Jean-Marie </client> </nomduclient></pre>



XQUERY – Expressions FLWOR

- L'expression FLWOR est un acronyme de For, Let, Where, Order, Return.
- On prononce cette expression flower
- D'une manière générale :
 - **For \$variable in doc(document xml)//élément** : donne un mécanisme d'itération en affectant la variable (ici \$x) successivement avec chaque item dans la séquence renvoyée après le in
 - Exemple 11 : Commandes du clients Bernard(ex11.xq) .

requête

```
xquery version "1.0";
for $x in doc("commande.xml")/boutique/commande
    where $x/client="Bernard"
return $x/numerodecommande
```

Réponse

```
<numerodecommande>1</numerodecommande>
<numerodecommande>3</numerodecommande>
<numerodecommande>4</numerodecommande>
```



XQUERY – Expressions FLWOR - Exercice

- Exercice : Ramener à l'aide d'une expression FLOWR, les villageois ayant participé à la bataille de Babaorum - lesGaulois_06.xq

NOM (5)	
	Abc Text
1	Abraracourcix
2	Antibiotix
3	Obelix
4	Orthopedix
5	Salamix



XQUERY – Expressions FLWOR - Exercice

- Exercice : Ramener à l'aide d'une expression FLOWR, les villageois ayant participé à la bataille de Babaorum - lesGaulois_06.xq

```
xquery version "1.0";
for $x in //lesgaulois/villageois
where $x/NOM_BATAILLE="Babaorum"
return $x/NOM
```



XQUERY – Expressions FLWOR

- **let \$variable := exp** affecte la variable \$variable avec la séquence "entièr"e retournée par exp
- Exemple n° 13 : On désire obtenir que les noms des produits se trouvant sur la commande n° 1 (ex13.xq)

requête

```
for $b in doc("commande.xml")//commande[1]/produit  
    let $al := $b/nom  
return $b
```

Réponse

```
<nom>Graveur * 16</nom>  
<nom>DVD-R</nom>
```



XQUERY – Expressions FLWOR

- **Where** : permet de filter à partir d'un ou plusieurs prédictats
- **Order by** : permet la réalisation d'un tri
- **Return \$variable** : génère le résultat de l'expression FLWOR
- Exemple n° 14 : On désire tous les noms des produits achetés par le client Bernard

requête

```
for $x in doc("commande.xml")/boutique/commande  
    where $x/client="Bernard"  
return $x/produit/nom
```

Réponse

```
<nom>Graveur * 16</nom>  
<nom>DVD-R</nom>  
<nom>Ecran 21 pouces</nom>  
<nom>Clavier</nom>
```



XQUERY – Expressions FLWOR - Exercice

- Afficher tous les villageois ayant ramené des casques de type **Centurion** – lesGaulois_07.xq

NOM (5)	
	Abc Text
1	Agecanonix
2	Asterix
3	Aventurepix
4	Obelix
5	Orthopedix



XQUERY – Expressions FLWOR - Exercice

- Afficher tous les villageois ayant ramené des casques de type **centurion**

```
xquery version "1.0";
for $x in /lesgaulois/villageois
    where
        $x/NOM_CASQUE="Centurion"
    return $x/NOM
```



XQUERY – Fonctions

- Le langage XQUERY inclut un nombre important de fonctions (date, chaîne de caractères, mathématiques, conversion...).

Fonctions d'**accès** : data()...

Fonctions **numériques** : abs(), floor(), ceiling(), round()...

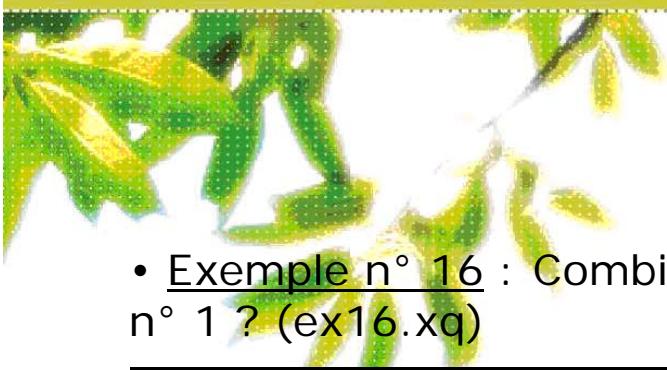
Fonctions de **chaînes** : upper-case(), lower-case(), contains(), substring()...

Fonctions **temporelles** : day-from-date(), daytime()...

Fonctions de **séquence** : distinct-values(), reverse(), exist()...

Fonctions d'**agrégat** : count(), avg(), sum(), min(), max()...

Fonctions de **contexte** : position(), last()...



XQUERY – Fonctions

- Exemple n° 16 : Combien de type de produits se trouve sur la commande n° 1 ? (ex16.xq)

requête	Réponse
<pre>for \$b in doc("commande.xml")//commande[1] let \$al := \$b/produit return count(\$al)</pre>	2

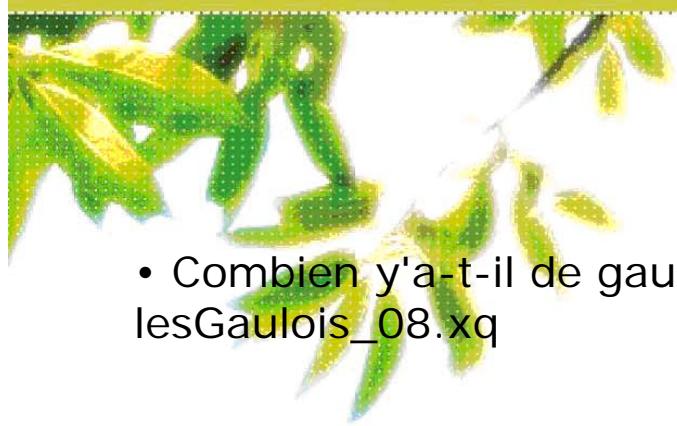
- Exemple n° 16bis : nombre total de commandes ? (ex16bis.xq)

requête	Réponse
<pre>xquery version "1.0"; let \$r:=doc("commande.xml") //commande return <nombre> {count(\$r)} </nombre></pre>	<nombre>5</nombre>



XQUERY – Fonctions - Exercices

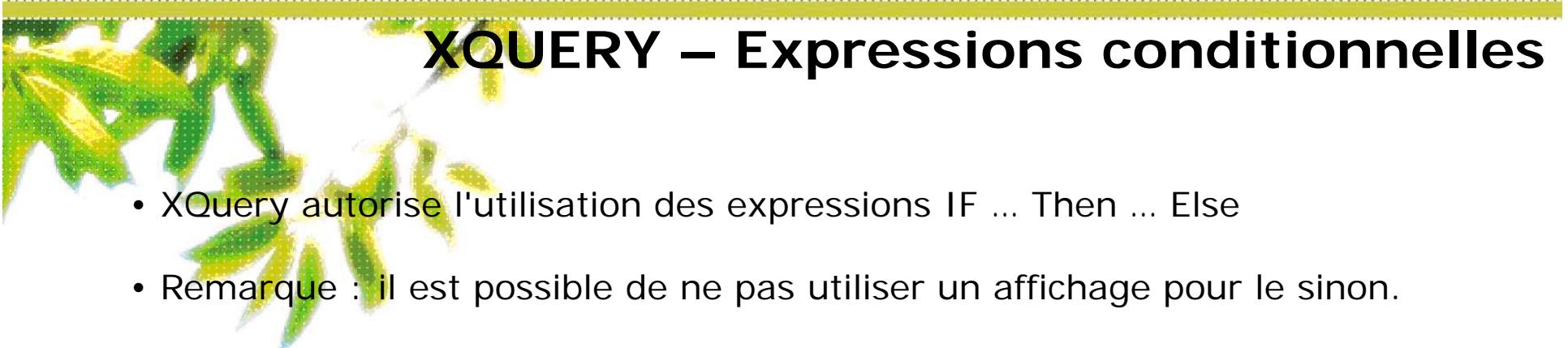
- Combien y'a-t-il de gaulois dans le fichier lesgaulois.xml ? –
lesGaulois_08.xq



XQUERY – Fonctions - Exercices

- Combien y'a-t-il de gaulois dans le fichier lesgaulois.xml ? – lesGaulois_08.xq

```
xquery version "1.0";
for $b in //lesgaulois
    let $al := $b/villageois/NOM
return count($al)
```



XQUERY – Expressions conditionnelles

- XQuery autorise l'utilisation des expressions IF ... Then ... Else
- Remarque : il est possible de ne pas utiliser un affichage pour le sinon.
- Cependant la syntaxe impose que le else() soit présent mais vide
- Exemple n° 19 : Faire apparaître les noms des clients qui ont effectué une commande de type papeterie (ex19.xq)

requête

```
for $x in doc("commande.xml")/boutique/commande  
return if ($x/@type= "papeterie")  
then $x/client  
else()
```

Réponse

```
<client origine="particulier">Jean-Marie</client>  
<client origine="particulier">Bea</client>
```



XQUERY – Compléments

- Il est possible d'éditer une réponse à l'intérieur de balises XHTML.

```
xquery version "1.0";
<html>
<body>

<table border="2" align="center"
cellspacing="1" bordercolor="blue">
<caption><h2>Liste des clients
</h2></caption>
<tr>
<td><client
origine="particulier">Bea</client></td>
<td><client
origine="entreprise">Bernard</client></td>
<td><client
origine="entreprise">Bernard</client></td>
<td><client origine="particulier">Jean-
Marie</client></td>
</tr>
</table>
</body>
</html>
```

```
<html><body>
<table border="2" align="center"
cellspacing="1" bordercolor="blue">
<caption><h2>Liste des clients
</h2></caption>
<tr>
<td><client
origine="particulier">Bea</client></td>
<td><client
origine="entreprise">Bernard</client></td>
<td><client
origine="entreprise">Bernard</client></td>
<td><client origine="particulier">Jean-
Marie</client></td>
</tr>
</table>
</body></html>
```



XQUERY – Exercice

- Sortir la fiche d'Astérix en HTML

XQUERY – Exercice

- Sortir la fiche d'Astérix en HTML

```
xquery version "1.0";
<html>
  <body>
    <h2>Astérix</h2>
    {
      for $x in doc("lesgaulois.xml")//villageois
      where $x/NOM = 'Astérix'
      return
      <div>
        <div style="float:left">
          
        </div>
        <div style="float:left;font-family:verdana;font-weight:bold;font-size:12px;margin-left:10px;" >
          Nom : {$x/NOM/text()}
          <br />
          ADRESSE : {$x/ADRESSE/text()}
          <br />
          Specialité : {$x/NOM_SPECIALITE/text()}
        </div>
      </div>
    }
  </body>
</html>
```



XQUERY – Jointures

- Il est possible de relier différents documents xml entre eux afin d'en tirer une ou plusieurs informations communes.

- Syntaxe générale :

- Utilisation d'une boucle pour stocker les différents documents à utiliser

```
for $variable in doc("document.xml")//element  
for $variable2 in doc("document2.xml")//element  
...
```

- Jointure sur un élément commun

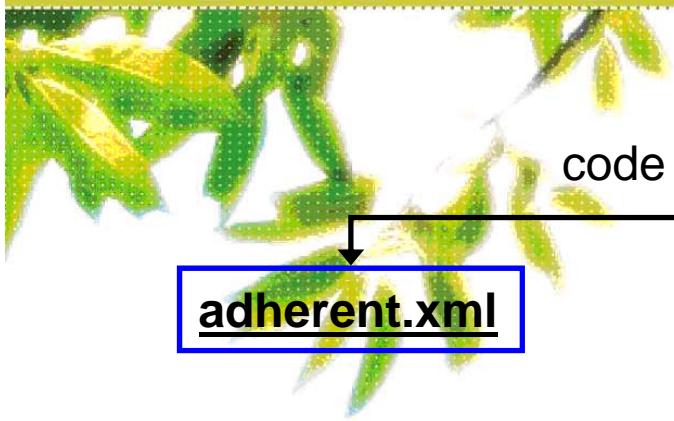
```
where $variable/element=$variable2/element
```

- Constructeur d'élément afin de construire un nœud réponse.

- Les accolades **{data(...)}** permettent de placer une expression au sein d'un constructeur

```
return  
<resultatrequete>  
    {data($variable/element)} ,{...} , ...  
</resultatrequete>
```

XQUERY – Jointures - Exemple



code

adherent.xml

Emprunt.xml

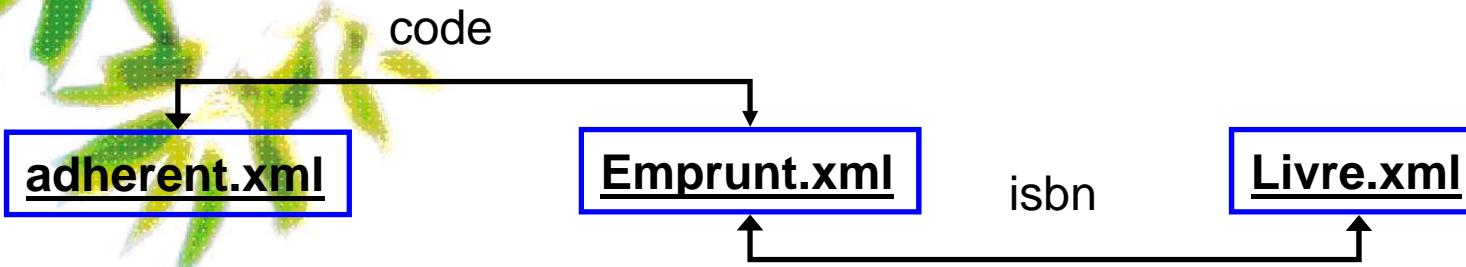
isbn

Livre.xml





XQUERY – Jointures - Exemple



Exemple n° 22 : On désire obtenir le nom, le n° d'isbn, les dates d'emprunt et de retour des livres empruntés (jointure1.xq)

```
xquery version "1.0";
for $adherent in doc("adherent.xml")//membre
for $emprunt in doc("emprunt.xml")//contenu
for $livre in doc("livre.xml")//livre
where
    $adherent/code=$emprunt/code
    and $livre/isbn= $emprunt/isbn
return
<resultat>
{data($adherent/nom)},
{data($emprunt/isbn)},
{data($emprunt/dateemprunt)},
{data($emprunt/dateretour)},
{data($livre/titre)}
</resultat>
```

fichiers xml
jointures

resultat (8)	
	Abc Text
1	Cardoni, 1234, 15/01/2007, 21/01/2007, ACCESS 20007
2	Cardoni, 567, 22/01/2007, 24/01/2007, XHTML et CSS
3	Lallement, 990, 22/01/2007, , Informatique de gestion
4	Lallement, 89, 31/01/2007, , C# par la pratique
5	
6	
7	
8	



XQUERY – Jointures - Exercice

lesGaulois_J01.xq

ID_LIEU

villageois.xml

lieu.xml

Effectuer une requête permettant de ramener les informations concernant les villageois, informations pour chaque villageois :

- nom , lieu d'habitation
- villageois :
 - ID_VILLAGEOIS : identifiant du villageois
 - **ID_LIEU : identifiant du lieu d'habitation**
- lieu :
 - **ID_LIEU : identifiant du lieu**



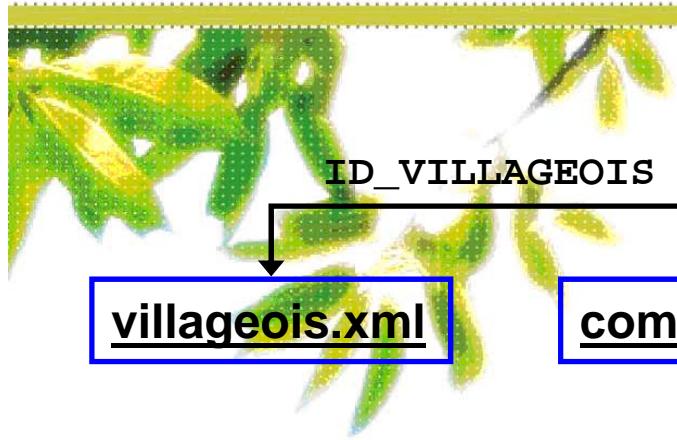
XQUERY – Jointures - Exercice

ID_LIEU

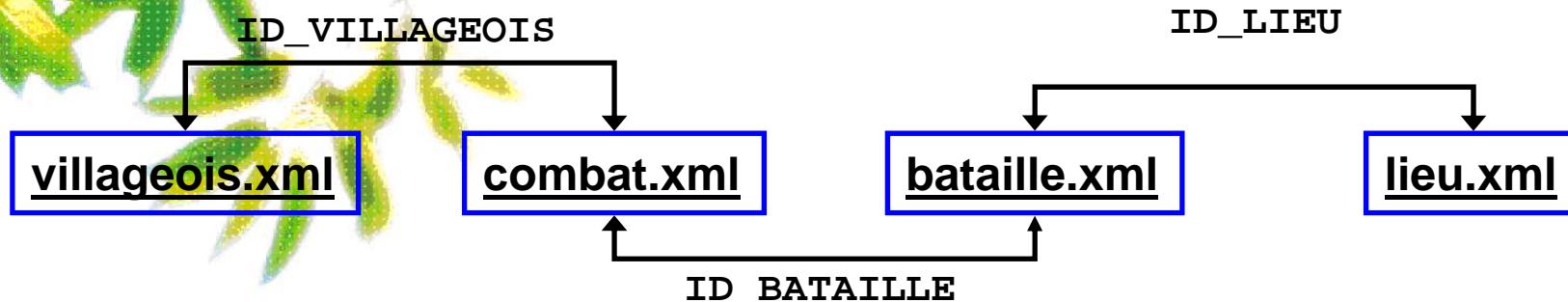
villageois.xml

lieu.xml

```
xquery version "1.0";
for $villageois in doc("villageois.xml")//villageois
for $lieu in doc("lieu.xml")//lieu
where $villageois/ID_LIEU=$lieu/ID_LIEU
return
<NOM> {data($villageois/NOM)}</NOM>
<LIEU> {data($lieu/NOM_LIEU)}</LIEU>
```



XQUERY – Jointures - Exercice



Effectuer une requête permettant de ramener les informations concernant les villageois, informations pour chaque villageois :

- Les batailles auxquelles il a participé
- Les lieux où elles se sont déroulées

lesGaulois_J02.xq

- villageois :
 - ID_VILLAGEOIS : identifiant du villageois
- combat :
 - ID_VILLAGEOIS : identifiant du villageois
 - ID_BATAILLE : identifiant de la bataille
- bataille :
 - ID_BATAILLE : identifiant de la bataille
 - ID_LIEU : identifiant du lieu
- lieu :
 - ID_LIEU : identifiant du lieu



XQUERY – Jointures - Exercice

[villageois.xml](#)[combat.xml](#)[bataille.xml](#)[lieu.xml](#)

```
xquery version "1.0";
for $villageois in doc("villageois.xml")//villageois
for $lieu in doc("lieu.xml")//lieu
for $combat in doc("combat.xml")//prise_casque
for $bataille in doc("bataille.xml")//bataille

where $villageois/ID_VILLAGEOIS=$combat/ID_VILLAGEOIS
and $combat/ID_BATAILLE=$bataille/ID_BATAILLE
and $bataille/ID_LIEU=$lieu/ID_LIEU

return
<resultat>
  <NOM> {data($villageois/NOM)}</NOM>
  <BATAILLE> {data($bataille/NOM_BATAILLE)} </BATAILLE>
  <LIEU> {data($lieu/NOM_LIEU)} </LIEU>
</resultat>
```



XQUERY – Jointures - Exercice

[villageois.xml](#)

[combat.xml](#)

[bataille.xml](#)

[lieu.xml](#)

A partir de la requête précédente , créer une extraction pour Astérix



XQUERY – Jointures - Exercice

[villageois.xml](#)

[combat.xml](#)

[bataille.xml](#)

[lieu.xml](#)

A partir de la requête précédente , créer la fiche html pour Astérix

```
xquery version "1.0";
for $villageois in doc("villageois.xml")//villageois
for $lieu in doc("lieu.xml")//lieu
for $combat in doc("combat.xml")//prise_casque
for $bataille in doc("bataille.xml")//bataille
where $villageois/ID_VILLAGEOIS=$combat/ID_VILLAGEOIS
and      $combat/ID_BATAILLE=$bataille/ID_BATAILLE
and      $bataille/ID_LIEU=$lieu/ID_LIEU
and $villageois/NOM = 'Astérix'
return
<resultat>
  <NOM> {data($villageois/NOM)}</NOM>
  <BATAILLE> {data($bataille/NOM_BATAILLE)} </BATAILLE>
  <LIEU> {data($lieu/NOM_LIEU)} </LIEU>
</resultat>
```



XSL-FO

eXtensible Stylesheet Language - Formatting Objects

<http://www.w3.org/TR/xsl/>

<http://www.w3.org/2005/11/Translations/Query?titleMatch=&lang=fr&search1=Submit>



XSL-FO - Introduction

- **XSL-FO** : **Vocabulaire** qui décrit les **mises en forme de documents** XML quel que soit le support : écran, papier, audio, etc.
- **XSL-FO** s'adresse principalement aux **typographes** afin de fournir avec les outils de gestion de documents, un outil typographique du niveau attendu par les publications imprimées.
- Il n'est pas prévu que les documents originaux soient rédigés avec XSL-FO, mais plutôt dans des dialectes adaptés (XHTML, DocBook, TEI, etc.).
- Ils peuvent alors être convertis en XSL-FO à l'aide de XSLT, une autre composante de la recommandation XSL.
- Finalement, un **processseur XSL-FO** permet de **générer** les **documents finaux** (par exemple pages imprimables en PDF ou PostScript ou autre).

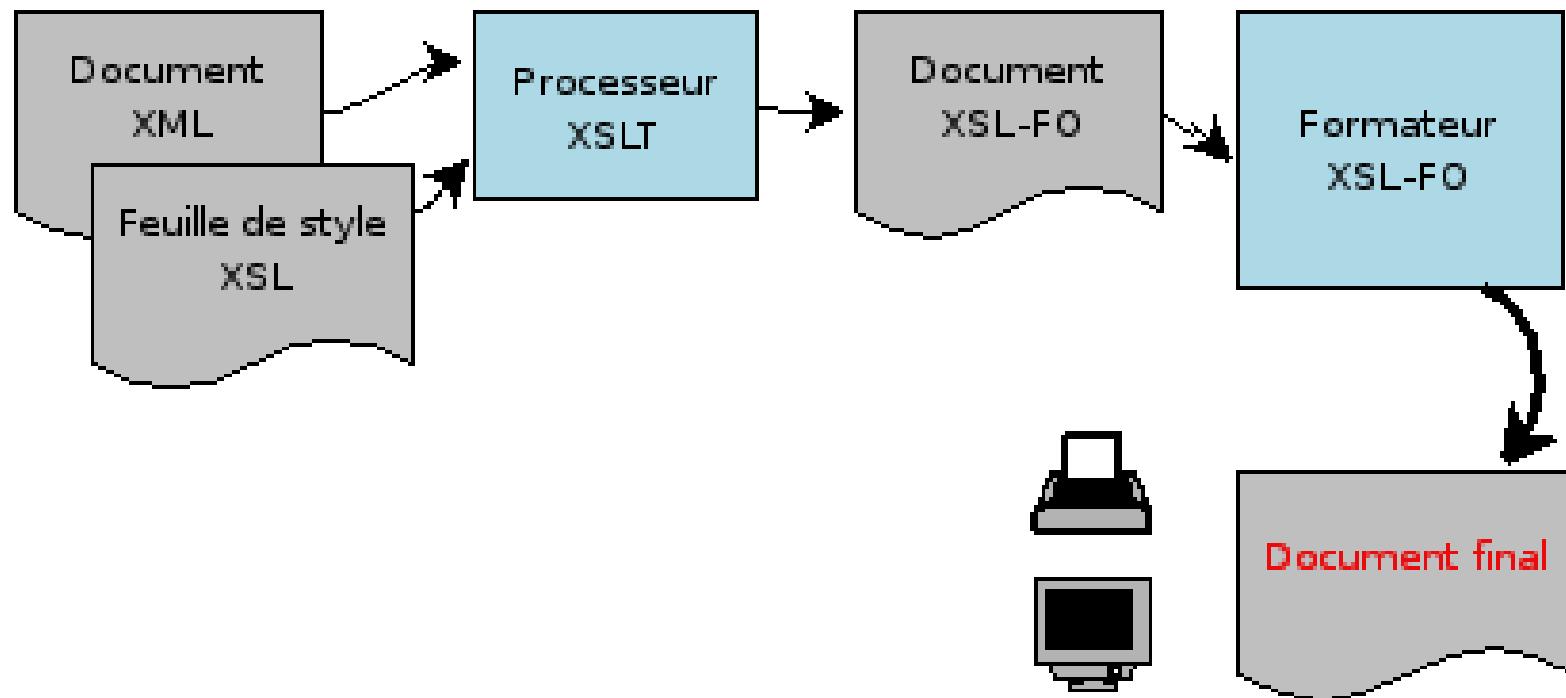


XSL-FO – Objectifs et principes

- L'objectif de XSL-FO est de **créer un arbre d'aires** où une aire est une **zone d'affichage** (visuelle ou auditive).
- Les aires sont de 2 types :
 - Les **aires de blocs** s'empilent les unes sur les autres,
 - Les **aires en-ligne** s'empilent les unes à côté des autres.
- XSL-FO fournit l'ensemble des commandes de contrôle de chaque aire :
 - Présentation du contenu,
 - Direction de l'empilement :
 - écriture de gauche à droite
 - ou inversement,
 - de haut en bas)



XSL-FO – Objectifs et principes



XSL-FO – Objectifs et principes - Exemple

Le langage XSL-FO emploie les CSS pour décrire les attributs formatés, comme :

- Les polices
- des couleurs
- des bordures
- Exemple : hello World

Élément racine fo:root, espace de nom :fo

```
<?xml version="1.0" encoding="utf-8"?>
<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
    <fo:layout-master-set>
        <fo:simple-page-master master-name="LetterPage" page-width="8.5in" page-height="11in">
            <fo:region-body region-name="PageBody" margin="0.7in"/>
        </fo:simple-page-master>
    </fo:layout-master-set>

    <fo:page-sequence master-reference="LetterPage">
        <fo:flow flow-name="PageBody">
            <fo:block>Hello World</fo:block>
        </fo:flow>
    </fo:page-sequence>
</fo:root>
```

La structure des pages est définie en utilisant fo:layout-master-set

Paragraphe de la page

PDF

Hello World



XSL-FO – fo:root

- C'est (comme son nom l'indique) l'élément racine du document XSL-FO.
- Exemple qui montre une structure typique de document :

```
<fo:root xmlns :fo="http://www.w3.org/1999/XSL/Format">  
--- --- --- le reste du document --- --- ---  
</fo:root>
```



XSL-FO – fo:layout-master-set

- **layout**, en anglais : arrangement, formation
- Spécifie la définition des pages
- Document simple : un seul layout peut être suffisant
- Document complexe : généralement plusieurs
 - Exemple :
 - 1° page différente,
 - 1° page des chapitres différentes des autres pages



XSL-FO – fo:simple-page-master

- Cet élément définit le layout d'une page particulière
- Exemple :

```
<fo:layout-master-set>
    <fo:simple-page-master
        master-name="simpleA4"
        page-height="21cm"
        page-width="29.7cm">

        <fo:region-body/>
    </fo:simple-page-master>
</fo:layout-master-set>
```

- On définit à l'intérieur de cette balise toutes les pages de notre document
- On y fait ensuite référence grâce à leur nom

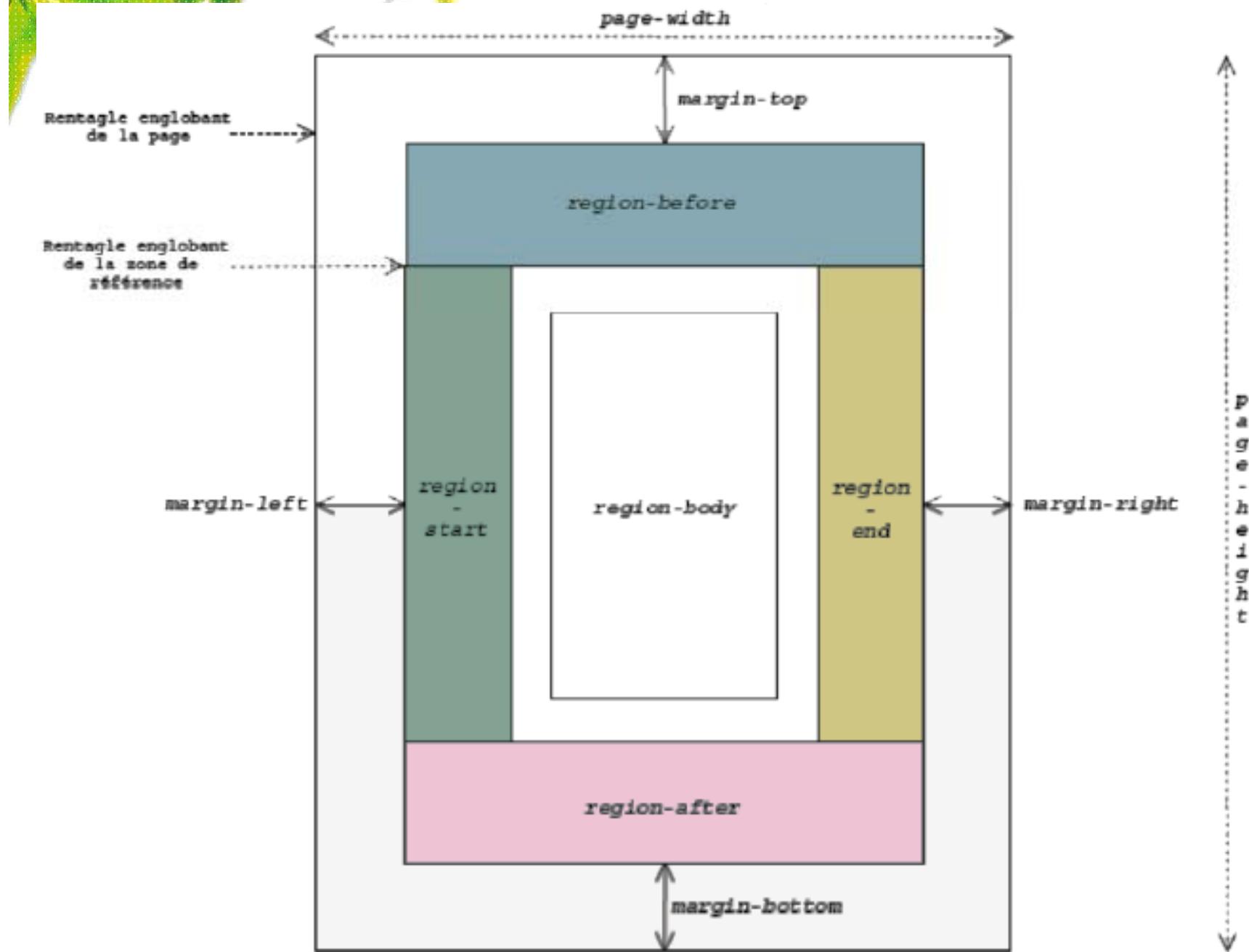


XSL-FO – fo:page-sequence-master

- Cet élément définit une séquence de layouts à utiliser dans le document
- Si nous voulons utiliser le simple-page-master appelé *simpleA4* pour toutes nos pages, il suffit de faire :

```
<fo:page-sequence-master reference="simpleA4">
```

XSL-FO – modèle simple page





XSL-FO – modèle simple page - Exemple

```
<fo:simple-page-master master-name="A4"

    page-width="21cm"
    page-height="29.7cm"

    margin-bottom="2cm"
    margin-top="2cm"
    margin-left="2.5cm"
    margin-right="2.5cm">

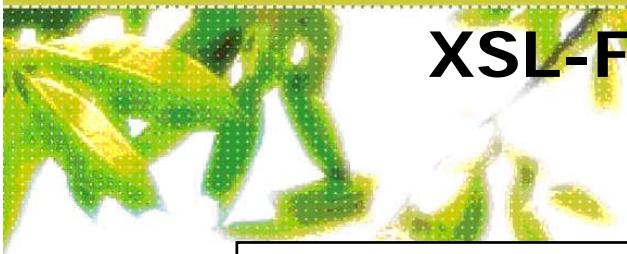
    ...

</fo:simple-page-master>
```



XSL-FO – Découpage en régions

- Une page est découpée en régions :
 - region-body,
 - region-before,
 - region-after,
 - region-start,
 - region-end.
- La taille d'une région est définie par l'attribut extent.
- Seule la taille de la région centrale ne peut pas être précisée.
- Les régions situées autour ont une taille mais pas de marge.



