XML et outils associés - Cefora/Technofutur TIC

Table des matières succinte

Chapitre 1 - Introduction à XML et aux normes associées

Table des matières de ce chapitre

XML, c'est quoi ?

XML = **Extensible Markup Language**, créé en 1998 par le W3C (World Wide Web Consortium)
XML est un langage de structuration de données, rien de plus

- Ce n'est pas un langage évolué (présenté à ses débuts comme l'**ASCII du futur**)
- Ce n'est pas un langage performant
- Ce n'est pas un langage révolutionnaire

Ses avantages:

- C'est un langage universel
- Il est adopté par tout le monde (il est de loin le langage informatique le plus utilisé au monde)
- Il se prête à d'innombrables utilisations
- C'est un langage stable et pérenne
- De nombreux outils ont été développés pour l'utiliser

XML, c'est quoi ?

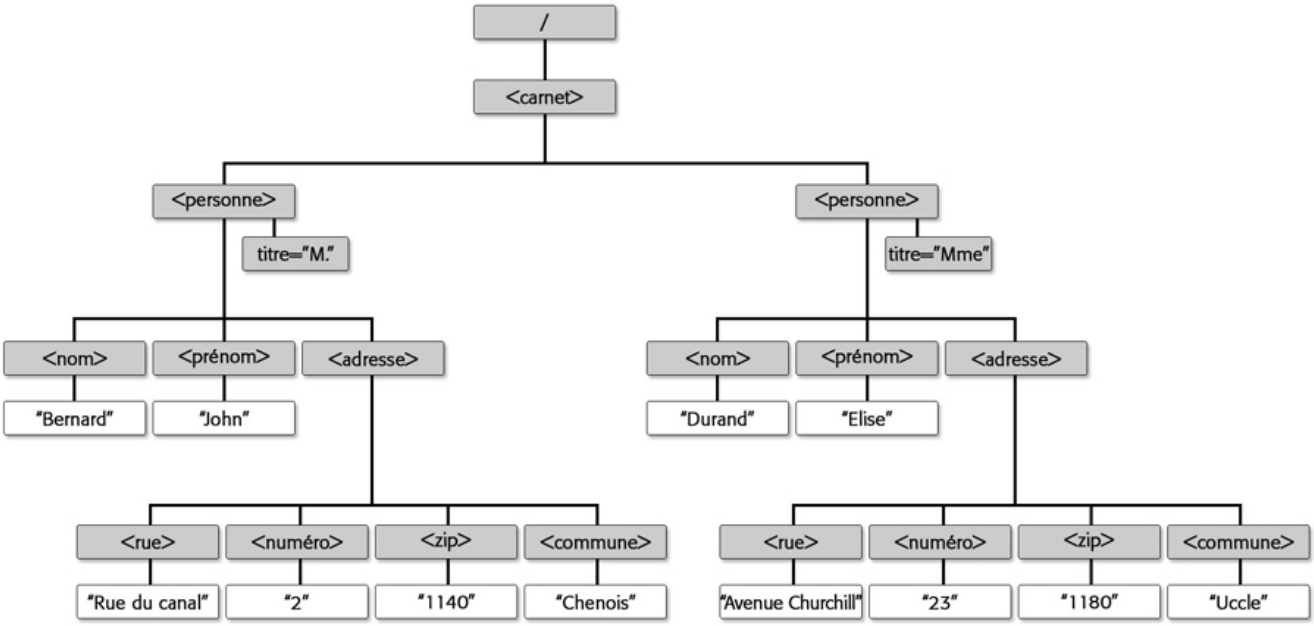
XML est un langage qui permet de créer des **documents XML**

- Comme HTML, il utilise des **balises** et des **attributs** pour marquer les données contenues dans le document

  Les balises et les attributs sont choisis librement par l'utilisateur et ne sont pas imposés dans le langage
- Le seul but est de donner une **structure** au document afin de caractériser son contenu
- Le document garde la même simplicité et la même lisibilité qu'HTML, mais permet de le manipuler de manière informatique

```
<?xml version="1.0" encoding="UTF-8"?>
<carnet>
  <personne titre="M.">
    <nom>Bernard</nom>
    <prénom>John</prénom>
    <adresse>
      <rue>Rue du canal</rue>
      <numéro>2</numéro>
      <zip>1140</zip>
      <commune>Chenois</commune>
    </adresse>
  </personne>

  <personne titre="Mme">
    <nom>Durand</nom>
    <prénom>Elise</prénom>
    <adresse>
      <rue>Avenue Churchill</rue>
      <numéro>23</numéro>
      <zip>1180</zip>
      <commune>Uccle</commune>
    </adresse>
  </personne>
</carnet>
```

Un arbre document (document tree)



un exemple XML comparé avec un exemple HTML

```xml
<?xml version="1.0" encoding="iso-8859-1"?>
<carnet>
  <personne titre="M.">
    <nom>Bernard</nom>
    <prénom>John</prénom>
    <adresse>
      <rue>Rue du canal</rue>
      <numéro>2</numéro>
      <zip>1140</zip>
      <commune>Chenois</commune>
    </adresse>
  </personne>

  <personne titre="Mme">
    <nom>Durand</nom>
    <prénom>Elise</prénom>
    <adresse>
      <rue>Avenue Churchill</rue>
      <numéro>23</numéro>
      <zip>1180</zip>
      <commune>Uccle</commune>
    </adresse>
  </personne>
</carnet>
```

```xml
<html>
  <head>
    <title>carnet d'adresses</title>
  </head>
  <body>
    <section style="font-family: Arial;">
      <h2>Carnet d'adresses</h2>
      <p>
        <b>M. John Bernard</b><br/>
        Rue du canal, 2<br/>
        1140 Chenois
      </p>
      <p>
        <b>Mme Elise Durand</b><br/>
        Avenue Churchill, 23<br/>
        1180 Uccle
      </p>
    </section>
  </body>
</html>
```

L'origine et les objectifs du XML

XML est un sous-ensemble de **SGML**, dans une version moderne, simplifiée et adaptée à l'Internet. SGML date de 1971 !

Buts recherchés lors de la conception de XML:

- il devait être utilisable sur Internet
- il devait servir à une large variété d'applications
- il devait rester compatible avec SGML (malheureusement)
- les documents devaient être manipulables par des programmes informatiques
- les applications devaient être simples à concevoir et à développer
- les documents devaient rester lisibles et faciles à créer

Les normes associées au XML

XML ne sert pas à grand chose seul, il est destiné à être traité par des applications qui vont donner une signification aux balises et aux attributs.

Certaines de ces applications vont utiliser des **langages**: XSLT, Xpath, XForms, XQuery...
Ces langages sont définis par des normes développées par le W3C, par OASIS, mais également l'IETF, l'ISO, etc. On parle de plusieurs centaines de normes.

D'autres normes vont mettre en oeuvre le XML pour structurer des informations. Celles-ci sont encore bien plus nombreuses (plusieurs dizaines de milliers de normes).

Les normes du socle de base

XML

- Le language existe en deux versions: 1.0 et 1.1
- Il n'est pas destiné à évoluer, afin de garantir la stabilité des données. La version 1.0 reste de loin la plus utilisée

DTD (Document Type Definition)

- Sorte de **grammaire** qui décrit un **vocabulaire** pour une **classe de documents XML** (issue du SGML, les DTD sont à l'heure actuelle un peu dépassées)

XML Namespaces (espaces de noms)

- Garantit le choix libre des noms de balises et d'attributs

XML Infoset

- Décrit ce qui fait réellement partie du document XML

XML schema

- Permet de créer des **schémas** pour définir la **structure** et le **contenu** d'une **classe de documents XML** (version moderne des DTD)

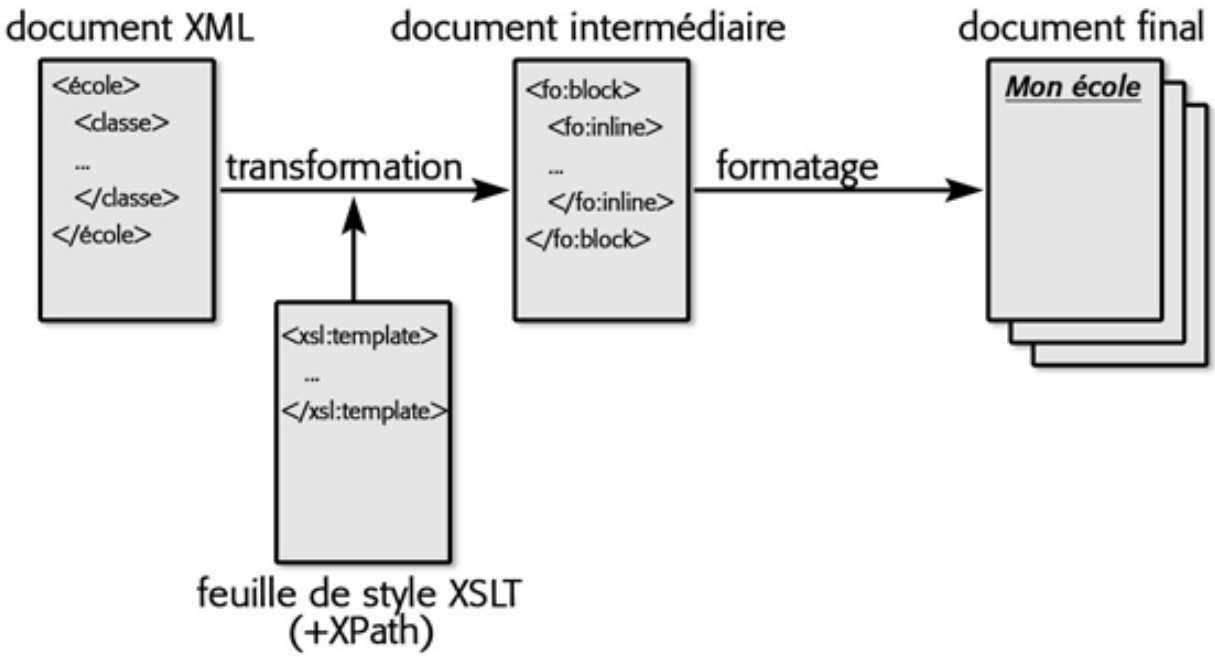
Les normes de mise en page (mise en forme)

CSS (Cascading Style Sheet)

- Permet de publier un document XML dans un navigateur web (solution basique)

Feuille de styles pour un document XML

- Une feuille de styles est définie par 3 langages: **XSLT**, **XPath** et **un langage de présentation**



XSLT (eXtended Stylesheet Language for Transformation)

- Sert à transformer un document XML en un autre document (XML, HTML, XHTML, texte...)
- Le terme de **feuille de transformation** est plus adéquat que celui de feuille de styles

XPath

- Sert à définir des **expressions**, similaires à celles utilisées dans d'autres langages informatiques, destinées à calculer un résultat (**ensemble de noeuds**, chaînes de caractères, valeurs numériques...)
- Ce résultat est souvent obtenu à partir du contenu du document XML. Exemple: **/carnet/personne[1]/nom**
- Xpath est toujours au service d'un autre langage: XSLT, XPointer, XForms, XQuery...

Langage de présentation ou **langage de sortie**

- Le langage utilisé pour le résultat d'une transformation XSLT peut être choisi (HTML, XHTML, SVG, SMIL, XSL...)
- Un des langages de présentation possible est le **XSL** (eXtended Stylesheet Language, parfois appelé XSL-FO), adapté au XML (est l'équivalent du HTML+CSS mais pour des pages de format fixe: A4, US-Letter...)

à retenir:

- **XSLT** est un langage générique de **transformation**
- **XPath** est un langage générique d'**interrogation**

Les normes pour utiliser XML dans un navigateur

XLink

- Définit des liens hypertextes (élaborés) entre documents XML

XPointer

- Etend la notion d'URI afin d'accéder à des parties d'un document XML

XML fragment

- Permet d'extraire des fragments d'un document XML en conservant le contexte

XForms

- Permet de créer des formulaires en XML

XFrames

- Permet de définir des frames, comme en HTML

XML Events

- Permet la gestion d'évènements au sein d'un document XML

Les normes techniques

XML Base

- Définit la base pour résoudre les URL relatives

XML Id

- Permet d'ajouter des identificateurs aux éléments d'un document XML

XML Inclusions (XInclude)

- Permet d'inclure un document XML ou une partie d'un document XML dans un autre document

XML Signature

- Permet de signer numériquement le tout ou une partie d'un document XML (authentification, contrôle d'intégrité...)

XML Key Management System (XKMS)

- Gestion de clés publiques

XML Encryption

- Encryption du tout ou une partie d'un document XML

XProc (An XML Pipeline Language)

- Permet d'automatiser des traitements (workflow) sur un document XML, en créant un pipeline XML (chaîne de transformation)

Les normes utilisées en programmation

DOM (Document Object Model)

- Interface de programmation orienté objets pour lire et éventuellement modifier un document XML

SAX (Simple API for XML)

- Interface de programmation pour lire un document XML, nettement plus rudimentaire que DOM, mais plus rapide et capable de traiter des documents très volumineux

SOAP (Simple Object Access Protocol)

- Format d'échange de messages dans un environnement client/serveur, utilisé dans les web services

WSDL (Web Service Description Language)

- Format XML de description d'un web service (décrit les méthodes, les paramètres, les réponses...)

Les normes pour les bases de données

XML Query est un projet visant à créer des **bases de données**, où les tables sont remplacées par des **documents XML**
(le but est de supplanter les bases de données relationnelles dont les principes ont plus de 45 ans !)

Le moteur de base de données peut au niveau physique librement implémenter la base de données, mais au niveau logique, les données seront des **arbres XML** (instances XDM).

- **XQuery** est un langage de requête au sein de ces bases de données (l'équivalent du SELECT en SQL)
- **XQuery Update Facility** permet de modifier la base de données (l'équivalent du INSERT, UPDATE et DELETE en SQL)

Les normes basées sur une structure XML

XHTML

〉 Reformulation de HTML en XML

RSS (Really Simple Syndication)

〉 Syndication = "Vente de contenu". Format de diffusion d'informations diverses (actualités, nouveautés, etc.)

RDF (Ressouce Description Framework)

〉 Meta-données au format XML pour décrire les ressources du web

SVG (Scalable Vector Graphics)

〉 Images vectorielles

SMIL (Synchronized Multimedia Integration Language)

〉 Animations multimédia

SSML (Speech Synthesis Markup Language)

〉 Synthèse vocale

InkML (Ink Markup Language)

〉 Digitalisation d'informations manuscrites (signatures...)

GML (Geography Markup Language)

〉 Informations géographiques

CML (Chemical Markup Language)

〉 Données chimiques (molécules, réactions, spectres...)

XBRL (eXtensible Business Reporting Language)

〉 Standard servant à communiquer des données financières

Des milliers d'autres...

