Syllabus Cours sur les services Web et XML

XML

OBJECTIFS

- **Maîtriser les bases d'XML**
- **Connaître les grands principes du méta-langage**
- **Découvrir les outils indispensables et la richesse des langages XML**

PUBLIC

Toute personne qui souhaite comprendre les apports et enjeux de la technologie XML et des technologies liées

PREREQUIS

Aucune connaissance particulière n'est nécessaire, mais des bases HTML sont utiles

WEBOGRAPHIE

https://www.w3schools.com/xml

https://fr.wikibooks.org/wiki/Accueil

XML : Cours et exercice (Modélisation - Schéma - Design patterns - XSLT - XPath - SOAP - XQuery - XSL-FO – SVG) - Auteur ; Alexandre Brillan – Ed. : ÉDITIONS EYROLLES

PROGRAMME DU COURS

I. Généralités XML

1. Définition

Extensible Markup Language (XML) est un langage de balisage et un format de fichier pour stocker, transmettre et reconstruire des données arbitraires. Il définit un ensemble de règles pour encoder les documents dans un format à la fois lisible par l'homme et lisible par la machine.

2. Historique/Origine et version

La communauté du Web voulant remédier aux faiblesses de HTML et la communauté SGML ne voulant pas rester en marge de l'essor de HTML ont collaboré pour définir un successeur (plus simple) à SGML. L'objectif était de définir un formalisme permettant d'échanger facilement des documents complexes. Elles ont constitué un groupe de travail en 1996 et le W3C a établi la recommandation de la version XML1.0 en 1998.

Il existe également une version 1.1 de XML, qui propose des différences mineures et nécessaires uniquement dans des contextes particuliers. Cette nouvelle version a été nécessaire pour des raisons de compatibilité des outils existants. Les évolutions principales de XML 1.1 sont de nouveaux caractères permis pour les noms d'éléments (pour suivre l'évolution d'Unicode depuis 1998), de nouvelles conventions pour les caractères de fin de ligne (pour la compatibilité avec des ordinateurs mainframe) et de nouveaux caractères de contrôle dans le contenu.

3. Rôle/utilité

Le rôle de l'informatique est d'offrir un cadre de stockage et de traitement de l'ensemble de ces informations. Pour être comprise, toute information doit être formalisée, c'est-à-dire représentée en respectant certaines règles. Le choix des mots, l'ordre des mots, etc., tout cela a du sens pour les acteurs de l'entreprise, qu'ils soient humains ou logiciels. Un document XML sert alors de vecteur à l'information : c'est une manière universelle de représenter des données et leur sens dans un cadre précis.

4. Quand / Où utiliser XML ?

- Echange de données
- Fichiers de configuration
- Manipulation et présentation de données

5. Document XML : orienté document ou orienté données ?

Lorsque les données sont élaborées par des êtres humains, on dit que les fichiers XML produits sont orientés document. Lorsque les données sont construites automatiquement par des programmes, on dit que les fichiers XML sont orientés données. Un fichier XML orienté document peut être, par exemple, un livre, un article, un message… Un fichier XML orienté donnée est, par exemple, un sous-ensemble d'une base de données.

Il faut noter que l'élaboration des fichiers XML nécessite des moyens de contrôle et d'édition plus ou moins sophistiqués. On n'utilisera pas pour fabriquer un ouvrage en XML un éditeur trop rudimentaire (comme le bloc-notes sous l'environnement Windows).

6. Outils pour manipuler les documents XML

a. Editeurs de texte évolué

Pour créer et ou gérer un fichier XML, nous avons besoin d'un éditeur de texte évolué qui offre des fonctionnalités telles que :

- le coloriage syntaxique
- l'auto complétion des mots et fonctions

Comme éditeurs évolué, nous avons :

- Notepad++ (avec plugins XML Tool)
- Eclipse (avec les extensions XML)
- Visual Studio Code (avec les extensions XML)

b. Processeur XML

Un processeur XML est une application ou une librairie qui peut lire un document XML et vérifier qu'il est bien formé ou qu'il est valide. Ainsi, Eclipse, les navigateurs Internet, … encapsulent des processeurs XML

c. Les parseurs XML

Un parseur est une libraire qui peut lire du XML et le rendre accessible à des langages de programmation tels Java, C#, Python, … .

Il a pour rôle d'analyser le document XML et de servir de lien avec une application de traitement. Il existe des parseurs non validants qui n'offrent qu'une vérification syntaxique et des parseurs validants qui offrent également le support des DTD/schéma W3C.