XSL-FO – Découpage en régions Exemple

regions.fo

```
<?xml version="1.0" encoding="UTF-8"?>
<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
  <fo:layout-master-set>
    <fo:simple-page-master master-name="page" page-height="297mm" page-width="210mm">
      <fo:region-body background-color="yellow" margin="20mm" />
      <fo:region-before background-color="blue" margin="20mm" />
      <fo:region-after background-color="green" extent="20mm" />
      <fo:region-start background-color="red" extent="20mm" />
      <fo:region-end background-color="lightgreen" extent="20mm" />
    </fo:simple-page-master>
  </fo:layout-master-set>

  <fo:page-sequence master-reference="page">
    <fo:flow flow-name="xsl-region-body">
      <fo:block>Contenu</fo:block>
    </fo:flow>
  </fo:page-sequence>

</fo:root>
```



XSL-FO – Découpage en régions Exemple





XSL-FO – Contenu du document

- Seconde partie d'un doc XSL-FO
 - Défini à l'aide de l'instruction :
 - fo:page-sequence (pour un ensemble de pages)
 - Contient les éléments :
 - fo:title,
 - fo:block
 - et fo:static-content :
 - un seul fo:flow est autorisé dans un élément fo:page-sequence
 - plusieurs fo:static-content sont autorisés



XSL-FO – Contenu du document : static

contenuStatic.fo

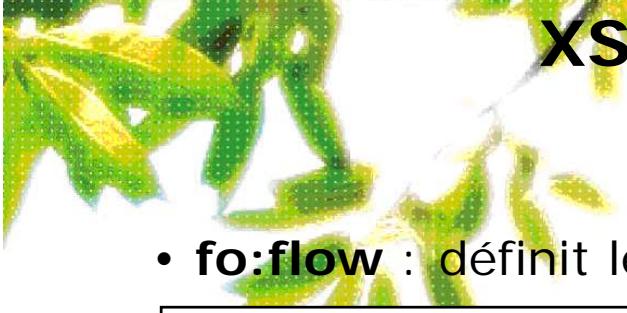
- **fo:static-content** : contenu défini pour toutes les pages

```
<?xml version="1.0" encoding="UTF-8"?>
<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
  <fo:layout-master-set>
    <fo:simple-page-master master-name="page" page-height="297mm" page-width="210mm">
      <fo:region-body background-color="yellow" margin="20mm" />
      <fo:region-before background-color="blue" margin="20mm" />
      <fo:region-after background-color="green" extent="20mm" />
      <fo:region-start background-color="red" extent="20mm" />
      <fo:region-end background-color="lightgreen" extent="20mm" />
    </fo:simple-page-master>
  </fo:layout-master-set>

  <fo:page-sequence master-reference="page">
    <fo:static-content flow-name="xsl-region-before">
      <fo:block>En tête</fo:block>
    </fo:static-content>

    <fo:flow flow-name="xsl-region-body">
      <fo:block>Contenu</fo:block>
    </fo:flow>

  </fo:page-sequence>
</fo:root>
```



XSL-FO – Contenu du document : flow

contenuFlow.fo

- **fo:flow** : définit le contenu (glissant) des pages

```
<?xml version="1.0" encoding="UTF-8"?>
<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
  <fo:layout-master-set>
    <fo:simple-page-master master-name="page" page-height="297mm" page-width="210mm">
      <fo:region-body background-color="yellow" margin="20mm" />
      <fo:region-before background-color="blue" margin="20mm" />
      <fo:region-after background-color="green" extent="20mm" />
      <fo:region-start background-color="red" extent="20mm" />
      <fo:region-end background-color="lightgreen" extent="20mm" />
    </fo:simple-page-master>
  </fo:layout-master-set>

  <fo:page-sequence master-reference="page">
    <fo:static-content flow-name="xsl-region-before" >
      <fo:block>En tête</fo:block>
    </fo:static-content>

    <fo:flow flow-name="xsl-region-body" >
      <fo:block>
        Lorem ipsum dolor sit amet, consectetur adipiscing elit. ...
        |egestas. Praesent magna odio, adipiscing non fringilla et, eleifend feugiat eros.
      </fo:block>
    </fo:flow>
  </fo:page-sequence>
</fo:root>
```



XSL-FO – Contenu du document : block

- **fo:block** : permet de formater des blocs de texte et d'images.
- Un bloc peut représenter un ensemble de lignes ou une partie de ligne
- Un bloc possède des attributs permettant de spécifier sa police, sa couleur, la taille de la police ...
- Un bloc vide permet de simuler un saut de ligne



XSL-FO – Contenu du document : block

contenuBlock.fo

```
<?xml version="1.0" encoding="UTF-8"?>
<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
  <fo:layout-master-set>
    <fo:simple-page-master master-name="page" page-height="297mm" page-width="210mm">
      <fo:region-body margin="20mm" />
      <fo:region-before margin="20mm" />
      <fo:region-after extent="20mm"/>
      <fo:region-start extent="20mm" />
      <fo:region-end extent="20mm" />
    </fo:simple-page-master>
  </fo:layout-master-set>

  <fo:page-sequence master-reference="page">
    <fo:static-content flow-name="xsl-region-before" >
      <fo:block>En tête</fo:block>
    </fo:static-content>

    <fo:flow flow-name="xsl-region-body" >
      <fo:block background-color="yellow" font-family="verdana" font-size="16pt" font-weight="bold" color="blue">
        Mauris rhoncus fringilla purus
      </fo:block>
      <fo:block font-family="verdana" font-size="12pt" font-weight="normal">
        Aliquam rhoncus bibendum ultrices. Phasellus eget leo sapien, bibendum mattis nisi. Nulla laoreet nulla at eros adipiscing bibendum. Pellentesque nec eros leo. Sed ac sem nec nibh congue luctus. Duis nec auctor sem. Vestibulum consequat feugiat faucibus. Maecenas rhoncus velit et massa posuere egestas. Praesent magna odio, adipiscing non fringilla et, eleifend feugiat eros.
      </fo:block>
    </fo:flow>

  </fo:page-sequence>
</fo:root>
```



XSL-FO – Contenu du document : listes

- **fo:list-block** : listes à puces
- Exemple :

```
<fo:list-block>
  <fo:list-item>
    <fo:list-item-label>...</fo:list-item-label>
    <fo:list-item-body>...</fo:list-item-body>
  </fo:list-item>
  ...
</fo:list-block>
```

XSL-FO – Contenu du document : listes - Exemple

```
<?xml version="1.0" encoding="UTF-8"?>
<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
  <fo:layout-master-set>
    <fo:simple-page-master master-name="page" page-height="297mm" page-width="210mm">
      <fo:region-body margin="20mm" />
      <fo:region-before margin="20mm" />
      <fo:region-after extent="20mm" />
      <fo:region-start extent="20mm" />
      <fo:region-end extent="20mm" />
    </fo:simple-page-master>
  </fo:layout-master-set>

  <fo:page-sequence master-reference="page">
    <fo:static-content flow-name="xsl-region-before">
      <fo:block>En tête</fo:block>
    </fo:static-content>

    <fo:flow flow-name="xsl-region-body" >
      <fo:block background-color="yellow" font-family="verdana" font-size="16pt" font-weight="bold" color="blue">
        Un titre
      </fo:block>

      <fo:list-block>
        <fo:list-item>
          <fo:list-item-label>
            <fo:block>1)</fo:block>
          </fo:list-item-label>
          <fo:list-item-body start-indent="body-start()">
            <fo:block>Contenu de la liste</fo:block>
          </fo:list-item-body>
        </fo:list-item>
      </fo:list-block>

    </fo:flow>
  </fo:page-sequence>
</fo:root>
```

contenuList.fo

body-start() :
Permet une indentation du texte par rapport à la puce ou n° de la liste



XSL-FO – Contenu - Exemple

personnes.xml - personnes.xsl

Application via un feuille de style xsl-fo : Création d'une impression PDF, permettant d'afficher les personnes et leurs métiers

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<personnes>
  <personne naissance="1912" mort="1954" id="p342">
    <nom>
      <prénom>Alan</prénom>
      <nom_famille>Turing</nom_famille>
    </nom>
    <profession>informaticien</profession>
    <profession>mathématicien</profession>
    <profession>cryptographe</profession>
  </personne>

  <personne naissance="1918" mort="1988" id="p4567">
    <nom>
      <prénom>Richard</prénom>
      <initiale_milieu>R</initiale_milieu>
      <nom_famille>Feynman</nom_famille>
    </nom>
    <profession>physicien</profession>
    <loisir>Jouer des bongos</loisir>
  </personne>
</personnes>
```

Alan Turing

- * **informaticien**
- * **mathématicien**
- * **cryptographe**

Richard Feynman

- * **physicien**
- **Jouer des bongos**



XSL-FO – Contenu - Exemple

personnesListe.xsl

- Feuille de style : template xsl et page fo

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform" xmlns:fo="http://www.w3.org/1999/XSL/Format">

    <!-- Définition du template, match sur l'élément racine -->
    <!-- On définit la page -->
    <xsl:template match="personnes"> ←
        <!-- Définition du template et de la page -->
        <fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
            <fo:layout-master-set>
                <fo:simple-page-master master-name="page" page-height="297mm" page-width="210mm">
                    <fo:region-body margin="20mm" />
                    <fo:region-before margin="20mm" />
                    <fo:region-after extent="20mm"/>
                    <fo:region-start extent="20mm" />
                    <fo:region-end extent="20mm" />
                </fo:simple-page-master>
            </fo:layout-master-set>
        </fo:root>
    </xsl:template>
</xsl:stylesheet>
```

Définition du template et de la page



XSL-FO – Contenu - Exemple

personnesListe.xsl

- Feuille de style : En-tête (region-before)

```
<fo:page-sequence master-reference="page">
  <fo:static-content flow-name="xsl-region-before" >
    <fo:block text-align="center">Les personnes</fo:block>
  </fo:static-content>
```



XSL-FO – Contenu - Exemple

personnesListe.xsl

- Feuille de style : Traitement des informations concernant le prénom et nom_famille de chaque personne

L'affichage se fait dans le corps du document (region-body)

On se positionne sur l'élément personne, on effectue une boucle permettant de traiter chaque personne

```
<fo:flow flow-name="xsl-region-body">
  <xsl:for-each select="//personne">
    <fo:block font-family="verdana" font-size="12pt" font-weight="bold">
      <xsl:apply-templates select=".//nom/prénom" />
      <xsl:text> </xsl:text>
      <xsl:apply-templates select=".//nom/nom_famille" />
    </fo:block>
  </xsl:for-each>
</fo:flow>
```

Un espace blanc

Affichage du prénom et nom_famille



XSL-FO – Contenu - Exemple

personnesListe.xsl

- Feuille de style : Traitement des informations de professions pouvant être multiples

On remonte sur l'élément profession, on utilise une boucle pour chaque profession

```
<xsl:for-each select=".//profession">
  <fo:list-block>
    <fo:list-item>
      <fo:list-item-label>
        <fo:block>*</fo:block>
      </fo:list-item-label>
      <fo:list-item-body start-indent="body-start()">
        <fo:block>
          <xsl:apply-templates select=".." />
        </fo:block>
      </fo:list-item-body>
    </fo:list-item>
  </fo:list-block>
</xsl:for-each>
```

Une liste pour l'affichage



XSL-FO – Contenu - Exemple

personnesListe.xsl

- Feuille de style : Traitement des informations de loisirs, même principe que les professions

```
<xsl:for-each select=".//loisir">
  <fo:list-block>
    <fo:list-item>
      <fo:list-item-label>
        <fo:block>-</fo:block>
      </fo:list-item-label>
      <fo:list-item-body start-indent="body-start()">
        <fo:block>
          <xsl:apply-templates select="." />
        </fo:block>
      </fo:list-item-body>
    </fo:list-item>
  </fo:list-block>
</xsl:for-each>

<xsl:text>-----</xsl:text>
```



XSL-FO – Contenu - Exemple

personnesListe.xsl

- Feuille de style : Fermeture des balises

```
    ...    ...    ...    </fo:block>
    ...    ...    ...    </xsl:for-each>
    ...    ...    ...    </fo:flow>

    |    </fo:page-sequence>
    |    </fo:root>
    |    </xsl:template>

</xsl:stylesheet>
```

Fermeture du bloc principal

**Fermeture de la boucle de traitement de
chaque personne et du flux**

**Fermeture de la sequence de la page, de
l'élément racine root de fo et du template**

Fermeture de la feuille de style

XSL-FO – Contenu - Exercice

lesGaulois_02.xml - gauloisXslFo_01.xsl

- Créer une feuille de style permettant d'afficher la liste des gaulois

```
<?xml version="1.0" encoding="UTF-8"?>
<lesgaulois>
  <gaulois>
    <nom>Obelix</nom>
    <specialite>Porteur de Menhir</specialite>
    <adresse/>
    <image>Obelix.jpg</image>
    <bataille date="0050-09-15" lieu="Laudanum">Babaorum</bataille>
  </gaulois>
  <gaulois>
    <nom>Abraracourcix</nom>
    <specialite>Guerrier</specialite>
    <adresse/>
    <image>Abraracourcix.jpg</image>
    <bataille date="0050-05-31" lieu="Babaorum">Mercenaires</bataille>
    <bataille date="0050-09-15" lieu="Laudanum">Babaorum</bataille>
  </gaulois>
```

The screenshot shows a software interface with a toolbar at the top containing icons for file operations and search. The main area displays a list titled "Les gaulois" on the right side. To the left of the title is a vertical sidebar with icons for document, folder, and other file-related functions. The list itself contains the names of the Gauls from the XML data:

- Obelix
- Abraracourcix
- Agecanonix
- Panoramix
- Cetautomatix
- Asterix
- Alambrix
- Allegorix
- Amerix
- Amnesix
- Analgesix
- Antibiotix
- Aplusbegalix



XSL-FO – Contenu - Correction

lesGaulois_02.xml - gauloisXslFo_01.xsl

- Créer une feuille de style permettant d'afficher la liste des gaulois

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform" xmlns:fo="http://www.w3.org/1999/XSL/Format">

    <xsl:template match="lesgaulois">
        <fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
            <fo:layout-master-set>
                <fo:simple-page-master master-name="page" page-height="297mm" page-width="210mm">
                    <fo:region-body margin="20mm" />
                    <fo:region-before margin="20mm" />
                    <fo:region-after extent="20mm"/>
                    <fo:region-start extent="20mm" />
                    <fo:region-end extent="20mm" />
                </fo:simple-page-master>
            </fo:layout-master-set>

            <fo:page-sequence master-reference="page">
                <fo:static-content flow-name="xsl-region-before" >
                    <fo:block text-align="center">Les gaulois</fo:block>
                </fo:static-content>

                <fo:flow flow-name="xsl-region-body" >
                    <xsl:for-each select="//gaulois">
                        <fo:block font-family="verdana" font-size="12pt" font-weight="bold" >
                            <xsl:apply-templates select=".//nom" />
                        </fo:block>
                    </xsl:for-each>
                </fo:flow>
            </fo:page-sequence>
        </fo:root>
    </xsl:template>
</xsl:stylesheet>
```



XSL-FO – Contenu - Exercice

lesGaulois_02.xml - gauloisXslFo_02.xsl

- A partie de la feuille de style de l'exercice précédent , ramener toutes les informations des gaulois, en utilisant des listes

Les gaulois

Obelix :

- * Adresse :
- * Specialite :Porteur de Mehnir
- * Image :Obelix.jpg
- * Bataille :Babaorum - date : 0050-09-15 - lieu : Laudanum

Abraracourcix :

- * Adresse :
- * Specialite :Guerrier
- * Image :Abraracourcix.jpg
- * Bataille :Mercenaires - date : 0050-05-31 - lieu : Babaorum
- * Bataille :Babaorum - date : 0050-09-15 - lieu : Laudanum

Agecanonix :

- * Adresse :
- * Specialite :Villageois



XSL-FO – Contenu - Correction

lesGaulois_02.xml - gauloisXslFo_02.xsl

- A partie de la feuille de style de l'exercice précédent , ramener toutes les informations des gaulois, en utilisant des listes

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform" xmlns:fo="http://www.w3.org/1999/XSL/Format">

    <xsl:template match="lesgaulois">
        <fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
            <fo:layout-master-set>
                <fo:simple-page-master master-name="page" page-height="297mm" page-width="210mm">
                    <fo:region-body margin="20mm" />
                    <fo:region-before margin="20mm" />
                    <fo:region-after extent="20mm"/>
                    <fo:region-start extent="20mm" />
                    <fo:region-end extent="20mm" />
                </fo:simple-page-master>
            </fo:layout-master-set>

            <fo:page-sequence master-reference="page">
                <fo:static-content flow-name="xsl-region-before" >
                    <fo:block text-align="center">Les gaulois</fo:block>
                </fo:static-content>
            </fo:page-sequence>
        </fo:root>
    </xsl:template>
</xsl:stylesheet>
```

Définition du template et de la page

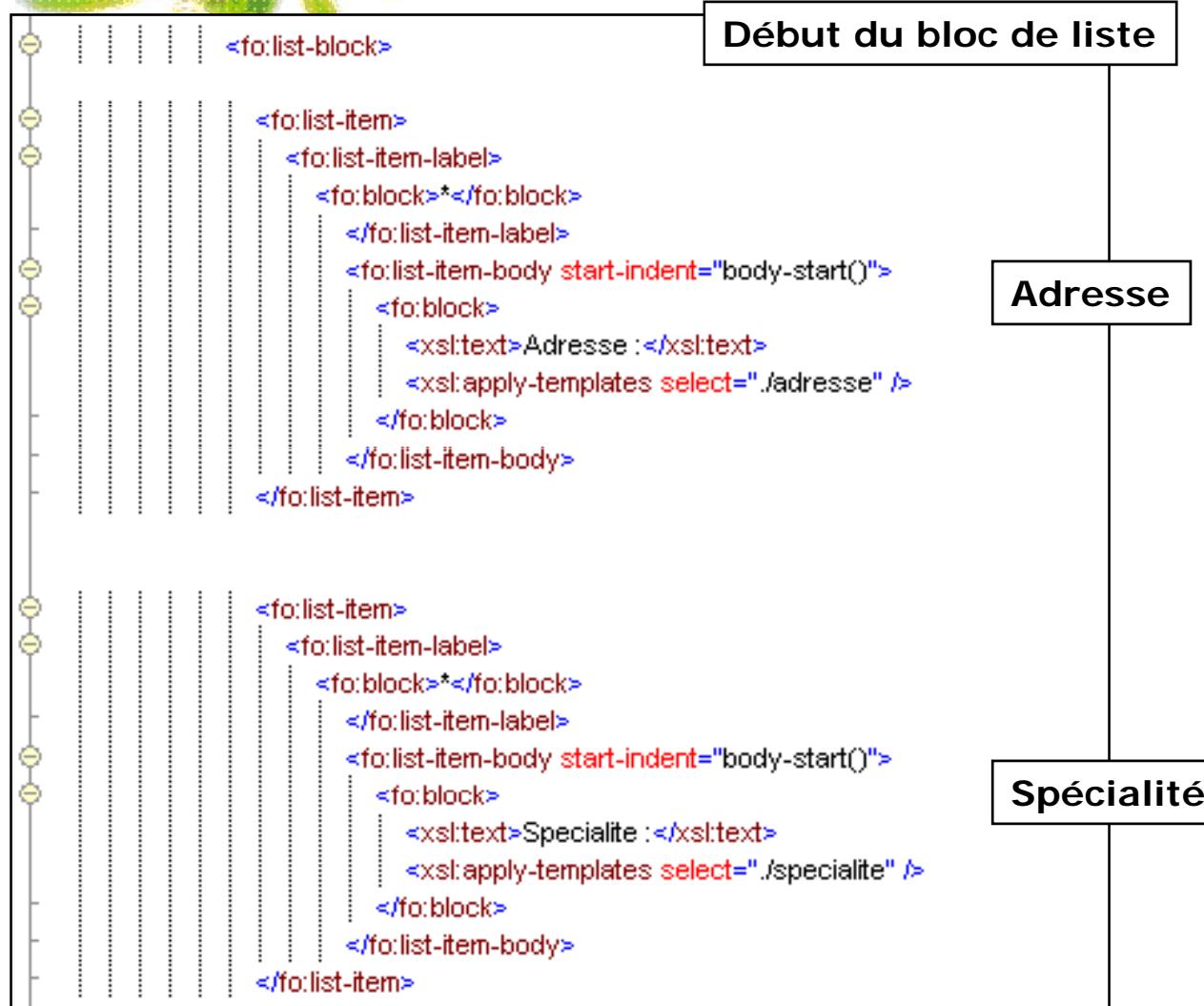
Entête de page

Début de la séquence d'affichage, on effectue une boucle sur chaque gaulois



XSL-FO – Contenu - Correction

lesGaulois_02.xml - gauloisXslFo_02.xsl



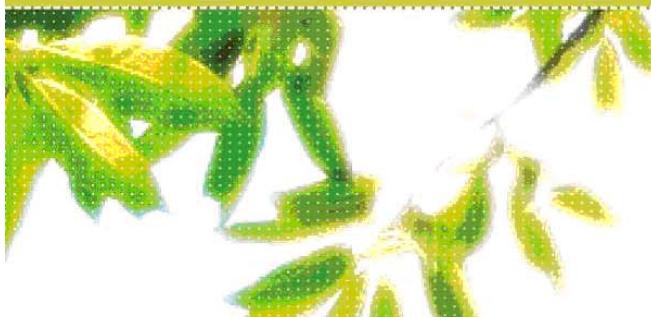


XSL-FO – Contenu - Correction

lesGaulois_02.xml - gauloisXsIFo_02.xsl

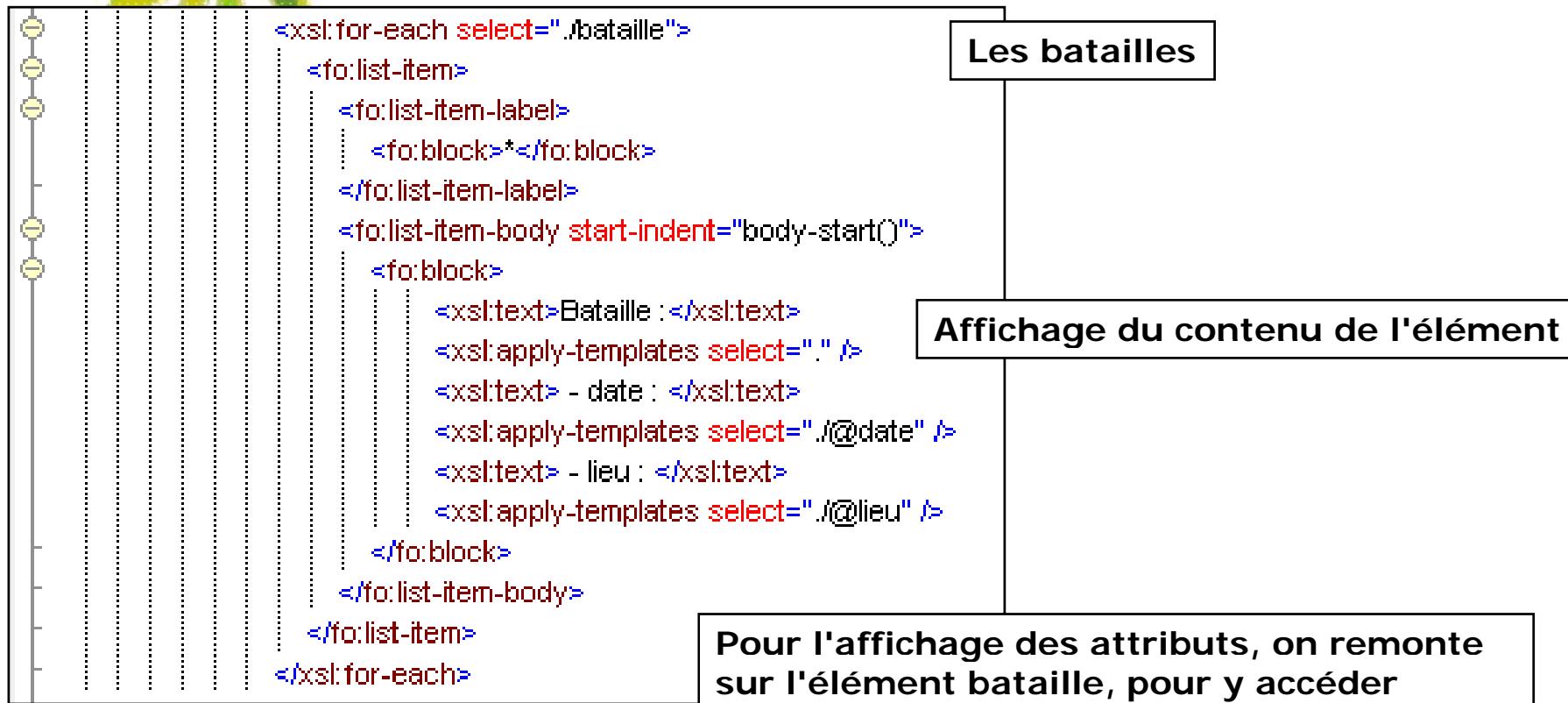
```
<fo:list-item>
  <fo:list-item-label>
    <fo:block>*</fo:block>
  </fo:list-item-label>
  <fo:list-item-body start-indent="body-start()">
    <fo:block>
      <xsl:text>Image :</xsl:text>
      <xsl:apply-templates select=".//image" />
    </fo:block>
  </fo:list-item-body>
</fo:list-item>
```

Image



XSL-FO – Contenu - Correction

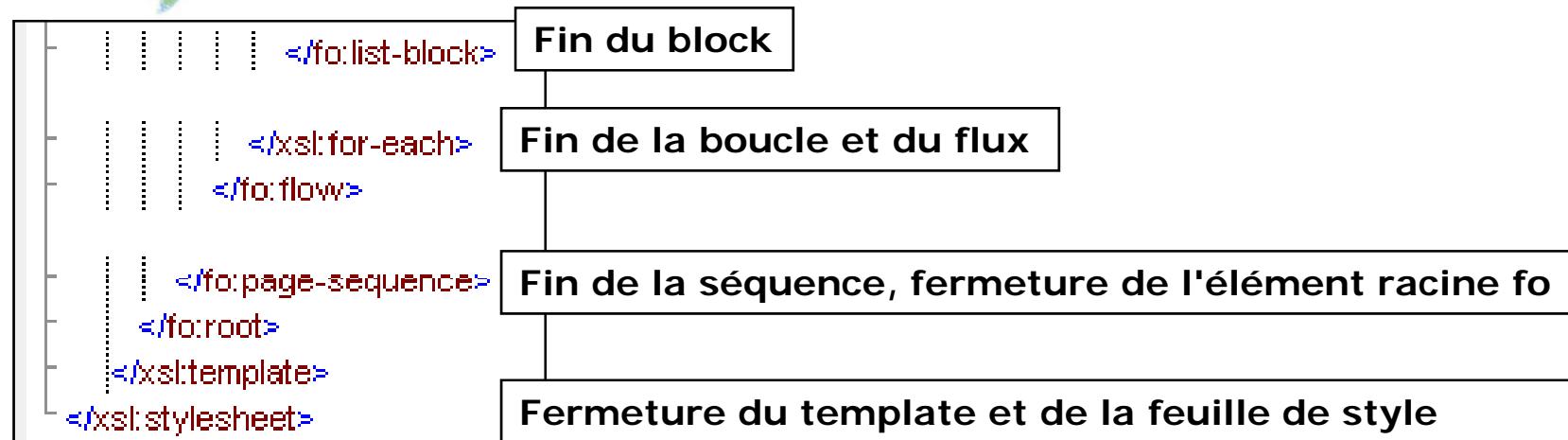
lesGaulois_02.xml - gauloisXslFo_02.xsl





XSL-FO – Contenu - Correction

lesGaulois_02.xml - gauloisXsIFo_02.xsl





XSL-FO – Les tableaux

- **fo:table-and-caption** : permet de définir un tableau et sa légende.
- **fo:table-caption** : définit la légende
- **fo:table** : définit le tableau
- Les dimensions de chaque colonne doivent d'abord être précisées avec **fo:table-column**
- Un tableau comprend trois parties :
 - **fo:tableheader** - entête,
 - **fo:table-body** – Corps,
 - et **fo:table-footer** – Pied