Chapitre 2 - Le langage XML

Table des matières de ce chapitre

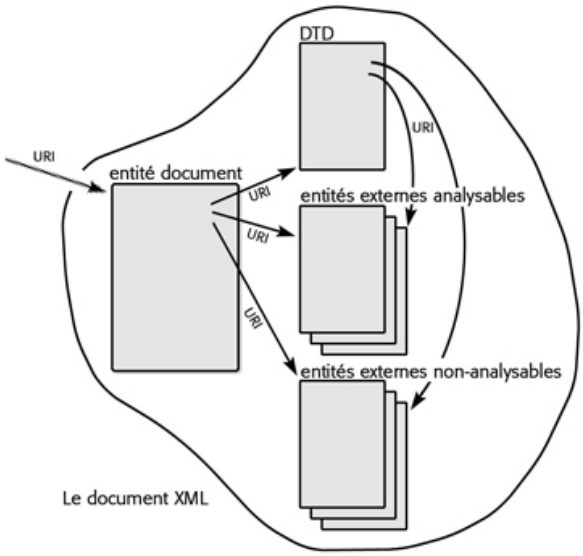
Le jargon XML

Un **parseur XML** (XML parser) ou **processeur XML (XML processor)**

- Module logiciel utilisé pour traiter un document XML et qui fonctionne sous le contrôle d'une **application**

Les **entités XML**

- Un document est composé de une ou plusieurs unités de stockage, appelées **entités**, référencées à l'aide d'URI
- L'entité principale s'appelle l'**entité document**

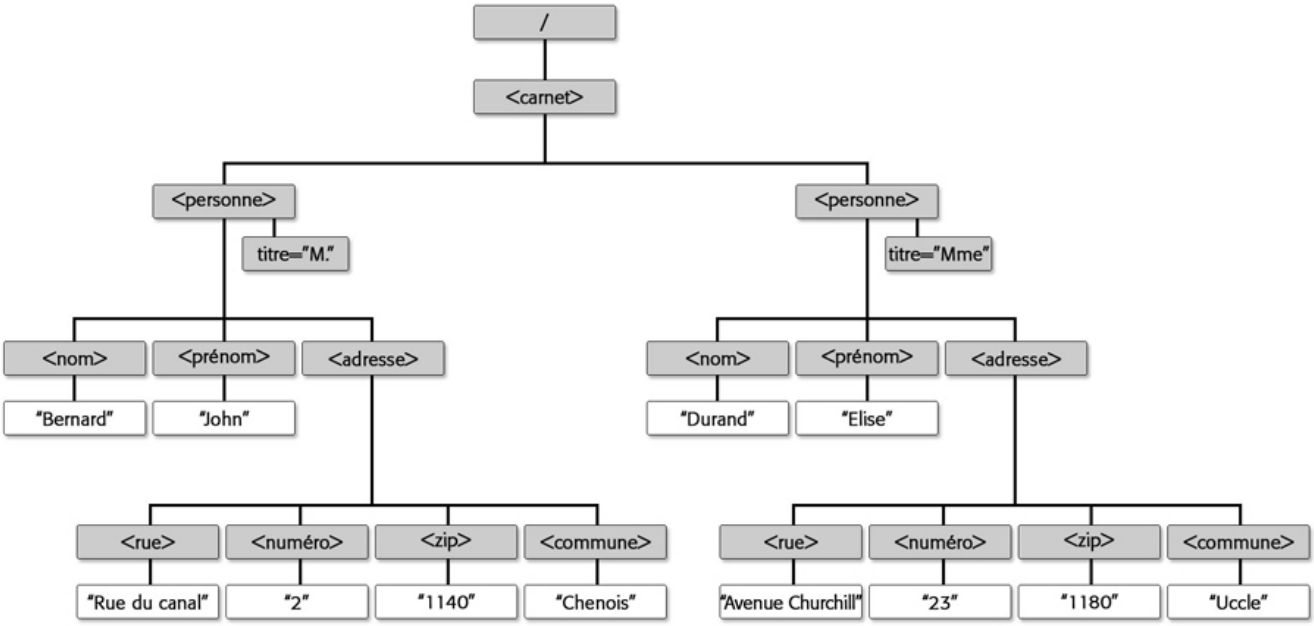


Les **éléments**

- Un élément est constitué:
  - d'une **balise ouvrante** avec éventuellement des **attributs**
  - d'un contenu
  - d'une **balise fermante**

Le jargon XML - suite

Un document XML peut être représenté sous la forme d'une arborescence de **noeuds**, appelée l'**arbre document** (document tree) ou **instance XDM**



On distingue les noeuds éléments, les noeuds attributs, les noeuds textes, les noeuds commentaires...
Les noeuds sont organisés suivant des relations **parent/enfants**

Les balises (qui forment des éléments)

Les **balises** doivent:

- toujours aller de pair (il existe une syntaxe abrégée pour les éléments vides)
- respecter les minuscules et les majuscules
- commencer par une lettre, un '_' ou un ':' suivi d'une combinaison de lettres, de chiffres, '_', '-', ':' ou '.'

La notion de lettre est à prendre au sens large (tel que le définit Unicode)

Remarque: Le ':' doit être réservé aux espaces de noms

```
<téléphone>
  ...
</téléphone>

<Téléphone>
  ...
</Téléphone>

<numéro-de-téléphone>
  ...
</numéro-de-téléphone>

<numéroDeTéléphone>
  ...
</numéroDeTéléphone>

<##>
  ...
</##>
```

Les balises vides

Les balises vides peuvent s'écrire:

```
<référence _empty="no"></référence>

<référence id="3287" _empty="no"></référence>

<référence/>

<référence id="3287"/>
```

Les attributs

Les **attributs** doivent:

- toujours posséder une valeur (qui peut être vide)
- toujours définir la valeur entre guillemets ou apostrophes
- respecter les minuscules et les majuscules
- commencer par une lettre, un '_' ou un ':' suivi d'une combinaison de lettres, de chiffres, '_', '-', ':' ou '.'

La notion de lettre est à prendre au sens large (tel que le définit Unicode)

Remarque: Le ':' doit être réservé aux espaces de noms

```
<livre isbn="312-12345-23">
  ...
</livre>

<article prix="320" devise="FB"/>

<## #="+32 2 650 22 22"/>
```

Les éléments

Les **éléments** sont les briques de base d'un document XML, ils sont composés:

- d'une balise ouvrante et d'une balise fermante
- du contenu entre les deux balises (autres éléments éventuels, données textuelles...)

```
<?xml version="1.0" encoding="UTF-8"?>
<livre>
  <titre>XML</titre>
  <chapitre>
    <titre>Introduction</titre>
    <p>XML est l'acronyme de...</p>
    <p>Comme HTML, XML utilise...</p>
  </chapitre>
  <chapitre>
    <titre>Le langage XML</titre>
    <p>Un document se compose...</p>
  </chapitre>
  <annexe>
    <titre>Glossaire</titre>
    <p>XML Schema...</p>
  </annexe>
</livre>
```

Les données textuelles

Les **données textuelles** sont contenues dans certains éléments et dans la valeur des attributs, elles sont composées :

- de caractères imprimables
- tabulations
- carriage return
- linefeed

(En XML 1.1, les autres caractères de contrôle, excepté le 0, sont acceptés sous la forme: &#x01; &#x02; ...)

```
<?xml version="1.0" encoding="UTF-8"?>
<memos>
  <memo date="30/5/2017">
    <p>
      Ce paragraphe pour expliquer le
      traitement des caractères blancs
      en XML
    </p>
  </memo>
</memos>
```

Les caractères **<** et **&** doivent être codés par **<** et **&**

Les caractères **blancs** sont préservés par défaut

Les données textuelles - zones CDATA

Une donnée textuelle peut contenir des **zones de données non structurées** dans lesquelles les caractères < et & sont traités comme tous les autres caractères

Elles débutent par **<![CDATA[** et se terminent par **]]>**

Les zones de données non structurées, ou **zones CDATA**, peuvent contenir n'importe quelle suite de caractères à l'exclusion de **]]>**

```
<?xml version="1.0" encoding="UTF-8"?>
<doc>
  <![CDATA[
     Ce document est</i> un document XML tout
    à fait <bold<underline>>bien formé<i>
  ]]>
</doc>
```

Les données textuelles - jeu de caractères et le type d'encodage

XML distingue le **jeu de caractères** utilisé et le **type d'encodage** utilisé.

le **jeu de caractères** utilisé est **UNICODE** ou **ISO/IEC10546**:

- jeu de caractères universel
- supporte toutes les langues et dialectes du monde
- http://www.unicode.org/

```
<?xml version="1.0" encoding="UTF-8"?>
<contact>
  <prénom>Amélie</prénom>
  <nom>Païva</nom>
  <adresse>
    <rue>drève de l'écluse</rue>
    <numéro>3</numéro>
    <zip>12180</zip>
    <commune>Bézier</commune>
  </adresse>
</contact>
```

```
<?xml version="1.0" encoding="ISO-8859-1"?>
```

le **type d'encodage** peut varier d'un document à l'autre:

- il doit être déclaré dans l'en-tête du document
- il doit toujours servir à encoder les caractères UNICODE
- UTF-8 et UTF-16 doivent être supportés par tous les parseurs
- ISO-8859-1 est utilisé pour les PC/Windows

```
<contact>
  <prénom>Amélie</prénom>
    <nom>Païva</nom>
    <adresse>
      <rue>drève de l'écluse</rue>
      <numéro>3</numéro>
        <zip>12180</zip>
        <commune>Bézier</commune>
    </adresse>
</contact>
```

Les données textuelles - références de caractère

Les **références de caractère** permettent de coder les caractères non directement supportés par le type d'encodage, en utilisant le code **Unicode** du caractère

La syntaxe :

- **&#** code_décimal **;**
- **&#x** code_hexadécimal **;**

```
<?xml version="1.0" encoding="UTF-8"?>
<contact>
  <prénom>Am&#233;lie</prénom>
  <nom>Pa&#xEF;va</nom>
  <adresse>
    <rue>dr&#232;ve de l'&#xE9;cluse</rue>
    <numéro>3</numéro>
    <zip>12180</zip>
    <commune>B&#xE9;zier</commune>
  </adresse>
</contact>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<data>
  <p>
    En cyrillique, le mot UNICODE s'écrit:
    &#x042E;&#x043D;&#x0438;&#x043A;&#x043E;&#x0434;
  </p>
</data>
```

Les donnes textuelles - références d'entité

- Les **références d'entité** font référence à une **entité** (pour rappel, une entité est une unité de stockage qui contient une partie du document XML)
- Le contenu des entités est appelé **texte de remplacement**
- Les entités peuvent être **internes** ou **externes**
- syntaxe **&nom;**

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE livre [
<!ENTITY dateParution "31 avril 2017">
]>
<livre>
  <p>Le livre est paru le &dateParution;.</p>
</livre>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE livre [
<!ENTITY dateParution SYSTEM "date.xml">
]>
<livre>
  <p>Le livre est paru le &dateParution;.</p>
</livre>
```

Contenu de l'entité:

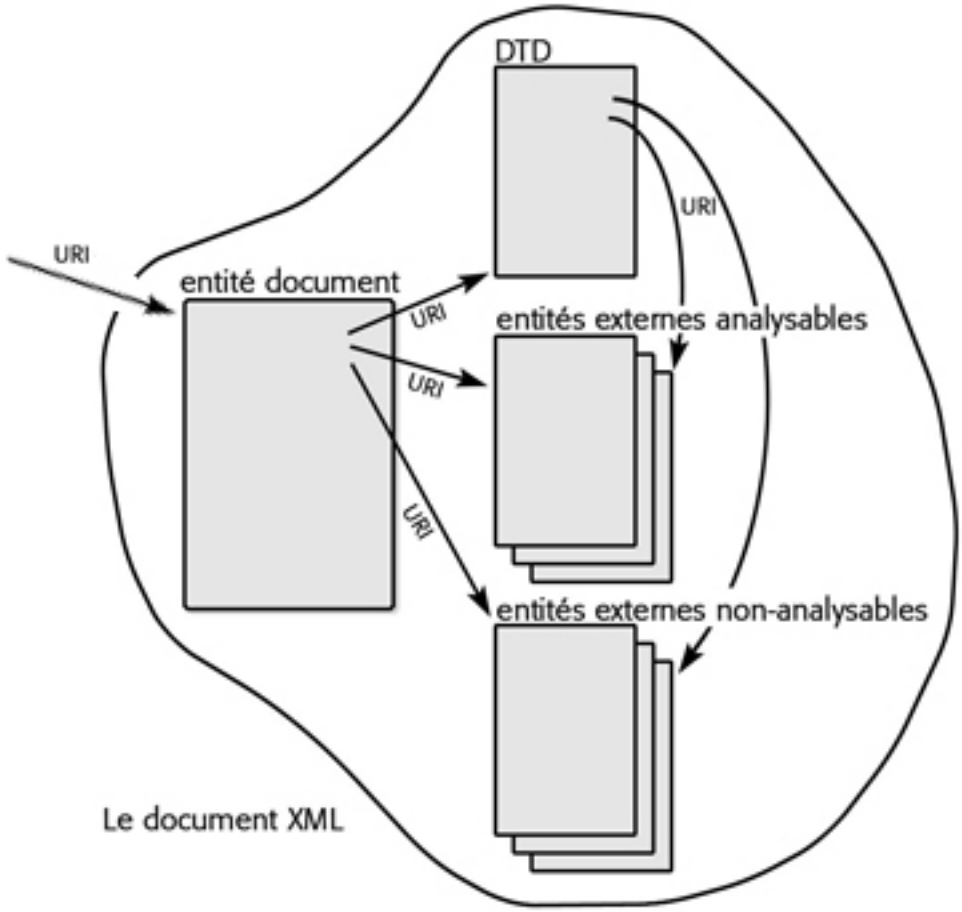
```
<?xml version="1.0" encoding="UTF-8"?>
31 avril 2017
```

Les entités

La structure physique est composée d'unités de stockage appelées **entités**. Elles peuvent être de différents types:

- **analysables** ou **non-analysables**
- **internes** ou **externes**
- **générales** ou **paramètres**

Un document XML possède toujours une entité appelée **entité document** (les autres sont de moins en moins souvent utilisées)



Le contenu d'une entité est appelé **texte de remplacement**. Le texte de remplacement d'une entité analysable peut être inclus dans le document à l'aide d'une référence d'entité:

- **&**nom**;** (pour les entités générales, au sein du code XML)
- **%**nom**;** (pour les entités paramètres, au sein de la DTD)

Les données textuelles - entités prédéfinies

Il n'y a que cinq entités prédéfinies en XML:

```
&amp;          &

&lt;           <

&gt;           >

&quot;         "  (guillemet)

&apos;         '  (apostrophe)
```

HTML en comptait plus de 250. Les caractères qu'elles représentaient sont pris en charge directement en UNICODE (il n'y a plus d'entités prédéfinies, par exemple, pour les caractères accentués)

Les commentaires

Un commentaire se note comme en HTML, entre **<!--** et **-->** (la séquence '--' ne peut pas se retrouver dans le commentaire)

```
<!-- commentaire -->

<!--
  un commentaire peut être placé sur plusieurs
  lignes de textes et contenir des <b>balises</b>
 -->
```

Les instructions de traitement

Les **instructions de traitement** sont destinées aux applications qui vont exploiter le contenu du document
Elles commencent par **<?** suivi d'un nom, d'une valeur et se terminent par **?>**

```
<?xml version="1.0"  encoding="ISO-8859-1"?>

<?xml-stylesheet href="cours.xsl" type="text/xsl"?>

<?cocoon-process type="xslt"?>
```

Les notations

La notion de notation est assez difficile à comprendre et n'est pratiquement jamais utilisée
Une notation désigne par un nom, appelé **nom de notation**, et associe un ou deux identificateurs, appelé **identificateur public** et **identificateur externe** :

- le contenu d'une entité externe non-analysable
- le contenu d'un élément qui possède un attribut de type NOTATION
- le contenu d'une instruction de traitement

Les attributs particuliers

L'attribut **xml:space** indique s'il faut préserver ou non les caractères blancs (il doit être déclaré dans la DTD)

```
<p xml:space="preserve">
  garder les blancs
</p>

<p xml:space="default">
  traiter normalement les blancs
</p>
```

L'attribut **xml:lang** spécifie la langue utilisée dans un élément (il doit être déclaré dans la DTD)

```
<p xml:lang="en">
  hello World !
</p>

<p xml:lang="fr">
  bonjour le monde !
</p>

<p xml:lang="fr-be">
  bonjour une fois !
</p>
```

Le document XML

Respecter la syntaxe XML permet de créer des documents XML **syntaxiquement corrects**. Il existe deux niveaux de "qualité" supplémentaires : les documents XML **bien-formés** et les documents XML **valides**

- Un documents XML bien-formé (well-formed document) doit :
  - être syntaxiquement correct
  - commencer par un **prologue** correctement formé
  - contenir un seul élément, appelé **élément racine**, qui doit respecter les **règles de contraintes**
- un document XML valide (valid document) doit :
  - être bien formé
  - respecter une DTD ou un schéma

Il est primordial qu'un document soit bien formé, il n'est pas primordial qu'il soit valide.

Le prologue d'un document bien formé

Le **prologue** d'un document XML bien formé doit commencer par:

- une instruction de traitement **<?xml ... ?>**

Il peut également contenir :

- une **déclaration de type de document** éventuelle (DTD)
- d'autres instructions de traitement
- des commentaires

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE carnet SYSTEM "carnet.dtd">
<?xml-stylesheet href="ulb.xsl" type="text/xsl"?>
<carnet>
  ...
</carnet>
```

L'instruction de traitement <?xml...?>

L'instruction de traitement **<?xml...?>** doit être la toute première chose rencontrée dans le document
Elle sert à définir :

- la version du langage XML utilisée (actuellement 1.0 ou 1.1)

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

- le type d'encodage utilisé dans le document (UTF-8 ou UTF-16 par défaut)

  ```
  <?xml version="1.0"" encoding="UTF-16"?>
  ```

- l'attribut standalone (yes ou no)

  ```
  <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
  ```

  Cet attribut indique s'il existe des déclarations de balisage externes à l'entité document qui puissent affecter la valeur finale du document (par exemple, une déclaration d'une valeur par défaut d'un attribut dans une DTD externe). Il servait à accélérer le traitement des documents en SGML