- Microsoft XML Core Services (MSXML : http://msdn.microsoft.com) est une API composée d'un parseur validant, compatible SAX et DOM, et d'un moteur de transformation 1.0.
- Xerces est disponible pour Java, C++ et Perl. C'est un logiciel Open Source réalisé par le groupe apache (http://xerces.apache.org/). Il s'agit probablement du parseur le

plus abouti du marché, quelle que soit la plate-forme, en terme de respect du standard et de l'API (SAX, DOM). Ses performances sont aussi remarquables.
- Un certain nombre de plates-formes, comme PHP et Java, disposent d'un parseur en standard.
- Expat est un parseur réalisé en C (http://expat.sourceforge.net/), utilisé par le projet Mozilla. Il dispose d'extensions pour SAX et DOM. Un ensemble de tests (benchmark) le présente comme beaucoup plus rapide que les autres parseurs (résultats disponibles à l'adresse http://www.xml.com/pub/a/Benchmark/article.html?page=3).
- Piccolo est un parseur non validant réalisé en Java (http://piccolo.sourceforge.net/). Les benchmarks disponibles, qui le présentent comme performant (http://piccolo.sourceforge.net/bench.html), peuvent être trompeurs car ils prennent en compte d'anciennes versions desautres parseurs ; par exemple, les dernières versions de Xerces donnent de meilleures performances.

II. Concepts de base

1. Prologue

Un document XML doit débuter par une ligne de ce type, indiquant la version de XML utilisé et, de manière optionnelle de type d'encodage et s'il est lié à une DTD

<?xml version="1.0" encoding="utf-8" standalone= "no" ?>

Le champ « standalone » désigne l'indépendance du document et est facultatif.

2. Commentaires

Les commentaires peuvent apparaître n'importe où dans un document en dehors d'autres balises. Les commentaires ne peuvent pas apparaître avant la déclaration XML. Les commentaires commencent par <!--et se terminent par -->.

3. Eléments ou balises

Une *balise* est une construction de balisage qui commence par <et se termine par >. Il existe trois types de balise :

- *balise de début*, telle que <section>;
- *balise de fin*, telle que </section>;
- *balise d'élément vide*, telle que <line-break />.

Les éléments ou balises obéissent à des règles qui sont les mêmes pour tout autre élément. Ce sont :

- les balises de début et de fin doivent être identiques

- les balises enfants doivent être fermées avant les balises parents
- les noms sont sensibles à la casse
- les noms doivent commencer par une lettre ou un souligné (_)
- un nom d'élément peut contenir des chiffres, des lettres.

a. Elément racine

C'est l'élément qui va contenir tous les autres éléments du document XML. Il ne peut pas y en avoir deux. Il est une balise paire.

b. Eléments enfants

Au sein de l'élément racine, il est possible de mettre d'autres éléments à l'aide de balises. Ces balises peuvent être paires ou orphelines et obéissent aux mêmes règles que celles citées plus haut.

c. Eléments vides

Les éléments qui n'en contiennent pas d'autres ou qui ne contiennent pas de données textuelles sont appelées des éléments vides. Ils sont spécifiés par des balises orphelines.

Syntaxe :

< element></ element > ou < element />

4. Attributs

Un attribut est un couple (clé, valeur) associé à la définition d'un élément. Il est localisé dans la balise ouvrante de l'élément. Un élément peut donc avoir de 0 à n attributs uniques. L'attribut est complémentaire de l'élément de par son rôle au sens où il ajoute une information à l'élément ou bien encore le complète dans sa définition. La valeur d'un attribut est toujours dans des quottes (simple ou double).

Exemples

<etudiant nom="Poutine" prenom="Avatar">…</ etudiant >

<contact email='poutine@avatar.tg'/>

nom et prenom sont des attributs de l'élément « etudiant » alors que email est un attribut de l'élément « contact »

Remarque : Équivalence attribut / élément

Un attribut peut toujours être représenté alternativement par un élément fils de l'élément qu'il caractérise, avec une signification du même ordre :

<element attribut="valeur_attribut"/>

<element>

<element_fils> valeur_attribut </ element_fils >

<element>

Ce deux (02) déclarations ci-dessus sont similaires.

Il est donc tout à fait possible de faire du XML sans utiliser d'attribut.

5. Entités prédéfinies

Ce sont des portions de textes qui commencent par le caractère & et finissent par un point-virgule. Ils sont souvent utilisés pour afficher des caractères réservés par XML

Symbole	Character entity
<	<
>	>
&	&
‘	"
"	'

Exemple

Incorrect	Correct
<message>salary < 1000</message>	<message>salary < 1000</message>

6. Blocs CDATA

On peut utiliser des blocs CDATA pour inclure dans le document XML du texte qui ne sera pas interprété par le processeur XML au lieu d’utiliser tout le document des entités prédéfinies.

On ne peut pas imbriquer un bloc CDATA dans un autre.

Syntaxe/exemple :

<![CDATA[

<Photo emplacement="C:/eclipse/photo.jpg" />

]]>

Le texte *<Photo emplacement="C:/eclipse/photo.jpg" />* ne sera pas interprété.