XSL-FO – Les tableaux

- Chaque partie est découpée en lignes avec : **fo:table-row**
- Chaque ligne est découpée en cellules avec : **fo:table-cell**
- L'attribut **table-layout** (de fo:table) définit comment faire le rendu du tableau :
 - **fixed** : la taille des colonnes est définie à partir de tailles précisées
 - **auto** : la taille des colonnes est définie à partir des données présentes dans les colonnes



XSL-FO – Les tableaux - Exemple

personnes.xml - personnesTab.xsl

- Affichage de la liste des personnes dans un tableau

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<personnes>
    <personne naissance="1912" mort="1954" id="p342">
        <nom>
            <prénom>Alan</prénom>
            <nom_famille>Turing</nom_famille>
        </nom>
        <profession>informaticien</profession>
        <profession>mathématicien</profession>
        <profession>cryptographe</profession>
    </personne>

    <personne naissance="1918" mort="1988" id="p4567">
        <nom>
            <prénom>Richard</prénom>
            <initiale_milieu>R</initiale_milieu>
            <nom_famille>Feynman</nom_famille>
        </nom>
        <profession>physicien</profession>
        <loisir>Jouer des bongos</loisir>
    </personne>
</personnes>
```

Nom	Prénom	Profession(s)	Loisir(s)
		* informaticien * mathématicien * cryptographe	*
		* physicien	* Jouer des bongos



XSL-FO – Les tableaux - Exemple

personnes.xml - personnesTab.xsl

- Affichage de la liste des personnes dans un tableau

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform" xmlns:fo="http://www.w3.org/1999/XSL/Format">

    <xsl:template match="personnes">

        <fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
            <fo:layout-master-set>
                <fo:simple-page-master master-name="page" page-height="297mm" page-width="210mm">
                    <fo:region-body margin="20mm" />
                    <fo:region-before margin="20mm" />
                    <fo:region-after extent="20mm"/>
                    <fo:region-start extent="20mm" />
                    <fo:region-end extent="20mm" />
                </fo:simple-page-master>
            </fo:layout-master-set>

            <fo:page-sequence master-reference="page">
                <fo:static-content flow-name="xsl-region-before">
                    <fo:block text-align="center">Les personnes</fo:block>
                </fo:static-content>
            </fo:page-sequence>
        </fo:root>
    </xsl:template>
</xsl:stylesheet>
```

Définition du template et de la page

Entête de page



XSL-FO – Les tableaux - Exemple

personnes.xml - personnesTab.xsl

- Affichage de la liste des personnes dans un tableau

```
<fo:flow flow-name="xsl-region-body">
  <fo:table border="1pt solid blue">
    <fo:table-column column-width="30mm"/>
    <fo:table-column column-width="30mm"/>
    <fo:table-column column-width="50mm"/>
    <fo:table-column column-width="50mm"/>

    <fo:table-header background-color="lightgray">
      <fo:table-row>
        <fo:table-cell>
          <fo:block font-weight="bold">Nom</fo:block>
        </fo:table-cell>
        <fo:table-cell>
          <fo:block font-weight="bold">Prénom</fo:block>
        </fo:table-cell>
        <fo:table-cell>
          <fo:block font-weight="bold">Profession(s)</fo:block>
        </fo:table-cell>
        <fo:table-cell>
          <fo:block font-weight="bold">Loisir(s)</fo:block>
        </fo:table-cell>
      </fo:table-row>
    </fo:table-header>
```

Définition du tableau et des largeurs de colonnes

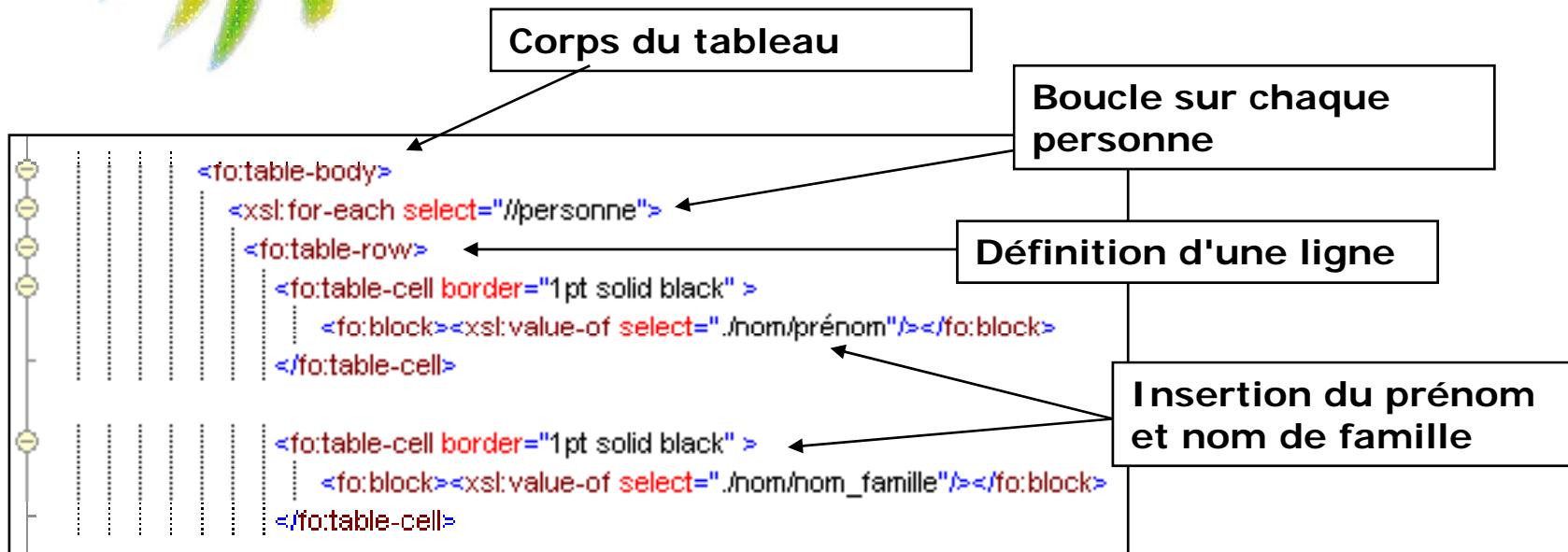
Entête du tableau



XSL-FO – Les tableaux - Exemple

personnes.xml - personnesTab.xsl

- Affichage de la liste des personnes dans un tableau





XSL-FO – Les tableaux - Exemple

personnes.xml - personnesTab.xsl

- Affichage de la liste des personnes dans un tableau

Boucle sur les professions

```
<fo:table-cell border="1pt solid black">
  <xsl:for-each select=".//profession">
    <fo:list-block>
      <fo:list-item>
        <fo:list-item-label>
          <fo:block>*</fo:block>
        </fo:list-item-label>
        <fo:list-item-body start-indent="body-start()">
          <fo:block>
            <xsl:apply-templates select="." />
          </fo:block>
        </fo:list-item-body>
      </fo:list-item>
    </fo:list-block>
  </xsl:for-each>
</fo:table-cell>
```



XSL-FO – Les tableaux - Exemple

personnes.xml - personnesTab.xsl

- Affichage de la liste des personnes dans un tableau

Boucle sur les loisirs

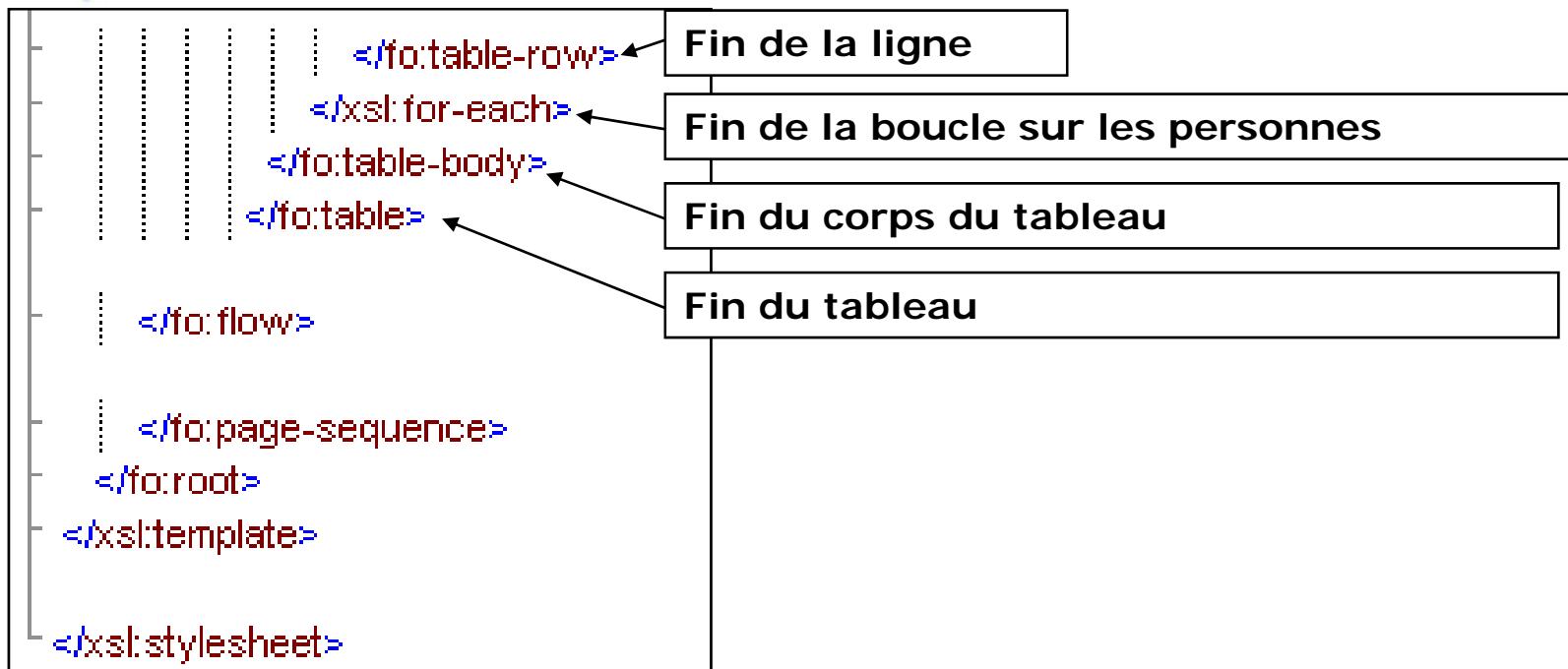
```
<fo:table-cell border="1pt solid black" >
<xsl:for-each select=".">
  <fo:list-block>
    <fo:list-item>
      <fo:list-item-label>
        <fo:block>*</fo:block>
      </fo:list-item-label>
      <fo:list-item-body start-indent="body-start()">
        <fo:block>
          <xsl:apply-templates select="loisir" />
        </fo:block>
      </fo:list-item-body>
    </fo:list-item>
  </fo:list-block>
</xsl:for-each>
</fo:table-cell>
```



XSL-FO – Les tableaux - Exemple

personnes.xml - personnesTab.xsl

- Affichage de la liste des personnes dans un tableau



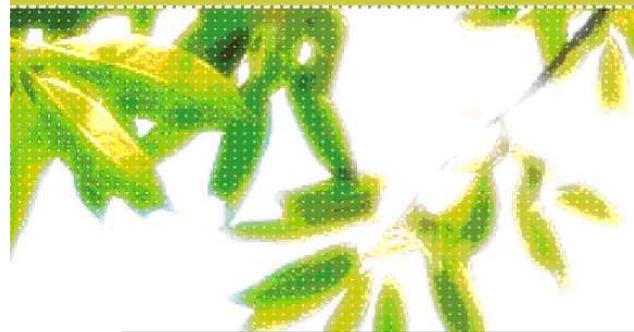


XSL-FO – Les tableaux - Exercice

lesGaulois_02.xml - gauloisXslFo_03.xsl

Créer une feuille de style permettant d'afficher les gaulois dans un tableau, utiliser un format paysage

Nom	Adresse	Specialite	Image	Participation aux batailles
Obelix		Porteur de Mehnir	Obelix.jpg	* Babaorum - date : 0050-09-15 - lieu : Laudanum
Abraracourcix		Guerrier	Abraracourcix.jpg	* Mercenaires - date : 0050-05-31 - lieu : Babaorum Babaorum - date : 0050-09-15 - lieu : Laudanum
Agecanonix		Villageois	Agecanonix.jpg	* Legion XII - date : 0050-10-08 - lieu : Grottes



XSL-FO – Les tableaux - Correction

lesGaulois_02.xml - gauloisXslFo_03.xsl

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform" xmlns:fo="http://www.w3.org/1999/XSL/Format">

<xsl:template match="lesgaulois">

    <fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
        <fo:layout-master-set>
            <fo:simple-page-master master-name="page" page-height="210mm" page-width="297mm">
                <fo:region-body margin="20mm" />
                <fo:region-before margin="20mm" />
                <fo:region-after extent="20mm" />
                <fo:region-start extent="20mm" />
                <fo:region-end extent="20mm" />
            </fo:simple-page-master>
        </fo:layout-master-set>

        <fo:page-sequence master-reference="page">
            <fo:static-content flow-name="xsl-region-before" >
                <fo:block text-align="center">Les gaulois</fo:block>
            </fo:static-content>
        </fo:page-sequence>
    </fo:root>
</xsl:stylesheet>
```

Définition du template et de la page

Entête de page



XSL-FO – Les tableaux - Correction

lesGaulois_02.xml - gauloisXslFo_03.xsl

```
<fo:flow flow-name="xsl-region-body">
  <fo:table border="1pt solid blue" table-layout="auto">
    <fo:table-header background-color="lightgray" font-family="verdana" font-size="12pt">
      <fo:table-row>
        <fo:table-cell>
          <fo:block font-weight="bold">Nom</fo:block>
        </fo:table-cell>
        <fo:table-cell>
          <fo:block font-weight="bold">Adresse</fo:block>
        </fo:table-cell>
        <fo:table-cell>
          <fo:block font-weight="bold">Specialite</fo:block>
        </fo:table-cell>
        <fo:table-cell>
          <fo:block font-weight="bold">Image</fo:block>
        </fo:table-cell>
        <fo:table-cell>
          <fo:block font-weight="bold">Participation aux batailles</fo:block>
        </fo:table-cell>
      </fo:table-row>
    </fo:table-header>
```

Définition du tableau

Entête du tableau

XSL-FO – Les tableaux - Correction

```
<fo:table-body font-family="verdana" font-size="10pt">
  <xsl:for-each select="//gaulois">
    <fo:table-row>
      <fo:table-cell border="1pt solid black" >
        | <fo:block><xsl:value-of select=".//nom"/></fo:block>
      </fo:table-cell>
      <fo:table-cell border="1pt solid black" >
        | <fo:block><xsl:value-of select=".//adresse"/></fo:block>
      </fo:table-cell>
      <fo:table-cell border="1pt solid black" >
        | <fo:block><xsl:value-of select=".//specialite"/></fo:block>
      </fo:table-cell>
      <fo:table-cell border="1pt solid black" >
        | <fo:block><xsl:value-of select=".//image"/></fo:block>
      </fo:table-cell>
      <fo:table-cell border="1pt solid black" >
        | <xsl:for-each select=".//bataille">
          <fo:list-block>
            <fo:list-item>
              <fo:list-item-label>
                <fo:block>*</fo:block>
              </fo:list-item-label>
              <fo:list-item-body start-indent="body-start()">
                <fo:block>
                  <xsl:for-each select="bataille">
                    <fo:block>
                      | <xsl:value-of select=".//date"/>
                      | <xsl:text> - date : </xsl:text><xsl:value-of select="@date"/>
                      | <xsl:text> - lieu : </xsl:text><xsl:value-of select="@lieu"/>
                    </fo:block>
                  </xsl:for-each>
                </fo:block>
              </fo:list-item-body>
            </fo:list-item>
          </fo:list-block>
        </xsl:for-each>
      </fo:table-cell>
    </u:table-ruvw>
  </xsl:for-each>
</fo:table-body>
```

Corps du tableau

lesGaulois_02.xml –
gauloisXslFo_03.xsl

Affichage des attributs



XSL-FO – Les images

- Il est possible d'insérer des images à l'aide de :
 - **fo:external-graphic**

```
<fo:external-graphic  
    src="/Users/reynier/images/logo.jpg"  
    width="99px" height="109px"/>
```

- On peut utiliser une variable xsl, facilitant l'utilisation des accès aux fichiers images
- Exemple :

```
<xsl:for-each select="//gaulois">  
    <xsl:variable name="fichierImage">  
        <xsl:value-of select=".//image"/>  
    </xsl:variable>  
    ...  
    <fo:block>  
        <fo:external-graphic src="imagesGaulois/{$fichierImage}" />  
    </fo:block>
```



XSL-FO – Les images - Exercice

lesGaulois_02.xml - gauloisXslFo_04.xsl

Utiliser la feuille de style précédente pour afficher les images des gaulois

Les gaulois



Obelix :

- * Adresse :
- * Specialite : Porteur de Mehnir
- * Image : Obelix.jpg
- * Bataille : Babaorum - date : 0050-09-15 -
lieu : Laudanum



Abraracourcix :

- * Adresse :
- * Specialite : Guerrier
- * Image : Abraracourcix.jpg
- * Bataille : Mercenaires - date : 0050-05-31 -
lieu : Babaorum
- * Bataille : Babaorum - date : 0050-09-15 -
lieu : Laudanum



XSL-FO – Les images - Correction

lesGaulois_02.xml - gauloisXslFo_04.xsl

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform" xmlns:fo="http://www.w3.org/1999/XSL/Format">
<xsl:template match="lesgaulois">
<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
<fo:layout-master-set>
<fo:simple-page-master master-name="page" page-height="210mm" page-width="297mm">
<fo:region-body margin="20mm" />
<fo:region-before margin="20mm" />
<fo:region-after extent="20mm"/>
<fo:region-start extent="20mm" />
<fo:region-end extent="20mm" />
</fo:simple-page-master>
</fo:layout-master-set>

<fo:page-sequence master-reference="page">
<fo:static-content flow-name="xsl-region-before" >
<fo:block text-align="center" font-family="Verdana" font-size="24pt" font-weight="bold" color="blue">Les gaulois</fo:block>
</fo:static-content>

<fo:flow flow-name="xsl-region-body" >
<fo:table>
<fo:table-column column-width="50mm"/>
<fo:table-column column-width="100mm"/>
```



XSL-FO – Les images - Correction

lesGaulois_02.xml - gauloisXsIFo_04.xsl

```
<fo:table-body font-family="verdana" font-size="10pt">
  <xsl:for-each select="//gaulois">
    <xsl:variable name="fichierImage">
      <xsl:value-of select=".//image"/>
    </xsl:variable>

    <fo:table-row font-family="Verdana">
      <fo:table-cell>
        <fo:block>
          <fo:external-graphic
            src="imagesGaulois/(${fichierImage})" />
        </fo:block>
      </fo:table-cell>
```



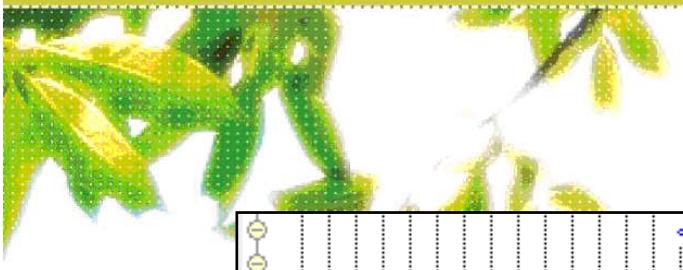
XSL-FO – Les images - Correction

lesGaulois_02.xml - gauloisXslFo_04.xsl

```
<fo:table-cell>
  <fo:block>
    <fo:table>
      <fo:table-column column-width="100mm"/>
      <fo:table-body font-family="verdana" font-size="10pt">

        <fo:table-row background-color="lightgray" font-family="Verdana">
          <fo:table-cell margin="10pt" >
            <fo:block font-family="verdana" font-size="12pt" font-weight="bold" >
              <xsl:apply-templates select=".//nom" />
              <xsl:text> : </xsl:text>
            </fo:block>

          <fo:list-block>
            <fo:list-item>
              <fo:list-item-label>
                <fo:block>*</fo:block>
              </fo:list-item-label>
              <fo:list-item-body start-indent="body-start()">
                <fo:block>
                  <xsl:text>Adresse : </xsl:text>
                  <xsl:apply-templates select=".//adresse" />
                </fo:block>
              </fo:list-item-body>
            </fo:list-item>
          </fo:list-block>
        </fo:table-cell>
      </fo:table-body>
    </fo:table>
  </fo:block>
</fo:table-cell>
```



XSL-FO – Les images - Correction

lesGaulois_02.xml - gauloisXslFo_04.xsl

```
<fo:list-item>
  <fo:list-item-label>
    <fo:block>*</fo:block>
  </fo:list-item-label>
  <fo:list-item-body start-indent="body-start()">
    <fo:block>
      <xsl:text>Specialité : </xsl:text>
      <xsl:apply-templates select=".//specialite" />
    </fo:block>
  </fo:list-item-body>
</fo:list-item>

<xsl:for-each select=".//bataille">
  <fo:list-item>
    <fo:list-item-label>
      <fo:block>*</fo:block>
    </fo:list-item-label>
    <fo:list-item-body start-indent="body-start()">
      <fo:block>
        <xsl:text>Bataille : </xsl:text>
        <xsl:apply-templates select=".//date" />
        <xsl:text> - date : </xsl:text>
        <xsl:apply-templates select=".//@date" />
        <xsl:text> - lieu : </xsl:text>
        <xsl:apply-templates select=".//@lieu" />
      </fo:block>
    </fo:list-item-body>
  </fo:list-item>
</xsl:for-each>

  </fo:list-block>
</fo:table-cell>
</fo:table-row>
```



XSL-FO – Les images - Correction

lesGaulois_02.xml - gauloisXsIFo_04.xsl

```
</fo:table-body>
</fo:table>
</fo:block>
</fo:table-cell>
</fo:table-row>
</xsl:for-each>
</fo:table-body>
</fo:table>

</fo:flow>

</fo:page-sequence>
</fo:root>
</xsl:template>

</xsl:stylesheet>
```



XSL-FO – Les liens

- Des liens entre documents peuvent être insérés grâce à
 - **fo:basic-link**
- Les liens peuvent être internes ou externes suivant les attributs :
 - **external-destination** :
 - lien externe : vers un autre document L'ancre de destination est positionnée par l'attribut '**id**' d'un élément fo:block
 - **internal-destination** :
 - lien interne : vers une partie du même document.
- Les propriétés graphiques de ces liens peuvent être spécifiées dans la balise **fo:basic-link**



XSL-FO – Les liens

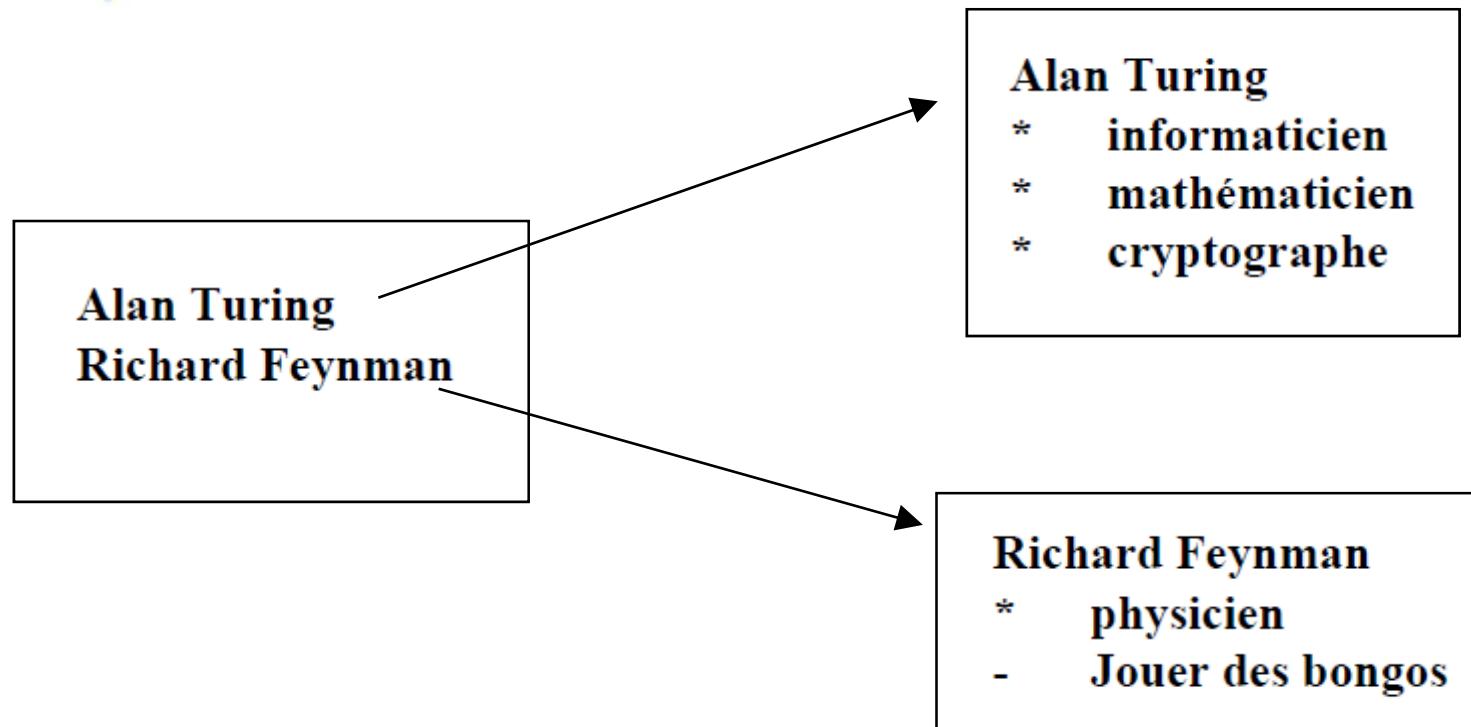
- Création d'un identifiant utilisé par les liens :
 - Demande à xsl de créer un identifiant et de l'utiliser pour créer les débranchements des liens aux ancrées :
- **<fo:basic-link internal-destination="{generate-id(.)}">**
 - Permet de générer un identifiant vers le lien
- **<fo:block id="{generate-id(.)}">** :
 - Block de débranchement du lien portant l'identifiant généré
- **<fo:block break-after="page"/>** : permet d'effectuer un saut de page



XSL-FO – Les liens - Exemple

personnes.xml - personnesLiens.xsl

- Créer la liste des personnes permettant un lien sur le descriptif de la personne





XSL-FO – Les liens - Exemple

personnes.xml - personnesLiens.xsl

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform" xmlns:fo="http://www.w3.org/1999/XSL/Format">

<xsl:template match="personnes">

    <fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
        <fo:layout-master-set>
            <fo:simple-page-master master-name="page" page-height="297mm" page-width="210mm">
                <fo:region-body margin="20mm" />
                <fo:region-before margin="20mm" />
                <fo:region-after extent="20mm"/>
                <fo:region-start extent="20mm" />
                <fo:region-end extent="20mm" />
            </fo:simple-page-master>
        </fo:layout-master-set>

        <fo:page-sequence master-reference="page">
            <fo:static-content flow-name="xsl-region-before">
                <fo:block text-align="center">Les personnes</fo:block>
            </fo:static-content>

            <fo:flow flow-name="xsl-region-body">
                <xsl:for-each select="//personne">
                    <fo:block font-family="verdana" font-size="12pt" font-weight="bold" >
                        <fo:basic-link internal-destination="(generate-id(.))">
                            <xsl:apply-templates select=".//nom/prénom" />
                            <xsl:text> </xsl:text>
                            <xsl:apply-templates select=".//nom/nom_famille" />
                        </fo:basic-link>
                    </fo:block>
                </xsl:for-each>
            </fo:flow>
        </fo:page-sequence>
    </fo:root>
</xsl:stylesheet>
```



XSL-FO – Les liens - Exemple

personnes.xml - personnesLiens.xsl

```
<xsl:for-each select="/personne">
  <fo:block font-family="verdana" font-size="12pt" font-weight="bold" id="(generate-id(.))">
    <xsl:apply-templates select=".//nom/prénom" />
    <xsl:text> </xsl:text>
    <xsl:apply-templates select=".//nom/nom_famille" />

    <xsl:for-each select=".//profession">
      <fo:list-block>
        <fo:list-item>
          <fo:list-item-label>
            <fo:block>*</fo:block>
            <fo:list-item-label>
              <fo:list-item-body start-indent="body-start()">
                <fo:block>
                  <xsl:apply-templates select=".." />
                </fo:block>
              </fo:list-item-body>
            </fo:list-item-label>
            <fo:list-item-body start-indent="body-start()">
              <fo:block>
                <xsl:apply-templates select=".." />
              </fo:block>
            </fo:list-item-body>
          </fo:list-item>
        </fo:list-block>
      </xsl:for-each>
      <xsl:for-each select=".//loisir">
        <fo:list-block>
          <fo:list-item>
            <fo:list-item-label>
              <fo:block>-</fo:block>
              <fo:list-item-label>
                <fo:list-item-body start-indent="body-start()">
                  <fo:block>
                    <xsl:apply-templates select=".." />
                  </fo:block>
                </fo:list-item-body>
              </fo:list-item-label>
            </fo:list-item>
          </fo:list-block>
        </xsl:for-each>
      </xsl:for-each>
    </xsl:for-each>
  </fo:block>
</xsl:for-each>
```



XSL-FO – Les liens - Exemple

personnes.xml - personnesLiens.xsl