Les règles de contrainte d'un document bien-formé

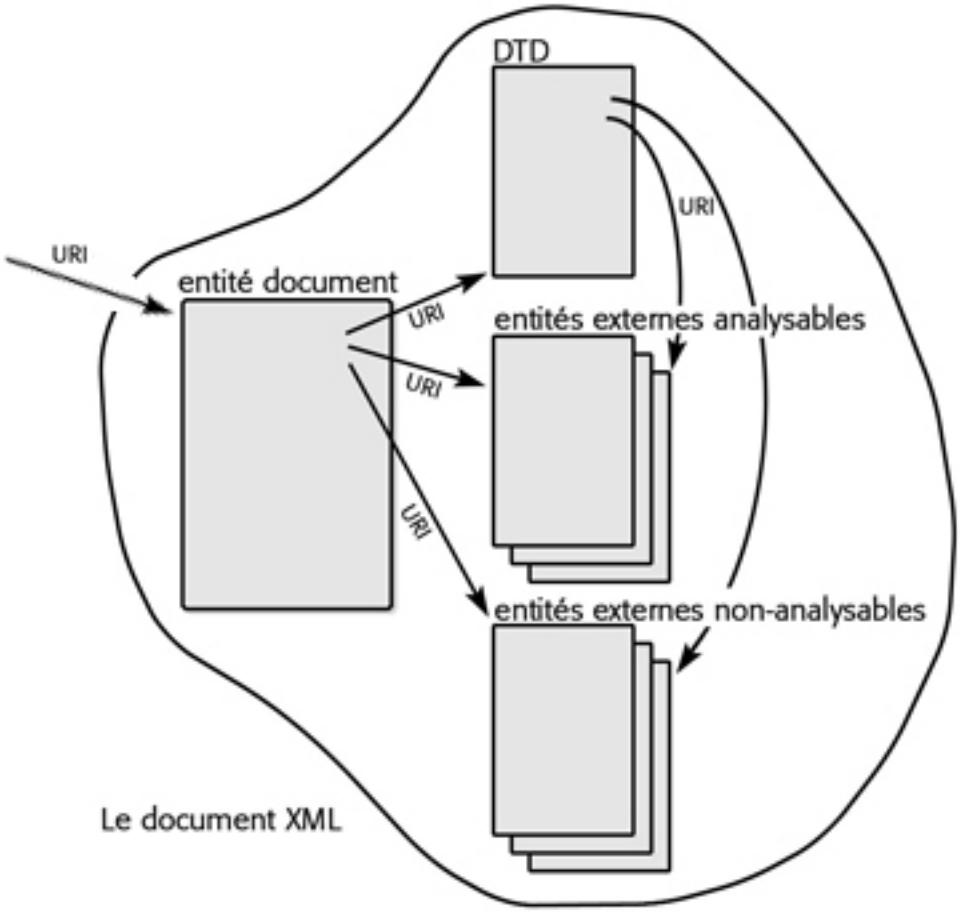
Pour qu'un document soit bien formé, il faut que son élément racine, ainsi que toutes les entités analysables externes, respectent les **régles de contraintes** suivantes :

- tous les éléments doivent être correctement imbriqués
- un même attribut ne peut apparaître qu'une fois dans une balise
- pas de référence dans un attribut vers une entité externe
- pas de référence dans un attribut vers une entité dont le texte contiendrait un <
- les références de caractère doivent représenter un caractère reconnu dans UNICODE
- les références d'entité doivent faire référence à une entité existante qui doit être analysable
- une entité ne peut pas se faire référence à elle-même
- les entités doivent être déclarées dans la DTD avant leur utilisation
- les entités paramètres dans le sous-ensemble interne de la DTD doivent apparaître là où une déclaration est attendue

La structure physique d'un document bien formé

Un document bien formé possède une structure physique et une structure logique.

La structure physique est composée d'unités de stockage appelées entités, avec au minimum une entité appelée l'**entité document**



Le contenu physique de ces entités est constitué de marqueurs de balisables (balises et attributs):

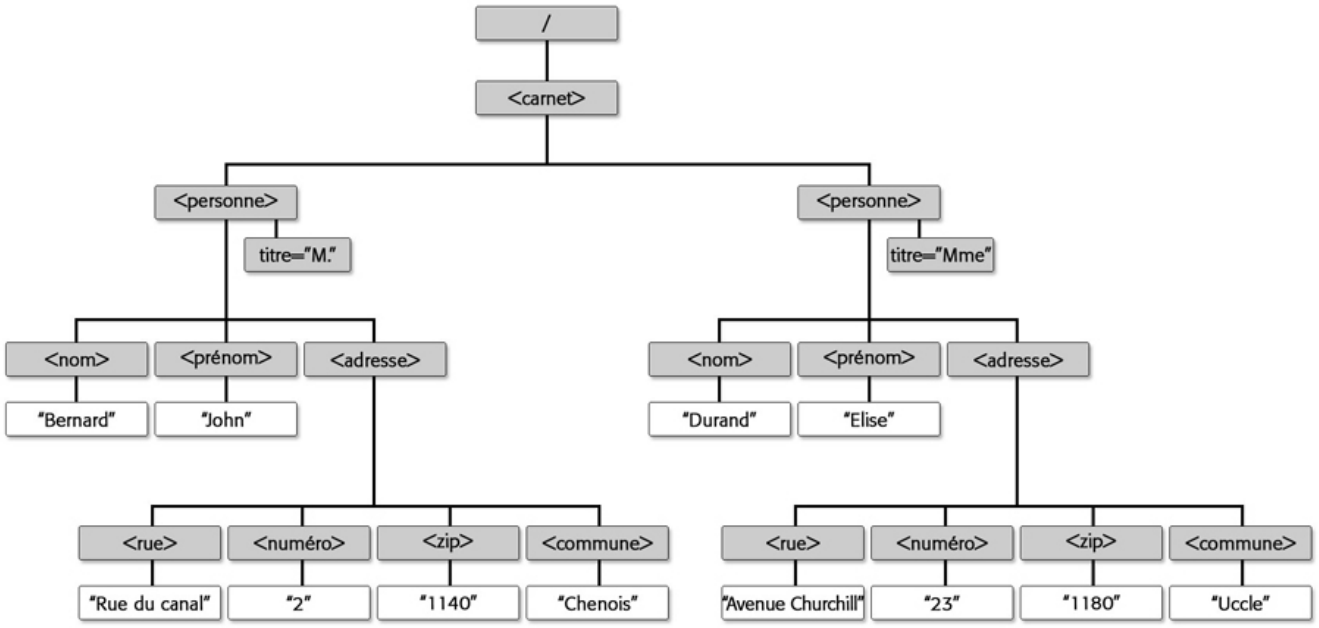
```
<?xml version="1.0" encoding="UTF-8"?>
<carnet>
  <personne titre="M.">
    <nom>Bernard</nom>
    <prénom>John</prénom>
    <adresse>
      <rue>Rue du canal</rue>
      <numéro>2</numéro>
      <zip>1140</zip>
      <commune>Chenois</commune>
    </adresse>
  </personne>

  <personne titre="Mme">
    <nom>Durand</nom>
    <prénom>Elise</prénom>
    <adresse>
      <rue>Avenue Churchill</rue>
```

```
      <numéro>23</numéro>
      <zip>1180</zip>
      <commune>Uccle</commune>
    </adresse>
  </personne>
</carnet>
```

La structure logique d'un document bien formé

Le rôle d'un parseur est de traduite la structure physique d'un document (qui potentiellement peut avoir des formes différentes) en une structure logique

Cette structure logique prend la forme d'un arbre document (document tree), ou instance XDM.



L'arbre document est constitué d'une arborescence de **noeuds** structurés suivant des relations **parent/enfants**.

Les noeuds sont typés: le noeud document, les noeuds éléments, les noeuds attributs, les noeuds textes, les noeuds commentaires et les noeuds instructions de traitement.

La structure logique - l'utilisation en DOM, XPath et XQuery

La structure logique est cette qui est manipulée dans une application qui fait appel à un parseur DOM :

```
doc.getDocumentElement().getFirstChild().getChildNodes().item(1).getFirstChild().getNodeValue();
```

Elle est également utilisée en XPath :

```
/child::carnet/child::personne[2]/child::nom/child::text()
```

```
/carnet/personne[2]/nom
```

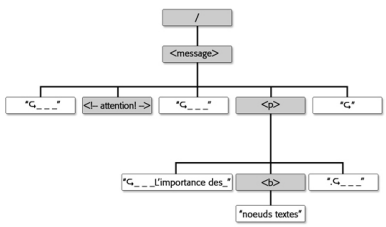
Ainsi qu'en XQuery :

```
for $zip in distinct-values(//zip)
order by $zip
return <area zip="{$zip}">
         {
         for $c in distinct-values(//commune[../zip = $zip])
         order by $c
         return <commune localité="{$c}">
                  {
                  for $p in /carnet/personne
                  where $p/adresse/zip = $zip and $p/adresse/commune = $c
                  return <nom>
                           {
                           concat($p/nom, ' ', $p/prenom)
                           }
                         </nom>
                  }
                </commune>
         }
       </area>
```

L'arbre document - tenir compte des blancs

En l'absence de DTD ou de schéma, l'arbre document tient compte de noeuds textes qui ne contiennent que des caractères blancs (espaces, tabulations, carriage returns et linefeeds)

```
<?xml version="1.0" encoding="UTF-8"?>
<message>
  <!-- attention! -->
  <p>
    L'importance des <b>noeuds textes</b>.
  </p>
</message>
```



Chapitre 3 - Les DTD

Table des matières de ce chapitre

Les documents valides

Un document XML bien formé ne garantit pas que son contenu soit correct. Pour ce faire, on doit lui associer une **DTD** ou un **schéma** qui définit une sorte de **grammaire** que devra respecter le document.
Les DTD, inventées avec SGML, ont plus de 45 ans. Elles restent fonctionnelles pour des documents XML simples, mais ne permettent que de valider la structure des documents et pas leur contenu.

DTD est l'acronyme de **Document Type Definition**.
Une DTD décrit de manière précise les éléments que peut contenir un document XML, dans quel ordre ils peuvent apparaître et quels peuvent être leurs attributs.

Un DTD se présente sous la forme d'une liste de déclarations de quatre types:

- les **déclarations d'éléments**
- les **déclarations de liste d'attributs**
- les **déclarations d'entités**
- les **déclarations de notations**

Les DTD sont traitées par des **parseurs validants**; elles peuvent être **internes** ou **externes**

Exemple de document valide, avec une DTD interne

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE livre [
<!ELEMENT livre (titre,chapitre+)>
<!ELEMENT chapitre (titre,p*)>
<!ATTLIST chapitre type (normal|annexe) "normal">
<!ELEMENT titre (#PCDATA)>
<!ELEMENT p (#PCDATA)>
]>
<livre>
  <titre>XML</titre>
  <chapitre>
    <titre>Introduction</titre>
    <p>XML est l'acronyme de...</p>
    <p>Comme HTML, XML utilise...</p>
  </chapitre>
  <chapitre>
    <titre>Le langage XML</titre>
    <p>Un document se compose...</p>
  </chapitre>
</livre>
```

Exemple de document valide, avec une DTD externe

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE livre SYSTEM "http://www.xml-it.be/dtd/livre.dtd">
<livre>
  <titre>XML</titre>
  <chapitre>
    <titre>Introduction</titre>
    <p>XML est l'acronyme de...</p>
    <p>Comme HTML, XML utilise...</p>
```

```
  </chapitre>
  <chapitre>
    <titre>Le langage XML</titre>
    <p>Un document se compose...</p>
  </chapitre>
</livre>
```

Exemple de document non valide

Un document qui possède une DTD n'est pas nécessairement valide. Il reste néanmoins utilisable, car il est bien-formé.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE livre [
<!ELEMENT chapitre (titre,p*)>
<!ATTLIST chapitre type (normal|annexe) "normal">
]>
<livre>
  <titre>XML</titre>
  <chapitre>
    <titre>Introduction</titre>
    <p>XML est l'acronyme de...</p>
    <p>Comme HTML, XML utilise...</p>
  </chapitre>
  <chapitre>
    <titre>Le langage XML</titre>
    <p>Un document se compose...</p>
  </chapitre>
</livre>
```

Les DTD: Document Type Definition

Une DTD utilise une syntaxe différente de XML (héritage de SGML)

Une DTD contient des **déclarations de marquage** de quatre types:

- des déclarations d'élément
- des déclarations de liste d'attributs
- des déclarations d'entité
- des déclarations de notation

plus éventuellement des commentaires et des instructions de traitement

Les DTD n'offrent pas toutes les fonctionnalités souhaitées, pour pallier leur manquements, il faut utiliser un **schéma**.

Un DTD peut se définir dans le **sous-ensemble interne** ou dans le **sous-ensemble externe** (les déclarations internes sont prioritaires aux déclarations externes)

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE doc SYSTEM "dtd/dtd1.dtd" [
<!ATTLIST titre isbn CDATA #REQUIRED>
]>
<doc>
  <titre isbn="2-1245-6745-2">Le langage XML</titre>
  <p>ligne 1</p>
  <p>ligne 2</p>
</doc>
```

La déclaration de type de document

Une DTD est définie par une **déclaration de type de document** :

Ce qui suit les mots-clés **PUBLIC** ou **SYSTEM** sont appelés les identificateurs **système** et **public** de la DTD

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xml:lang="en" lang="en">
  <head>
    <title>Le langage XHTML</title>
  </head>
  <body>
    <p>Consultez le <a href="http://www.w3.org/">site du W3C</a>.</p>
  </body>
</html>
```

Exemple de DTD dans le sous-ensemble externe

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE carnet SYSTEM "addr.dtd">
<carnet>
  <personne titre="M.">
    <nom>Bernard</nom>
    <prénom>John</prénom>
    <adresse>
      <rue>Rue du canal</rue>
      <numéro>2</numéro>
      <zip>1140</zip>
      <commune>Chenois</commune>
    </adresse>
  </personne>
  <personne titre="Mme">
    <nom>Durand</nom>
    <prénom>Elise</prénom>
    <adresse>
      <rue>Avenue Churchill</rue>
      <numéro>23</numéro>
      <zip>1180</zip>
      <commune>Uccle</commune>
    </adresse>
  </personne>
</carnet>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT carnet (personne)* >
<!ELEMENT personne (nom,prénom,adresse)>
<!ATTLIST personne titre CDATA #IMPLIED>
<!ELEMENT nom (#PCDATA)>
<!ELEMENT prénom (#PCDATA)>
<!ELEMENT adresse (rue,numéro,zip,commune)>
<!ELEMENT rue (#PCDATA)>
<!ELEMENT numéro (#PCDATA)>
<!ELEMENT zip (#PCDATA)>
<!ELEMENT commune (#PCDATA)>
```

Exemple de DTD dans le sous-ensemble interne

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE carnet [
<!ELEMENT carnet (personne)* >
<!ELEMENT personne (nom,prénom,adresse)>
<!ATTLIST personne titre CDATA #IMPLIED>
<!ELEMENT nom (#PCDATA)>
<!ELEMENT prénom (#PCDATA)>
<!ELEMENT adresse (rue,numéro,zip,commune)>
<!ELEMENT rue (#PCDATA)>
<!ELEMENT numéro (#PCDATA)>
<!ELEMENT zip (#PCDATA)>
```

```
<!ELEMENT commune (#PCDATA)>
]>
<carnet>
  <personne titre="M.">
    <nom>Bernard</nom>
    <prénom>John</prénom>
    <adresse>
      <rue>Rue du canal</rue>
      <numéro>2</numéro>
      <zip>1140</zip>
      <commune>Chenois</commune>
    </adresse>
  </personne>
  <personne titre="Mme">
    <nom>Durand</nom>
    <prénom>Elise</prénom>
    <adresse>
      <rue>Avenue Churchill</rue>
      <numéro>23</numéro>
      <zip>1180</zip>
      <commune>Uccle</commune>
    </adresse>
  </personne>
</carnet>
```

Exemple de DTD dans le sous-ensemble interne et le sous-ensemble externe

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE carnet SYSTEM "addr1.dtd" [
<!ATTLIST personne titre CDATA #REQUIRED>
]>
<carnet>
  <personne titre="M.">
    <nom>Bernard</nom>
    <prénom>John</prénom>
    <adresse>
      <rue>Rue du canal</rue>
      <numéro>2</numéro>
      <zip>1140</zip>
      <commune>Chenois</commune>
    </adresse>
  </personne>
  <personne titre="Mme">
    <nom>Durand</nom>
    <prénom>Elise</prénom>
    <adresse>
      <rue>Avenue Churchill</rue>
      <numéro>23</numéro>
      <zip>1180</zip>
      <commune>Uccle</commune>
    </adresse>
  </personne>
</carnet>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT carnet (personne)* >
<!ELEMENT personne (nom,prénom,adresse)>
<!ATTLIST personne titre CDATA #IMPLIED>
<!ELEMENT nom (#PCDATA)>
<!ELEMENT prénom (#PCDATA)>
<!ELEMENT adresse (rue,numéro,zip,commune)>
<!ELEMENT rue (#PCDATA)>
<!ELEMENT numéro (#PCDATA)>
<!ELEMENT zip (#PCDATA)>
<!ELEMENT commune (#PCDATA)>
```

La déclaration d'un élément

Une déclaration d'élément sert à définir son nom et son contenu. La syntaxe d'une déclaration est la suivante:

<!ELEMENT *nom_de_l'élément format_du_contenu***>**

Pour que le document soit valide, tous les éléments doivent être déclarés et leur contenu doit être conforme au format donné dans la déclaration

On distingue les formats suivants:

- les **formes simples**
- les **formes mélangées**
- les **formes composées**

La déclaration d'un élément - formes simples

Permet de déclarer des éléments soit vides, soit pouvant contenir n'importe quoi (mais qui doit être déclaré)

<!ELEMENT *nom_de_l'élément* **EMPTY>**
<!ELEMENT *nom_de_l'élément* **ANY>**

```
<!ELEMENT flag EMPTY>


<flag _empty="no"></flag>

<flag/>


<!ELEMENT root ANY>


<root>
  Cet <i>élément</i> contient <b>n'importe quel</b> autre élément
</root>
```

La déclaration d'un élément - formes mélangées

Permet de déclarer des éléments contenant une donnée textuelle, éventuellement mélangées avec des éléments

<!ELEMENT *nom_de_l'élément* **(#PCDATA)>**
<!ELEMENT *nom_de_l'élément* **(#PCDATA** | *elem1* | *elem2* | *...***)*>**

```
<!ELEMENT nom (#PCDATA)>


<nom>Jean Delicourt</nom>

<nom>Pierre Canni&#xE8;re</nom>
```

```
<!ELEMENT p (#PCDATA | i | b | u)*>
```

```
<p>texte simple</p>

<p>texte en <b>gras</b> et en <i>italique</i>.</p>
```

La déclaration d'un élément - formes composées 1/4

Permet de déclarer des éléments contenant uniquement des sous-éléments

<!ELEMENT *nom_de_l'élément* **(***elem1***)>**	rien *... une fois*
<!ELEMENT *nom_de_l'élément* **(***elem1***)?>**	**?** *... zéro ou une fois*
<!ELEMENT *nom_de_l'élément* **(***elem1***)*>**	***** *... zéro, une ou plusieurs fois*
<!ELEMENT *nom_de_l'élément* **(***elem1***)+>**	**+** *... une ou plusieurs fois*

```
<!ELEMENT info (message)>
```

```
<info>
  <message>...</message>
</info>
```

```
<!ELEMENT data (p)?>
```

```
<data/>

<data>
  <p>...</p>
</data>
```

```
<!ELEMENT carnet (personne)*>
```

```
<carnet/>

<carnet>
  <personne>...</personne>
</carnet>

<carnet>
  <personne>...</personne>
  <personne>...</personne>
  <personne>...</personne>
</carnet>
```

La déclaration d'un élément - formes composées 2/4

La **,** exprime une site, le **|** exprime un choix

<!ELEMENT *nom_de_l'élément* **(** *elem1*, *elem2*, *elem3* **)>**
<!ELEMENT *nom_de_l'élément* **(** *elem1* | *elem2* | *elem3* **)>**

```
<!ELEMENT html (head, body)>
```

```
<html>
  <head>...</head>
  <body>...</body>
</html>
```

```
<!ELEMENT contacts (tel | gsm | fax)*>
```

```
<contacts/>
```

```
<contacts>
  <gsm>...</gsm>
</contacts>
```

```
<contacts>
  <tel>...</tel>
  <tel>...</tel>
  <gsm>...</gsm>
  <fax>...</fax>
  <tel>...</tel>
</contacts>
```

La déclaration d'un élément - formes composées 3/4

Les indicateurs d'occurence peuvent se mettre dans une suite ou dans un choix

```
<!ELEMENT head (title?, meta*)>
```

```
<head/>
```

```
<head>
  <meta/>
</head>
```

```
<head>
  <title>...</title>
  <meta>...</meta>
  <meta>...</meta>
</head>
```

```
<!ELEMENT section (div | p*)>
```

```
<section>
  <div>...</div>
</section>
```

```
<section/>

<section>
  <p>...</p>
  <p>...</p>
</section>
```

La déclaration d'un élément - formes composées 4/4

Les cas plus complexes sont traités avec des parenthèses

```
<!ELEMENT contact (tel,(tel|gsm|fax)*,email?,addr+)>


<contact>
  <tel>02 650 37 15</tel>
  <addr>97 Av. Buyl, 1050 Ixelles</addr>
</contact>

<contact>
  <tel>02 650 37 15</tel>
  <fax>02 650 37 40</fax>
  <email>Cellule.Web@ulb.ac.be</email>
  <addr>97 Av. Buyl, 1050 Ixelles</addr>
  <addr>50 Av. Roosevelt, 1050 Ixelles</addr>
</contact>
```

Caractères blancs dans une forme composée

Les caractères "blancs" entre les éléments sont ignorés dans une forme composée :

```
<!ELEMENT liste (item)*>


<liste>
   <item>A123A</item>
   <item>A2XQS297</item>
</liste>
```

Ce n'est pas le cas dans une forme mélangée (ce qui est la valeur par défait si la DTD n'est pas présente) :

```
<!ELEMENT liste (#PCDATA|item)*>


<liste>
   <item>A123A</item>
   <item>A2XQS297</item>
</liste>
```

La déclaration d'une liste d'attributs

Une déclaration de liste d'attributs sert à définir la liste des attributs d'un élément :

<!ATTLIST *nom_de_l'élément attribut1 attribut2 ...*>

Chaque déclaration d'attribut contient :

- le nom de l'attribut
- le type de valeur de l'attribut
- **#REQUIRED**, **#IMPLIED** ou une valeur par défaut de l'attribut

Pour que le document soit valide, tous les attributs utilisés doivent être déclarés, les attributs obligatoires doivent être mentionnés et la valeur des attributs doit être conforme au type déclaré

Pour un même élément: soit une déclaration pour plusieurs attributs, soit plusieurs déclarations pour chaque attribut

```
<!ATTLIST livre ref CDATA #REQUIRED isbn CDATA #REQUIRED>

<!ATTLIST livre ref CDATA #REQUIRED>
<!ATTLIST livre isbn CDATA #REQUIRED>
```

Le type d'un attribut - type CDATA

Le type **CDATA** sert à déclarer un attribut qui pourra contenir n'importe quelle donnée textuelle (Caracter Data)

```
<!ELEMENT chapitre (paragraphe)+>
<!ATTLIST chapitre titre CDATA #REQUIRED>

<chapitre titre="le langage XML">...</chapitre>



<!ELEMENT periode EMPTY>
<!ATTLIST periode heures CDATA #REQUIRED minutes CDATA #REQUIRED secondes CDATA #REQUIRED>

<periode heures="10" minutes="43" secondes="00"/>
```

Le type d'un attribut - énumération de valeurs

Le type **CDATA** peut être remplacé par une énumération de valeurs, séparées par des |, entre parenthèses

```
<!ELEMENT data (#PCDATA)>
<!ATTLIST data public (on|off) #REQUIRED>

<data public="on">...</data>
<data public="off">...</data>



<!ELEMENT personne (#PCDATA)>
<!ATTLIST personne titre (M.|Mme|Mlle) #REQUIRED>

<personne titre="Mlle">...</personne>
<personne titre="M.">...</personne>
```

Le type d'un attribut - types NMTOKEN et NMTOKENS

Le type **NMTOKEN** sert à déclarer un attribut qui pourra contenir un nom (name token) composé de lettres, de chiffres, de "-", "_", ":" ou "."

```
<!ELEMENT data EMPTY>
<!ATTLIST data ref NMTOKEN #REQUIRED>

<data ref="N203"/>
```

Le type **NMTOKENS** est une liste de NMTOKEN séparés par des espaces

```
<!ELEMENT data EMPTY>
<!ATTLIST data refs NMTOKENS #REQUIRED>

<data refs="N203 N765 J_67 K091"/>
```

Le type d'un attribut - types ID, IDREF et IDREFS

Le type **ID** sert à déclarer un attribut qui pourra contenir un identificateur qui va servir de clé primaire, dont la valeur est composée de lettres, de chiffres, de "-", "_", ":" ou ".". Il doit commencer par une lettre ou un '_' ou un ':'

- la valeur doit être unique dans tout le document
- un seul attribut de type ID par élément
- pas de valeur par défaut

```
<!ELEMENT personne (#PCDATA)>
<!ATTLIST personne matricule ID #REQUIRED>

<personne matricule="P203">Marc Declerck</personne>
<personne matricule="P345">Serge Flament</personne>
<personne matricule="P409">Katia Jansens</personne>
<personne matricule="P198">François Albert</personne>
<personne matricule="P217">Michel Vandevenne</personne>
```

Le type **IDREF** sert à déclarer un attribut qui pourra contenir un identificateur qui va servir de clé secondaire

- la valeur faire référence à un attribut ID existant

```
<!ELEMENT service ANY>
<!ATTLIST service directeur IDREF #REQUIRED>

<service directeur="P198">...</service>
```

Le type **IDREFS** est une liste d'IDREF séparés par des espaces

```
<!ATTLIST service composition IDREFS #REQUIRED>

<service directeur="P198" composition="P345 P409 P217">...</service>
```

Le type d'un attribut - types ENTITY, ENTITIES et NOTATION

Le type **ENTITY** sert à déclarer un attribut qui pourra contenir le nom d'une entité externe non-analysable

```
<!ELEMENT image EMPTY>
<!ATTLIST image src ENTITY #REQUIRED>

<!ENTITY img1 SYSTEM "img1.gif" NDATA gif>
<!NOTATION gif PUBLIC "GIF V.23 01121999">

<image src="img1"/>
```

Le type **ENTITIES** est une liste d'ENTITY séparés par des espaces

Le type **NOTATION** sert à déclarer un attribut qui pourra contenir un nom de notation à choisir parmi une énumération de notations

```
<!ELEMENT graphe (#PCDATA)>
<!ATTLIST graphe format NOTATION (autocad|indesign) #REQUIRED>
<!NOTATION autocad PUBLIC "xxxx">
<!NOTATION indesign PUBLIC "yyyy">

<graphe format="autocad">...</graphe>
```

Déclarer un attribut #REQUIRED, #IMPLIED ou avec une valeur par défaut

Déclarer un attribut obligatoire : **#REQUIRED**

```
<!ELEMENT data ANY>
<!ATTLIST data public (yes|no) #REQUIRED>

<data public="yes">...</data>
```

Déclarer un attribut optionnel : **#IMPLIED**

```
<!ELEMENT data ANY>
<!ATTLIST data public (yes|no) #IMPLIED>

<data>...</data>
```

Déclarer un attribut avec une valeur par défaut : **"xxxx"** ou **'xxxx'**

```
<!ELEMENT data ANY>
<!ATTLIST data public (yes|no) "yes">

<data>...</data>
```

Déclarer un attribut avec une valeur fixe : **#FIXED "xxxx"** ou **#FIXED 'xxxx'**

```
<!ELEMENT p ANY>
```

```
<!ATTLIST p xml:lang (fr) #FIXED "fr">

<p>...</p>
```

Les déclarations d'entité

Les entités sont de moins en moins utilisées à l'heure actuelle. La déclaration va dépendre du type de l'entité: générale ou paramètre, interne ou externe, analysable ou non-analysable

- entité générale interne:

```
<!ENTITY nom "texte de remplacement">
```

- entité générale externe analysable:

```
<!ENTITY nom SYSTEM "uri">
<!ENTITY nom PUBLIC "id_public" "uri">
```

- entité générale externe non-analysable:

```
<!ENTITY nom SYSTEM "uri" NDATA notation>
<!ENTITY nom PUBLIC "id_public" "uri" NDATA notation>
```

- entité paramètre interne:

```
<!ENTITY % nom "texte de remplacement">
```

- entité paramètre externe analysable:

```
<!ENTITY % nom SYSTEM "uri">
<!ENTITY % nom PUBLIC "id_public" "uri">
```

Les déclarations de notation

Les notations ont été très peu utilisées, et ne le sont pratiquement plus du tout à l'heure actuelle.

```
<!NOTATION nom SYSTEM "id_externe">
<!NOTATION nom PUBLIC "id_public">
<!NOTATION nom PUBLIC "id_public" "id-externe">
```

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE document [
<!ELEMENT document (p|image)*>
<!ELEMENT p (#PCDATA)>
<!ELEMENT image EMPTY>
<!ATTLIST image src ENTITY #REQUIRED>
<!NOTATION gif PUBLIC "GIF V.23 01121999" >
<!NOTATION jpg PUBLIC "JPEG images" >
<!ENTITY img1 SYSTEM "img/i1.gif" NDATA gif>
<!ENTITY img2 SYSTEM "sig.gif" NDATA jpg>
]>
```

```
<document>
  <image src="img1"/>
  <p>...</p>
  <p>...</p>
  <image src="img2"/>
  <p>...</p>
</document>
```

Inclusion et exclusion de déclarations dans la DTD

Une DTD peut contenir des sections à inclure ou à exclure dans une DTD. On les rend conditionnelles en utilisant une entité paramètre.

- Inclusion de déclarations: **<![INCLUDE[...]]>**
- Exclusion de déclarations: **<![IGNORE[...]]>**

```
<!ENTITY % draft 'INCLUDE'>
<!ENTITY % final 'IGNORE'>

<![%draft;[
<!ELEMENT p ANY>
]]>

<![%final;[
<!ELEMENT p (#PCDATA|b|u|i)*>
<!ELEMENT b (#PCDATA|b|u|i)*>
<!ELEMENT u (#PCDATA|b|u|i)*>
<!ELEMENT i (#PCDATA|b|u|i)*>
]]>
```

Chapitre 4 - Les espaces de noms XML

Table des matières de ce chapitre

Les espaces de noms XML

- **Conflit potentiel**

  Quand on a commencé à mélanger dans le même document XML des données provenant de plusieurs sources différentes, un conflit potentiel pouvait avoir lieu si deux personnes avaient choisis le même nom de balise ou d'attribut pour des données différentes

- **La solution: les espaces de noms**

  Les espaces de noms ont été inventés pour résoudre ce problème: chaque personne peut continuer à choisir le nom des balises et des attributs comme elle l'entend, à condition de le faire dans des espaces de noms différents pour chaque personne

Les espaces de noms vont également permettre aux applications de pouvoir reconnaître "leurs" éléments parmi tous les autres (c'est-à-dire les éléments qu'elles doivent traiter)

Pour éviter de confondre deux espaces de noms, chaque espace de noms va utiliser une **signature**.

La signature d'un espace de noms

La signature d'un espace de noms prend la forme d'une URI (IRI dans la norme 1.1).

Principe:

- Tout document XML qui utilise un espace de noms doit déclarer celui-ci en utilisant la signature
- Tout logiciel compatible avec un espace de noms doit vérifier si la signature est bien celle de l'espace de noms pour lequel il a été conçu

L'URI ne correspond à rien de physique, elle ne sert que de signature. Elle ne doit pas exister et aucune connexion n'est établie à cette URI.

Exemples:

- L'espace de noms de XSLT est http://www.w3.org/1999/XSL/Transform
- L'espace de noms de XSL est http://www.w3.org/1999/XSL/Format
- L'espace de noms de XML Schema est http://www.w3.org/2001/XMLSchema
- L'espace de noms d'une application http://www.ulb.be/catalogue/cours
- L'espace de noms d'une application http://www.ulb.be/catalogue/enseignants
- L'espace de noms d'une application http://www.ulb.be/catalogue/programme

La déclaration et l'identificateur d'un espace de noms

L'identificateur ainsi que la signature de l'espace de noms doivent être déclarés dans le document XML, à l'aide d'un attribut **xmlns:identificateur="uri"** (xmlns est l'abréviation de XML namespace).

Les éléments et les attributs qui font partie d'un espace de noms doivent être préfixés par cet identificateur et un deux-points

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

```
  <xsl:template match="/carnet/personne">
    <ul>
      <xsl:for-each select="nom">
        <li><xsl:value-of select="."/></li>
      </xsl:for-each>
    </ul>
  </xsl:template>

</xsl:stylesheet>
```

Choix de l'identificateur

L'identificateur de l'espace de noms peut être librement choisi. Seule la signature est importante.

L'identificateur doit commencer par une lettre ou un '_', suivi d'une combinaison quelconque de lettres, de chiffres et de caractères '.', '-' et '_'

```
<?xml version="1.0" encoding="UTF-8"?>
<xyz:stylesheet xmlns:xyz="http://www.w3.org/1999/XSL/Transform">

  <xyz:template match="/carnet/personne">
    <ul>
      <xyz:for-each select="nom">
        <li><xyz:value-of select="."/></li>
      </xyz:for-each>
    </ul>
  </xyz:template>

</xyz:stylesheet>
```

Déclaration d'un espace de noms par défaut

Il est possible de déclarer un **espace de noms par défaut** à l'aide de l'attribut **xmlns="..."**

Il sera valable pour tous les éléments non préfixés par un identificateur

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

  <element name="carnet">
    <complexType>
      <sequence>
        <element minOccurs="0" maxOccurs="unbounded" ref="personne"/>
      </sequence>
    </complexType>
  </element>
  ...
</schema>
```

Plusieurs espaces de noms dans un même document

Plusieurs espaces de noms peuvent être utilisés au sein d'un document XML

Chaque espace de nom aura son propre identificateur

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" xmlns:xt="http://www.jclark.com/xt">
```

```
  <xsl:template match="/carnet/personne">
    <xt:document href="liste.html">
      <ul>
        <xsl:for-each select="nom">
          <li><xsl:value-of select="."/></li>
        </xsl:for-each>
      </ul>
    </xt:document>
  </xsl:template>

</xsl:stylesheet>
```

Plusieurs espaces de noms - exemple

Cet exemple déclare trois espaces de noms (import/export avec Microsoft Dynamics NAV database)
Un des espaces de noms est déclaré par défaut

```
<?xml version="1.0" encoding="UTF-8"?>
<Root xmlns="urn:nav:schema:all" xmlns:navField="urn:nav:schema:field" xmlns:navTable="urn:nav:schema:table">
  <navTable:SalesHeader>
    <navField:DocType>Order</navField:DocType>
    <navField:DocNo>101005</navField:DocNo>
    <navField:SellToCustNo>30000</navField:SellToCustNo>
    <navField:SellToCustName>John Haddock Insurance Co.</navField:SellToCustName>
    <navField:BillToCustNo>30000</navField:BillToCustNo>
    <navField:BillToCustName>John Haddock Insurance Co.</navField:BillToCustName>
  </navTable:SalesHeader>
  ...
</Root>
```

Utilisation des espaces de noms pour les attributs

Exemple en XLink:

```
<?xml version="1.0" encoding="UTF-8"?>
<biblio xmlns:link="http://www.w3.org/1999/xlink">
  <livre>
    <titre>Le langage XML</titre>
    <auteur link:type="simple" link:href="a1.xml">...</auteur>
  </livre>
</biblio>
```

Exemple en XML Events:

```
<?xml version="1.0" encoding="UTF-8"?>
<doc xmlns:ev="http://www.w3.org/2001/xml-events">
  <trigger>
    <reset ev:event="DOMActivate"/>
  </trigger>
</doc>
```

Redéclaration d'un espace de noms

Les espaces de noms peuvent être redéclarés (un même préfixe peut être réattribué à un autre espace de noms).