7. Règles d'un document XML bien formé

Il obéit à quelques règles qui sont :

- Chaque document XML doit avoir un élément racine
- Les éléments XML doivent avoir une balise fermante (</element> ou />)
- Les éléments XML doivent être correctement imbriqués
- Les tags XML sont sensibles à la casse
- Les valeurs des éléments XML doivent être mises entre quottes (simples ou doubles)

8. Exercice appliqué : Création d'un livre en XML

On souhaite écrire un livre en utilisant le formalisme XML. Le livre est structuré en sections (au moins 2), en chapitres (au moins 2) et en paragraphes (au moins 2).

Le livre doit contenir la liste des auteurs (avec nom et prénom).

Tous les éléments doivent posséder un titre, sauf le paragraphe qui contient du texte.

Proposez une structuration XML de ce document (avec 2 auteurs, 2 sections, 2 chapitres par section et 2 paragraphes par chapitre).

Vérifiez, à l'aide de l'éditeur, que votre document est bien formé.

Attention :

- ne pas utiliser d'attributs ;
- l'encodage utilisé est utf-8 ;
- nom du document sera livre1.xml.

9. Espaces de noms

Pour délimiter la portée d'une balise, d'un attribut ou d'une valeur d'attribut, nous disposons d'espaces de noms (namespace). Par analogie, cette notion est reprise dans la plupart des langages de programmation récents, comme les packages en java ou les namespaces dans l'environnement .net. La notion d'espace de noms peut être perçue comme un groupe d'appartenance ou une famille. L'utilisation des espaces de noms garantit une forme de traçabilité de la balise et évite les ambiguïtés d'usage.

Les espaces de noms sont très souvent employés dans les spécifications W3C car ces documents peuvent être mélangés à d'autres et entraîner des conflits. On en trouve, par exemple, dans les feuilles de styles (XSLT) et dans les schémas W3C.

a. Vocabulaire : qualification des éléments

Un élément qui est connu dans un espace de noms est dit qualifié ; dans le cas contraire, il est dit non qualifié. Lorsqu'on ignore l'espace de noms d'un élément, on dit qu'on s'intéresse à sa forme locale. Ce vocabulaire est à connaître lorsqu'on travaille avec un parseur dans les techniques dites SAX ou DOM.

b. Espace de noms par défaut

Un premier usage consiste à utiliser simplement l'espace de noms par défaut. Ce dernier est précisé par un pseudo-attribut xmlns (retenir que ns est pour namespace). La valeur associée sera une URL garantissant l'unicité de l'espace de noms. L'espace de noms par défaut s'applique à l'élément où se situe sa déclaration et à tout son contenu.

Exemple :

```
<chapitre xmlns="http://www.masociete.com">
    <paragraphe>
    ...
    </paragraphe>
</chapitre>
```

Nous pouvons changer l'espace de noms par défaut même dans les éléments enfants : dans ce cas, une règle de priorité est appliquée. Attention, les espaces de noms ne sont pas imbriqués ; on ne peut appliquer qu'un seul espace de noms à la fois.

Exemple :

```
<chapitre xmlns="http://www.masociete.com">
    <paragraphe xmlns="http://www.autresociete.com">
    ...
    </paragraphe>
</chapitre>
```

L'élément paragraphe n'appartient pas à l'espace de noms http://www.masociete.com mais uniquement à l'espace de noms http://www.autresociete.com.

c. Espace de noms explicite

L'espace de noms par défaut présente l'inconvénient d'être peu contrôlable sur un document de taille importante. En effet, tout ajout ou modification d'un tel espace va se répercuter sur la totalité du contenu.

On déclare un préfixe comme un pseudo-attribut commençant par xmlns:prefixe. Une fois déclaré, il est employable uniquement dans l'élément le déclarant et dans son contenu. L'emploi consiste à ajouter en tête de l'élément, de l'attribut ou d'une valeur d'attribut, le préfixe suivi de :.

Exemple :

<p:resultat xmlns:p="http://www.masociete.com">

</p:resultat>

L'élément « resultat » est dans l'espace de noms « http://www.masociete.com » grâce au préfixe p.

On peut déclarer et utiliser plusieurs espaces de noms grâce aux préfixes.

Exemple :

<p:res xmlns:p=http://www.masociete.com xmlns:p2="http://www.autresociete.com">

<p2:res>

</p2:res>

</p:res>

d. Nom qualifié et nom universel

Nom qualifié composé :

- d'un préfixe (optionnel)
- du type de l'élément (nom local)

Nom universel :

- composé d'une URI (optionnelle) et du type de l'élément (nom local)
- obtenu à partir du nom qualifié en remplaçant le préfixe par sa définition

Exemple

```
<document>
    <art:film xmlns:art='http://www.pariscope.fr/'>
        <com:acteur xmlns:com='http://www.comedie.fr/'
            com:nom='Juliette Binoche' />
    </art:film>
</document>
```

Nom qualifié	Nom universel
art:film	http://www.pariscope.fr/:film
com:acteur	http://www.comedie.fr/:acteur

III. Validation des documents XML : Schéma XML

Un document XML avec une syntaxe correcte est appelé « document bien formé ». Un document XML avec une syntaxe correcte et validé par un schéma XML est appelé « document bien formé » et « valide ».