```
</fo:block>
<fo:block break-after="page"/>
</xsl:for-each>
</fo:flow>

</fo:page-sequence>
</fo:root>
</xsl:template>

</xsl:stylesheet>
```



XSL-FO – Les signets / bookmark

- Des liens entre documents peuvent être insérés grâce à
 - **fo:bookmark-tree**
- **fo:bookmark-title** : permet de spécifier un libellé
- **fo:bookmark** : permet de définir la destination
 - Utilise l'attribut **internal-destination**, pour spécifier la destination



XSL-FO – Les signets / bookmark

personnes.xml - personnesSignets.xsl

- Créer un système de signets pour les personnes

The screenshot shows a software interface with a dark-themed sidebar on the left labeled 'Signets'. The sidebar contains icons for file operations (New, Open, Save, Delete, Copy, Paste) and a list of two entries: 'Alan Turing' and 'Richard Feynman', each preceded by a small icon. To the right of the sidebar is a large white content area containing the names 'Alan Turing' and 'Richard Feynman' in a bold, dark blue font.

Alan Turing
Richard Feynman



XSL-FO – Les signets / bookmark

personnes.xml - personnesSignets.xsl

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform" xmlns:fo="http://www.w3.org/1999/XSL/Format">

<xsl:template match="personnes">

    <fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
        <fo:layout-master-set>
            <fo:simple-page-master master-name="page" page-height="297mm" page-width="210mm">
                <fo:region-body margin="20mm" />
                <fo:region-before margin="20mm" />
                <fo:region-after extent="20mm" />
                <fo:region-start extent="20mm" />
                <fo:region-end extent="20mm" />
            </fo:simple-page-master>
        </fo:layout-master-set>

        <fo:bookmark-tree>
            <xsl:for-each select="//personne">
                <fo:bookmark internal-destination="{generate-id(.)}">
                    <fo:bookmark-title>
                        <xsl:apply-templates select=".//nom/prénom" />
                        <xsl:text> </xsl:text>
                        <xsl:apply-templates select=".//nom/nom_famille" />
                    </fo:bookmark-title>
                </fo:bookmark>
            </xsl:for-each>
        </fo:bookmark-tree>

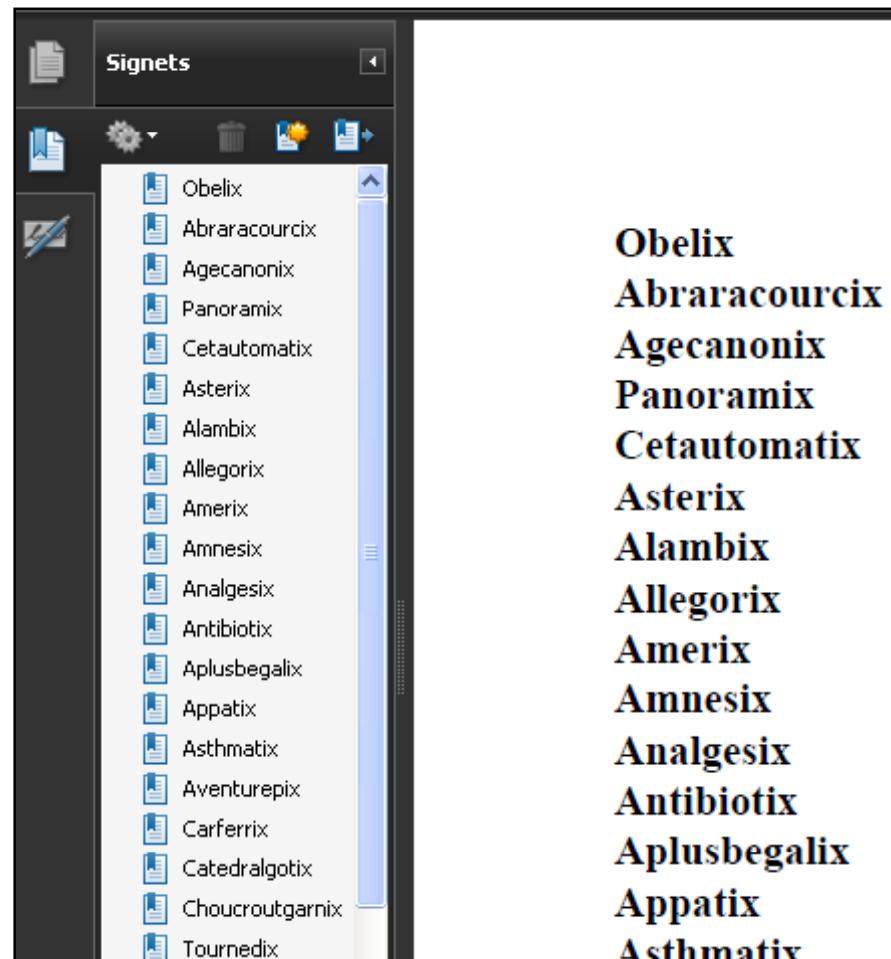
        <fo:page-sequence master-reference="page">
            <fo:static-content>
                <fo:flow>
                    <fo:page-number>Page </fo:page-number>
                </fo:flow>
            </fo:static-content>
        </fo:page-sequence>
    </fo:root>
</xsl:stylesheet>
```



XSL-FO – Les liens - Exercice

lesGaulois_02.xml - gauloisXslFo_05.xsl

A partir de l'exercice précédent créer une table des matières et des liens permettant d'aller sur le gaulois concerné, ainsi qu'un système de signets





XSL-FO – Les liens - Correction

lesGaulois_02.xml - gauloisXslFo_05.xsl

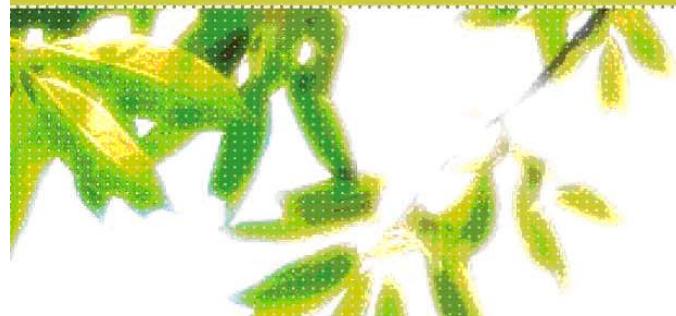
```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform" xmlns:fo="http://www.w3.org/1999/XSL/Format">

<xsl:template match="lesgaulois">

    <fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">

        <fo:layout-master-set>
            <fo:simple-page-master master-name="page" page-height="297mm" page-width="210mm">
                <fo:region-body margin="20mm" />
                <fo:region-before margin="20mm" />
                <fo:region-after extent="20mm"/>
                <fo:region-start extent="20mm" />
                <fo:region-end extent="20mm" />
            </fo:simple-page-master>
        </fo:layout-master-set>

        <fo:bookmark-tree>
            <fo:bookmark internal-destination="{generate-id(.)}" starting-state="show">
                <fo:bookmark-title>Les Gaulois</fo:bookmark-title>
                <xsl:for-each select="//gaulois">
                    <fo:bookmark internal-destination="{generate-id(.)}">
                        <fo:bookmark-title>
                            <xsl:apply-templates select=".//nom" />
                        </fo:bookmark-title>
                    </fo:bookmark>
                </xsl:for-each>
            </fo:bookmark>
        </fo:bookmark-tree>
    </fo:root>
</xsl:template>
</xsl:stylesheet>
```



XSL-FO – Les liens - Correction

lesGaulois_02.xml - gauloisXslFo_05.xsl

```
<fo:page-sequence master-reference="page">
  <fo:static-content flow-name="xsl-region-before" >
    |  <fo:block text-align="center" font-family="Verdana" font-size="24pt" font-weight="bold" color="blue">Les gaulois</fo:block>
  </fo:static-content>

  <fo:flow flow-name="xsl-region-body" >
    <xsl:for-each select="//gaulois">
      <fo:block font-family="verdana" font-size="12pt" font-weight="bold" >
        <fo:basic-link internal-destination="{generate-id(.)}">
          |  <xsl:apply-templates select=".//nom" />
        </fo:basic-link>
      </fo:block>
    </xsl:for-each>

    <fo:block break-after="page"/>

    <fo:table>
      <fo:table-column column-width="50mm"/>
      <fo:table-column column-width="100mm"/>

      <fo:table-body font-family="verdana" font-size="10pt">
        <xsl:for-each select="//gaulois">
```



XSL-FO – Pagination - notes

- On peut obtenir le numéro de la page : `<fo:page-number/>`
- On peut faire des notes de bas de page :
- Exemple :

```
<fo:footnote>
    <fo:inline>(1)</fo:inline>
        <fo:footnote-body>
            (1) En effet...
        </fo:footnote-body>
    </fo:footnote>
```



XSL-FO – Mise en page complexe

- **fo:page-sequence-master** permet de définir une pagination pour un ensemble de pages
- Cette instruction peut contenir :
 - des définitions de pages simples avec :
 - **single-page-master-reference**
 - des définitions de présentations portant chacune sur un ensemble de pages avec :
 - **repeatable-page-master-reference**
 - **repeatable-page-master-alternatives**



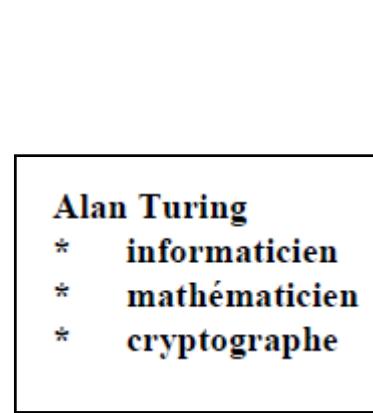
XSL-FO – Mise en page complexe - Exemple

personnes.xml - personnesLivret.xsl

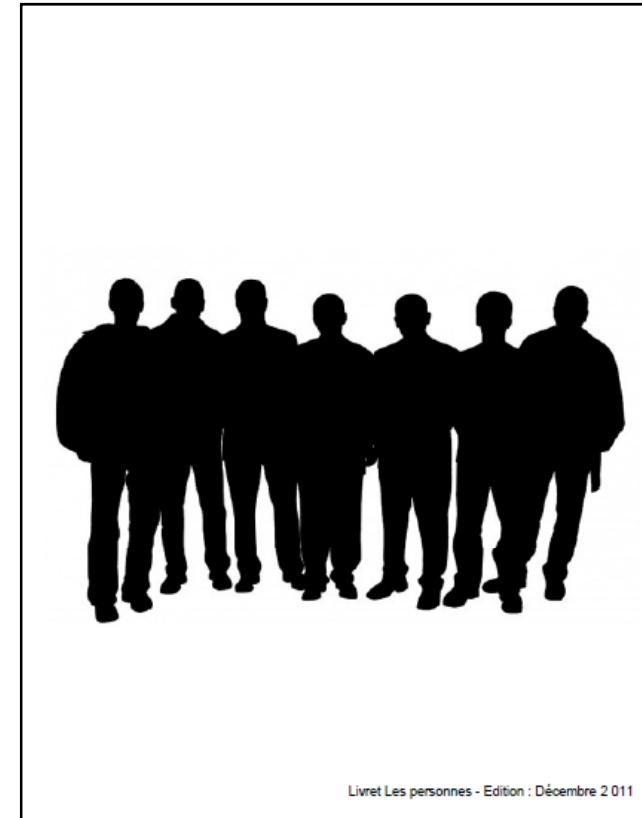
Création d'un livret sur les personnes



couverture



contenu



dos



XSL-FO – Mise en page complexe - Exemple

personnes.xml - personnesLivret.xsl

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform" xmlns:fo="http://www.w3.org/1999/XSL/Format">

<xsl:template match="personnes">

    <fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
        <fo:layout-master-set>
            <fo:simple-page-master master-name="couverture" page-height="297mm" page-width="210mm" >
                <fo:region-body margin="10mm" background-image="imagesLivre/personnes.jpg"/>
                <fo:region-before margin="10mm" />
                <fo:region-after extent="10mm"/>
                <fo:region-start extent="10mm" />
                <fo:region-end extent="10mm" />
            </fo:simple-page-master>

            <fo:simple-page-master master-name="page" page-height="297mm" page-width="210mm">
                <fo:region-body margin="10mm" />
                <fo:region-before margin="10mm" />
                <fo:region-after extent="10mm"/>
                <fo:region-start extent="10mm" />
                <fo:region-end extent="10mm" />
            </fo:simple-page-master>

            <fo:simple-page-master master-name="dos" page-height="297mm" page-width="210mm">
                <fo:region-body margin="10mm" background-image="imagesLivre/personnes.jpg"/>
                <fo:region-before margin="10mm" />
                <fo:region-after extent="10mm"/>
                <fo:region-start extent="10mm" />
                <fo:region-end extent="10mm" />
            </fo:simple-page-master>
        </fo:layout-master-set>
    </fo:root>
</xsl:template>
</xsl:stylesheet>
```



XSL-FO – Mise en page complexe - Exemple

personnes.xml - personnesLivret.xsl

```
<fo:bookmark-tree>
  <xsl:for-each select="//personne">
    <fo:bookmark internal-destination="(generate-id(.))">
      <fo:bookmark-title>
        <xsl:apply-templates select=".//nom/prénom" />
        <xsl:text> </xsl:text>
        <xsl:apply-templates select=".//nom/nom_famille" />
      </fo:bookmark-title>
    </fo:bookmark>
  </xsl:for-each>
</fo:bookmark-tree>
```

XSL-FO – Mise en page complexe - Exemple

personnes.xml - personnesLivret.xsl

```
<fo:page-sequence master-reference="couverture">
  <fo:flow flow-name="xsl-region-body">
    <fo:block
      font-family="Helvetica"
      font-size="28pt"
      color="#000000"
      margin-right="20mm"
      margin-top="20mm"
      text-align="end"
      space-after="220mm"
    >
      Livret Les personnes
    </fo:block>
    <fo:block
      font-family="Helvetica"
      font-size="18pt"
      text-align="end"
      color="#000000"
      margin-right="5mm"
    >
      Copyright formation XML - JM Fino
    </fo:block>
    <fo:block
      font-size="18pt"
      text-align="end"
      color="#000000"
      margin-right="5mm"
    >
      Formation E.N.S.
    </fo:block>
  </fo:flow>
</fo:page-sequence>
```



XSL-FO – Mise en page complexe - Exemple

personnes.xml - personnesLivret.xsl

```
<fo:page-sequence master-reference="page">
  <fo:static-content flow-name="xsl-region-before" >
    <fo:block text-align="center">Les personnes</fo:block>
  </fo:static-content>

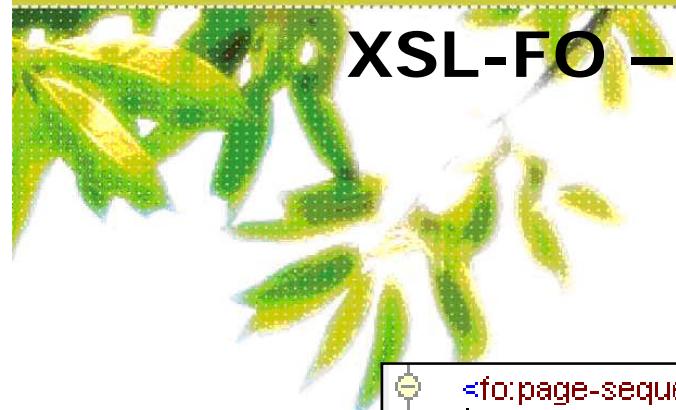
  <fo:flow flow-name="xsl-region-body" >
    <xsl:for-each select="//personne">
      <fo:block font-family="verdana" font-size="12pt" font-weight="bold" >
        <fo:basic-link internal-destination="(generate-id(.))" >
          <xsl:apply-templates select=".//nom/prénom" />
          <xsl:text> </xsl:text>
          <xsl:apply-templates select=".//nom/nom_famille" />
        </fo:basic-link>
      </fo:block>
    </xsl:for-each>
  </fo:flow>
</fo:page-sequence>
```



XSL-FO – Mise en page complexe - Exemple

personnes.xml - personnesLivret.xsl

```
<xsl:for-each select="//personne">
  <fo:block font-family="verdana" font-size="12pt" font-weight="bold" id="(generate-id(.))">
    <xsl:apply-templates select=".//nom/prénom" />
    <xsl:text> </xsl:text>
    <xsl:apply-templates select=".//nom/nom_famille" />
    <xsl:for-each select=".//profession">
      <fo:list-block>
        <fo:list-item>
          <fo:list-item-label><fo:block>*</fo:block></fo:list-item-label>
          <fo:list-item-body start-indent="body-start()">
            <fo:block><xsl:apply-templates select=". /></fo:block>
          </fo:list-item-body>
        </fo:list-item>
      </fo:list-block>
    </xsl:for-each>
    <xsl:for-each select=".//loisir">
      <fo:list-block>
        <fo:list-item>
          <fo:list-item-label><fo:block>-</fo:block></fo:list-item-label>
          <fo:list-item-body start-indent="body-start()">
            <fo:block><xsl:apply-templates select=". /></fo:block>
          </fo:list-item-body>
        </fo:list-item>
      </fo:list-block>
    </xsl:for-each>
  </fo:block>
  <fo:block break-after="page"/>
</xsl:for-each>
</fo:flow>
</fo:page-sequence>
```



XSL-FO – Mise en page complexe - Exemple

personnes.xml - personnesLivret.xsl

```
<fo:page-sequence master-reference="dos">
  <fo:flow flow-name="xsl-region-body">
    <fo:block
      font-family="Helvetica"
      font-size="12pt"
      color="#000000"
      margin-right="2mm"
      margin-top="250mm"
      text-align="end">
      >
      Livret Les personnes - Edition : Décembre 2 011
    </fo:block>
  </fo:flow>
</fo:page-sequence>

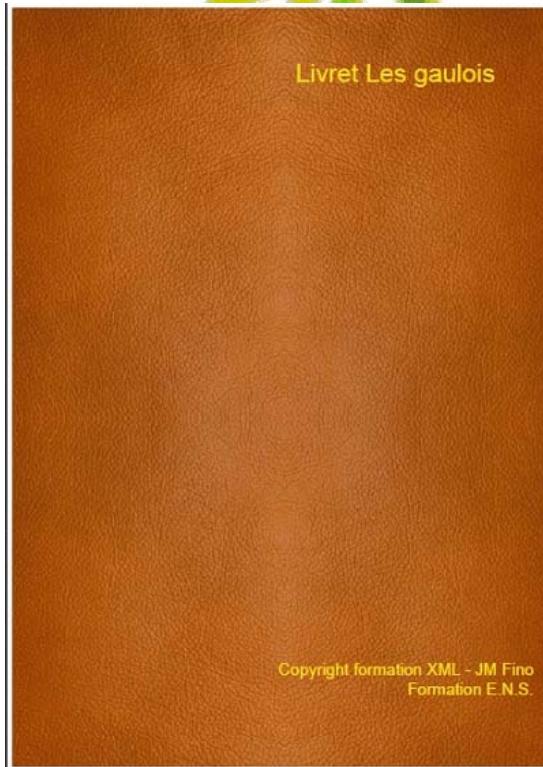
</fo:root>
</xsl:template>
</xsl:stylesheet>
```



XSL-FO – Mise en page complexe

lesGaulois_02.xml - gauloisXslFo_06.xsl

Créer un livret sur les gaulois



couverture

Les gaulois

Panoramix :

- Adresse : Rocheville • Brûlée
- image : Panoramix.jpg
- Babille : Boëdechwingum • date : 0050-12-02 • lieu : En mer

Cetautomax :

- Adresse : 2987A • Artisan
- image : Cetautomax.jpg
- Babille : Ladaunum • date : 0050-12-05 • lieu : Babacrum

Asterix :

- Adresse : code carriens
- image : Asterix.jpg
- Babille : Mercenarie • date : 0050-09-15 • lieu : Ladaunum
- Babille : Babacrum • date : 0050-01-01 • lieu : Je chene a crote de la carriere

Alamix :

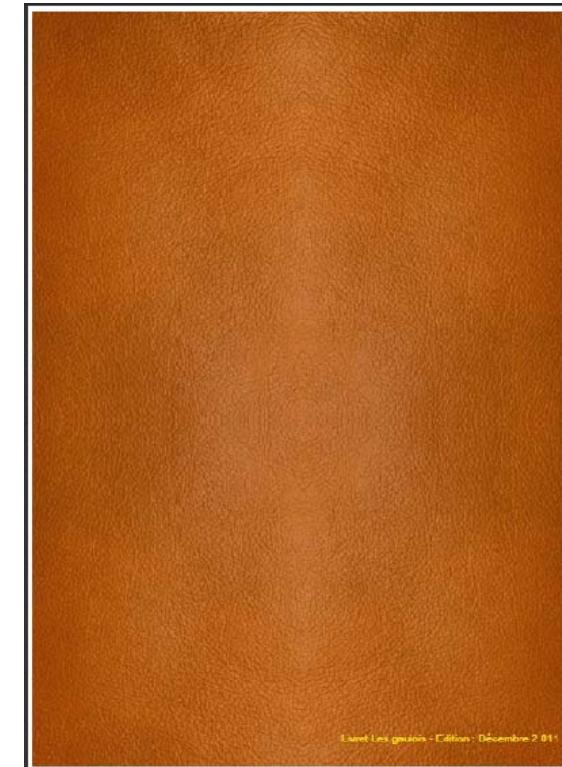
- Adresse : 2987B • charpentier
- image : Alamix.jpg
- Babille : Domene XII • date : 0050-09-03 • lieu : Vilage gaulois

Allégorix :

- Adresse : 2987C • charpentier
- image : Allégorix.jpg
- Babille : Conchte II • date : 0050-09-01 • lieu : Champs de Nouveutechogix
- Babille : Gladiateurs • date : 0050-01-01 • lieu : Je chene a crote de la carriere

3

contenu



dos



Relax NG

Schémas



Relax NG - Introduction

- **R**Egular **L**anguage for **X**ML **N**ext **G**eneration
- **RELAX NG** est un langage de **schéma pour XML**, basé sur **RELAX** et **TREX**.
- Un schéma RELAX NG spécifie des **motifs** qui décrivent la **structure et le contenu d'un document XML**.
- Un tel schéma RELAX NG identifie une classe de documents XML qui correspond aux documents qui se conforment à cet ensemble de motifs.
- Un schéma **RELAX NG** est en soit un **document XML**



Relax NG - Eléments

- Balise : **<element>**
- nom de l'élément : attribut **name**
- **Motif : description de la structure des éléments :**
 - **<zeroOrMore>** : aucun ou plusieurs
 - **<oneOrMore>** : Un ou plusieurs

Relax NG - Exemple

repertoire.xml – repertoire.rng

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <répertoire>
3   <carte>
4     <nom>John Smith</nom>
5     <courriel>js@exemple.com</courriel>
6   </carte>
7   <carte>
8     <nom>Fred Bloggs</nom>
9     <courriel>fb@exemple.net</courriel>
10  </carte>
11 </répertoire>
12
```

Motif

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <element name="répertoire" xmlns="http://relaxng.org/ns/structure/1.0">
3   <zeroOrMore>
4     <element name="carte">
5       <element name="nom">
6         <text/>
7       </element>
8       <element name="courriel">
9         <text/>
10      </element>
11    </element>
12  </zeroOrMore>
13 </element>
```



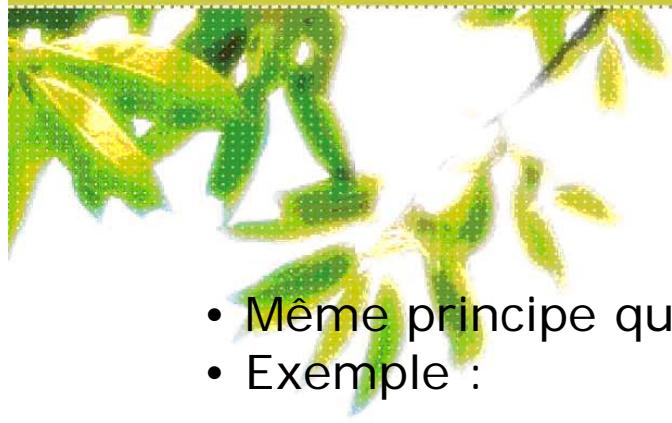
Relax NG – Eléments Optionnel

repertoire.xml – repertoire2.rng

- Balise : **<optional>**
- Permet de décrire un élément optionnel
- Exemple :

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <répertoire>
3    <carte>
4      <nom>John Smith</nom>
5      <courriel>js@example.com</courriel>
6    </carte>
7    <carte>
8      <nom>Fred Bloggs</nom>
9      <courriel>fb@example.net</courriel>
10   </carte>
11 </répertoire>
12
```

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <element name="répertoire" xmlns="http://relaxng.org/ns/structure/1.0">
3    <zeroOrMore>
4      <element name="carte">
5        <element name="nom">
6          <text/>
7        </element>
8        <element name="courriel">
9          <text/>
10       </element>
11       <optional>
12         <element name="note">
13           <text/>
14         </element>
15       </optional>
16     </element>
17   </zeroOrMore>
18 </element>
```



Relax NG – Espace de noms

repertoire.xml – repertoire3.rng

- Même principe que schémas W3C
- Exemple :

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <répertoire>
3  <carte>
4      <nom>John Smith</nom>
5      <courriel>js@exemple.com</courriel>
6  </carte>
7  <carte>
8      <nom>Fred Bloggs</nom>
9      <courriel>fb@exemple.net</courriel>
10 </carte>
11 </répertoire>
12
```

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <rng:element name="répertoire" xmlns:rng="http://relaxng.org/ns/structure/1.0">
3  <rng:zeroOrMore>
4  <rng:element name="carte">
5  <rng:element name="nom">
6      <rng:text/>
7  </rng:element>
8  <rng:element name="courriel">
9      <rng:text/>
10 </rng:element>
11 </rng:element>
12 </rng:zeroOrMore>
13 </rng:element>
14
```

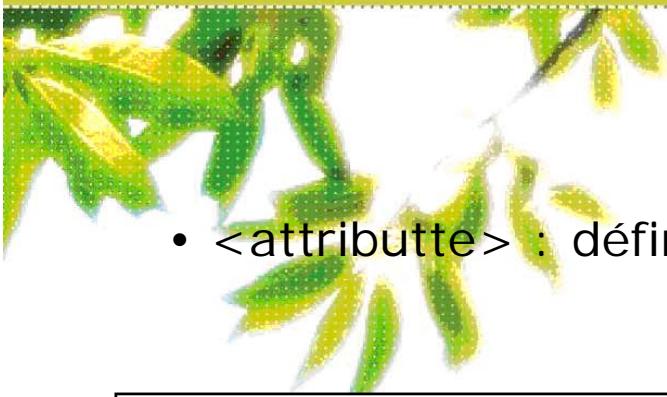
Relax NG – Choix et regroupement

repertoire2.xml – repertoire4.rng

- <choice> : alternative
- <group> : élément composé d'autres éléments

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <r  pertoire>
3  <carte>
4      <p  renom>John</p  renom>
5      <n  om-de-famille>Smith</n  om-de-famille><!--
6      <courriel>js@example.com</courriel>
7  </carte>
8  <carte>
9      <n  om>Fred Bloggs</n  om><!--
10     <courriel>fb@example.net</courriel>
11  </carte>
12 </r  pertoire>
13
```

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <element name="r  pertoire">
3  <zeroOrMore>
4      <element name="carte">
5          <choice>
6              <element name="nom">
7                  <text/>
8              </element>
9              <group>
10                 <element name="p  renom">
11                     <text/>
12                 </element>
13                 <element name="n  om-de-famille">
14                     <text/>
15                 </element>
16             </group>
17         </choice>
18         <element name="courriel">
19             <text/>
20         </element>
21         <optional>
22             <element name="note">
23                 <text/>
24             </element>
25         </optional>
26     </element>
27 </zeroOrMore>
28 </element>
```



Relax NG – Attributs

repertoire3.xml – repertoire5.rng

- <attributte> : définition d'attribut

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <répertoire>
3      <carte nom="John Smith" courriel="js@example.com"/>
4      <carte nom="Fred Bloggs" courriel="fb@example.net"/>
5  </répertoire>
```

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <element name="répertoire" xmlns="http://relaxng.org/ns/structure/1.0">
3      <zeroOrMore>
4          <element name="carte">
5              <attribute name="nom">
6                  <text/>
7              </attribute>
8              <attribute name="courriel">
9                  <text/>
10             </attribute>
11         </element>
12     </zeroOrMore>
13 </element>
```



Relax NG – Exercice

librairie.xml – librairie.rng

- Créer un schéma Relax NG validant ce document :

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <librairie>
3
4  <livre isbn="2212110472" categorie="XML">
5    <titre>
6      Services Web avec XML, SOAP, WSDL, UDDI, ebXML ...
7    </titre>
8    <auteur>Jean-Marie Chauvet</auteur>
9    <editeur>Eyrolles</editeur>
10   </livre>
11
12 <livre isbn="2100065203" categorie="XML">
13   <titre>XML Manuel de référence</titre>
14   <auteur>R. Wyke</auteur>
15   <auteur>S. Rehman</auteur>
16   <auteur>B. Leupen</auteur>
17   <editeur>Microsoft Press</editeur>
18   </livre>
19
20 <livre isbn="1928994474" categorie="XML">
21   <titre>XML.NET Developer's Guide</titre>
22   <auteur>Collectif</auteur>
23   <editeur>Syngress</editeur>
24   </livre>
25
26 </librairie>
27
```



Relax NG – Correction

librairie.xml – librairie.rng

- Créer un schéma Relax NG validant ce document :

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <element name="librairie" xmlns="http://relaxng.org/ns/structure/1.0">
3    <zeroOrMore>
4      <element name="livre">
5        <attribute name="isbn">
6          <text/>
7        </attribute>
8        <attribute name="categorie">
9          <text/>
10         </attribute>
11        <group>
12          <element name="titre">
13            <text/>
14          </element>
15          <oneOrMore>
16            <element name="auteur">
17              <text/>
18            </element>
19          </oneOrMore>
20          <element name="editeur">
21            <text/>
22          </element>
23        </group>
24      </element>
25    </zeroOrMore>
26  </element>
```



Relax NG – Motifs nommés

repertoire.xml – repertoire6.rng

- **grammar**, **start** et **define**
- Un élément **grammar** a :
 - un **unique** élément fils **start**,
 - **aucun** ou **plusieurs** éléments fils **define**

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <répertoire>
3    <carte>
4      <nom>John Smith</nom>
5      <courriel>js@example.com</courriel>
6    </carte>
7    <carte>
8      <nom>Fred Bloggs</nom>
9      <courriel>fb@example.net</courriel>
10   </carte>
11 </répertoire>
12
```