```
...
<test:elem1 xmlns:test="http://www.xml-it.com/namespace1">
  <test:elem2>...</test:elem2>
  <test:elem3 xmlns:test="http://www.xml-it.com/namespace2">
    <test:elem4>...</test:elem4>
  </test:elem3>
  <test:elem5>...</test:elem5>
</test:elem1>
...
```

Il en va de même pour l'espace de noms par défaut:

```
...
<elem1 xmlns="http://www.xml-it.com/namespace1">
  <elem2>...</elem2>
  <elem3 xmlns="">
    <elem4>...</elem4>
  </elem3>
  <elem5>...</elem5>
</elem1>
...
```

Exemple concret d'utilisation des espaces de noms dans un document XHTML

```
<?xml version="1.0" encoding="UTF-8"?>
<html xmlns="http://www.w3.org/2002/06/xhtml2" xmlns:xforms="http://www.w3.org/2002/xforms">
  <head>
    <title>Exemple 1</title>
    <xforms:model xmlns:xsd="http://www.w3.org/2001/XMLSchema">
      <xforms:instance>
        <personne xmlns="">
          <nom xsd:type="xsd:string">Durand</nom>
          <prénom xsd:type="xsd:string">Elise</prénom>
          <date xsd:type="xsd:date">1975-12-29</date>
        </personne>
      </xforms:instance>
      <xforms:submission id="f1" method="post" action="..."/>
    </xforms:model>
  </head>

  <body>
    <p>Veuillez remplir le formulaire suivant:</p>

    <xforms:input ref="/personne/nom">
      <xforms:label>Nom: </xforms:label>
    </xforms:input>
    <br/>

    <xforms:input ref="/personne/prénom">
      <xforms:label>Prénom: </xforms:label>
    </xforms:input>
    <br/>

    <xforms:submit submission="f1">
      <xforms:label>Envoyer</xforms:label>
    </xforms:submit>
```

```
    <xforms:trigger xmlns:ev="http://www.w3.org/2001/xml-events">
      <xforms:label>Reset</xforms:label>
      <xforms:reset ev:event="DOMActivate"/>
    </xforms:trigger>
  </body>
</html>
```

Chapitre 5 - XML schema

Table des matières de ce chapitre

Le langage XML schema

Le langage **XML schema** décrit la syntaxe de documents XML appelés des schémas

Un **schéma** permet, comme une DTD, de valider la structure mais également le contenu d'une classe de documents XML

Le langage utilise pour cela des **types de données**

Ces types sont très complets et très polyvalents. Ils servent à de nombreuses applications (schémas, XQuery, XForms...)

Un type peut être prédéfini dans le language (xs:string, xs:integer...) ou être créé dans le schéma

Les types simples et les types complexes

Deux catégories de types de données seront utilisées:

- les **types simples**: servent à définir valeur qui prend la forme d'une donnée textuelle
- les **types complexes**: servent à définir tout ce qui ne prend pas la forme d'une donnée textuelle

- Un élément peut être du type simple ou du type complexe
- Un attribut est toujours du type simple

Ces éléments seront de type simple :

```
<nom>Marc Declerck</nom>

<date>2017-08-24</date>
```

Ces éléments seront de type complexe :

```
<nom date="2017-08-24">Marc Declerck</nom>

<personne>
  <nom>Declerck</nom>
  <prénom>Marc</prénom>
</personne>
```

Exemple de schéma

Un exemple de document XML validé par un schéma "schema.xsd" (situé dans le même répertoire que le document XML). Le schéma est lié grâce à l'attribut `noNamespaceSchemaLocation`. Ce n'est pas obligatoire d'établir cette liaison (l'URL du schéma peut être donnée dynamiquement lors de la validation)

```
<?xml version="1.0" encoding="UTF-8"?>
<école xsi:noNamespaceSchemaLocation="schema.xsd" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <directeur>Peter Van Rossem</directeur>
  <classe année="1">
    <professeur>Eric Piraux</professeur>
    <élève>Serge Declerck</élève>
    <élève>Manon Libert</élève>
    <élève>Marc Albert</élève>
  </classe>
```

```
</école>
```

Un schéma est un document XML, dans lequel la syntaxe du langage XML schema doit être respectée. L'élément racine doit s'appeler <schema> et doit être placé dans l'espace de noms du langage

La seule obligation est de définir l'élément racine du document XML, c'est-à-dire <école> dans cet exemple

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

  <xs:element name="école" type="typeEcole"/>

  <xs:complexType name="typeEcole">
    <xs:sequence>
      <xs:element name="directeur" type="xs:string"/>
      <xs:element name="classe" type="typeClasse" minOccurs="1" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="typeClasse">
    <xs:sequence>
      <xs:element name="professeur" type="xs:string"/>
      <xs:element name="élève" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="année" type="xs:positiveInteger"/>
  </xs:complexType>

</xs:schema>
```

Documenter un schéma grâce aux annotations

Les annotations, définies par des éléments `<xs:annotation>`, peuvent apparaître à de nombreux endroits dans un schéma. Elles permettent de documenter le contenu du schéma.

Elles contiennent des éléments `<xs:documentation>` (destinés à une personne humaine) et/ou des éléments `<xs:appinfo>` (destinés à une application)

Des outils comme XML-Spy ou oXygen vont s'en servir dans des infos-bulles, des légendes, etc. pour renseigner l'utilisateur sur le contenu xml attendu par le schéma.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="customers">
    <xs:annotation>
      <xs:documentation>Liste de clients (contient des éléments customer)</xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="customer" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:element name="customer">
    <xs:annotation>
      <xs:documentation>Définition d'un client</xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:sequence>
```

```
        <xs:element name="lastname" type="xs:string" minOccurs="0">
          <xs:annotation>
            <xs:documentation>Nom de famille du client</xs:documentation>
          </xs:annotation>
        </xs:element>
        <xs:element name="firstname" type="xs:string" minOccurs="0">
          <xs:annotation>
            <xs:documentation>Prénom de famille du client</xs:documentation>
          </xs:annotation>
        </xs:element>
        <xs:element name="birthdate" minOccurs="0">
          <xs:annotation>
            <xs:documentation>Date de naissance au format JJ/MM/AAAA</xs:documentation>
          </xs:annotation>
          <xs:simpleType>
            <xs:restriction base="xs:string">
              <xs:pattern value="(0[1-9]|[12][0-9]|3[01])/(0[1-9]|1[012])/([0-9]{4})"/>
            </xs:restriction>
          </xs:simpleType>
        </xs:element>
      </xs:sequence>
      <xs:attribute name="id">
        <xs:annotation>
          <xs:documentation>Numéro de référence du client</xs:documentation>
        </xs:annotation>
      </xs:attribute>
    </xs:complexType>
  </xs:element>

</xs:schema>
```

La déclaration d'une élément

Chaque élément utilisé dans un document XML doit avoir été déclaré dans le schéma. La déclaration va servir à indiquer quelle valeur cet élément pourra contenir (à l'aide d'un type simple ou d'un type complexe)

Il faut également indiquer où un élément peut être placé dans le document XML. En général, cette indication est donnée dans la déclaration de l'élément parent

La déclaration d'un élément se fait à l'aide d'un élément `<xs:element>` possèdant un attribut `name="..."` donnant le nom de l'élément déclaré et un attribut `type="..."` donnant le type de l'élément :

```
<xs:element name="personne" type="..."/>
```

Le type de l'élément peut également être défini au sein de la déclaration, sans utiliser d'attribut `type` :

```
<xs:element name="personne">
  <xs:complexType>
    ...
  </xs:complexType>
</xs:element>
```

La déclaration d'une élément - déclaration globale

La déclaration sera **globale** si elle est placée au plus haut niveau (dans l'élément `<xs:schema>`) :

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
```

```
  ...
  <xs:element name="personne">
    ...
  </xs:element>
  ...
</xs:schema>
```

Là où l'élément peut être placé dans le document XML, on fait référence à la déclaration globale avec un élément `<xs:element ref="...">` (appelé *particle* en anglais, traduit par particule):

```
...
<xs:element name="carnet">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="personne" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
...
```

La déclaration d'une élément - déclaration locale

La déclaration sera **locale** si elle est faite là où où l'élément est utilisé (impossible alors d'y faire référence ailleurs) :

```
...
<xs:element name="carnet">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="personne" maxOccurs="unbounded">
        ...
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
...
```

La déclaration d'une élément - le type de données

Chaque déclaration d'un élément doit déclarer le type de données qui sera imposé au contenu de l'élément

Type de données prédéfini dans le langage

Si le type de données est choisi parmi les types prédéfinis dans le langage (des types simples, pour la plupart), il sera mentionné grâce à un attribut `type="xs:xxxx"` où le nom `xxxx` du type prédéfini est mentionné dans l'espace de nom du langage :

```
<xs:element name="firstname" type="xs:string"/>

<xs:element name="lastname" type="xs:string"/>

<xs:element name="birthdate" type="xs:date"/>
```

Type de données global (explicite)

Si le type de données est défini au niveau global, il possède un nom qui sera également mentionné grâce à un attribut `type="xxxx"` mais sans l'identificateur de l'espace de noms du langage :

```
<xs:element name="carnet" type="cType"/>

<xs:complexType name="cType">
  ...
</xs:complexType>
```

Type de données local (anonyme)

On peut également définir le type de données là où on en a besoin, sans lui donner de nom :

```
<xs:element name="carnet">
  <xs:complexType>
    ...
  </xs:complexType>
</xs:element>
```

Exemple de déclarations globales avec des types anonymes

Exemple de schéma qui déclare tous les éléments de manière globale et des types anonymes :

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

  <xs:element name="carnet">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="personne" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:element name="personne">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="nom"/>
        <xs:element ref="prénom"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:element name="nom" type="xs:string"/>

  <xs:element name="prénom" type="xs:string"/>

</xs:schema>
```

Exemple de déclarations globales avec des types explicites

Exemple de schéma qui déclare tous les éléments de manière globale et des types explicites :

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

  <xs:element name="carnet" type="cType"/>
```

```
  <xs:complexType name="cType">
    <xs:sequence>
      <xs:element ref="personne" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>

  <xs:element name="personne" type="pType"/>

  <xs:complexType name="pType">
    <xs:sequence>
      <xs:element ref="nom"/>
      <xs:element ref="prénom"/>
    </xs:sequence>
  </xs:complexType>

  <xs:element name="nom" type="xs:string"/>

  <xs:element name="prénom" type="xs:string"/>

</xs:schema>
```

Exemple de déclarations locales

Exemple de schéma qui déclare les éléments de manière locale (excepté l'élément racine qui doit toujours être déclaré globalement) :

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

  <xs:element name="carnet">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="personne" maxOccurs="unbounded">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="nom" type="xs:string"/>
              <xs:element name="prénom" type="xs:string"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

</xs:schema>
```

Exemple de déclarations mixtes, globales et locales

Exemple de schéma qui déclare les éléments de deux manières afin de rendre le schéma plus compréhensible :

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

  <xs:element name="carnet">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="personne" maxOccurs="unbounded"/>
```

```
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:element name="personne">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="nom" type="xs:string"/>
        <xs:element name="prénom" type="xs:string"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

</xs:schema>
```

Exercice n°1

Créer un schéma permettant de valider le document XML suivant (utiliser le type `xs:string` pour toutes les valeurs)

```
<?xml version="1.0" encoding="UTF-8"?>
<Root xsi:noNamespaceSchemaLocation="customers1.xsd" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <customers>
    <customer>
      <id>009E52FGT5</id>
      <emailAddress>info@tanis.be</emailAddress>
      <telephoneNumber>003270398740</telephoneNumber>
      <billingAddress>
        <city>Wavre</city>
        <postalCode>1300</postalCode>
        <streetName>Rue basse</streetName>
        <streetNumber>18</streetNumber>
        <countryCode>BE</countryCode>
      </billingAddress>
    </customer>
    <customer>
      <id>00DG55479V</id>
      <emailAddress>staff@newbatis.be</emailAddress>
      <telephoneNumber>003226502222</telephoneNumber>
      <telephoneNumber>003226502245</telephoneNumber>
      <billingAddress>
        <city>Ixelles</city>
        <postalCode>1050</postalCode>
        <streetName>Avenue de l'université</streetName>
        <streetNumber>10</streetNumber>
        <countryCode>BE</countryCode>
      </billingAddress>
    </customer>
  </customers>
</Root>
```

→ Solution [53]