L'extension du fichier schéma XML est « .xsd » qui est l'acronyme de « XML Schema Definition ».

En utilisant des schémas XML, les applications peuvent valider les fichiers XML qu'ils s'échangent. Au cours de ce chapitre, nous verrons ce que c'est qu'un schéma XML, pourquoi nous en avons besoin et comment en créer. Nous allons aussi voir les différents types qui sont disponibles lorsque nous créons un fichier schéma XML.

Les différences entre une définition de schéma XML (XSD) et une définition de type de document (DTD) incluent : **les schémas XML sont écrits en XML tandis que les DTD sont dérivés de la syntaxe SGML** . Les schémas XML définissent les types de données pour les éléments et les attributs tandis que DTD ne prend pas en charge les types de données. Les schémas XML permettent la prise en charge des espaces de noms, contrairement à DTD.

1. Qu'est-ce qu'un schéma XML

Un schéma XML définit une grammaire pour un document XML. Il peut être interne ou externe au document XML.

Il est utilisé pour spécifier ce qui doit apparaître dans un document XML en termes de :

- Éléments
- Attributs
- Espaces de noms (namespaces)
- Ordre dans lequel les éléments doivent apparaître
- Nombre d'occurrences de chaque élément
- Restrictions

Un document XML qui respecte une définition de schéma XML est un document valide. La définition du schéma XML est également un document XML. Toutefois, il porte l'extension .xsd (XML Schema Definition). Tous les éléments au sein d'une définition de schéma XML sont fournis par le W3C.

2. Pourquoi utiliser un schéma XML ?

Avec le Schéma XML, vos fichiers XML peuvent contenir une description de leur propre format.

Avec le Schéma XML, des groupes de personnes indépendants peuvent s'entendre sur une norme d'échange de données.

Avec le Schéma XML, vous pouvez vérifier les données.

3. Schémas XML utilisent la syntaxe XML

Une autre grande force des schémas XML est qu'ils sont écrits en XML :

- Vous n'avez pas besoin d'apprendre une nouvelle langue
- Vous pouvez utiliser votre éditeur XML pour modifier vos fichiers de schéma
- Vous pouvez utiliser votre analyseur XML pour analyser vos fichiers schéma
- Vous pouvez manipuler vos schémas avec le DOM XML
- Vous pouvez transformer vos schémas avec XSLT

4. Syntaxe

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

<xs:schema>

..
```

</xs:schema>

Exemple de syntaxe

<?xml version="1.0"?>

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

targetNamespace="https://www.w3schools.com"

xmlns="https://www.w3schools.com"

elementFormDefault="qualified">

...

</xs:schema>

Le fragment suivant :

xmlns:xs="http://www.w3.org/2001/XMLSchema"

indique que les éléments et les types de données utilisés dans le schéma proviennent de l'espace de noms "http://www.w3.org/2001/XMLSchema". Il précise également que les éléments et les types de données provenant de l'espace de noms "http://www.w3.org/2001/XMLSchema" doivent être préfixés par xs :

Ce fragment :

targetNamespace="https://www.w3schools.com"

indique que les éléments définis par ce schéma (note, to, from, header, body.) proviennent de l'espace de noms "https://www.w3schools.com".

Ce fragment :

xmlns="https://www.w3schools.com"

indique que l'espace de noms par défaut est "https://www.w3schools.com".

Ce fragment :

elementFormDefault="qualified"

indique que tous les éléments utilisés par le document d'instance XML qui ont été déclarés dans ce schéma doivent être qualifiés d'espace de noms.

5. Exemple

Note.xml

Ce document XML contient une référence à un schéma XML :

```
<?xml version="1.0" encoding="UTF-8"?>
<note  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation=" Note.xsd">>
        <to>Tove</to>
        <from>Jani</from>
        <heading>Reminder</heading>
        <body>Don't forget me this weekend!</body>
</note>
```

Le fragment suivant :

xmlns="https://www.w3schools.com"

spécifie la déclaration d'espace de noms par défaut. Cette déclaration indique au validateur de schéma que tous les éléments utilisés dans ce document XML sont déclarés dans l'espace de noms "https://www.w3schools.com".

Une fois que vous disposez de l'espace de noms XML Schema Instance :

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

vous pouvez utiliser l'attribut schemaLocation. Cet attribut a deux valeurs, séparées par un espace. La première valeur est l'espace de noms à utiliser. La deuxième valeur est l'emplacement du schéma XML à utiliser pour cet espace de noms :

xsi:schemaLocation="https://www.w3schools.com note.xsd"

Note.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
        <xs:element name="note">
                <xs:complexType>
                        <xs:sequence>
                                <xs:element name="to" type="xs:string"/>
                                <xs:element name="from" type="xs:string"/>
                                <xs:element name="heading" type="xs:string"/>
                                <xs:element name="body" type="xs:string"/>
                        </xs:sequence>
                </xs:complexType>
        </xs:element>
</xs:schema>
```