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <grammar>
3    <start>
4      <element name="répertoire">
5        <zeroOrMore>
6          <element name="carte">
7            <ref name="contenu-de-carte"/>
8          </element>
9        </zeroOrMore>
10       </element>
11     </start>
12
13    <define name="contenu-de-carte">
14      <element name="nom">
15        <text/>
16      </element>
17      <element name="courriel">
18        <text/>
19      </element>
20    </define>
21  </grammar>
22
```

Relax NG – Motifs nommés

repertoire.xml – repertoire6.rng

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <répertoire>
3    <carte>
4      <nom>John Smith</nom>
5      <courriel>js@exemple.com</courriel>
6    </carte>
7    <carte>
8      <nom>Fred Bloggs</nom>
9      <courriel>fb@exemple.net</courriel>
10   </carte>
11 </répertoire>
12
```

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <grammar>
3    <start>
4      <element name="répertoire">
5        <zeroOrMore>
6          <element name="carte">
7            <ref name="contenu-de-carte"/>
8          </element>
9        </zeroOrMore>
10       </element>
11     </start>
12
13   <define name="contenu-de-carte">
14     <element name="nom">
15       <text/>
16     </element>
17     <element name="courriel">
18       <text/>
19     </element>
20   </define>
21 </grammar>
22
```

Relax NG – Motifs nommés

repertoire.xml – repertoire7.rng

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <r  ertoire>
3   <carte>
4     <nom>John Smith</nom>
5     <courriel>js@exemple.com</courriel>
6   </carte>
7   <carte>
8     <nom>Fred Bloggs</nom>
9     <courriel>fb@exemple.net</courriel>
10  </carte>
11 </r  ertoire>
12
```

En utilisation proche des DTD

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <grammar>
3   <start>
4     <ref name="R  ertoire"/>
5   </start>
6   <define name="R  ertoire">
7     <element name="r  ertoire">
8       <zeroOrMore>
9         <ref name="Carte"/>
10        </zeroOrMore>
11      </element>
12    </define>
13   <define name="Carte">
14     <element name="carte">
15       <ref name="Nom"/>
16       <ref name="Courriel"/>
17     </element>
18   </define>
19   <define name="Nom">
20     <element name="nom">
21       <text/>
22     </element>
23   </define>
24   <define name="Courriel">
25     <element name="courriel">
26       <text/>
27     </element>
28   </define>
29 </grammar>
```



Relax NG – Typage des données

- Relax NG permet l'utilisation de définition de données externes
- Par exemple : utilisation des structures définies par W3C Schema DataTypes
- Exemple :

```
<element name="nombre">
    <data type="integer"
        datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes"/>
</element>
```



- L'attribut datatypeLibrary prend pour valeur l'URI identifiant la librairie de type de données
- Il n'est pas utile de définir systématiquement l'URI dataType
- Elle se transmet par héritage aux autres éléments, les éléments utilisent la définition du datatype du parent le plus proche



Relax NG – Typage des données

```
<element name="point"
datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes">

    <element name="x">
        <data type="double"/>
    </element>

    <element name="y">
        <data type="double"/>
    </element>

</element>
```

X et y utilise la librairie de point



Relax NG – Typage des données - paramètre

- <param>
- Les types de données peuvent avoir des paramètres
- Exemple :
 - Chaîne de caractères limitée à 127 caractères

```
<element name="courriel">
    <data type="string">
        <param name="maxLength">127</param>
    </data>
</element>
```



Relax NG – Enumération

- <choice>
- Permet de préciser une liste de valeurs autorisées
- Exemple : l'attribut préférence-de-format ne peut avoir comme valeurs que html ou texte

```
<element name="carte">
    <attribute name="nom"/>
    <attribute name="courriel"/>
    <attribute name="préférence-de-format">
        <choice>
            <value>html</value>
            <value>text</value>
        </choice>
    </attribute>
</element>
```



Relax NG – Enumération

- Exemple pour les éléments : l'attribut préférence-de-format ne peut avoir comme valeurs que html ou texte

```
<element name="carte">
    <element name="nom">
        <text/>
    </element>
    <element name="courriel">
        <text/>
    </element>
    <element name="préférence-de-format">
        <choice>
            <value>html</value>
            <value>text</value>
        </choice>
    </element>
</element>
```

```
<carte>
    <nom>John Smith</nom>
    <courriel>js@exemple.com</courriel>
    <préférence-de-format>
        html
    <préférence-de-format>
</carte>

<carte>
    <nom>Fred Bloggs</nom>
    <courriel>fb@exemple.net</courriel>
    <préférence-de-format>
        text
    <préférence-de-format>
</carte>
```



Relax NG – Liste

- Le motif **list** décrit une séquence de segments séparés par un espace
- Exemple :
 - L'élément vecteur contient 2 réels

```
<element name="vecteur">
  <list>
    <data type="float"/>
    <data type="float"/>
  </list>
</element>
```

```
<vecteur>
  17.4 20.12
</vecteur>
```

- Une ou plusieurs valeurs de type float

```
<element name="vecteur">
  <list>
    <oneOrMore>
      <data type="double"/>
    </oneOrMore>
  </list>
</element>
```

```
<vecteur>
  17.4 20.12 33.77
</vecteur>

<vecteur>
  17.4
</vecteur>
```



Relax NG – Interleave

- Le motif **interleave**, permet à des éléments fils / attributs d'apparaître dans n'importe quel ordre
- Exemple :

```
<element name="répertoire">
  <zeroOrMore>
    <element name="carte">
      <interleave>
        <element name="nom">
          <text/>
        </element>
        <element name="courriel">
          <text/>
        </element>
      </interleave>
    </element>
  </zeroOrMore>
</element>
```

Nom et courriel peuvent apparaître dans un ordre quelconque



Présentation générale des Web Services



Vue Globale

- Type d'architecture reposant sur les standards de l'Internet
- Alternative aux architectures classiques :
 - Client/serveur
 - n/tiers
- Orientée services permettant à des applications de communiquer sans préoccupation des technologies d'implantations utilisées de part et d'autre
- Priorité : Interopérabilité
- Née fin 90 (Microsoft, IBM, SAP)
 - Basée sur les technologies XML



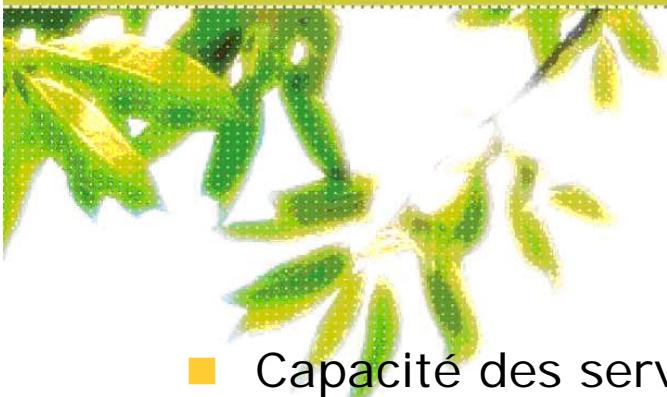
Architecture Web Services

- **Web-services** : logiciel qui interagit avec d'autres au moyen de protocoles & langages universels (http, xml ...)
- Deux formes de services-web :
 - **XML-RPC** : Remote Procedure Call
 - **SOAP** : Simple Object Access Protocol
- Présentent 2 caractéristiques :
 - Enregistrement (facultatif) auprès d'un service de recherche (**UDDI**)
 - Interface publique avec laquelle le client invoque le service Web (**WSDL**)



Architecture Web Services

- **UDDI** : *Universal Description, Discovery and Integration* peut être vu comme les pages blanches (ou jaunes) des services-web.
 - C'est un annuaire permettant à des fournisseurs de présenter leurs services à des '*clients*'.
- **WSDL** : *Web Service Description Language* est un langage reposant sur XML dont on se sert pour décrire les services-web.
 - Il est indispensable à UDDI pour permettre aux clients de trouver les méthodes leur permettant d'invoquer les services web.
- **SOAP** : *Simple Object Access Protocol* est un protocole basé sur XML et qui définit les mécanismes d'échanges d'information entre les clients et les fournisseurs de service-web.
 - Les messages SOAP sont susceptibles d'être transportés en HTTP, SMTP, FTP...
- **XML-RPC** : protocole RPC (*Remote Procedure Call*) basé sur XML. Permet donc l'invocation de procédure distante sur internet.



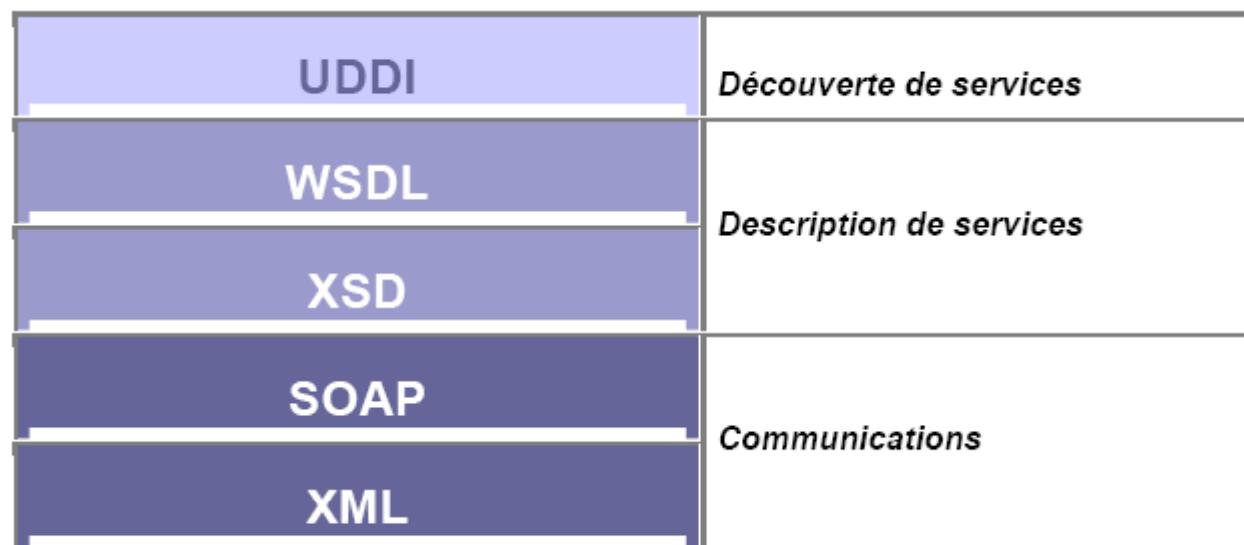
Interopérabilité

- Capacité des services Web à faire converser des **applications & des composants hétérogènes**
- Exemple :
 - Réalisation d'un service Web permettant de donner le cours d'une action en bourse, fonctionnant sous **Linux en Java**
 - Et l'interroger depuis une page Web **Asp.net** en même temps que depuis une application **PERL ou PHP**



Fonctionnement - Couches

- Toutes ces technologies sont disposées en couches et constituent l'architecture services Web



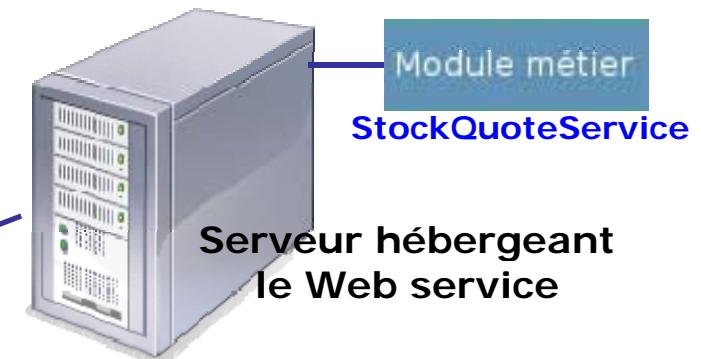
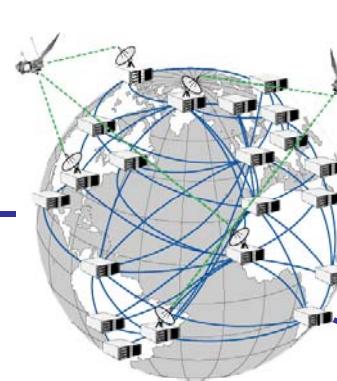
Architecture Web Services - Exemple



Un utilisateur désire consulter le cours de l'action IBM au moyen d'un service Web *StockQuoteService* qui retourne le cours d'une action en bourse, à partir de son code de cotation



Utilisateur
MacBook

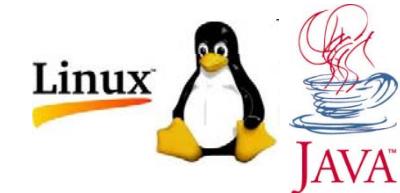
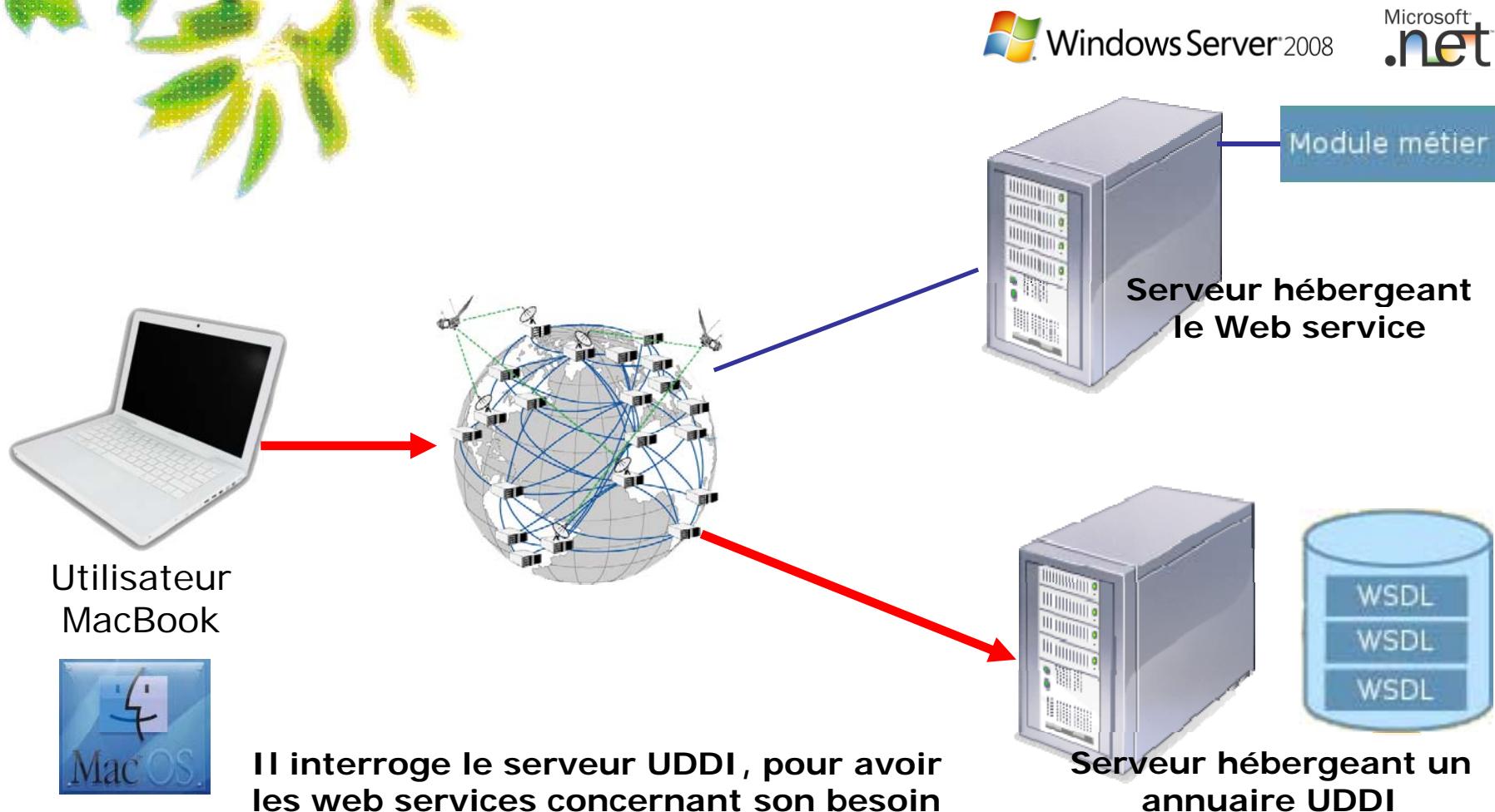


Serveur hébergeant un annuaire UDDI



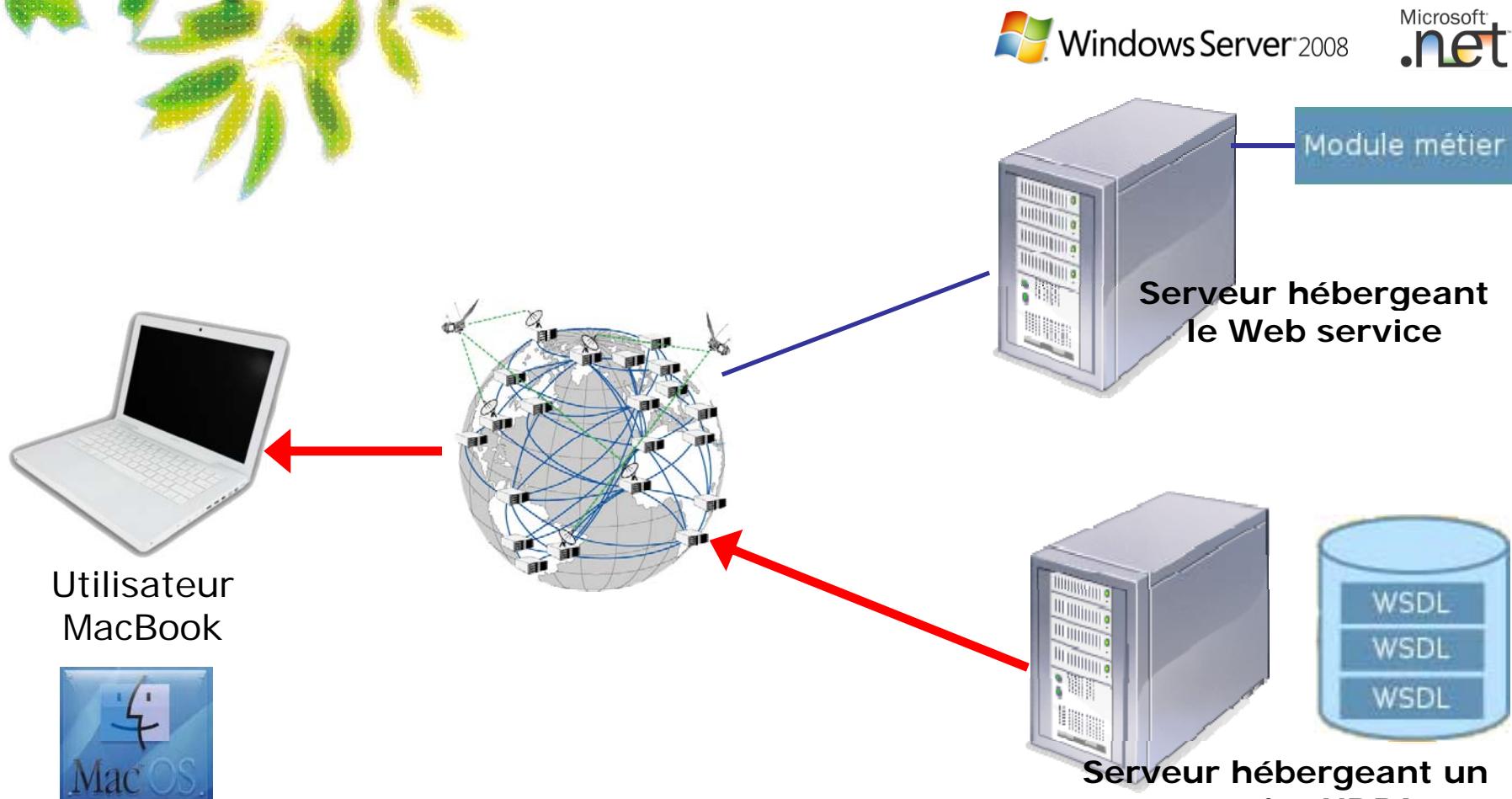


Architecture Web Services - Exemple



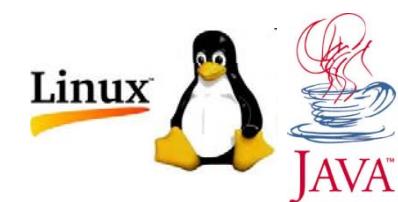


Architecture Web Services - Exemple



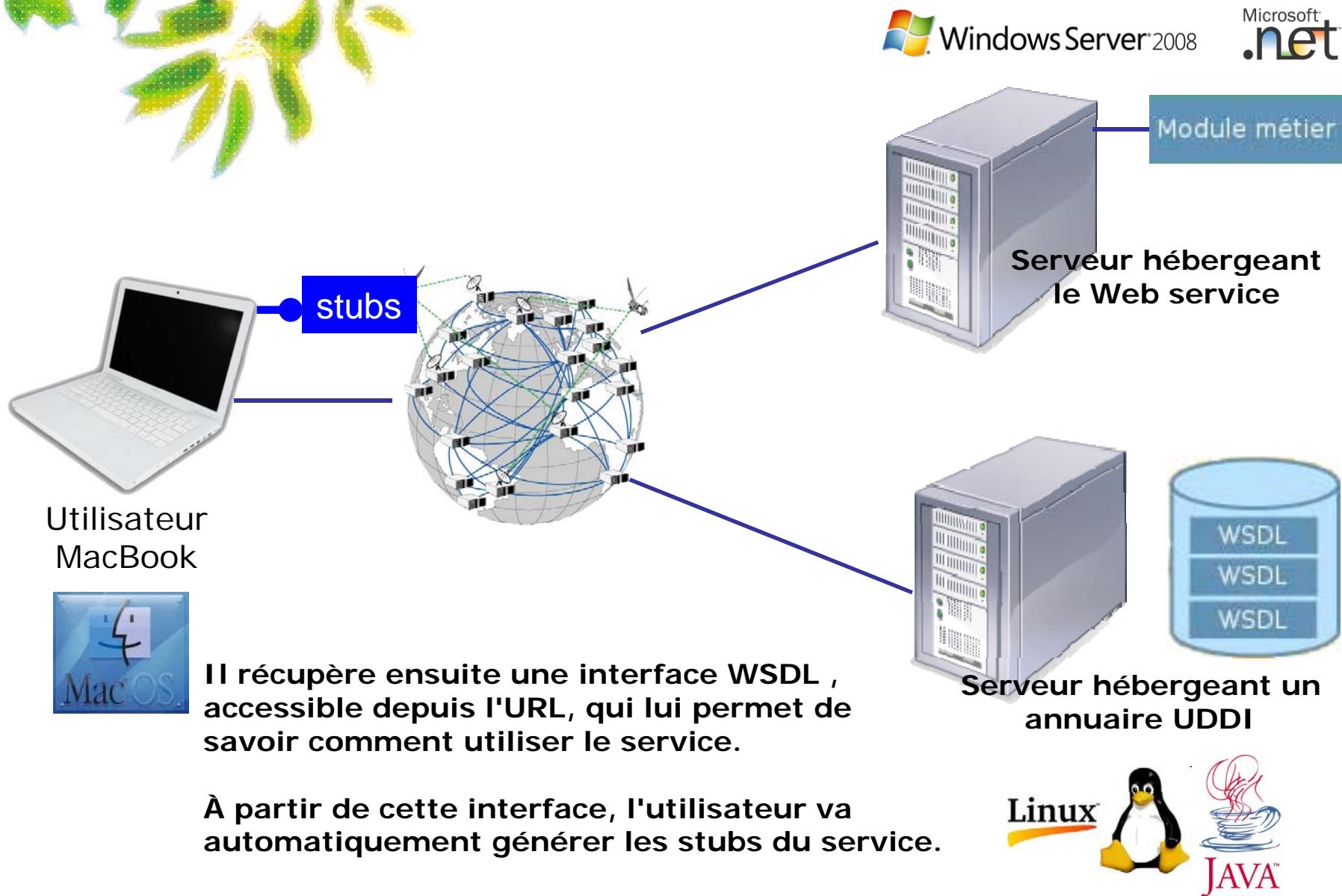
Le serveur lui retourne la liste des possibilités parmi lesquelles il en sélectionne une

À ce stade, l'utilisateur ne possède qu'une URL pointant vers le service sélectionné.



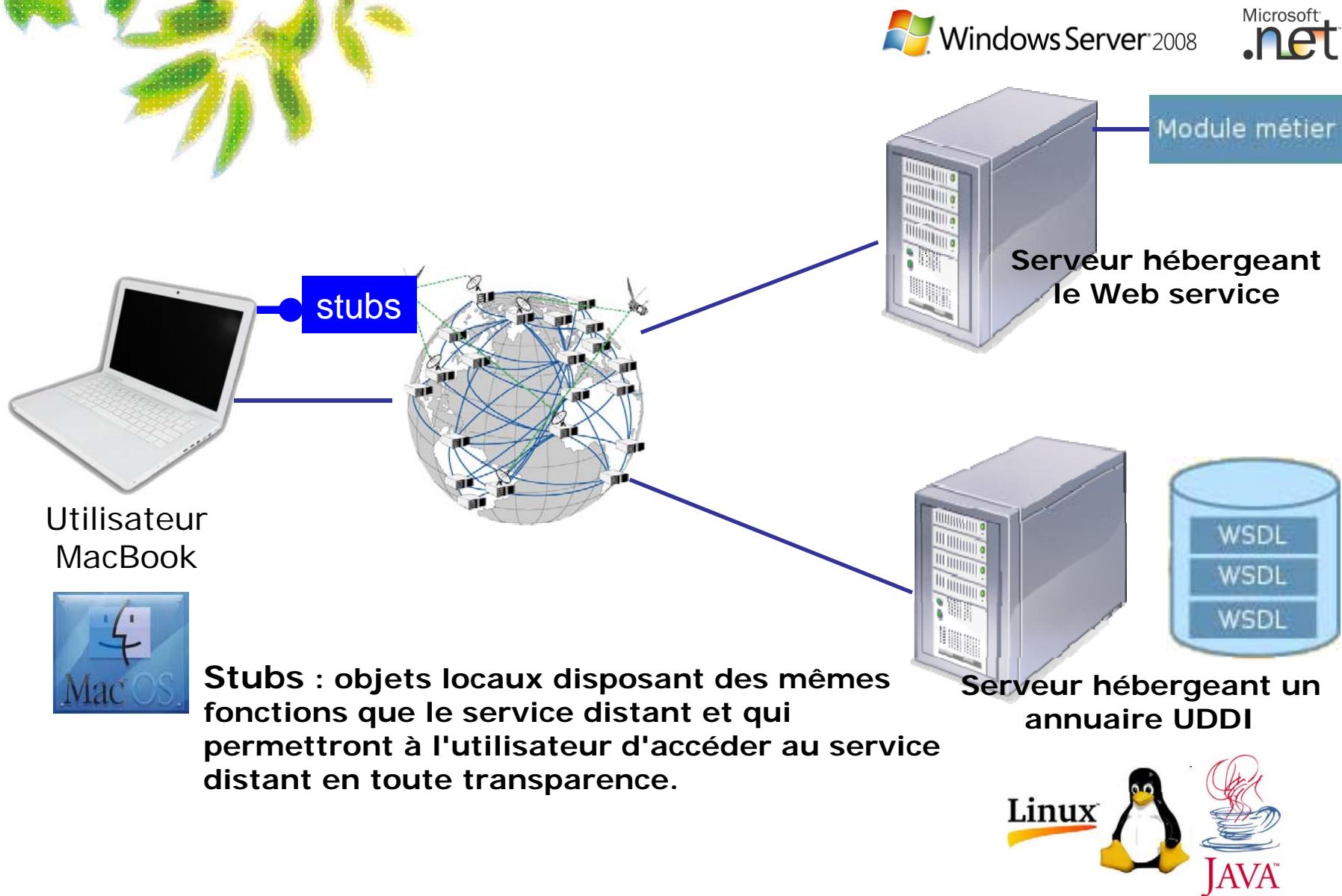


Architecture Web Services - Exemple



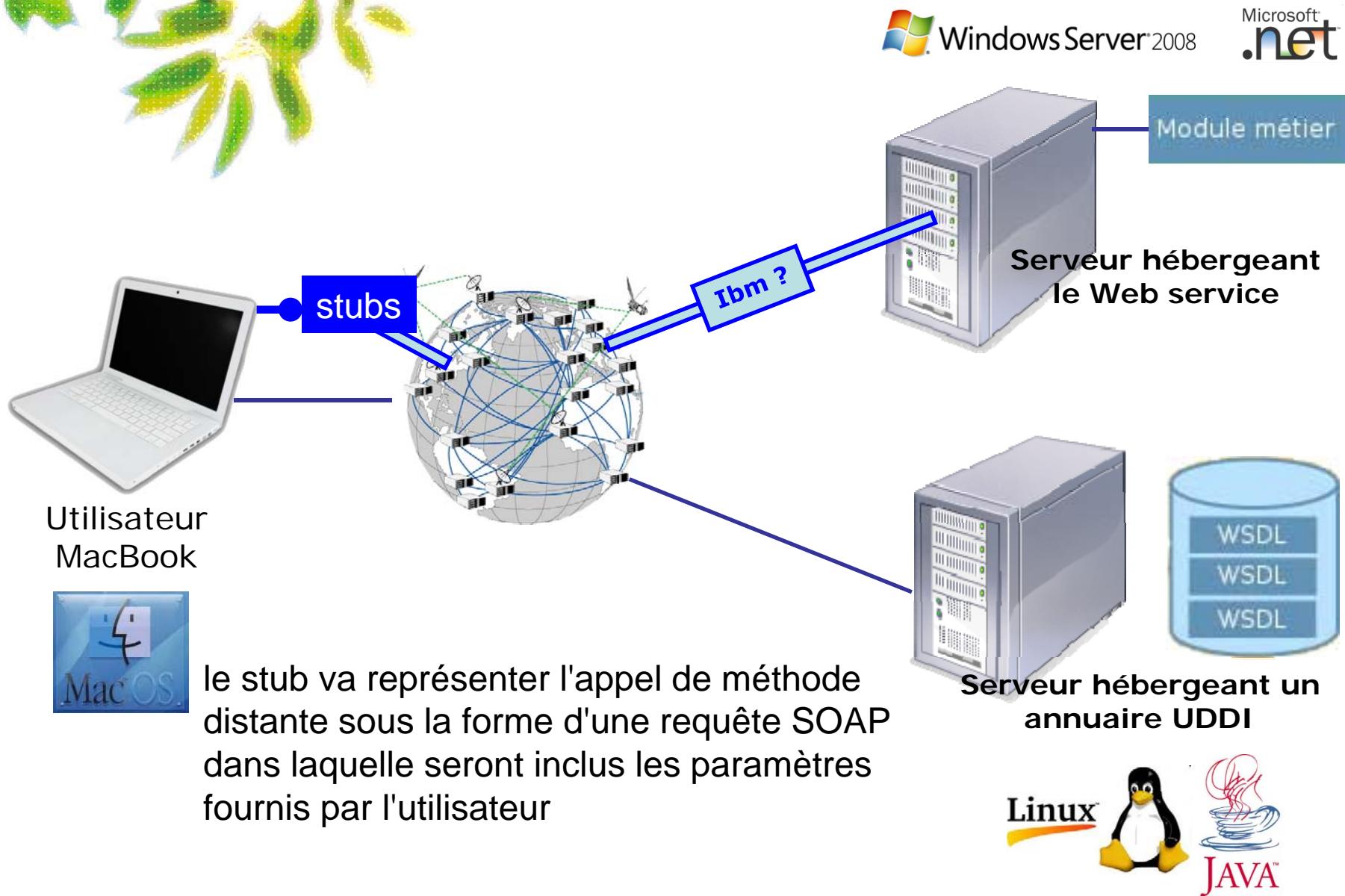


Architecture Web Services - Exemple



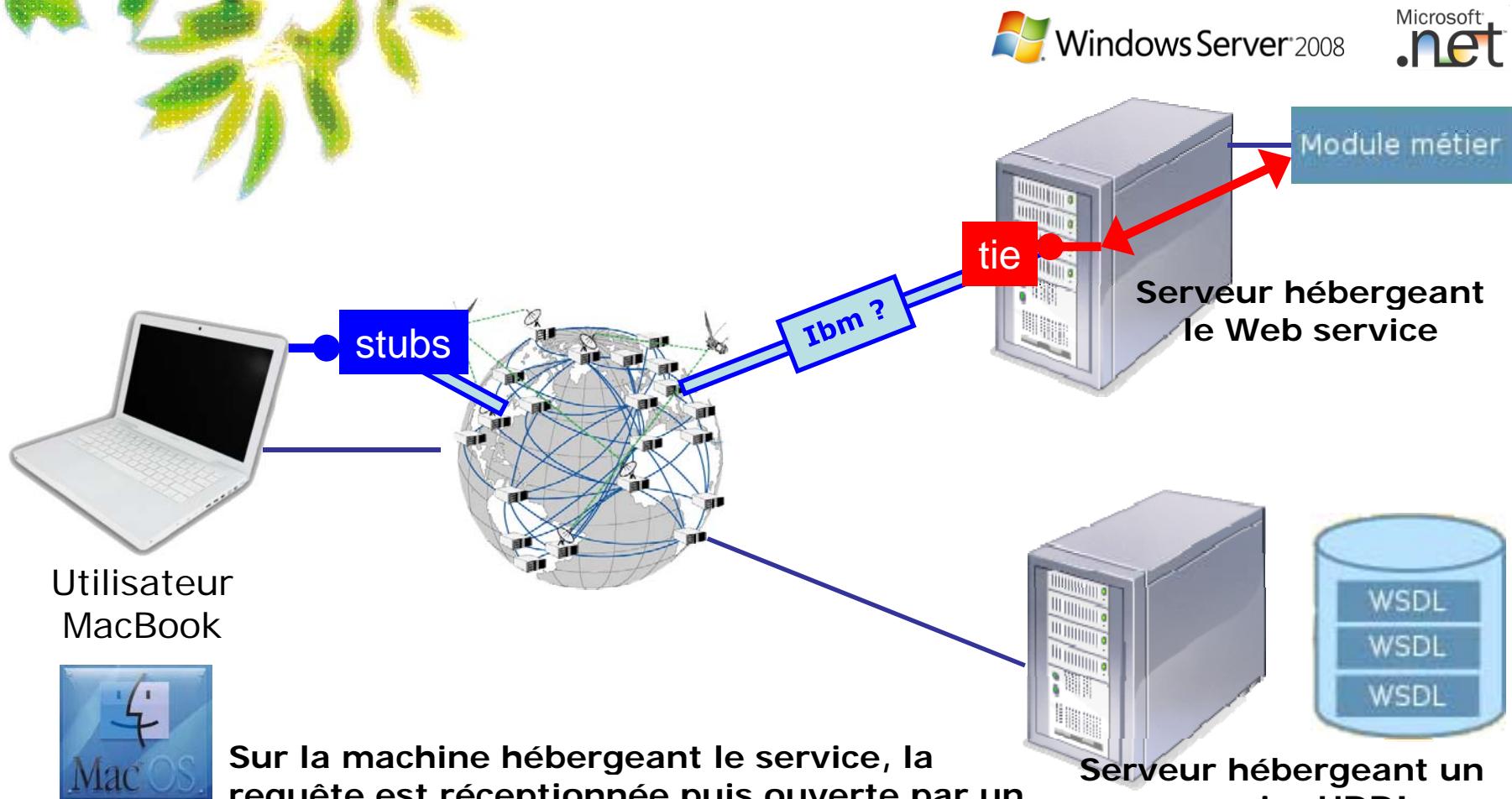


Architecture Web Services - Exemple





Architecture Web Services - Exemple



Sur la machine hébergeant le service, la requête est réceptionnée puis ouverte par un Tie

Le service Web, une fois la requête comprise, interroge sa base de données et récupère le cours de l'action IBM

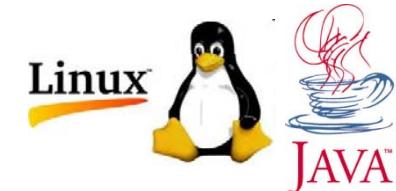
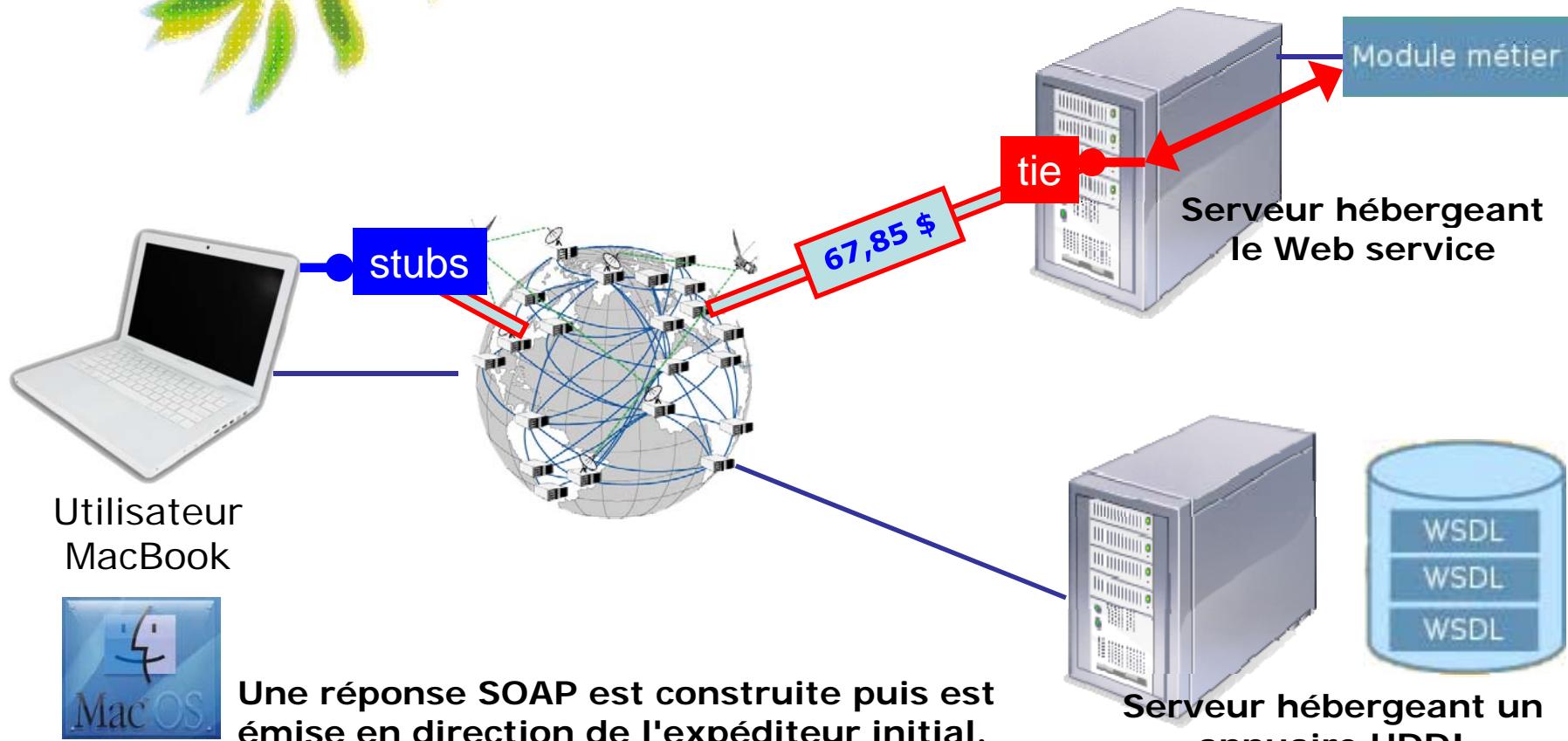




Architecture Web Services - Exemple

Windows Server 2008

Microsoft .NET

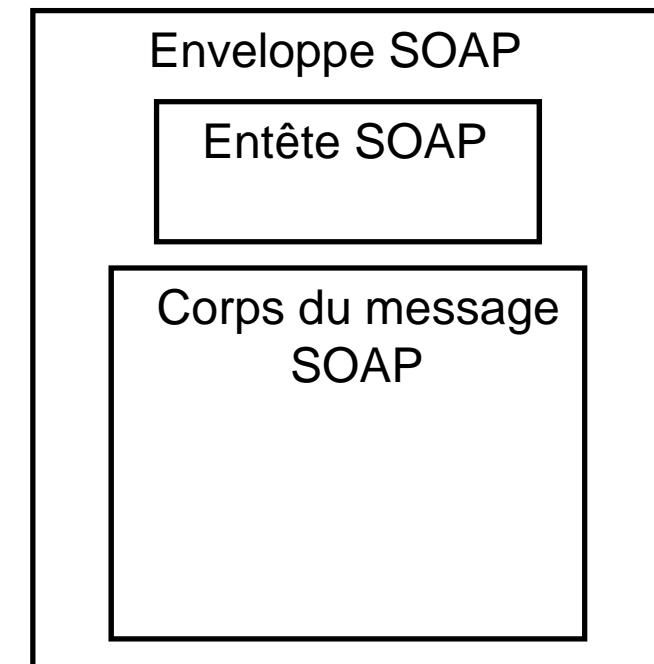




Structure des messages - SOAP

- Document XML, constitué :

- D'une enveloppe :
 - Un entête – header (facultatif)
 - Un corps – body
- D'un message





Enveloppe SOAP - Exemple

Requête demande valeur d'un titre

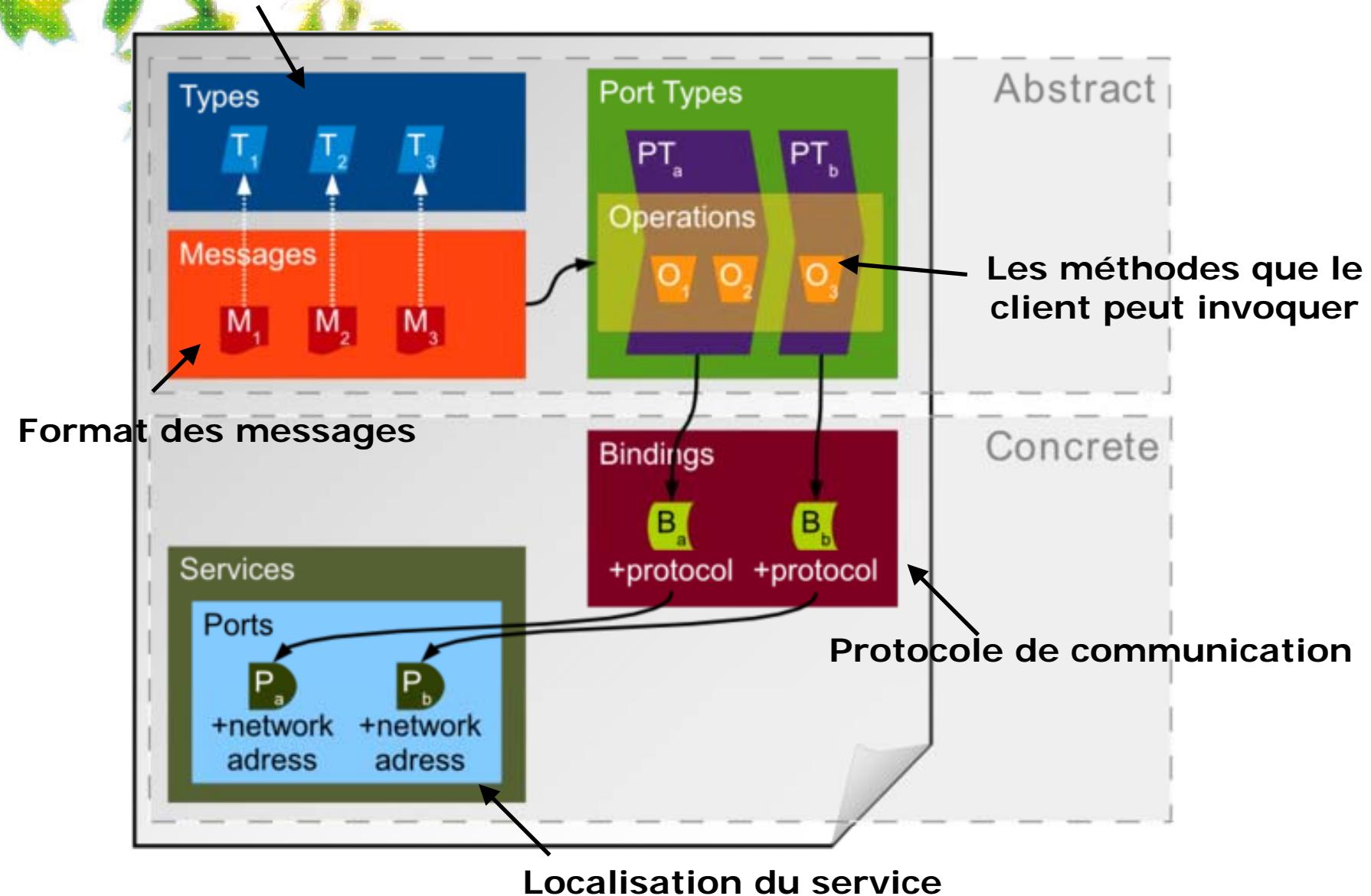
```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
    <SOAP-ENV:Body>
        Requete pour la valeur d'un titre
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Réponse du serveur

```
<?xml version="1.0" encoding="UTF-8" ?>
<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
    <SOAP-ENV:Body>
        Valeur du titre
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

WSDL

Types de données échangées (en entrée et en sortie)

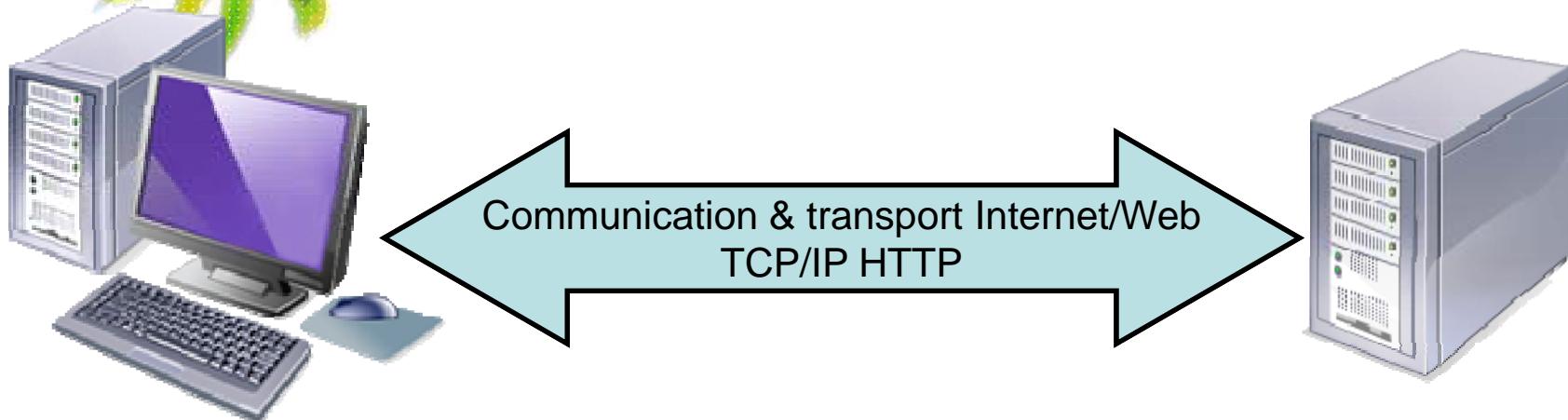




Architecture des services Web



Fondations de l'architecture



Le client peut être soit un navigateur Web (demande faite par un utilisateur humain) soit une application (demande de service automatique)

Le serveur est un serveur HTTP (Apache/PHP par ex.) + un serveur d'application (J2EE, .NET par ex.)



SOAP - Communication



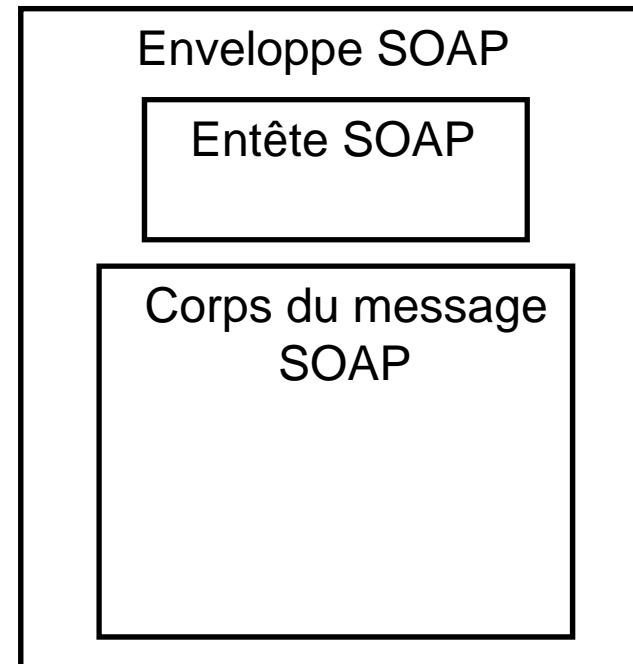
Simple Object Access Protocol

- SOAP : Spécification de communication entre services Web par échange de messages XML
- Indépendant des langages de programmation / systèmes d'exploitation
- Repose sur XML nameSpace & XML Schéma en particulier
- Se décompose en 4 parties :
 - L'enveloppe SOAP
 - Les règles de codage SOAP
 - Un protocole RPC
 - La définition de l'utilisation de HTTP
- Spécifications : <http://www.w3.org/TR/SOAP/>



Structure des messages

- Document XML, constitué :
 - D'une enveloppe :
 - Un entête – header (facultatif)
 - Un corps - body
 - D'un message



Enveloppe SOAP - <SOAP-ENV:Envelope>

- Racine du document XML contenant le message SOAP
 - Balise <Enveloppe>
 - Tous les attributs et ceux des balises imbriquées doivent être associés à un namespace
- Deux namespaces fréquemment utilisés :
- SOAP-ENV : Pour l'enveloppe
 - V1.1 - URI : <http://schemas.xmlsoap.org/soap/envelope/>
 - V1.2 - URI : <http://www.w3.org/2001/06/soap-envelope/>
- SOAP-ENV : Pour la définition des formats de type de données :
 - V1.1 – URI : <http://schemas.xmlsoap.org/soap/encoding/>
 - V1.2 - URI : <http://www.w3.org/2001/06/soap-encoding/>



Enveloppe SOAP - Exemple

Requête demande valeur d'un titre

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
    <SOAP-ENV:Body>
        Requete pour la valeur d'un titre
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Réponse du serveur

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
    <SOAP-ENV:Body>
        Valeur du titre
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```



Entête SOAP : <SOAP-ENV:Header>

- Permet de passer dans le message des informations complémentaires (facultatif)
 - Si l'est présent, doit être le premier dans l'enveloppe SOAP
 - Exemple : info d'authentification de l'émetteur, contexte d'une transaction ...
- Attributs :
 - mustUnderStand :
 - =1 : le récepteur dit reconnaître l'information présente dans l'entête, son traitement est obligatoire
 - =0 : l'entête peut être ignoré
 - Intérêt : Un serveur peut ignorer cette information, un autre peut la prendre en compte



Entête SOAP : Exemple

Entête transportant une adresse électronique