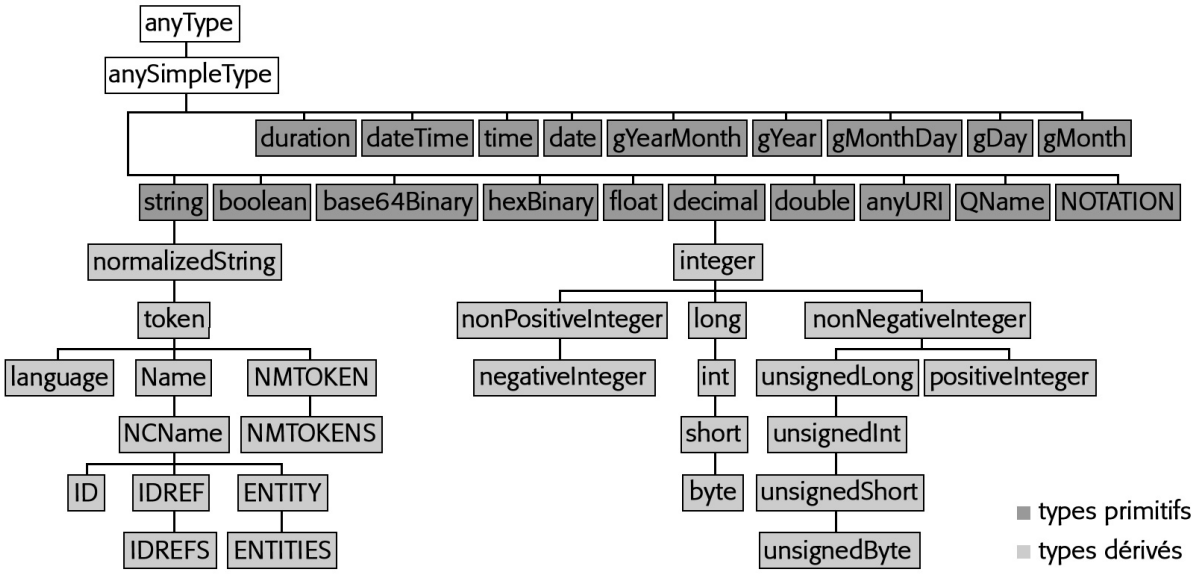
Solution de l'exercice n°1

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

```
  <xs:element name="Root">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="customers">
          <xs:complexType>
            <xs:sequence>
              <xs:element ref="customer" maxOccurs="unbounded"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:element name="customer">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="id" type="xs:string"/>
        <xs:element name="emailAddress" type="xs:string"/>
        <xs:element name="telephoneNumber" type="xs:string" maxOccurs="unbounded"/>
        <xs:element ref="billingAddress"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:element name="billingAddress">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="city" type="xs:string"/>
        <xs:element name="postalCode" type="xs:string"/>
        <xs:element name="streetName" type="xs:string"/>
        <xs:element name="streetNumber" type="xs:string"/>
        <xs:element name="countryCode" type="xs:string"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Les types simples prédéfinis

Les types simples prédéfinis:



Les types simples prédéfinis - les types chaînes de caractères

xs:string

N'importe quelle chaîne de caractères

xs:normalizedString

`xs:string` où tous les caractères blancs (tabulations, carriage returns, linefeed) sont remplacés par des espaces

xs:token

`xs:normalizedString` où toute suite d'espaces est remplacée par une espace où les espaces en début et en fin de chaîne sont supprimés

xs:language

`xs:token` qui représente une langue, éventuellement spécifique à un pays (voir RFC 3066)

〉 `en`, `en-uk`, `en-us`, `fr`, `fr-fr`, `fr-be`, `fr-ca`...

xs:NMTOKEN

`xs:token` qui correspond au type `NMTOKEN` dans les DTD

xs:NMTOKENS

Liste de `xs:NMTOKEN` qui correspond au type `NMTOKENS` dans les DTD

xs:NAME

`xs:token` qui correspond au format des noms de balises ou d'attributs en XML (y compris avec des ':')

xs:NCNAME

`xs:token` qui correspond au format des noms de balises ou d'attributs en XML (sans ':', NCNAME = non-colonized name)

xs:ID

`xs:NCNAME` qui correspond au type `ID` dans une DTD (clé primaire)

xs:IDREF

`xs:NCNAME` qui correspond au type `IDREF` dans une DTD (clé secondaire)

xs:IDREFS

Liste de `xs:NCNAME` qui correspond au type `IDREFS` dans une DTD (liste de clés secondaires)

xs:ENTITY

`xs:NCNAME` qui correspond au type `ENTITY` dans une DTD

xs:ENTITIES

Liste de `xs:ENTITY` qui correspond au type `ENTITIES` dans une DTD

Les types simples prédéfinis - les types de dates et de durées

xs:date

Représente une date au format `yyyy-mm-dd`, l'année peut être précédée d'un '-'. L'ensemble peut être suivi d'un `Z` (UTC), ou d'une timezone `+hh:mm` ou `-hh:mm`

〉 `2017-24-02`, `-0054-12-31Z`, `2032-01-15+05:00`

xs:dateTime

Représente une date et une heure, au format `yyyy-mm-ddThh:mm:ss`, l'année peut être précédée d'un '-', les secondes peuvent être suivie de décimales (avec un '.'). L'ensemble peut être suivi d'un `Z` (UTC), ou d'une timezone `+hh:mm` ou `-hh:mm`

〉 `2017-24-02T00:00:00`, `-0054-12-31T23:59:59Z`, `2032-01-15T12:00:00+05:00`

xs:dateTimeStamp

Nouveauté en XML Schema 1.1. Idem que `xs:dateTime` excepté que la mention d'une timezone est obligatoire

〉 `-0054-12-31T23:59:59Z`, `2032-01-15T12:00:00+05:00`

xs:duration

Représente une durée, au format `PyyyyYmmMddDThhHmmMssS`, cette valeur peut être précédée d'un '-' pour exprimer une durée négative, les secondes peuvent être suivie de décimales (avec un '.'). Chacune des composantes peut être omise (la valeur et la lettre). Si une composante vaut 0, la lettre seule peut être utilisée. Si aucune des composantes de l'heure sont présentes, le 'T' peut être omis.

〉 `P1Y`, `P2M15D`, `-PT1H30MS`

xs:yearMonthDuration

Nouveauté en XML Schema 1.1. Idem que `xs:duration` mais seules les composantes `Y` (année) et `M` (mois) peuvent être utilisées.

〉 `P1Y`, `P10Y2M`, `-P1Y6M`

xs:dayTimeDuration

Nouveauté en XML Schema 1.1. Idem que `xs:duration` mais les composantes `Y` (année) et `M` (mois) ne peuvent pas être utilisées.

〉 `P1D`, `P1DT6H30M`, `-PT12H30M25.5S`

xs:gYear

Représente un nombre d'années, au format `yyyy`, éventuellement précédée d'un '-' pour exprimer une année avant JC. L'ensemble peut être suivi d'un `Z` (UTC), ou d'une timezone +`hh:mm` ou -`hh:mm`

〉 `2017`, `-0034`, `1999Z`, `2032+05:00`

xs:gYearMonth

Représente un nombre d'années et de mois, au format `yyyy-mm`, éventuellement précédée d'un '-' pour exprimer une année avant JC. L'ensemble peut être suivi d'un `Z` (UTC), ou d'une timezone +`hh:mm` ou -`hh:mm`

〉 `2017-04`, `-0034-01`, `1999-12Z`, `2032-06+05:00`

xs:gMonth

Représente un nombre de mois, au format `--mm`. L'ensemble peut être suivi d'un `Z` (UTC), ou d'une timezone +`hh:mm` ou -`hh:mm`

〉 `--04`, `--12Z`, `--06+02:00`

xs:gMonthDay

Représente un nombre de mois et de jours, au format `--mm-dd`. L'ensemble peut être suivi d'un `Z` (UTC), ou d'une timezone +`hh:mm` ou -`hh:mm`

〉 `--04-01`, `--12-31Z`, `--06-15+02:00`

xs:gDay

Représente un nombre de jours, au format `---dd`. L'ensemble peut être suivi d'un `Z` (UTC), ou d'une timezone +`hh:mm` ou -`hh:mm`

〉 `---01`, `--31Z`, `---15+02:00`

Les types simples prédéfinis - les types numériques

xs:decimal

Valeur numérique avec une partie entière, positive ou négative, et une partie décimale (minimum 18 chiffres de précision).

〉 `12, 3.141592, -1.5`

xs:double
Nombre réel en 64 bits de précision (correspond au double dans de nombreux langages)

〉 `12, 3.141592, -1.5, 1.323e10, -0.1e-12`

xs:float
Nombre réel en 32 bits de précision (correspond au float dans de nombreux langages)

〉 `12, 3.141592, -1.5, 1.323e3, -0.1e-7`

xs:integer
Valeur numérique entière, positive ou négative (minimum 18 chiffres de précision).

〉 `12, -1, 219210192122231497`

xs:positiveInteger, **xs:negativeInteger**, **xs:nonPositiveInteger**, **xs:nonNegativeInteger**
Valeurs numériques entières >0, <0, <=0 et >=0

xs:long
Valeur numérique entière, en 64 bits.

〉 `19734, -435600131, 219210192122231497`

xs:int
Valeur numérique entière, en 32 bits.

〉 `78912, -11214134, 219211497`

xs:short
Valeur numérique entière, en 16 bits (-32768 à 32768).

〉 `1024, -1123, +23434`

xs:byte
Valeur numérique entière, en 8 bits (-128 à 127).

〉 `12, -1, +64`

xs:unsignedLong, **xs:unsignedInt**, **xs:unsignedShort**, **xs:unsignedByte**
Valeurs numériques entières non signées

Les types simples prédéfinis - les autres types

xs:boolean

Valeur booléenne

〉 `true`, `false`, `1`, `0`

xs:anyURI

Valeur de type URI (absolue ou relative)

〉 `http://www.ulb.be/`, `../design/img/logo.png`

xs:QNAME

Correspond au format des noms de balises ou d'attributs en XML avec espace de noms (avec un ':')

〉 `xlink:href`, `catalog:C8121`

xs:NOTATION, **xs:base64Binary**, **xs:hexBinary**

Création d'un nouveau type simple

Un nouveau type simple est créé à l'aide du composant de définition de type `<xs:simpleType>`. Le nouveau type sera créé par restriction ou par extension d'un (parfois plusieurs) type existant, appelé le **type de base**.

Par **restriction** du type de base `xs:string`, cet exemple crée un type `title` qui est limité à 128 caractères:

```
<xs:simpleType name="title">
  <xs:restriction base="xs:string">
    <xs:maxLength value="128"/>
  </xs:restriction>
</xs:simpleType>
```

Par **extension** du type de base `xs:integer`, cet exemple crée un type `integers` qui est une liste d'entiers:

```
<xs:simpleType name="integers">
  <xs:list itemType="xs:integer"/>
</xs:simpleType>
```

Création d'un nouveau type simple - type global et type local anonyme

Un nouveau type simple peut être déclaré au niveau global de **manière explicite**, en lui donnant un **nom**:

```
<xs:simpleType name="typePercent">
  <xs:restriction base="xs:integer">
    <xs:minInclusive value="0"/>
    <xs:maxInclusive value="100"/>
  </xs:restriction>
</xs:simpleType>

<xs:element name="result" type="typePercent"/>
```

Un nouveau type simple peut être déclaré au niveau local (là où il doit être utilisé), de **manière anonyme** (sans lui donner de nom):

```
<xs:element name="result">
  <xs:simpleType>
    <xs:restriction base="xs:integer">
      <xs:minInclusive value="0"/>
      <xs:maxInclusive value="100"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

Création d'un nouveau type simple - les restrictions 1/6

Plusieurs types de restriction peuvent être utilisés et combinés les uns avec les autres.

Restrictions sur la longueur (en nombre de caractères, en nombres d'octets dans des valeurs binaires ou d'éléments dans une liste):

```
<xs:restriction base="xs:string">
  <xs:length value="10"/>
</xs:restriction>

<xs:restriction base="xs:string">
  <xs:minLength value="2"/>
  <xs:maxLength value="128"/>
</xs:restriction>
```

On peut combiner avec `<xs:whiteSpace>` pour traiter les caractères blancs:

```
<xs:whiteSpace value="preserve"/>
<xs:whiteSpace value="replace"/>
<xs:whiteSpace value="collapse"/>
```

Création d'un nouveau type simple - les restrictions 2/6

Restrictions sur des valeurs minimales ou maximales, celles-ci étant incluses ou exclues:

```
<xs:restriction base="xs:integer">
  <xs:minInclusive value="0"/>
  <xs:maxInclusive value="100"/>
</xs:restriction>

<xs:restriction base="xs:integer">
  <xs:minExclusive value="0"/>
  <xs:maxExclusive value="100"/>
</xs:restriction>
```

Création d'un nouveau type simple - les restrictions 3/6

Restrictions sur le nombre de chiffres

```
<xs:restriction base="xs:integer">
  <xs:totalDigits value="10"/>
</xs:restriction>

<xs:restriction base="xs:integer">
```

```
  <xs:fractionDigits value="3"/>
</xs:restriction>
```

Création d'un nouveau type simple - les restrictions 4/6

Restrictions à l'aide d'une énumération de valeurs

```
<xs:restriction base="xs:string">
  <xs:enumeration value="Monsieur"/>
  <xs:enumeration value="Madame"/>
  <xs:enumeration value="Mademoiselle"/>
</xs:restriction>
```

Création d'un nouveau type simple - les restrictions 5/6

Restrictions à l'aide d'une expression régulière

```
<xs:restriction base="xs:string">
  <xs:pattern value="[0-9]{3}-[0-9]{5}-[0-9]{2}"/>
</xs:restriction>
```

Plusieurs éléments `<xs:pattern>` peuvent être utilisés, pour réaliser une union de types

```
<xs:simpleType name="typeColor">
  <xs:restriction base="xs:string">
    <xs:pattern value="white|black|red|green|blue"/>
    <xs:pattern value="#[0-9a-fA-F]{6}"/>
    <xs:pattern value="rgb\((0|[1-9][0-9]{0,2})(\,(0|[1-9][0-9]{0,2})){2}\)"/>
  </xs:restriction>
</xs:simpleType>
```

Création d'un nouveau type simple - les restrictions 6/6

Il est possible de créer un nouveau type par dérivation d'un type existant (ce qui peut être important au niveaux des applications, de XPath et de XQuery)

```
<xs:simpleType name="EUcountries">
  <xs:restriction base="xs:string">
    <xs:enumeration value="Belgium"/>
    <xs:enumeration value="France"/>
    <xs:enumeration value="Germany"/>
    <xs:enumeration value="Italy"/>
    <xs:enumeration value="Luxemburg"/>
    <xs:enumeration value="Netherlands"/>
    <xs:enumeration value="Portugal"/>
    <xs:enumeration value="Spain"/>
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="BENELUXcountries">
  <xs:restriction base="EUcountries">
    <xs:enumeration value="Belgium"/>
    <xs:enumeration value="Luxemburg"/>
    <xs:enumeration value="Netherlands"/>
```

```
  </xs:restriction>
</xs:simpleType>
```

L'attribut `final` permet d'empêcher la dérivation d'un type (il peut valoir `#all` ou une combinaison de `restriction`, `list` ou `union`)

```
<xs:simpleType name="EUcountries" final="#all">
  <xs:restriction base="xs:string">
    ...
  </xs:restriction>
</xs:simpleType>
```

L'attribut `fixed="true"` permet d'empêcher la réutilisation d'une contrainte dans une dérivation

```
<xs:simpleType name="typePercent">
  <xs:restriction base="xs:integer">
    <xs:minInclusive value="0" fixed="true"/>
    <xs:maxInclusive value="100" fixed="true"/>
  </xs:restriction>
</xs:simpleType>
```

Création d'un nouveau type simple - extension sous forme de liste

Un nouveau type simple peut être crée sous forme d'une liste d'un type simple de base, à l'aide d'un élément `<xs:list>`. Le type de base est soit donné par un attribut `itemType`, soit défini au sein de cet élément `<xs:list>`.
Les valeurs dans une liste sont séparées par des espaces.

Dans cet exemple, le type de base est donné par l'attribut `itemType`

```
<xs:simpleType name="integers">
  <xs:list itemType="xs:integer"/>
</xs:simpleType>
```

〉 Par exemple: "`198`", "`76 546 34 9863`"

Dans cet exemple, le type de base est défini au sein de l'élément

```
<xs:simpleType name="références">
  <xs:list>
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:pattern value="REF[0-9]{3}"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:list>
</xs:simpleType>
```

〉 Par exemple: "`REF764`", "`REF769 REF546 REF863`"

Sur un liste, il est possible d'exprimer les restrictions `<xs:length>`, `<xs:minLength>` et `<xs:maxLength>`

```
<xs:simpleType name="refs">
  <xs:restriction base="références">
```

```
    <xs:minLength value="2"/>
    <xs:maxLength value="4"/>
  </xs:restriction>
</xs:simpleType>
```

〉 Par exemple: "`REF764 REF973`", "`REF769 REF546 EF546 REF863`"

Création d'un nouveau type simple - extension sous forme d'union

Un nouveau type simple peut être crée sous forme d'une union de plusieurs types simples de base, à l'aide d'un élément `<xs:union>`. Les types de base sont soit donnés par un attribut `memberTypes` (liste de types séparés par des espaces), soit définis au sein de cet élément `<xs:union>`.