Le schéma ci-dessus est interprété comme ceci :

- <xs:element name="note"> définit l'élément appelé "note"
- <xs:complexType> l'élément "note" est un type complexe
- <xs:sequence> le type complexe est une séquence d'éléments
- <xs:element name="to" type="xs:string"> l'élément "to" est de type string (texte)
- <xs:element name="from" type="xs:string"> l'élément "from" est de type string
- <xs:element name="heading" type="xs:string"> l'élément "heading" est de type string
- <xs:element name="body" type="xs:string"> l'élément "body" est de type string

6. Types de données XSD : Chaine

a. Type de données de chaîne

Le type de données chaîne peut contenir des caractères, des sauts de ligne, des retours chariot et des tabulations.

XML	XSD
<customer>John Smith</customer>	<xs:element name="customer" type="xs:string"/>

Remarque : Le processeur XML ne modifiera pas la valeur si vous utilisez le type de données chaîne.

b. Type de données NormalizedString

Le type de données normalizedString, (dérivé du type de données String) contient également des caractères, mais le processeur XML supprimera les sauts de ligne, les retours chariot et les tabulations.

XML	XSD
<customer>John Smith</customer>	<xs:element name="customer" type="xs:normalizedString"/>

Remarque : Dans l'exemple ci-dessus, le processeur XML remplacera les tabulations par des espaces.

c. Type de données de jeton

Le type de données de jeton (dérivé du type de données String) contient également des caractères, mais le processeur XML supprimera les sauts de ligne, les retours chariot, les tabulations, les espaces de début et de fin et les espaces multiples.

XML	XSD
<customer>John Smith</customer>	<xs:element name="customer" type="xs:token"/>

Remarque : Dans l'exemple ci-dessus, le processeur XML supprimera les onglets.

d. Types de données de chaîne

Notez que tous les types de données ci-dessous dérivent du type de données String (à l'exception de la chaîne elle-même) !

Name	Description
ENTITIES	
ENTITY	
ID	A string that represents the ID attribute in XML (only used with schema attributes)
IDREF	A string that represents the IDREF attribute in XML (only used with schema attributes)
IDREFS	
language	A string that contains a valid language id
Name	A string that contains a valid XML name
NCName	
NMTOKEN	A string that represents the NMTOKEN attribute in XML (only used with schema attributes)
NMTOKENS	
normalizedString	A string that does not contain line feeds, carriage returns, or tabs
QName	
string	A string
token	A string that does not contain line feeds, carriage returns, tabs, leading or trailing spaces, or multiple spaces

7. Types de données XSD : Date/Heure

Les types de données date et heure sont utilisés pour les valeurs qui contiennent la date et l'heure.

a. Type de données Date

Le type de données date est utilisé pour spécifier une date.

La date est précisée sous la forme suivante « AAAA-MM-JJ » où :

- AAAA indique l'année
- MM indique le mois
- JJ indique le jour

Remarque : Tous les composants sont requis !

XML	XSD
<start>2002-09-24</start>	<xs:element name="start" type="xs:date"/>

Pour spécifier un fuseau horaire, vous pouvez soit entrer une date en heure UTC en ajoutant un "Z" derrière la date - comme ceci :

<start>2002-09-24Z</start>

ou vous pouvez spécifier un décalage par rapport à l'heure UTC en ajoutant une heure positive ou négative derrière la date - comme ceci :

<start>2002-09-24-06:00</start>

OU

<start>2002-09-24+06:00</start>

b. Type de données Time

Le type de données time est utilisé pour spécifier une heure.

L'heure est spécifiée sous la forme suivante "hh:mm:ss" où :

- hh indique l'heure
- mm indique les minutes
- ss indique la seconde

Remarque : Tous les composants sont requis !

XML	XSD
<start>09:00:00</start>	<xs:element name="start" type="xs:time"/>

c. Type de données DateTime

Le type de données dateTime est utilisé pour spécifier une date et une heure.

La dateHeure est spécifiée sous la forme suivante "AAAA-MM-JJThh:mm:ss" où :

- AAAA indique l'année
- MM indique le mois
- JJ indique le jour
- T indique le début de la tranche horaire souhaitée
- hh indique l'heure
- mm indique les minutes
- ss indique la seconde

Remarque : Tous les composants sont requis !

XML	XSD
<startdate>2022-05-30T09:00:00</startdate>	<xs:element name="startdate" type="xs:dateTime"/>

d. Type de données de durée

Le type de données de durée est utilisé pour spécifier un intervalle de temps.