```
<?xml version="1.0"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" >
<SOAP-ENV:header>
    <exemple:emetteur xmlns:exemple="http://monURII SOAP-
        ENV:mustUnderstand="0">
        MonNom@MonDomaine.com
    </exemple:emetteur>
</SOAP-ENV:header>
</SOAP-ENV:Envelope>
```



Corps message SOAP : <SOAP-ENV:Body>

- Constitué du seul élément Body, contenant un ou plusieurs sous-éléments
- Sous-éléments :
 - Fault : indique une erreur ou une défaillance en réponse à une requête
 - De données destinées au destinataire du message, à un format défini par les règles de codage SOAP
- Dans le cas de la défaillance, de nouveaux sous éléments signalent le type d'erreur
- Les codes d'erreurs sont normalisés par la spécification SOAP
- SOAP définit 4 sous éléments de l'élément fault



Fautes SOAP – élément faultcode

- Obligatoire : code de l'erreur
- Codes :
 - **VersionMismatch** : espace de noms invalide associé à l'élément envelope
 - **MustUnderstand** : Il existe un élément SOAP header contenant un attribut mustUnderstand à 1 non compris
 - **Client** : des erreurs sont survenues dans la mise en forme du message SOAP (exemple : demande d'une valeur inexistante)
 - **Server** : le message ne peut être traité, il n'y a pas d'erreur de mise en forme, mais d'impossibilité au serveur de répondre (hors-ligne par exemple)
- Ces codes sont extensibles à l'aide de la notation point, exemple :
 - Server.EchecStationMeteo
 - Server.EchecStationMeteo.inconnue



Fautes SOAP – Autres éléments

- **faultstring** (Obligatoire) : description de l'erreur
 - But : fournir une description de la faute qui soit compréhensible par un humain
- **faultactor** : Responsable de l'erreur
 - Fournit un indication sur le système responsable de l'erreur
- **detail** : Erreur dans body requête
 - Fournit une information relatives aux erreurs sur l'élément body du message de requête
 - Inclus seulement si erreur sur l'élément body



Fautes SOAP – exemple

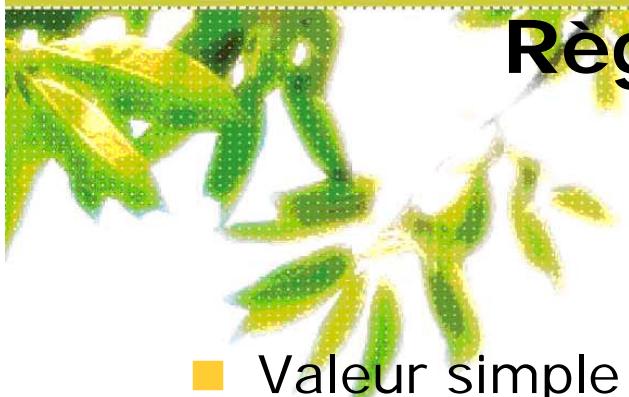
Réponse à une demande invalide d'échelle à un service météo