Dans cet exemple, les types de base sont donnés par l'attribut `memberTypes`

```
<xs:simpleType name="startDate">
  <xs:union memberTypes="xs:date xs:dateTime"/>
</xs:simpleType>
```

〉 Par exemple: "`2017-02-31`", "`2017-03-01T12:00:00`"

Dans cet exemple, les types de base sont donnés au sein de l'élément

```
<xs:simpleType name="colorCode">
  <xs:union>
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:pattern value="#[0-9A-Fa-f]{6}"/>
      </xs:restriction>
    </xs:simpleType>
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:enumeration value="red"/>
        <xs:enumeration value="green"/>
        <xs:enumeration value="blue"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:union>
</xs:simpleType>
```

〉 Par exemple: "`#FFFFFF`", "`red`"

Exercice n°2

Reprendre le document et le schéma créé dans l'exercice n°1 [53]. Modifier le schéma pour:

〉 forcer l'identificateur à dix caractères

〉 forcer le numéro de téléphone à commencer par 0032 suivi de 8 chiffres ou 0033 suivi de 9 chiffres

〉 Limiter le nom de la ville à 64 caractères et le nom de rue à 128 caractères

〉 forcer le code postal entre 1000 et 6999 ou entre 12000 et 82999

〉 Restreindre le code pays aux deux valeurs "`BE`" et "`FR`"

→ Solution [64]

Solution de l'exercice n°2

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

  <xs:element name="Root">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="customers">
          <xs:complexType>
            <xs:sequence>
              <xs:element ref="customer" maxOccurs="unbounded"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:element name="customer">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="id">
          <xs:simpleType>
            <xs:restriction base="xs:string">
              <xs:length value="10"/>
            </xs:restriction>
          </xs:simpleType>
        </xs:element>
        <xs:element name="emailAddress" type="xs:string"/>
        <xs:element name="telephoneNumber" maxOccurs="unbounded">
          <xs:simpleType>
            <xs:restriction base="xs:string">
              <xs:pattern value="0032[0-9]{8}"/>
              <xs:pattern value="0033[0-9]{9}"/>
            </xs:restriction>
          </xs:simpleType>
        </xs:element>
        <xs:element ref="billingAddress"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:element name="billingAddress">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="city">
          <xs:simpleType>
            <xs:restriction base="xs:string">
              <xs:maxLength value="64"/>
            </xs:restriction>
          </xs:simpleType>
        </xs:element>
        <xs:element name="postalCode">
          <xs:simpleType>
            <xs:union>
              <xs:simpleType>
                <xs:restriction base="xs:integer">
```

```
                <xs:minInclusive value="1000"/>
                <xs:maxInclusive value="6999"/>
              </xs:restriction>
            </xs:simpleType>
            <xs:simpleType>
              <xs:restriction base="xs:integer">
                <xs:minInclusive value="12000"/>
                <xs:maxInclusive value="82999"/>
              </xs:restriction>
            </xs:simpleType>
          </xs:union>
        </xs:simpleType>
      </xs:element>
      <xs:element name="streetName">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:maxLength value="128"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
      <xs:element name="streetNumber" type="xs:string"/>
      <xs:element name="countryCode">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:enumeration value="BE"/>
            <xs:enumeration value="FR"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

Création d'un nouveau type complexe

Le seul type complexe prédéfini dans le langage est `xs:anyType`. Tous les autres types complexes doivent être créés dans le schéma

Un nouveau type complexe est créé à l'aide du composant de définition de type `<xs:complexType>`

Exemple de type complexe constitué d'une séquence d'éléments:

```
<xs:complexType name="Personne">
  <xs:sequence>
    <xs:element name="nom" type="xs:string"/>
    <xs:element name="prénom" type="xs:string"/>
    <xs:element name="contacts" type="Contacts"/>
  </xs:sequence>
</xs:complexType>
```

Exemple de type complexe constitué d'un choix d'éléments:

```
<xs:complexType name="Contacts">
  <xs:choice maxOccurs="unbounded">
    <xs:element nom="tel" type="xs:string"/>
    <xs:element nom="gsm" type="xs:string"/>
    <xs:element nom="email" type="xs:string"/>
  </xs:choice>
```

```
</xs:complexType>
```

Contenu d'un type complexe

Un type complexe <xs:complexType> peut contenir un des éléments suivants:

<xs:sequence>	une séquence (d'éléments, de séquences, de choix...)
<xs:choice>	un choix (d'éléments, de séquences, de choix...)
<xs:all>	un ensemble d'éléments devant tous apparaître dans n'importe quel ordre
<xs:group>	un groupe (défini ailleurs) qui contient une séquence, un choix ou un ensemble d'éléments
<xs:simpleContent>	un contenu simple dérivé d'un type simple existant
<xs:complexContent>	un contenu complexe dérivé d'un type complexe existant

Ce contenu peut être suivi d'une liste d'attributs, définis par des éléments <xs:attribute>, <xs:attributeGroup> et/ou <xs:anyAttribute>

Une séquence <xs:sequence> ou un choix <xs:choice> peuvent contenir un des éléments suivants:

<xs:element>	un élément
<xs:any>	n'importe quel élément
<xs:sequence>	une autre séquence (d'éléments, de séquences, de choix...)
<xs:choice>	un autre choix (d'éléments, de séquences, de choix...)
<xs:all>	un ensemble d'éléments devant tous apparaître dans n'importe quel ordre
<xs:group>	un groupe (défini ailleurs) qui contient une séquence, un choix ou un ensemble d'éléments

Un ensemble d'éléments <xs:all> peut contenir un des éléments suivants:

<xs:element>	un élément

Création d'un nouveau type complexe - type global et type local anonyme

Un nouveau type complexe peut être déclaré au niveau global de **manière explicite**, en lui donnant un **nom**:

```
<xs:complexType name="Contacts">
  <xs:choice maxOccurs="unbounded">
    <xs:element nom="tel" type="xs:string"/>
    <xs:element nom="gsm" type="xs:string"/>
    <xs:element nom="email" type="xs:string"/>
  </xs:choice>
</xs:complexType>

<xs:element name="contact" type="Contacts"/>
```

Un nouveau type complexe peut être déclaré au niveau local (là où il doit être utilisé), de **manière anonyme** (sans lui donner de nom):

```
<xs:element name="contact">
  <xs:complexType>
    <xs:choice maxOccurs="unbounded">
      <xs:element nom="tel" type="xs:string"/>
      <xs:element nom="gsm" type="xs:string"/>
      <xs:element nom="email" type="xs:string"/>
    </xs:choice>
  </xs:complexType>
</xs:element>
```

Type complexe - les séquences <xs:sequence>

Un type complexe peut contenir une séquence déclarée par l'élément `<xs:sequence>`. Celle-ci peut contenir des: `<xs:element>`, `<xs:sequence>`, `<xs:choice>`, `<xs:all>`, `<xs:group>` et `<xs:any>`

```
<xs:complexType name="Personne">
  <xs:sequence>
    <xs:element name="nom" type="xs:string"/>
    <xs:element name="prénom" type="xs:string"/>
    <xs:element name="contacts" type="Contacts"/>
  </xs:sequence>
</xs:complexType>


<xs:complexType name="Contacts">
  <xs:sequence>
    <xs:element nom="tel" type="xs:string"/>
    <xs:choice minOccurs="0" maxOccurs="unbounded">
      <xs:element nom="tel" type="xs:string"/>
      <xs:element nom="gsm" type="xs:string"/>
      <xs:element nom="email" type="xs:string"/>
    </xs:choice>
  </xs:sequence>
</xs:complexType>
```

Type complexe - les choix <xs:choice>

Un type complexe peut contenir un choix déclaré par l'élément `<xs:choice>`. Celui-ci peut contenir des: `<xs:element>`, `<xs:sequence>`, `<xs:choice>`, `<xs:all>`, `<xs:group>` et `<xs:any>`

```
<xs:complexType name="PreferredContact">
  <xs:choice>
    <xs:element name="email" type="xs:string"/>
    <xs:element name="phone" type="xs:string"/>
  </xs:choice>
</xs:complexType>

<xs:complexType name="B2BContact">
  <xs:choice>
    <xs:sequence>
      <xs:element name="department" type="xs:string"/>
      <xs:element name="company" type="xs:string"/>
    </xs:sequence>
    <xs:sequence>
```

```
      <xs:element name="firstname" type="xs:string"/>
      <xs:element name="lastname" type="xs:string"/>
      <xs:element name="company" type="xs:string"/>
    </xs:sequence>
  </xs:choice>
</xs:complexType>
```

Type complexe - les ensembles d'éléments <xs:all>

Un type complexe peut contenir un ensemble d'éléments déclaré par l'élément `<xs:all>`. Celui-ci ne peut contenir que des `<xs:element>` ou des `<xs:any>`

Les éléments mentionnés devront apparaître, mais dans un ordre quelconque.

L'attribut `minOccurs="0"` peut éventuellement apparaître pour indiquer que l'élément est optionnel (`minOccurs` ne peut valoir que 0 ou 1, `maxOccurs` doit toujours être égal à 1).

```
<xs:complexType name="Contacts">
  <xs:all>
    <xs:element name="email" type="xs:string"/>
    <xs:element name="phone" type="xs:string"/>
    <xs:element name="www" type="xs:string" minOccurs="0"/>
    <xs:element name="fax" type="xs:string" minOccurs="0"/>
  </xs:all>
</xs:complexType>
```

Type complexe - les indicateurs d'occurence minOccurs et maxOccurs

Les indicateurs d'occurences (`?`, `*` et `+` dans les DTD) sont données par deux attributs **minOccurs** et **maxOccurs**. Ces attributs peuvent apparaître sur chaque élément qui constitue le contenu d'un type complexe, à savoir: `<xs:element>`, `<xs:sequence>`, `<xs:choice>`, `<xs:all>`, `<xs:group>` et `<xs:any>`

Si un de ces attributs n'est pas mentionné, la valeur `1` est utilisée par défaut. `maxOccurs` peut valoir "unbounded" pour indiquer une répétition sans limite supérieure

```
<xs:complexType name="record">
  <xs:sequence>
    <xs:element name="recordDate" type="xs:date" minOccurs="0"/>
    <xs:element name="customerLanguage" type="xs:language" maxOccurs="3"/>
    <xs:sequence minOccurs="0" maxOccurs="unbounded">
      <xs:element name="controlType" type="xs:string"/>
      <xs:element name="controlDate" type="xs:date"/>
      <xs:element name="controlResult" type="xs:boolean"/>
    </xs:sequence>
  </xs:sequence>
</xs:complexType>
```

Type complexe - les attributs <xs:attribute>

La définition d'un attribut dans un type complexe se fait après avoir défini son contenu (après la `<xs:sequence>`, le `<xs:choice>`...).

On peut déclarer un attribut à l'aide d'une **déclaration locale** ou faire référence à une **déclaration globale**.

```
<xs:complexType name="record">
  <xs:sequence>
    <xs:element name="date" type="xs:date"/>
    <xs:element name="customerLanguage" type="xs:language"/>
```

```
  </xs:sequence>
  <xs:attribute name="recordId" type="xs:ID"/>
  <xs:attribute ref="customerId"/>
</xs:complexType>

<xs:attribute name="customerId">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="[A-Z]{3}-[0-9]{8}"/>
    </xs:restriction>
  </xs:simpleType>
</xs:attribute>
```

Type complexe - attributs optionnel ou obligatoire, valeur par défaut

L'attribut `use="required"`, `use="optional"` (valeur par défaut) ou `use="prohibited"` permet d'indiquer que l'attribut est obligatoire, optionnel ou interdit (utilisé dans les dérivations de type)

```
<xs:complexType name="record">
  <xs:sequence>
    <xs:element name="date" type="xs:date"/>
    <xs:element name="customerLanguage" type="xs:language"/>
  </xs:sequence>
  <xs:attribute name="recordId" type="xs:ID" use="required"/>
  <xs:attribute ref="customerId" type="xs:IDREF" use="optional"/>
</xs:complexType>
```

L'attribut `default` permet de donner une valeur par défaut à un attribut.

```
<xs:complexType name="record">
  <xs:sequence>
    <xs:element name="date" type="xs:date"/>
  </xs:sequence>
  <xs:attribute ref="customerLanguage" type="xs:language" use="optional" default="fr"/>
</xs:complexType>
```

Type complexe - élément vide ne contenant que des attributs

Définir des éléments vides ne contenant que des attributs se fait à l'aide d'un type complexe sans mentionner de contenu (sans `<xs:sequence>`, le `<xs:choice>`...).

```
<xs:complexType name="record">
  <xs:attribute name="date" type="xs:date"/>
  <xs:attribute name="customerLanguage" type="xs:language"/>
  <xs:attribute name="recordId" type="xs:ID"/>
</xs:complexType>
```

Type complexe - les formes mélangées

Définir des éléments contenant du texte mélangé avec des balises se fait à l'aide d'un type complexe possédant un attribut **`mixed="true"`**.

```
<xs:element name="p">
  <xs:complexType mixed="true">
    <xs:choice minOccurs="0" maxOccurs="unbounded">
```

```
      <xs:element name="b" type="paragraphe"/>
      <xs:element name="i" type="paragraphe"/>
      <xs:element name="u" type="paragraphe"/>
    </xs:choice>
  </xs:complexType>
</xs:element>


<p>Un paragraphe de <b>texte formaté avec <i>italique</i> et <u>souligné</u></b></p>
```

Type complexe - dérivation à partir d'un type simple existant

Définir un élément ayant un contenu simple mais possédant des attributs se fait à l'aide d'un type complexe contenant un `<xs:simpleContent>`, qui peut exprimer une **extension** ou une **restriction** d'un type simple de base. Ces deux derniers éléments vont contenir les attributs désirés.

contenu simple par **extension** (l'élément `<xs:extension>` ne peut contenir que des d'attributs: `<xs:attribute>`, `<xs:attributeGroup>` ou `<xs:anyAttribute>`)

```
<xs:complexType>
  <xs:simpleContent>
    <xs:extension base="xs:integer">
      <xs:attribute name="unité" type="xs:string" use="required"/>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
```

contenu simple par **restriction** (l'élément `<xs:restriction>` peut contenir des restrictions suivies d'attributs: `<xs:attribute>`, `<xs:attributeGroup>` ou `<xs:anyAttribute>`)

```
<xs:complexType>
  <xs:simpleContent>
    <xs:restriction base="xs:string">
      <xs:maxLength value="64"/>
      <xs:attribute name="codePays" type="xs:language" use="required"/>
    </xs:restriction>
  </xs:simpleContent>
</xs:complexType>
```

Type complexe - dérivation à partir d'un type complexe existant

La dérivation d'un type complexe à partir d'un autre type complexe peut se faire par extension ou par restriction.

Le type complexe contient un élément `<cs:complexContent>` contenant lui-même un élément `<xs:extension>` ou `<xs:restriction>`. Ces deux derniers éléments possèdent un attribut `base` donnant le type de base.

```
<xs:complexType>
  <xs:complexType>
    <xs:complexContent>
      <xs:extension base="complexBaseType">
        ...
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
```

```
  <xs:complexContent>
    <xs:restriction base="complexBaseType">
      ...
    </xs:restriction>
  </xs:complexContent>
</xs:complexType>
```

Type complexe - dérivation par extension

La dérivation par extension d'un type complexe permet de rajouter des éléments et des attributs à un type existant.

L'élément `<xs:extension>` peut contenir une séquence, un choix ou un ensemble d'éléments qui viendront s'ajouter au contenu du type de base, ainsi que des attributs qui viendront s'ajouter à ceux du type de base. Elle se réalise à l'aide d'un élément `<xs:complexContent>` qui contient à son tour un élément `<xs:extension>`.

Dans cet exemple, le type `postalForeignAddress` est obtenu par extension du type `localForeignAddress`: un élément `<country>` et un attribut `countryCode` ont été rajoutés.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="diary">
    <xs:complexType>
      <xs:choice maxOccurs="unbounded">
        <xs:element name="localAddress" type="postalLocalAddress"/>
        <xs:element name="foreignAddress" type="postalForeignAddress"/>
      </xs:choice>
    </xs:complexType>
  </xs:element>