L'intervalle de temps est spécifié sous la forme suivante "PnYnMnDTnHnMnS" où :

- P indique la période (obligatoire)
- nY indique le nombre d'années
- nM indique le nombre de mois
- nD indique le nombre de jours
- T indique le début d'une section de temps (requis si vous allez spécifier des heures, des minutes ou des secondes)
- nH indique le nombre d'heures
- nM indique le nombre de minutes
- nS indique le nombre de secondes

XML	XSD
<period>P5Y</period>	<xs:element name="period" type="xs:duration"/>

Remarque : Pour spécifier une durée négative, entrez un signe moins avant le P

e. Types de données de date et d'heure

Name	Description
date	Defines a date value
dateTime	Defines a date and time value
duration	Defines a time interval
gDay	Defines a part of a date - the day (DD)

gMonth	Defines a part of a date - the month (MM)
gMonthDay	Defines a part of a date - the month and day (MM-DD)
gYear	Defines a part of a date - the year (YYYY)
gYearMonth	Defines a part of a date - the year and month (YYYY-MM)
time	Defines a time value

8. Types de données numériques

a. Type de données décimal

Le type de données décimal est utilisé pour spécifier une valeur numérique.

XML	XSD
<price>999.50</price> OU <price>+999.50</price> OU <price>-999.50</price>	<xs:element name="price" type="xs:decimal"/>

b. Type de données entier

Le type de données entier est utilisé pour spécifier une valeur numérique sans composante fractionnaire.

XML	XSD
<price>999</price> OU <price>+999</price> OU <price>-999</price>	<xs:element name="price" type="xs:integer"/>

c. Types de données numériques

Notez que tous les types de données ci-dessous dérivent du type de données Decimal (à l'exception de decimal lui-même) !

Name	**Description**
byte	A signed 8-bit integer
decimal	A decimal value
int	A signed 32-bit integer

integer	An integer value
long	A signed 64-bit integer
negativeInteger	An integer containing only negative values (..,-2,-1)
nonNegativeInteger	An integer containing only non-negative values (0,1,2,..)
nonPositiveInteger	An integer containing only non-positive values (..,-2,-1,0)
positiveInteger	An integer containing only positive values (1,2,..)
short	A signed 16-bit integer
unsignedLong	An unsigned 64-bit integer
unsignedInt	An unsigned 32-bit integer
unsignedShort	An unsigned 16-bit integer
unsignedByte	An unsigned 8-bit integer

9. Types de données divers XSD

Les autres types de données divers sont boolean, base64Binary, hexBinary, float, double, anyURI, QName et NOTATION.

a. Type de données booléen

Le type de données booléen est utilisé pour spécifier une valeur vraie ou fausse.

XML	XSD
<price disabled="true">999</price>	<xs:attribute name="disabled" type="xs:boolean"/>

b. Types de données binaires

Les types de données binaires sont utilisés pour exprimer des données au format binaire.

Nous avons deux types de données binaires :

- base64Binary (données binaires encodées en Base64)
- hexBinary (données binaires codées en hexadécimal)

c. Type de données AnyURI

Le type de données anyURI est utilisé pour spécifier un URI.

XML	XSD

<pic src="https://www.w3schools.com/images/smiley.gif" />	<xs:attribute name="src" type="xs:anyURI"/>

Remarque : si un URI contient des espaces, remplacez-les par %20.

d. Types de données divers

Name	Description
anyURI	
base64Binary	
boolean	
double	
float	
hexBinary	
NOTATION	
QName	

10. Éléments simples XSD

a. Qu'est-ce qu'un élément complexe ?

Un élément simple est un élément XML qui ne contient que du texte. Il ne peut contenir aucun autre élément ou attribut. Le texte peut être de plusieurs types différents. Il peut s'agir de l'un des types inclus dans la définition du schéma XML (booléen, chaîne, date, etc.) ou d'un type personnalisé que vous pouvez définir vous-même.

Vous pouvez également ajouter des restrictions (facettes) à un type de données afin de limiter son contenu, ou vous pouvez exiger que les données correspondent à un modèle spécifique.

b. Syntaxe d'élément simple

```
<xs:element name="xxx" type="yyy" [default|fixed=valeur] />
```

où xxx est le nom de l'élément et yyy est le type de données de l'élément.

XML Schema a beaucoup de types de données intégrés. Les types les plus courants sont :

- xs : chaîne
- xs : décimal

- xs : entier
- xs : booléen
- xs :date
- xs : heure

c. Exemple

Xml Voici quelques éléments XML :	xsd Et voici les définitions d'éléments simples correspondantes :
<lastname>Refsnes</lastname> <age>36</age> <dateborn>1970-03-27</dateborn>	<xs:element name="lastname" type="xs:string"/> <xs:element name="age" type="xs:integer"/> <xs:element name="dateborn" type="xs:date"/>

d. Attributs XSD

Les éléments simples ne peuvent pas avoir d'attributs.