```
<?xml version="1.0"?>
<SOAP-ENV:Envelope xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/" >
<SOAP-ENV:body>
  <SOAP-ENV:Fault>
    <faultcode>SOAP-ENV:Client</faultcode>
    <faultstring>Erreur du client</faultstring>
    <faultactor>http://www.mindstrm.com/LocalWeather</faultactor>
    <detail>
      <m:detailsfautemeteo xmlns:m="StationMeteo">
        <message>Pas d'echelle : Calcium</message>
        <error_code>1234</error_code>
      </m:detailsfautemeteo>
    </detail>
  </SOAP-ENV:Fault>
</SOAP-ENV:body>
</SOAP-ENV:Envelope>
```



Règles de codage des différents types de données

- Appel à des balises définies dans le second namespace SOAP (<http://schemas.xmlsoap.org/soap/encoding/>)
- Possible de spécifier des namespaces de substitution propres à l'utilisateur
 - Définir de nouvelles balises
 - Redéfinir des balises existantes
- Doivent suivre des règles imposées



Règles de codage – valeur simple

- Valeur simple (par ex. entier ou chaînes)
 - doit apparaître directement entre les balises
 - Ne contient aucune partie nommée, mais simplement une donnée

- Exemple :
 - <a xsi:type="xsd:int">10
 - <x xsi:type="xsd:float">3.14159</x>
 - <s xsi:type="xsd:string">SOAP</s>

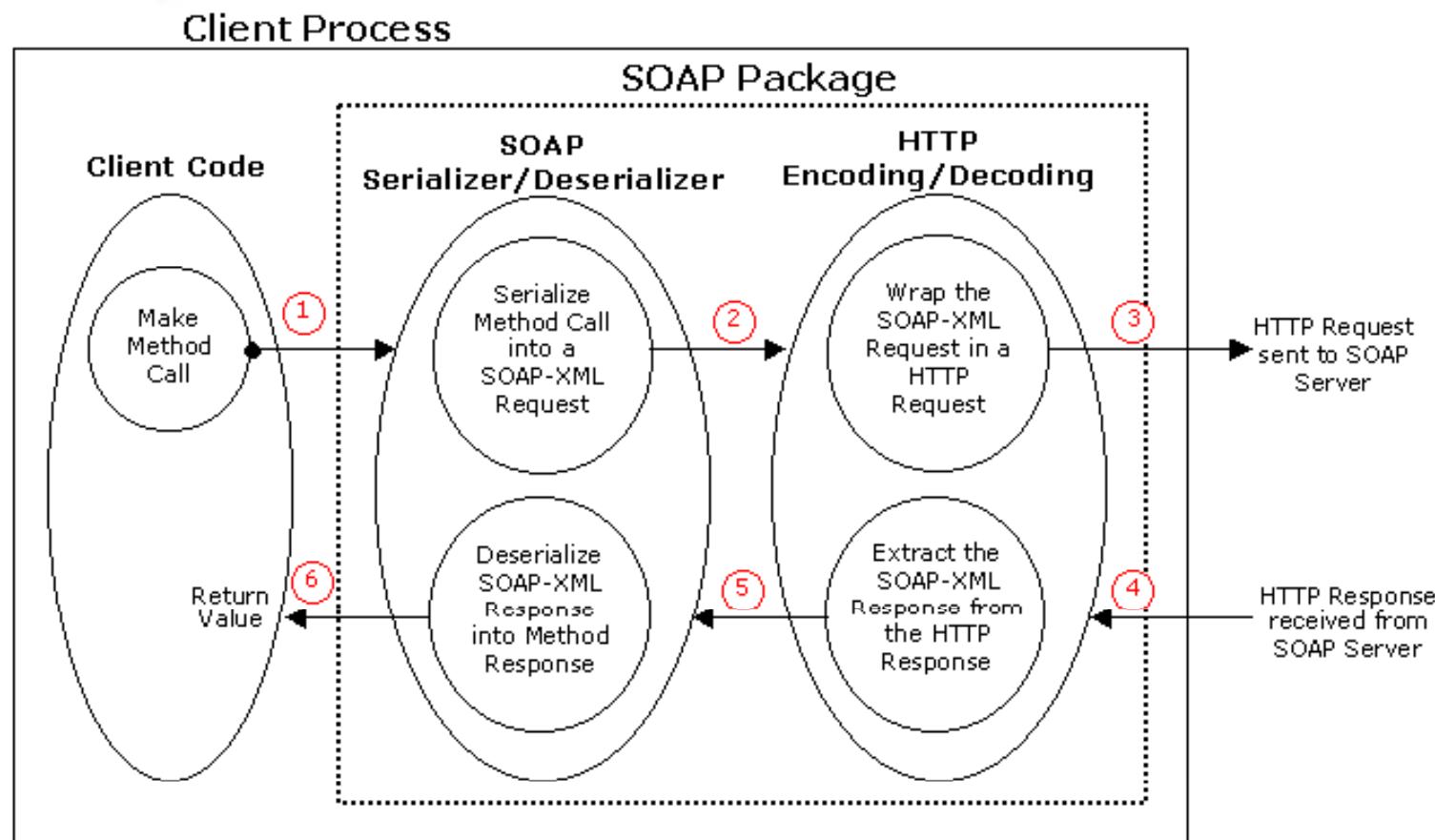


Règles de codage – valeur composée

- Valeur composée (par ex. entier ou chaînes)
 - Contient des données multiples en relation entre elles
 - Accès aux données individuelles en indiquant une position ordinaire (idem tableau) ou clé
- Exemple :
 - <iTableau xsi:type="SOAP-ENC:Array" SOAP-ENC:arrayType="xsd:int[3]">
 - <val>10</val>
 - <val>20</val>
 - <val>30</val>
 - <iTableau>

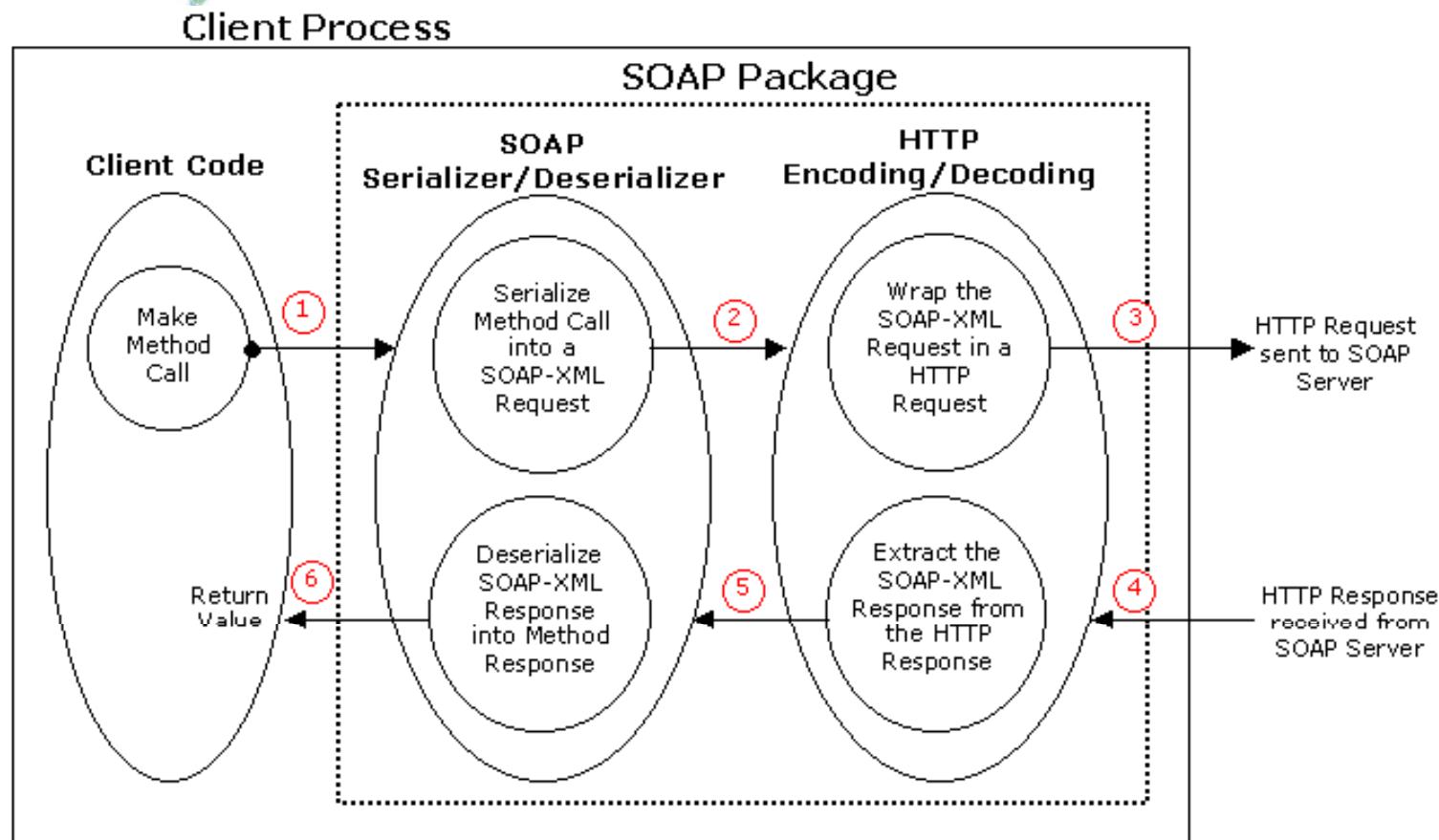
SOAP – coté client

- Exemple HTTP
- Utilisation d'un package SOAP (s'affranchir des détails de la sérialisation)



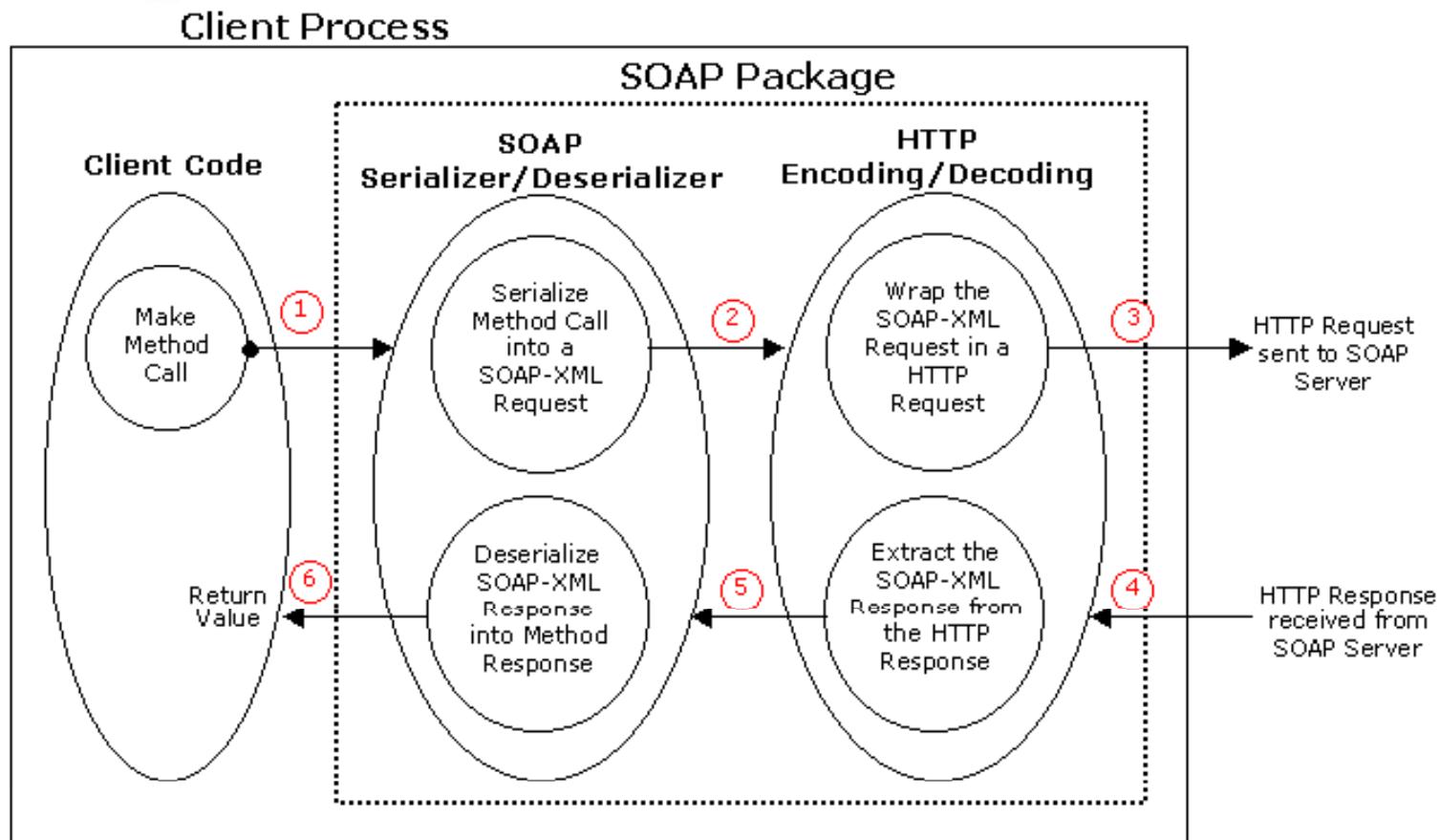
SOAP – coté client

(1) Le code client crée un appel de service en invoquant la méthode appropriée du package SOAP



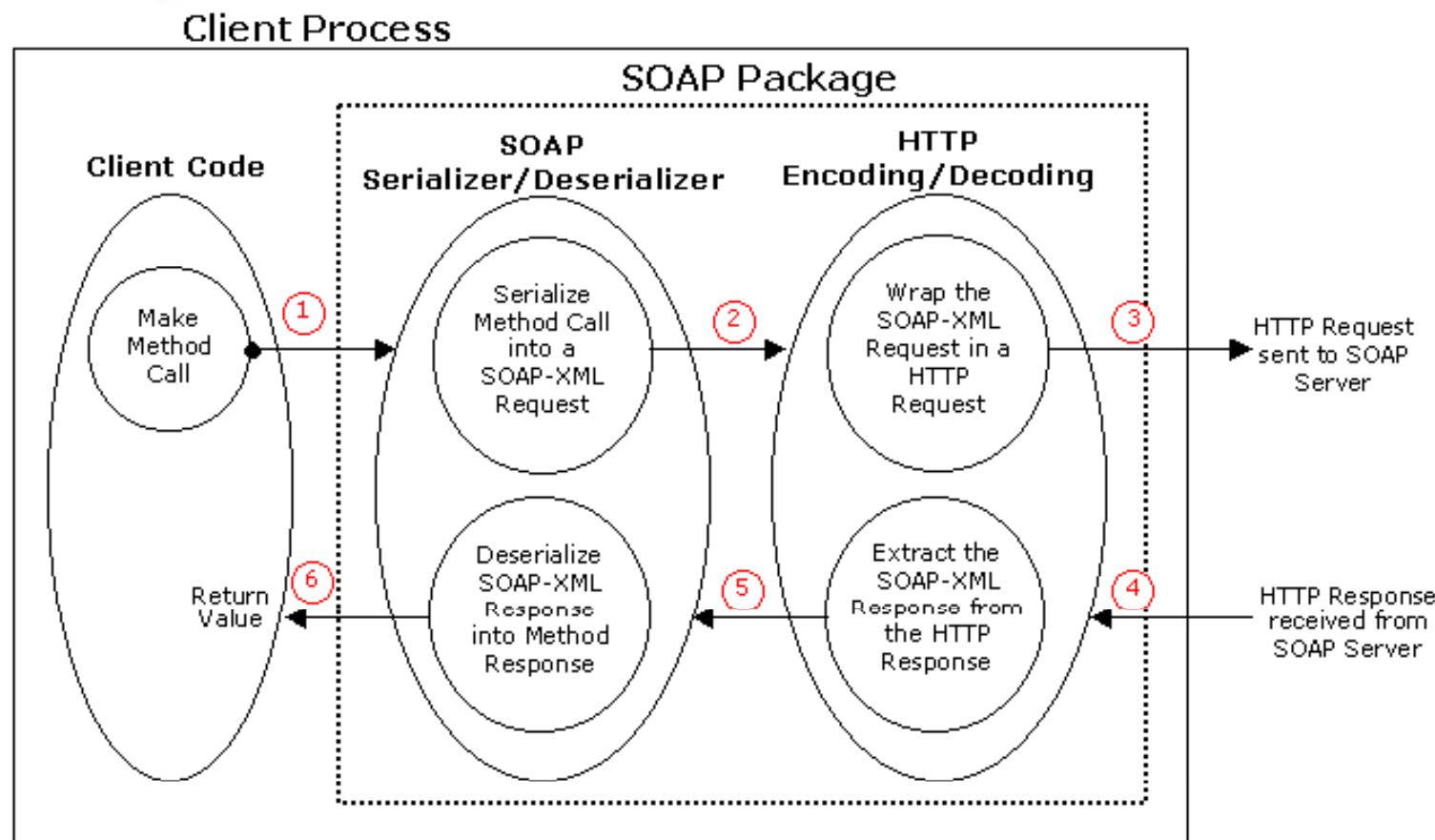
SOAP – coté client

(2) Le serialiseur SOAP du package SOAP convertit cette invocation en requête SOAP et l'envoie à l'encodeur HTTP



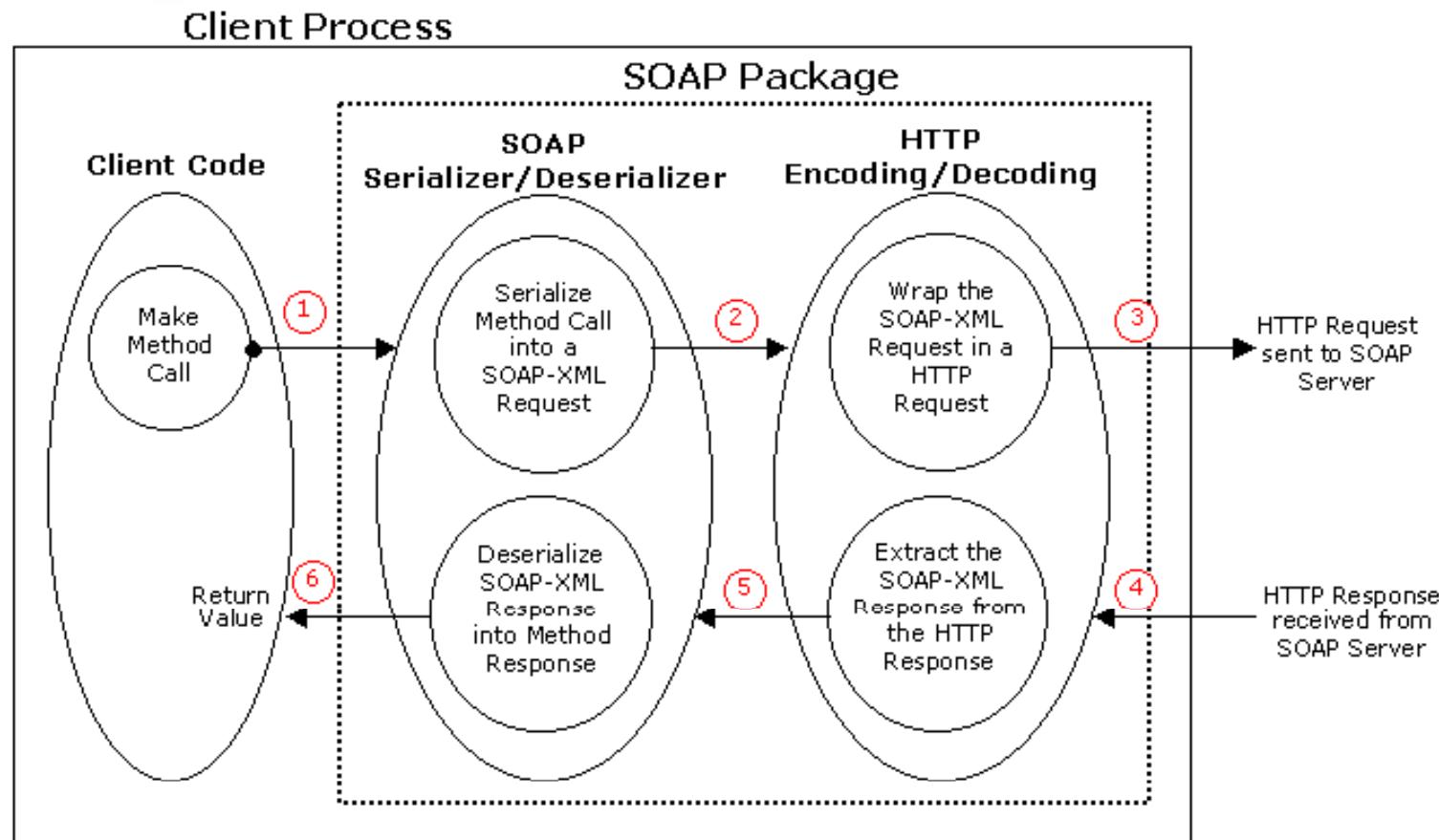
SOAP – coté client

(3) L'encodeur HTTP enveloppe le message SOAP dans une requête HTTP et l'envoie au serveur SOAP



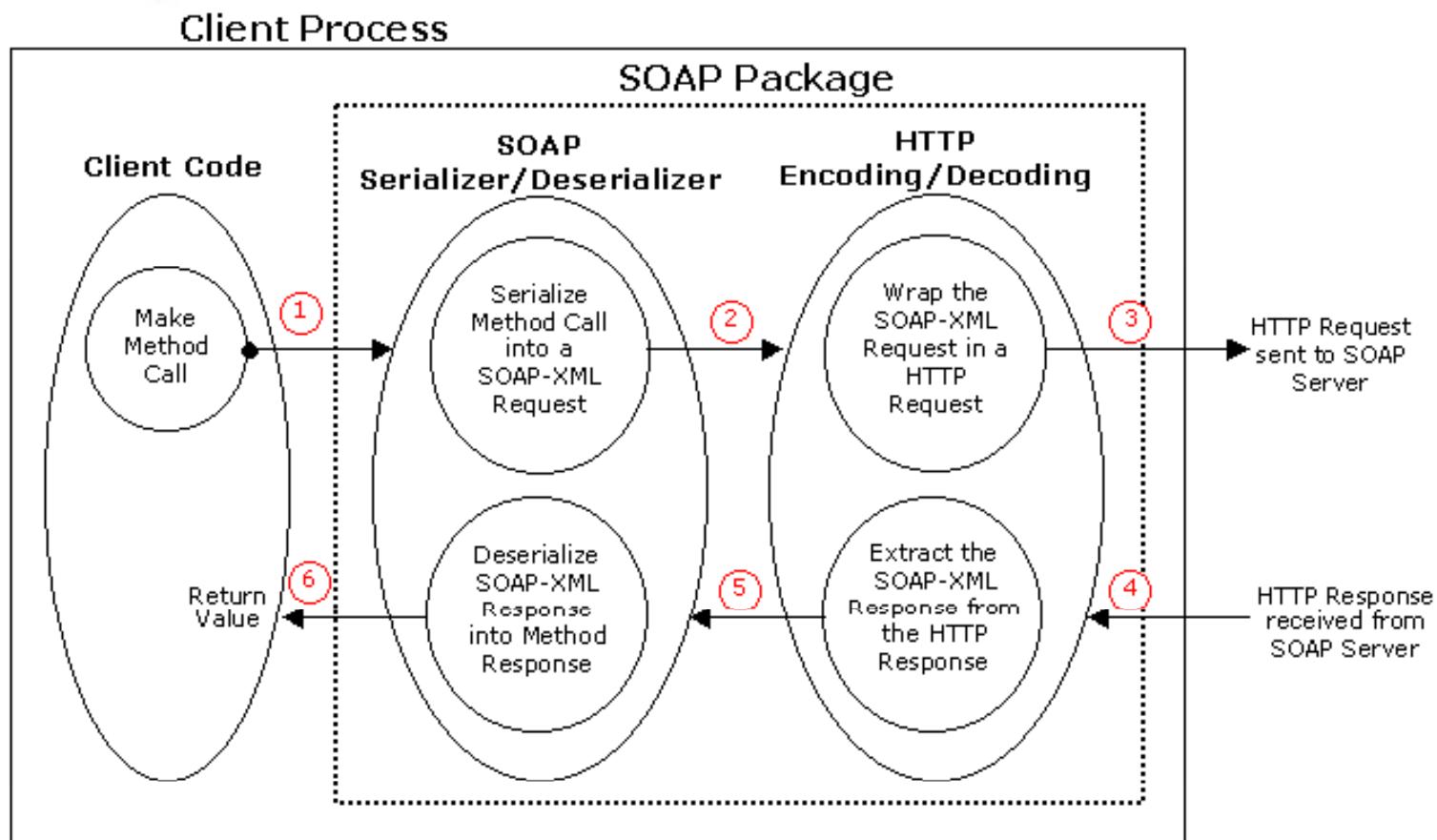
SOAP – coté client

(4) La réponse est reçue par le module d'encodage/décodage HTTP du serveur SOAP



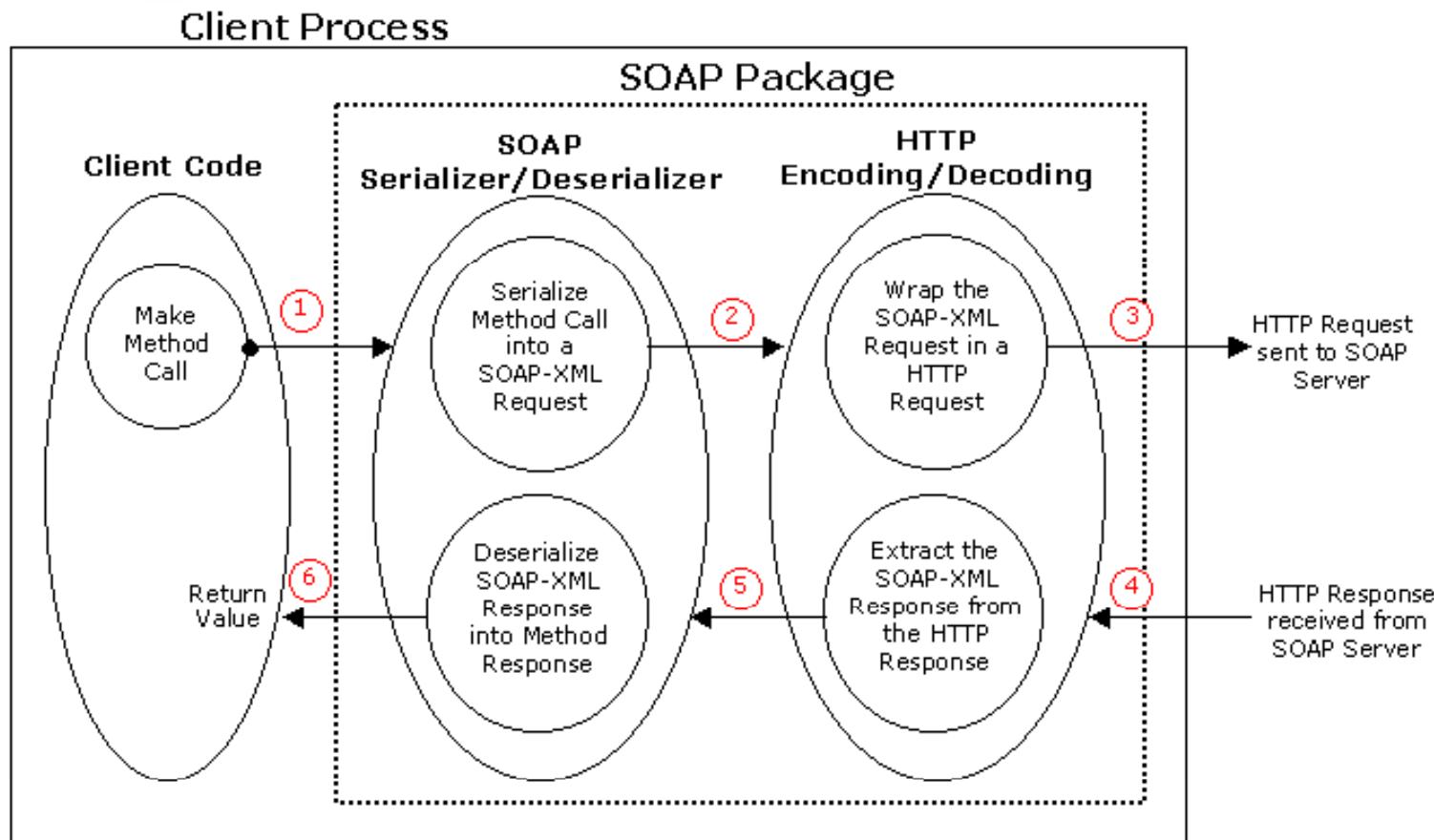
SOAP – coté client

(5) ce module décode et extrait la réponse SOAP qui la remet au deserialiseur SOAP



SOAP – coté client

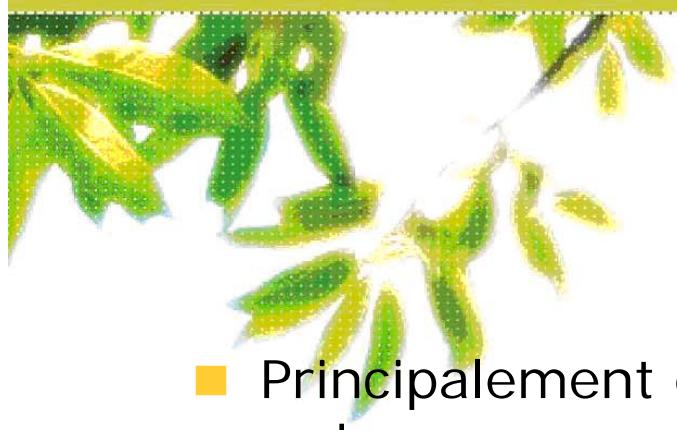
(6) Le deserialiseur SOAP deserialise le message et passe le résultat au code client comme valeur de retour de l'invocation originale





SOAP – coté client

Type d'Appli	Plate-Forme	Langage	Package SOAP
Applications Client Autonomes (Stand-Alone)	Win32 & Unix	Java	Apache SOAP for Java
		C++	Systinet WASP for C++
	Win32	Visual Basic & Excel Macros	Microsoft SOAP Toolkit
JSPs, Servlets ou Pages Web à partir d'Applets	Win32 & Unix	Java	Apache SOAP for Java
Pages Web Perl/CGI	Win32 & Unix	Perl	SOAP::Lite for Perl
Pages Web ASP	Win32	ASP Script	Microsoft SOAP Toolkit
Daemons	Win32 & Unix	C++	Systinet WASP for C++
	Win32 & Unix	Perl	SOAP::Lite for Perl



Usage SOAP (en résumé)

- Principalement employé pour transporter entre services web :
 - Des requêtes
 - Le résultat de leur exécution
- Une requête SOAP contient le nom d'une méthode ou fonction invoquée sur le site web distant avec l'ensemble de ses paramètres
- La réponse SOAP contient le résultat de l'exécution de cette fonction ou un code erreur



Usage SOAP - exemple

- Opération sur un compte bancaire

Méthode Java coté server

```
Package banque  
Public void operation{  
    public int compte;  
    public float montant;  
}
```

Corps du message SOAP

Une instance apparaît dans le corps du message SOAP

Accesseurs :

- compte
- montant

```
<t:operation xmlns:t=http://monServer/banque>  
    <compte>125612</compte>  
    <montant>250.75</montant>  
</t:operation>
```



WSDL – Description & configuration



WSDL - Principes

- Un langage (en XML) de description des services fournis par un serveur.
- Une description de type « boîte noire » de ces services :
 - Quelles sont les opérations disponibles ?
 - Comment on y accède (adresse, protocole,...)
 - Quel est le format des messages échangés entre le client et le serveur:
 - Pour invoquer le service
 - Pour interpréter les résultats
- ...mais rien sur ce qu'ils font vraiment (leur sémantique).



WSDL - Présentation

- Dans WSDL, les services communiquent par échange de messages entre des terminaisons constituées d'un ou plusieurs ports, chacun doté d'un type
- WSDL 1.0 définit les entités suivantes :
 - **Types** : Système de types applicables à des données, utilisation de XML Schéma
 - **Message** : Forme des données échangées entre services Web
 - **Opération** : Action implémentée par un service Web
 - **Type de port** : Ensemble d'opérations implémentée par une terminaison donnée
 - **Liaison (binding)** : Protocole d'accès à un port et un format de spécification des messages et des données pour ce port
 - **Port** : point de terminaison identifié de manière unique par la combinaison adresse internet & liaison
 - **Service Web** : collection de ports



Document WSDL

- Le document WSDL est constitué d'une collection de définitions des différentes briques du service Web
- Utilisation intensive des schémas XML
- Définition objet => Génération par certains frameworks de programmes d'accès (Axis par exemple)

```
<wsdl:definitions>
```

```
  <wsdl:Types>
```

```
  ...
```

```
  </wsdl:Types>
```

```
  <wsdl:Messages>
```

```
  ...
```

```
  </wsdl:Messages>
```

```
  <wsdl:PortType>
```

```
  ...
```

```
  </wsdl:PortType>
```

```
  <wsdl:Service>
```

```
  ...
```

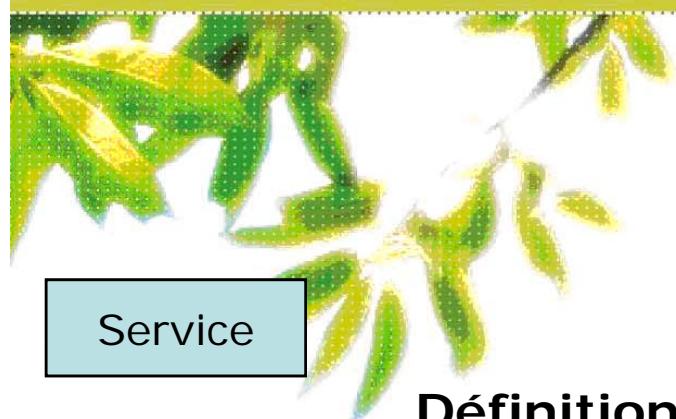
```
  </wsdl:Service>
```

```
</wsdl:definitions>
```



Document WSDL - Exemple

- Service boursier, permettant à un client de récupérer la valeur d'un titre :
 - Définition d'une opération derniersCours :
 - Argument code SICOVAM ou le nom du titre
 - Renvoie la valeur du cours
 - Les deux messages, la requête et la réponse constitueront cette seule opération d'un unique port WSDL de ce service WEB
 - Le document WSDL devra préciser comment sont concrètement constitués les message SOAP du dialogue client-service Web



Document WSDL - Exemple

Service

Port

Binding

Port Types

Operations

Messages

Types

Définition des namespaces de référence pour ce service web

```
<definitions name="CoursDesTitres"
    targetNamespace="http://example.com/titres.wsdl"
    xmlns:tns="http://example.com/titres.wsdl"
    xmlns:xsd="http://example.com/titres.xsd"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns="http://schemas.xmlsoap.org/wsdl/">
```

Document WSDL - Exemple

Composant 1:

Définition des types de données pour ce service web

Service

```
<types>
  <schema targetNamespace="http://example.com/titres.xsd"
    xmlns=http://www.w3.org/1999/XMLSchema/>
```

Port

```
  <element name="RequeteValeurCours">
```

```
    <complexType>
```

```
      <all>
```

```
        <element name="Symbole" type="string"/>
```

```
      </all>
```

```
    </complexType>
```

```
  </element>
```

```
  <element name="ValeurCours">
```

```
    <complexType>
```

```
      <all>
```

```
        <element name="prix" type="float"/>
```

```
      </all>
```

```
    </complexType>
```

```
  </element>
```

2 types de données
RequeteValeurCours
ValeurCours

Port Types

Operations

```
</schema>
```

```
</type>
```

Messages

Types



Document WSDL - Exemple

Composant 2.

Définition des messages : requête & réponse

Service

Port

Binding

Port Types

Operations

Messages

Types

```
<message name="dernierCours_Input">
    <part name="body"
element="xsd:RequeteValeurCours"/> </message>

<message name="dernierCours_Output">
    <part name="body" element="xsd:ValeurCours"/>
</message>
```

Description des 2 messages (requête & réponse) avec les éléments nécessaires



Document WSDL - Exemple

Service

Port

Binding

Port Types

Operations

Messages

Types

Composant 3.

Définition du type de port WSDL : une seule opération