  <xs:complexType name="postalLocalAddress">
    <xs:sequence>
      <xs:element name="streetName" type="xs:string"/>
      <xs:element name="streetNumber" type="xs:integer"/>
      <xs:element name="city" type="xs:string"/>
      <xs:element name="postalCode" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="postalForeignAddress">
    <xs:complexContent>
      <xs:extension base="postalLocalAddress">
        <xs:sequence>
          <xs:element name="country" type="xs:string"/>
        </xs:sequence>
        <xs:attribute name="countryCode" type="xs:string"/>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
</xs:schema>
```

Type complexe - dérivation par restriction

La syntaxe et l'utilité d'une dérivation par restriction n'est pas évidente de prime abord: il faut en effet réécrire dans l'élément `<xs:restriction>` le contenu complet d'un type complexe. Ce contenu sera pratiquement identique à celui du type de base.

L'intérêt est lié à cette règle de base: un élément devra se conformer tant au type dérivé qu'au type de base. Le type dérivé procédera par "petites touches" pour modifier le type de base. Il pourra:

- limiter les valeur possibles de `minOccurs` et `maxOccurs`;
- définir des valeurs par défaut différentes;
- fixer des valeurs ou restreindre les valeurs possibles.

- Cet exemple [72] définit trois types `postalAddressBE`, `postalAddressDE` et `postalAddressFR` à partir d'un type complexe de base `postalAddress`.
  Chacun de ces types dérivés va contraindre le format du code postal (`postalCode`) et fixer la valeur du code pays (`country`)
- Dans un document XML, comme le montre cet exemple [73], les éléments de type `postalAddress` pourront être validés par un des trois types dérivés, si ces éléments possèdent un attribut `xsi:type`

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="diary">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="address" type="postalAddress" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:complexType name="postalAddress">
    <xs:sequence>
      <xs:element name="streetName" type="xs:string"/>
      <xs:element name="streetNumber" type="xs:integer"/>
      <xs:element name="city" type="xs:string"/>
      <xs:element name="postalCode" type="postalCode"/>
      <xs:element name="country" type="countries"/>
    </xs:sequence>
  </xs:complexType>

  <xs:simpleType name="postalCode">
    <xs:restriction base="xs:string">
      <xs:pattern value="[0-9]{3,6}"/>
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="countries">
    <xs:restriction base="xs:string">
      <xs:enumeration value="BE"/>
      <xs:enumeration value="DE"/>
      <xs:enumeration value="FR"/>
    </xs:restriction>
  </xs:simpleType>

  <xs:complexType name="postalAddressBE">
    <xs:complexContent>
      <xs:restriction base="postalAddress">
        <xs:sequence>
          <xs:element name="streetName" type="xs:string"/>
          <xs:element name="streetNumber" type="xs:integer"/>
```

```
          <xs:element name="city" type="xs:string"/>
          <xs:element name="postalCode">
            <xs:simpleType>
              <xs:restriction base="postalCode">
                <xs:pattern value="[0-9]{4}"/>
              </xs:restriction>
            </xs:simpleType>
          </xs:element>
          <xs:element name="country" type="countries" fixed="BE"/>
        </xs:sequence>
      </xs:restriction>
    </xs:complexContent>
  </xs:complexType>

  <xs:complexType name="postalAddressFR">
    <xs:complexContent>
      <xs:restriction base="postalAddress">
        <xs:sequence>
          <xs:element name="streetName" type="xs:string"/>
          <xs:element name="streetNumber" type="xs:integer"/>
          <xs:element name="city" type="xs:string"/>
          <xs:element name="postalCode">
            <xs:simpleType>
              <xs:restriction base="postalCode">
                <xs:pattern value="[0-9]{5}"/>
              </xs:restriction>
            </xs:simpleType>
          </xs:element>
          <xs:element name="country" type="countries" fixed="FR"/>
        </xs:sequence>
      </xs:restriction>
    </xs:complexContent>
  </xs:complexType>

  <xs:complexType name="postalAddressDE">
    <xs:complexContent>
      <xs:restriction base="postalAddress">
        <xs:sequence>
          <xs:element name="streetName" type="xs:string"/>
          <xs:element name="streetNumber" type="xs:integer"/>
          <xs:element name="city" type="xs:string"/>
          <xs:element name="postalCode">
            <xs:simpleType>
              <xs:restriction base="postalCode">
                <xs:pattern value="[0-9]{5}"/>
              </xs:restriction>
            </xs:simpleType>
          </xs:element>
          <xs:element name="country" type="countries" fixed="DE"/>
        </xs:sequence>
      </xs:restriction>
    </xs:complexContent>
  </xs:complexType>
</xs:schema>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<diary xsi:noNamespaceSchemaLocation="test1.xsd" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <address xsi:type="postalAddressBE">
    <streetName>Rue de la poste</streetName>
```

```
    <streetNumber>1</streetNumber>
    <city>Bruxelles</city>
    <postalCode>1050</postalCode>
    <country>BE</country>
  </address>
  <address xsi:type="postalAddressFR">
    <streetName>Avenue Général de Gaule</streetName>
    <streetNumber>12</streetNumber>
    <city>Sens</city>
    <postalCode>43678</postalCode>
    <country>FR</country>
  </address>
  <address xsi:type="postalAddressDE">
    <streetName>Avenue Général de Gaule</streetName>
    <streetNumber>12</streetNumber>
    <city>Dortmund</city>
    <postalCode>43678</postalCode>
    <country>DE</country>
  </address>
</diary>
```

Exercice n°3

En repartant de l'exercice n°2 [63], écrire un schéma permettant de valider le document suivant:

```
<?xml version="1.0" encoding="UTF-8"?>
<Root xsi:noNamespaceSchemaLocation="customer3.xsd" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <customers>
    <customer>
      <id>009E52FGT5</id>
      <contacts>
        <emailAddress>info@tanis.be</emailAddress>
        <telephoneNumber>003270398740</telephoneNumber>
      </contacts>
      <postalAddress>
        <city>Wavre</city>
        <postalCode>1300</postalCode>
        <streetName>Rue basse</streetName>
        <streetNumber>18</streetNumber>
        <countryCode>BE</countryCode>
      </postalAddress>
      <billing>
        <billingAddress>
          <city>Wavre</city>
          <postalCode>1300</postalCode>
          <streetName>Rue basse</streetName>
          <streetNumber>18</streetNumber>
          <countryCode>BE</countryCode>
        </billingAddress>
        <destinee>Att. M. Paul Smith</destinee>
      </billing>
    </customer>
    <customer>
      <id>00DG55479V</id>
      <contacts>
        <faxNumber>003226502222</faxNumber>
        <telephoneNumber>003226502222</telephoneNumber>
        <emailAddress>staff@newbatis.be</emailAddress>
      </contacts>
      <postalAddress>
```

```
        <city>Ixelles</city>
        <postalCode>1050</postalCode>
        <streetName>Avenue de l'université</streetName>
        <streetNumber>10</streetNumber>
        <countryCode>BE</countryCode>
      </postalAddress>
      <billing>
        <postalAddress/>
      </billing>
      <remarks>Contacts by email only</remarks>
    </customer>
    <customer>
      <id>EF0875479V</id>
      <contacts>
        <telephoneNumber>003223432982</telephoneNumber>
        <emailAddress>staff@newbatis.be</emailAddress>
      </contacts>
      <postalAddress>
        <city>Uccle</city>
        <postalCode>1080</postalCode>
        <streetName>Rue Vanderkindere</streetName>
        <streetNumber>478</streetNumber>
        <countryCode>BE</countryCode>
      </postalAddress>
      <billing>
        <postalAddress/>
        <destinee>Mrs Anna Gionacalitto</destinee>
      </billing>
    </customer>
  </customers>
</Root>
```

→ Solution [75]

Solution de l'exercice n°3

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

  <xs:element name="Root">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="customers">
          <xs:complexType>
            <xs:sequence>
              <xs:element ref="customer" maxOccurs="unbounded"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:element name="customer">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="id">
          <xs:simpleType>
            <xs:restriction base="xs:string">
```

```
            <xs:length value="10"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
      <xs:element name="contacts">
        <xs:complexType>
          <xs:all>
            <xs:element name="emailAddress" type="xs:string"/>
            <xs:element name="telephoneNumber" type="phoneNumber"/>
            <xs:element name="faxNumber" type="phoneNumber" minOccurs="0"/>
          </xs:all>
        </xs:complexType>
      </xs:element>
      <xs:element name="postalAddress" type="Address"/>
      <xs:element name="billing">
        <xs:complexType>
          <xs:sequence>
            <xs:choice>
              <xs:element name="postalAddress">
                <xs:complexType/>
              </xs:element>
              <xs:element name="billingAddress" type="Address"/>
            </xs:choice>
            <xs:element name="destinee" type="xs:string" minOccurs="0"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="remarks" type="xs:string" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:simpleType name="phoneNumber">
  <xs:restriction base="xs:string">
    <xs:pattern value="0032[0-9]{8}"/>
    <xs:pattern value="0033[0-9]{9}"/>
  </xs:restriction>
</xs:simpleType>


<xs:complexType name="Address">
  <xs:sequence>
    <xs:element name="city">
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:maxLength value="64"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:element>
    <xs:element name="postalCode">
      <xs:simpleType>
        <xs:union>
          <xs:simpleType>
            <xs:restriction base="xs:integer">
              <xs:minInclusive value="1000"/>
              <xs:maxInclusive value="6999"/>
            </xs:restriction>
          </xs:simpleType>
          <xs:simpleType>
            <xs:restriction base="xs:integer">
              <xs:minInclusive value="12000"/>
```

```
              <xs:maxInclusive value="82999"/>
            </xs:restriction>
          </xs:simpleType>
        </xs:union>
      </xs:simpleType>
    </xs:element>
    <xs:element name="streetName">
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:maxLength value="128"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:element>
    <xs:element name="streetNumber" type="xs:string"/>
    <xs:element name="countryCode">
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:enumeration value="BE"/>
          <xs:enumeration value="FR"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:element>
  </xs:sequence>
</xs:complexType>

</xs:schema>
```

Les schémas et les espaces de noms 1/6

Un schéma peut mentionner un espace de noms dans un attribut `targetNamespace`. Comme cet espace de noms sera utilisé au sein du schéma, il faut également le déclarer avec l'attribut `xmlns`.

Le schéma ne peut mentionner qu'un seul espace de noms. Si les documents XML à valider en nécessitent plusieurs, il faudra créer plusieurs schémas (chaque espace de noms sera validé avec son propre schéma).

Tout élément et tout attribut défini globalement dans le schéma appartiendra à l'espace de noms en question ("qualified"). Il devront donc être utilisés avec cet espace de noms dans le document XML.

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

  <xs:element name="customers">
    ...
  </xs:element>

  <xs:element name="customer">
    ...
  </xs:element>

  <xs:attribute name="id">
    ...
  </xs:attribute>
</xs:schema>
```

La liaison avec le schéma se fera avec un attribut `schemaLocation` mentionnant l'espace de nom suivi de l'URL du schéma.

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<cust:customers xmlns:cust="http://www.mycompany.com/customers" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:schemaLocation="http://www.mycompany.com/customers customer1.xsd">

  <cust:customer cust:id="CUST043455">
    ...
  </cust:customer>

  <cust:customer cust:id="CUST123498">
    ...
  </cust:customer>

</cust:customers>
```

Les schémas et les espaces de noms 2/6

Tout élément et tout attribut appartenant à un espace de noms doit être mentionné dans le schéma avec cet espace de noms:

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

  <xs:element name="customers">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="cust:customer" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:element name="customer">
    <xs:complexType>
      <xs:sequence>
        ...
      </xs:sequence>
      <xs:attribute ref="cust:id"/>
    </xs:complexType>
  </xs:element>

  <xs:attribute name="id">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:pattern value="CUST[0-9]{6}"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>
</xs:schema>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<cust:customers xmlns:cust="http://www.mycompany.com/customers" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:schemaLocation="http://www.mycompany.com/customers customer1.xsd">

  <cust:customer cust:id="CUST043455">
    ...
  </cust:customer>

  <cust:customer cust:id="CUST123498">
    ...
```

```
  </cust:customer>

</cust:customers>
```

Les schémas et les espaces de noms 3/6

Par défaut, tout élément et tout attribut déclaré localement n'appartient pas à l'espace de noms du schéma ("unqualified"):

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

  <xs:element name="customers">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="cust:customer" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:element name="customer">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="firstname" type="xs:string"/>
        <xs:element name="lastname" type="xs:string"/>
      </xs:sequence>
      <xs:attribute ref="cust:id" use="required"/>
      <xs:attribute name="custNumber" type="xs:integer" use="required"/>
    </xs:complexType>

  </xs:element>

  <xs:attribute name="id">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:pattern value="CUST[0-9]{6}"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>

</xs:schema>


<?xml version="1.0" encoding="UTF-8"?>
<cust:customers xmlns:cust="http://www.mycompany.com/customers" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:schemaLocation="http://www.mycompany.com/customers customer1.xsd">

  <cust:customer cust:id="CUST043455" custNumber="1">
    <firstname>Paul</firstname>
    <lastname>Smith</lastname>
  </cust:customer>

  <cust:customer cust:id="CUST123498" custNumber="2">
    <firstname>Carla</firstname>
    <lastname>Vandervelde</lastname>
  </cust:customer>

</cust:customers>
```

Les schémas et les espaces de noms 4/6

Pour chaque élément ou attribut déclaré localement, on peut changer le comportement par défaut en utilisant un attribut `form="qualified"` ou `form="unqualified"`:

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

  <xs:element name="customers">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="cust:customer" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:element name="customer">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="firstname" type="xs:string" form="qualified"/>
        <xs:element name="lastname" type="xs:string" form="qualified"/>
      </xs:sequence>
      <xs:attribute ref="cust:id" use="required"/>
      <xs:attribute name="custNumber" type="xs:integer" use="required" form="qualified"/>
    </xs:complexType>

  </xs:element>

  <xs:attribute name="id">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:pattern value="CUST[0-9]{6}"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>

</xs:schema>


<?xml version="1.0" encoding="UTF-8"?>
<cust:customers xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://
www.mycompany.com/customers customer1.xsd" xmlns:cust="http://www.mycompany.com/customers">

  <cust:customer cust:id="CUST043455" cust:custNumber="1">
    <cust:firstname>Paul</cust:firstname>
    <cust:lastname>Smith</cust:lastname>
  </cust:customer>

  <cust:customer cust:id="CUST123498" cust:custNumber="2">
    <cust:firstname>Carla</cust:firstname>
    <cust:lastname>Vandervelde</cust:lastname>
  </cust:customer>

</cust:customers>
```

Les schémas et les espaces de noms 5/6

Il est possible de changer le comportement par défaut pour tous les éléments et les attributs déclarés localement, grâce aux attributs `elementFormDefault="qualified"` (`elementFormDefault="unqualified"`) ou `attributeFormDefault="qualified"` (`attributeFormDefault="unqualified"`):

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:cust="http://www.mycompany.com/
customers" targetNamespace="http://www.mycompany.com/customers" elementFormDefault="qualified"
attributeFormDefault="qualified">

  <xs:element name="customers">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="cust:customer" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:element name="customer">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="firstname" type="xs:string"/>
        <xs:element name="lastname" type="xs:string"/>
      </xs:sequence>
      <xs:attribute ref="cust:id" use="required"/>
      <xs:attribute name="custNumber" type="xs:integer" use="required"/>
    </xs:complexType>
  </xs:element>

  <xs:attribute name="id">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:pattern value="CUST[0-9]{6}"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>

</xs:schema>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<cust:customers xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://
www.mycompany.com/customers customer1.xsd" xmlns:cust="http://www.mycompany.com/customers">

  <cust:customer cust:id="CUST043455" cust:custNumber="1">
    <cust:firstname>Paul</cust:firstname>
    <cust:lastname>Smith</cust:lastname>
  </cust:customer>

  <cust:customer cust:id="CUST123498" cust:custNumber="2">
    <cust:firstname>Carla</cust:firstname>
    <cust:lastname>Vandervelde</cust:lastname>
  </cust:customer>

</cust:customers>
```

Les schémas et les espaces de noms 6/6

Les éléments `<xs:any>` et `<xs:anyAttribute>` permettent de définir un élément ou un attribut d'un autre espace de noms que celui du schéma courant.

Leur contenu ne peut pas être déclaré dans le schéma courant, mais peut l'être dans un autre schéma possèdant un `targetNamespace` égal à l'espace de noms de l'élément ou de l'attribut en question.

Ce premier schéma insère dans la séquence un élément défini dans un autre schéma et un autre espace de noms:

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

  ...
  <xs:element name="customer">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="firstname" type="xs:string"/>
        <xs:element name="lastname" type="xs:string"/>
        <xs:any processContents="strict" namespace="http://www.mycompany.com/countries"/>
      </xs:sequence>
      <xs:attribute ref="cust:id" use="required"/>
    </xs:complexType>
  </xs:element>
  ...

</xs:schema>
```

Le deuxième schéma doit définir globalement l'élément à insérer dans la séquence du premier schéma:

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

  ...
  <xs:element name="countryCode">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:enumeration value="BE"/>
        <xs:enumeration value="DE"/>
        <xs:enumeration value="FR"/>
        <xs:enumeration value="NL"/>
        ...
      </xs:restriction>
    </xs:simpleType>
  </xs:element>
  ...
</xs:schema>
```

Le document XML devra être validé conjointement par les deux schémas:

```
<?xml version="1.0" encoding="UTF-8"?>
<cust:customers xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://
www.mycompany.com/customers customer1.xsd http://www.mycompany.com/countries countries1.xsd"
xmlns:cust="http://www.mycompany.com/customers" xmlns:country="http://www.mycompany.com/countries">
```

```
  <cust:customer cust:id="CUST043455">
    <cust:firstname>Paul</cust:firstname>
    <cust:lastname>Smith</cust:lastname>
    <country:contryCode>BE</country:contryCode>
  </cust:customer>

  <cust:customer cust:id="CUST123498">
    <cust:firstname>Carla</cust:firstname>
    <cust:lastname>Vandervelde</cust:lastname>
    <country:contryCode>NL</country:contryCode>
  </cust:customer>

</cust:customers>
```

Les éléments <xs:any> et <xs:anyAttribute>

Les éléments `<xs:any>` et `<xs:anyAttribute>` permettent de définir n'importe quel élément ou attribut dans la déclaration d'un type complexe.

Ils sont régis par deux attributs `namespace` et `processContents`.

L'attribut `namespace` va indiquer à quel espace de noms l'élément ou l'attribut doit appartenir:

`##any`	n'importe quel espace de noms
`##other`	n'importe quel espace de noms, sauf celui donné par `targetNamespace`
`##targetNamespace`	l'espace de noms donné par `targetNamespace`
`##local`	aucun espace de noms
URI	L'espace de noms dont la signature est égale à cette URI

Les trois dernières valeurs peuvent être combinées dans une liste pour accepter un élément ou un attribut de plusieurs espaces de noms différents.

L'attribut `processContents` indique comment cet élément ou attribut va être validé:

`##strict`	il devra être validé à l'aide d'une déclaration globale dans un autre schéma déclarant l'espace de noms adéquat
`##lax`	il devra être validé, à condition qu'il soit défini à l'aide d'une déclaration globale dans un autre schéma déclarant l'espace de noms adéquat
`##skip`	il ne sera pas validé

Table des matières complète