11. Éléments complexes XSD

a. Qu'est-ce qu'un élément complexe ?

Un élément complexe est un élément XML qui contient d'autres éléments et/ou attributs.
Il existe quatre types d'éléments complexes :

- éléments vides
- éléments qui ne contiennent que d'autres éléments
- éléments qui ne contiennent que du texte
- éléments contenant à la fois d'autres éléments et du texte

Remarque : Chacun de ces éléments peut également contenir des attributs !

b. Comment définir un élément complexe

Regardez cet élément XML complexe, "employee", qui ne contient que d'autres éléments :

XML
<employee> <firstname>John</firstname> <lastname>Smith</lastname> </employee>

<table>
<tr><td colspan="2">XSD
Nous pouvons définir un élément complexe dans un schéma XML de deux manières différentes :</td></tr>
<tr><td rowspan="2">1. L'élément "employé" peut être déclaré directement en nommant l'élément, comme ceci :</td><td><xs:element name="employee">
<xs:complexType>
<xs:sequence>
<xs:element name="firstname" type="xs:string"/>
<xs:element name="lastname" type="xs:string"/>
</xs:sequence>
</xs:complexType>
</xs:element></td></tr>
<tr><td>Si vous utilisez la méthode décrite ci-dessus, seul l'élément "employé" peut utiliser le type complexe spécifié. Notez que les éléments enfants, "firstname" et "lastname", sont entourés de l'indicateur <sequence>. Cela signifie que les éléments enfants doivent apparaître dans le même ordre qu'ils sont déclarés. Vous en apprendrez plus sur les indicateurs dans le chapitre Indicateurs XSD.</td></tr>
<tr><td>2. L'élément "employee" peut avoir un attribut type qui fait référence au nom du type complexe à utiliser :</td><td><xs:element name="employee" type="personinfo"/>

<xs:complexType name="personinfo">
<xs:sequence>
<xs:element name="firstname" type="xs:string"/>
<xs:element name="lastname" type="xs:string"/>
</xs:sequence>
</xs:complexType></td></tr>
</table>

Si vous utilisez la méthode décrite ci-dessus, plusieurs éléments peuvent faire référence au même type complexe, comme ceci :

XSD

```
<xs:element name="employee" type="personinfo"/>
<xs:element name="student" type="personinfo"/>
<xs:element name="member" type="personinfo"/>

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

XSD

```
<xs:element name="employee" type="fullpersoninfo"/>

<xs:complexType name="personinfo">
 <xs:sequence>
  <xs:element name="firstname" type="xs:string"/>
  <xs:element name="lastname" type="xs:string"/>
 </xs:sequence>
</xs:complexType>