```
<portType name="Type_PortValeurCours">
    <operation name="dernierCours">
        <input message="tns: dernierCours_ Input " />
        <output message="tns: dernierCours_ Output " />
    </operation>
</portType>
```

Le port ne porte qu'une opération attendant un message dernierCours_Input en entrée, et émettant le message dernierCours_Output en sortie

Document WSDL - Exemple

Composant 4:

Définition de la liaison WSDL/SOAP pour ce port

Service

```
<binding name="Liaison_PortValeurCours" type="tns:Type_PortValeurCours">
    <soap:binding style="document"
        transport="http://schemas.xmlsoap.org/soap/http"/>
```

Port

```
    <operation name=" dernierCours ">
        <soap:operation soapAction=
            "http://example.com/dernierCours "/>
        <input>
            <soap:body use="literal" namespace=
                "http://example.com/titres.xsd"
                encodingStyle=
                    "http://schemas.xmlsoap.org/soap/encoding//"/>
        </input>
        <output>
```

Port Types

```
            <soap:body use="literal" namespace=
                http://example.com/titres.xsd
                encodingStyle=
                    "http://schemas.xmlsoap.org/soap/encoding//"/>
```

Operations

```
        </output>
    </operation>
```

Messages

```
</binding>
```

Types



Document WSDL - Exemple

Composant 5.

Service

Port

Binding

Port Types

Operations

Messages

Types

Définition du service web, instantiation d'un port WSDL

```
<service name="ServiceWebBoursier">
    <documentation>Mon premier service WSDL I
    </documentation>
    <signature>
    </signature>

    <port name="PortValeurCours"
        binding="tns:Liaison_PortValeurCours ">
        <soap:address location="http://example.com/titres"/>
    </port>
```

```
</service>
</definitions>
```

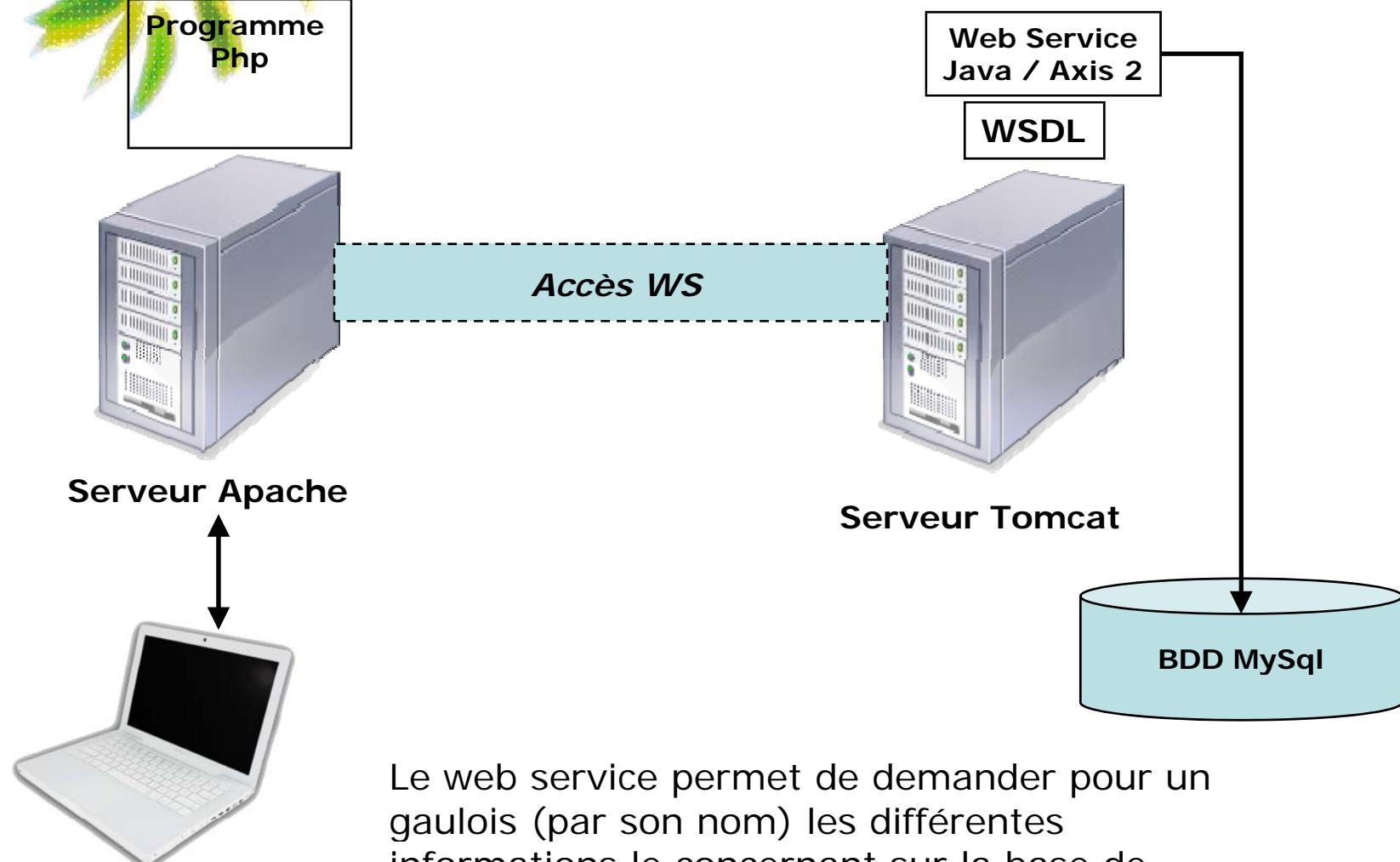


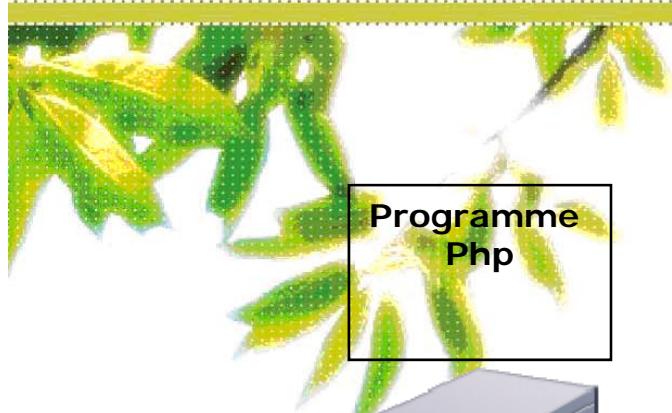
Web Services

Exemple PHP / Java



WS Java / Php - Exemple





WS Java / Php - Exemple

Programme
Php



Web Service
Java / Axis 2

WSDL



Accès WS

En entrée :

On spécifie le nom
du gaulois désiré
(String)

Serveur Tomcat

En retour :

On fournit un objet
UnGaulois

Un Gaulois		
nom	String	
specialite	String	
lieuHabitation	String	
adresse	String	
sesBatailles	[*]	



WSDL - Entête

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<wsdl:definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:tns="http://localhost:8080/lesGaulois2WS/"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    name="lesGaulois2WS"
    targetNamespace="http://localhost:8080/lesGaulois2WS/">
```



WSDL – Les types

- Les types utilisés par le Web Services sont définis par un schéma XML

```
<wsdl:types>
  <xsd:schema targetNamespace="http://localhost:8080/lesGaulois2WS/">
```

```
    <xsd:element name="DonneGaulois">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="nom" type="xsd:string"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
```

Méthode demandée

```
    <xsd:element name="DonneGauloisResponse">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="leGaulois" type="tns:UnGaulois"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
```

Méthode en réponse

Réponse fournie



WSDL – Les types – Structure UnGaulois

```
<xsd:complexType name="UnGaulois">
  <xsd:sequence>
    <xsd:element name="nom" type="xsd:string"></xsd:element>
    <xsd:element name="specialite" type="xsd:string"></xsd:element>
    <xsd:element name="lieuHabitation" type="xsd:string"></xsd:element>
    <xsd:element name="adresse" type="xsd:string"></xsd:element>
    <xsd:element name="sesBatailles" type="tns:LesBatailles"
      minOccurs="0" maxOccurs="unbounded"></xsd:element>
  </xsd:sequence>
</xsd:complexType>
```

```
<xsd:complexType name="LesBatailles"> ←
  <xsd:sequence>
    <xsd:element name="nomBataille" type="xsd:string"></xsd:element>
    <xsd:element name="dateBataille" type="xsd:date"></xsd:element>
    <xsd:element name="lieuBataille" type="xsd:string"></xsd:element>
  </xsd:sequence>
</xsd:complexType>
</xsd:schema>
```



WSDL – Les types – Messages en entrée/sortie

```
<wsdl:message name="DonneGauloisRequest">
    <wsdl:part element="tns:DonneGaulois" name="parameters"/>
</wsdl:message>

<wsdl:message name="DonneGauloisResponse">
    <wsdl:part element="tns:DonneGauloisResponse" name="parameters"/>
</wsdl:message>
```

WSDL – Les types – Le port et binding

```
<wsdl:portType name="lesGaulois2WS">
    <wsdl:operation name="DonneGaulois">
        <wsdl:input message="tns:DonneGauloisRequest"/>
        <wsdl:output message="tns:DonneGauloisResponse"/>
    </wsdl:operation>
</wsdl:portType>

<wsdl:binding name="lesGaulois2WSSOAP" type="tns:lesGaulois2WS">
    <soap:binding style="document"
                  transport="http://schemas.xmlsoap.org/soap/http" />
    <wsdl:operation name="DonneGaulois">
        <soap:operation
                      soapAction="http://localhost:8080/lesGaulois2WS/NewOperation" />
        <wsdl:input>
            <soap:body use="literal" />
        </wsdl:input>

        <wsdl:output>
            <soap:body use="literal" />
        </wsdl:output>
    </wsdl:operation>
</wsdl:binding>
```

Pour la génération de
l'enveloppe SOAP



WSDL – Les types – Le Service Web

```
<wsdl:service name="lesGaulois2WS">
  <wsdl:port binding="tns:lesGaulois2WSSOAP" name="lesGaulois2WSSOAP">
    <soap:address location="http://localhost:8080"/>
  </wsdl:port>
</wsdl:service>

</wsdl:definitions>
```

WS Java / Php – Interrogation WSDL

- Interrogation du WSDL du Web Service (Tomcat) :

<http://localhost:8080/axis2/services/lesGaulois2WS?wsdl>



Mozilla Firefox

Fichier Édition Affichage Historique Marque-pages Outils ?

http://localhost/web...ClientLesGaulois.php http://localhost:808.../lesGaulois2WS?wsdl +

localhost:8080/axis2/services/lesGaulois2WS?wsdl

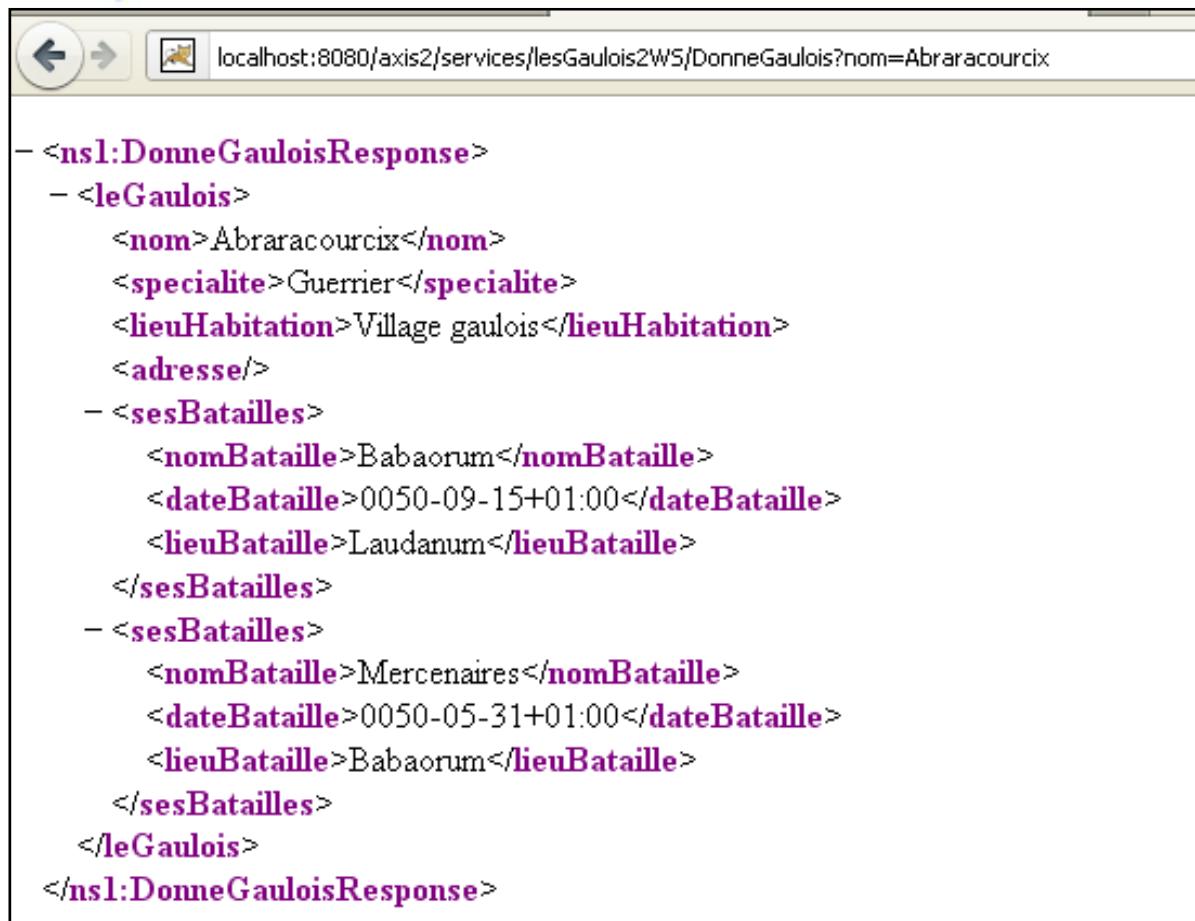
```
</xsd:schema>
</wsdl:types>
- <wsdl:message name="DonneGauloisRequest">
    <wsdl:part name="parameters" element="tns:DonneGaulois"></wsdl:part>
</wsdl:message>
- <wsdl:message name="DonneGauloisResponse">
    <wsdl:part name="parameters" element="tns:DonneGauloisResponse"></wsdl:part>
</wsdl:message>
- <wsdl:portType name="lesGaulois2WS">
    - <wsdl:operation name="DonneGaulois">
        <wsdl:input message="tns:DonneGauloisRequest"></wsdl:input>
        <wsdl:output message="tns:DonneGauloisResponse"></wsdl:output>
    </wsdl:operation>
</wsdl:portType>
- <wsdl:binding name="lesGaulois2WSSOAP" type="tns:lesGaulois2WS">
    <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
    - <wsdl:operation name="DonneGaulois">
        <soap:operation soapAction="http://localhost:8080/lesGaulois2WS/NewOperation"/>
        - <wsdl:input>
            <soap:body use="literal"/>
        </wsdl:input>
        - <wsdl:output>
            <soap:body use="literal"/>
        </wsdl:output>
```



WS Java / Php – Interrogation d'un gaulois

- Interrogation du WSDL du Web Service (Tomcat) pour un gaulois :

<http://localhost:8080/axis2/services/lesGaulois2WS/DonneGaulois?nom=Abraracourcix>



The screenshot shows a web browser window with the URL `localhost:8080/axis2/services/lesGaulois2WS/DonneGaulois?nom=Abraracourcix` in the address bar. The page content displays an XML response:

```
- <ns1:DonneGauloisResponse>
  - <leGaulois>
    <nom>Abraracourcix</nom>
    <specialite>Guerrier</specialite>
    <lieuHabitation>Village gaulois</lieuHabitation>
    <adresse/>
  - <sesBatailles>
    <nomBataille>Babaorum</nomBataille>
    <dateBataille>0050-09-15+01:00</dateBataille>
    <lieuBataille>Laudanum</lieuBataille>
  </sesBatailles>
  - <sesBatailles>
    <nomBataille>Mercenaires</nomBataille>
    <dateBataille>0050-05-31+01:00</dateBataille>
    <lieuBataille>Babaorum</lieuBataille>
  </sesBatailles>
</leGaulois>
</ns1:DonneGauloisResponse>
```

WS Java / Php – Interrogation d'un gaulois

- Programme PHP :

```
<?php
// Accès au WSDL
$wsdl= "http://localhost:8080/axis2/services/lesGaulois2WS?wsdl";
// Instanciation du service Web
$service=new SoapClient($wsdl, array('encoding'=>'ISO-8859-1'));
// Accès à la méthode du service Web
$Reponse=$service->DonneGaulois(array("nom"=>"Abraracourcix"));

// Recuperation des éléments
echo "Nom : ".$Reponse->leGaulois->nom."<br>";
echo "Specialite : ".$Reponse->leGaulois->specialite."<br>";
echo "lieu d habitation : ".$Reponse->leGaulois->lieuHabitation."<br>";
echo "adresse : ".$Reponse->leGaulois->adresse."<br>";
// Ses Batailles
echo "Ses Batailles : ".count($Reponse->leGaulois->sesBatailles)."<br>";
for($i=0;$i<count($Reponse->leGaulois->sesBatailles);$i++)
{
    echo "  |--> "
    . $Reponse->leGaulois->sesBatailles[$i]->nomBataille.
    " - "
    . $Reponse->leGaulois->sesBatailles[$i]->dateBataille.
    " - "
    . $Reponse->leGaulois->sesBatailles[$i]->lieuBataille.
    "<br>";
}
```

WS Java / Php – Interrogation d'un gaulois

- Programme PHP :



```
http://localhost/webSe.../ClientLesGaulois.php +  
localhost/webServicePhp/ClientLesGaulois.php  
  
Nom : Abraracourcix  
Specialite : Guerrier  
lieu d habitation : Village gaulois  
adresse :  
Ses Batailles : 2  
|--> Babaorum - 0050-09-15+01:00 - Laudanum  
|--> Mercenaires - 0050-05-31+01:00 - Babaorum
```



Langages d'interface utilisateur

XML



SVG - Introduction

- SVG signifie Scalable Vector Graphics
- SVG est utilisé pour définir des graphiques vectoriels pour le Web
- SVG définit les graphismes au format XML
- Les graphiques SVG ne perdent pas de qualité si elles sont agrandies ou redimensionnées
- Chaque élément et chaque attribut dans les fichiers SVG peuvent être animés
- SVG est une recommandation du W3C
- SVG intègre les normes du W3C tels que le DOM et XSL



SVG - Avantages

- Avantages de l'utilisation de SVG dans autres formats d'image (comme JPEG et GIF) sont:
 - Images SVG peuvent être créés et édités avec n'importe quel éditeur de texte
 - Images SVG peuvent être recherchés, indexés, scripté, et comprimé
 - Images SVG sont évolutives
 - Images SVG peuvent être imprimés de haute qualité à toute résolution
 - Images SVG sont zoomables (et l'image peut être agrandie sans dégradation)
 - SVG est un standard ouvert
 - Les fichiers SVG sont pures XML
- Le principal concurrent de SVG est Flash
- Le plus grand avantage de SVG a plus de Flash est le respect des autres normes (par exemple, XSL et DOM)
- Flash s'appuie sur une technologie exclusive qui n'est pas open source.



SVG - Exemple

Cette DTD réside à [w3.org](http://www.w3.org), et contient tous les éléments admissibles SVG.

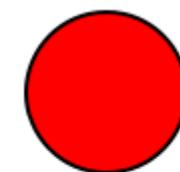
```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG11/DTD/svg11-flat-20030114.dtd">

<svg xmlns="http://www.w3.org/2000/svg" width="100%" height="100%">
  <desc>
    Un cercle
  </desc>
  <circle
    cx="100"
    cy="50"
    r="40"
    stroke="black"
    stroke-width="2"
    fill="red" />
</svg>
```

Elément racine

Description du graphique

Création d'un cercle





XAML – eXtensible Application Markup Language

- XAML est l'acronyme de eXtensible Application Markup Language
 - Prononcer 'Zammel'
 - Langage à balises pour applications extensibles.
 - Sont extensibles les applications avec l'ajout de composant pendant l'exécution et le langage lui-même avec de nouveaux composants.
- Dans le nom XAML, la lettre A originellement représentait Avalon, le nom provisoire de la couche graphique de Windows Vista, renommée par la suite WPF (Windows Présentation Foundation).
- Utilise la plate-forme .NET et le plug-in Silverlight
- XAML est un format XML dans lequel peuvent être intégrées des commandes, associées à des propriétés des éléments de la même façon que dans JavaScript:
Exemple: appel de la fonction OnClick() quand on clique sur le bouton.

```
<button Click="OnClick"> </button>
```



XAML – Exemples

- Afficher le message "Salut le Monde!"

```
<Page xmlns="">
    <TextBlock> Salut, le Monde! </TextBlock>
</Page>
```

- Une fenêtre complète

```
<windows Width="600" Height="480" Text="Mon Programme">
    <FlowPanel>
        <Label Name="Montexte" FontSize="20"> Mon application </label>
        <Button Width="80" Click="BoutonClic"> Fermer </Button>
    </FlowPanel>
</windows>
```

XUL – XML-based User interface Language

- **Histoire :**

- XUL a été défini pour construire l'interface utilisateur de Mozilla, navigateur open source.
- Le nom vient d'une créature du film Ghostbuster.
- XUL se prononce "zool".

- **But :**

- XUL fournit à l'interface utilisateur les bases pour une application portable
- De même que DHTML a été créé pour l'interface des pages web, XUL l'est pour les applications.

- **En quoi consiste XUL?**

- XUL décrit un contenu avec un comportement, puis une présentation pour ce contenu, et ensuite on peut localiser l'interface pour la langue d'un pays.

- Les composantes de chaque niveau sont les suivants:

- 1) niveau contenu: XUL, XBL, JavaScript,
- 2) comportement: gestion des évènements,
- 3) présentation: CSS, images,
- 4) local: DTD, fichiers de propriétés.



XUL – Exemple

Exemple04.xul

```
<?xml version='1.0'?>
<window
  id="menu"
  title="Menu"
  width="400"
  height="300"
  xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul" >

<toolbox flex="1">
  <menubar>
    <menu label="Fichiers">
      <menupopup>
        <menuitem id="openMenu" label="Ouvrir" onCommand="openFun" />
        <menuitem id="closeMenu" label="Fermer" />
        <menuseparator />
        <menuitem id="exitMenu" label="Quitter" />
      </menupopup>
    </menu>
    <menu label="Aide" >
      <menupopup>
        <menuitem label="Manuel" />
      </menupopup>
    </menu>
  </menubar>
</toolbox>
...
```

MXML – Macromédia Flex Markup Language

- Deux types principaux de composants sont disponibles :
 - les **conteneurs** (boîtes, panneaux, fenêtres, etc.)
 - les **éléments de contrôle** (champs texte, listes, datagrids, tree, etc.)
- Eléments d'interface évolués propriétaire
- Collaboration avec Action Script

MXML – Exemple



SimpleComboBox.mxml

```
<?xml version="1.0"?>
<!-- Simple example to demonstrate the ComboBox control. -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>
        <![CDATA[
            [Bindable]
            public var cards: Array = [ {label:"Visa", data:1},
                {label:"MasterCard", data:2}, {label:"American Express", data:3} ];

            [Bindable]
            public var selectedItem:Object;
        ]]>
    </mx:Script>

    <mx:Panel title="ComboBox Control Example"
        height="75%" width="75%" layout="horizontal"
        paddingTop="10" paddingBottom="10" paddingLeft="10" paddingRight="10">

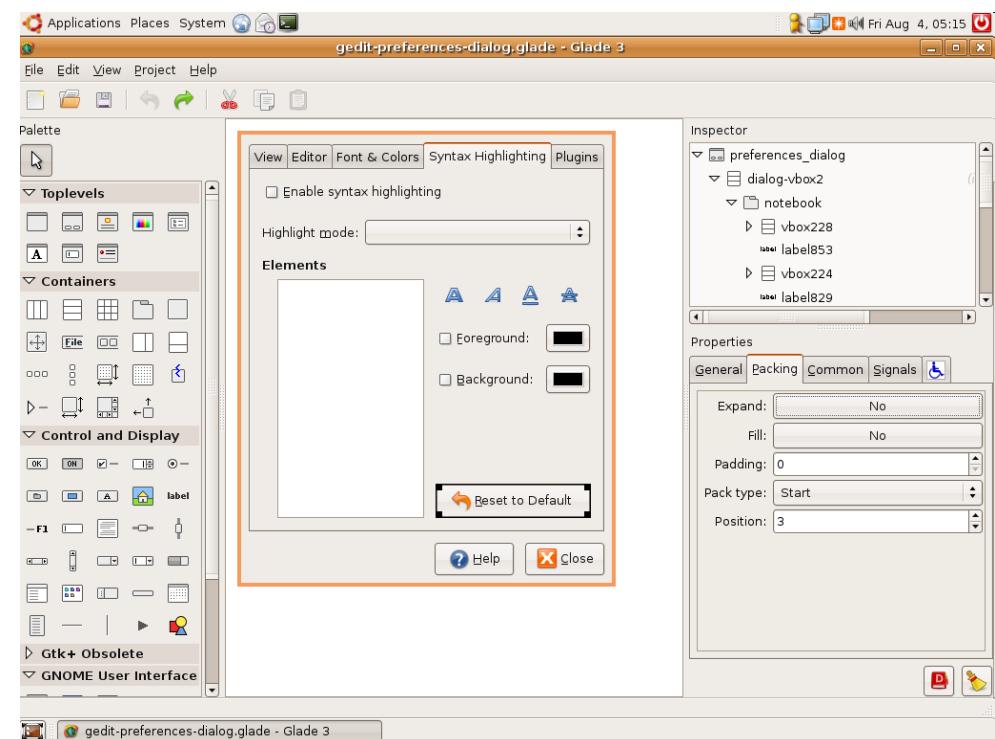
        <mx:ComboBox dataProvider="{cards}" width="150"
            close="selectedItem=ComboBox(event.target).selectedItem"/>

        <mx:VBox width="250">
            <mx:Text width="200" color="blue" text="Select a type of credit card."/>
            <mx:Label text="You selected: {selectedItem.label}"/>
            <mx:Label text="Data: {selectedItem.data}"/>
        </mx:VBox>

    </mx:Panel>
</mx:Application>
```

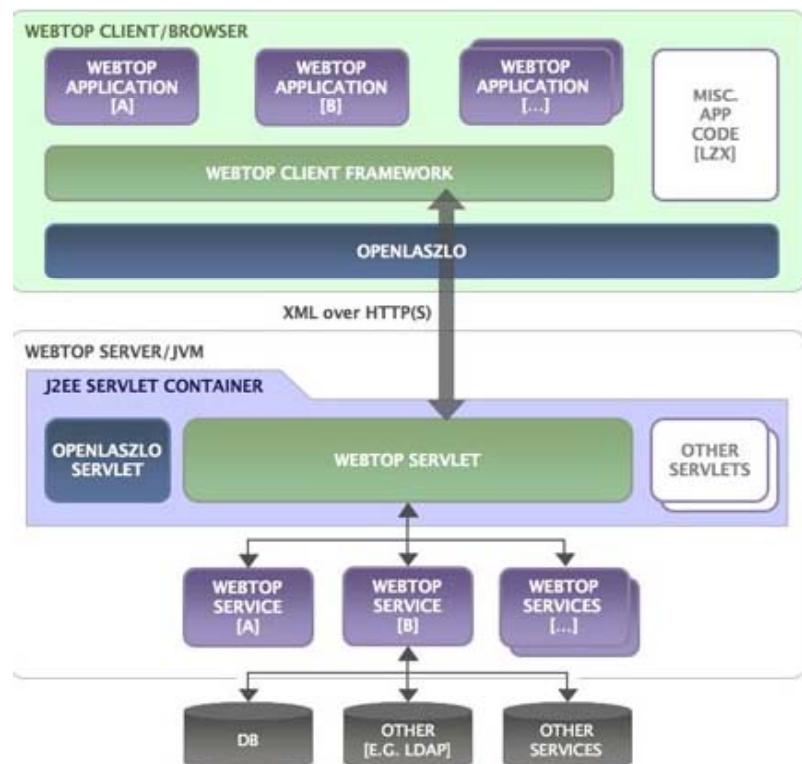
GladeXML

- **GladeXML** est le format XML utilisé par le logiciel de création d'interface graphique Glade.
- Il crée des formulaires qui seront utilisés en conjonction avec la bibliothèque **libglade** basée sur GTK+.
- Glade fournit une interface de développement comme Visual Studio, C++ Builder ...



OpenLaszlo

- OpenLaszlo est une plateforme open source pour le développement d'applications web avec une interface utilisateur graphique.
- Il est distribué sous licence OSI Common Public License .
- La plateforme est constituée du langage de programmation LZX et du serveur OpenLaszlo.





UIML - User Interface Markup Language

- Le but de UIML est de créer un standard ouvert de description d'interface utilisateur en XML, qui puisse être implémenté facilement.
- L'intérêt est :
 - de faciliter l'existence de meilleurs outils pour la création d'interfaces utilisateur
 - et qui fonctionne sur toutes les plateformes actuelles,
 - mais qui permette aussi aux interfaces actuelles d'évoluer en un format leur permettant de tourner sur les plateformes anciennes.



XForms

- Le standard XForms a été défini par le W3C pour combiner XML et formulaires sur le web.
- Le standard est conçu pour être plus général et permet d'entrer des données dans les applications bureautiques.
- Il remplace dans XHTML le format de formulaire utilisé jusqu'ici avec HTML.
- Il a trois parties:
 - XForms User Interface, l'interface utilisateur fournit les contrôles graphiques destinés à remplacer ceux des formulaires HTML. Ces contrôles peuvent être utilisés dans les documents XML et donc dans les formats dérivés de XML.
 - XForms définit aussi un format XML structuré pour les données collectées avec les contrôles XForms.
 - La troisième partie, XForms Submit Protocol, (protocole de soumission XForms) définit comment les données sont envoyées et reçues.

xWidglet

- Une offre commerciale pour un autre langage d'interface basé sur XML.
- Pour son utilisation, un atelier de développement est fourni ainsi qu'un navigateur spécialisé qui affiche les interfaces.
- Basé sur Java.