<xs:complexType name="fullpersoninfo">
 <xs:complexContent>
  <xs:extension base="personinfo">
   <xs:sequence>
    <xs:element name="address" type="xs:string"/>
    <xs:element name="city" type="xs:string"/>
    <xs:element name="country" type="xs:string"/>
   </xs:sequence>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>
```

c. Attributs XSD

Les éléments simples ne peuvent pas avoir d'attributs. Si un élément a des attributs, il est considéré comme étant de type complexe. Mais l'attribut lui-même est toujours déclaré comme un type simple.

d. Syntaxe d'un attribut XSD

<xs:attribute name="xxx" type="yyy"[default|fixed=valeur] [use="required|optional"]/>

où xxx est le nom de l'attribut et yyy spécifie le type de données de l'attribut.

XML Schema a beaucoup de types de données intégrés. Les types les plus courants sont :

- xs : chaîne
- xs : décimal
- xs : entier
- xs : booléen
- xs :date
- xs : heure

e. Exemple d'un attribut XSD

Xml Voici quelques éléments XML :	xsd Et voici la définition d'attribut correspondante :
<lastname lang="EN">Smith</lastname>	<xs:attribute name="lang" type="xs:string"/>

f. Éléments vides complexes

XML Un élément XML vide :
<product prodid="1345" />

XSD
<xs:element name="product"> <xs:complexType> <xs:attribute name="prodid" type="xs:positiveInteger"/> </xs:complexType> </xs:element>
OU
<xs:element name="product" type="prodtype"/> <xs:complexType name="prodtype"> <xs:attribute name="prodid" type="xs:positiveInteger"/> </xs:complexType>

g. Types complexes contenant uniquement des éléments

XML
<person> <firstname>John</firstname> <lastname>Smith</lastname> </person>

XSD	
<xs:element name="person"> <xs:complexType> <xs:sequence> <xs:element name="firstname" type="xs:string"/> <xs:element name="lastname" type="xs:string"/> </xs:sequence> </xs:complexType> </xs:element>	Remarquez la balise <xs:sequence>. Cela signifie que les éléments définis ("firstname" et "lastname") doivent apparaître dans cet ordre à l'intérieur d'un élément "person".

OU	
<xs:element name="person" type="persontype"/> <xs:complexType name="persontype"> <xs:sequence> <xs:element name="firstname" type="xs:string"/> <xs:element name="lastname" type="xs:string"/> </xs:sequence> </xs:complexType>	vous pouvez donner un nom à l'élément complexType, et laisser l'élément "person" avoir un attribut type qui fait référence au nom du complexType (si vous utilisez cette méthode, plusieurs éléments peuvent faire référence au même type complexe)

h. Éléments textuels complexes

Ce type ne contient que du contenu simple (texte et attributs), nous ajoutons donc un élément simpleContent autour du contenu. Lorsque vous utilisez un contenu simple, vous devez définir une extension OU une restriction dans l'élément simpleContent, comme ceci :

<xs:element name="somename"> <xs:complexType> <xs:simpleContent> <xs:extension base="basetype"> .... </xs:extension> </xs:simpleContent> </xs:complexType> </xs:element>	OU	<xs:element name="somename"> <xs:complexType> <xs:simpleContent> <xs:restriction base="basetype"> .... </xs:restriction> </xs:simpleContent> </xs:complexType> </xs:element>

Conseil : Utilisez l'élément extension/restriction pour développer ou limiter le type simple de base de l'élément.

XML
<shoesize country="france">35</shoesize>

XSD	
L'exemple suivant déclare un complexType, "shoesize". Le contenu est défini comme une valeur entière, et l'élément	<xs:element name="shoesize"> <xs:complexType> <xs:simpleContent> <xs:extension base="xs:integer"> <xs:attribute name="country" type="xs:string" />

"shoesize" contient également un attribut nommé "country":	</xs:extension> </xs:simpleContent> </xs:complexType> </xs:element>
OU	
Nous pourrions également donner un nom à l'élément complexType, et laisser l'élément "shoesize" avoir un attribut type qui fait référence au nom du complexType (si vous utilisez cette méthode, plusieurs éléments peuvent faire référence au même type complexe) :	<xs:element name="shoesize" type="shoetype"/> <xs:complexType name="shoetype"> <xs:simpleContent> <xs:extension base="xs:integer"> <xs:attribute name="country" type="xs:string" /> </xs:extension> </xs:simpleContent> </xs:complexType>

i. Types complexes avec contenu mixte

Un élément de type complexe mixte peut contenir des attributs, des éléments et du texte.

XML
Un élément XML, "letter", qui contient à la fois du texte et d'autres éléments :

```
<letter>
 Dear Mr. <name>John Smith</name>.
 Your order <orderid>1032</orderid>
 will be shipped on <shipdate>2001-07-13</shipdate>.
</letter>
```

XSD
Le schéma suivant déclare l'élément "lettre" :

```
<xs:element name="letter">
 <xs:complexType mixed="true">
  <xs:sequence>
   <xs:element name="name" type="xs:string"/>
   <xs:element name="orderid" type="xs:positiveInteger"/>
   <xs:element name="shipdate" type="xs:date"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

Remarque : Pour permettre aux données de caractères d'apparaître entre les éléments enfants de "letter", l'attribut mixte doit être défini sur "true". La balise <xs:sequence>

signifie que les éléments définis (name, orderid et shipdate) doivent apparaître dans cet ordre à l'intérieur d'un élément "letter".

On pourrait aussi donner un nom à l'élément complexType, et laisser l'élément "letter" avoir un attribut type qui fait référence au nom du complexType (si vous utilisez cette méthode, plusieurs éléments peuvent faire référence au même type complexe) :

XSD
<xs:element name="letter" type="lettertype"/> <xs:complexType name="lettertype" mixed="true"> <xs:sequence> <xs:element name="name" type="xs:string"/> <xs:element name="orderid" type="xs:positiveInteger"/> <xs:element name="shipdate" type="xs:date"/> </xs:sequence> </xs:complexType>

12. Indicateurs XSD

Il y a sept indicateurs regroupés en trois parties

Indicateurs de commande ::

- All
- Choice
- Sequence

Indicateurs d'occurrence:

- maxOccurs
- minOccurs

Indicateurs Groupe:

- Group name
- attributeGroup name

a. Indicateurs de commande

Les indicateurs d'ordre permettent de définir l'ordre des éléments.

L'indicateur <all> spécifie que les éléments enfants peuvent apparaître dans n'importe quel ordre et que chaque élément	<xs:element name="person"> <xs:complexType> <xs:all> <xs:element name="firstname" type="xs:string"/>

enfant ne doit apparaître qu'une seule fois : **Remarque :** Lorsque vous utilisez l'indicateur <all>, vous pouvez régler l'indicateur <minOccurs> sur 0 ou 1 et l'indicateur <maxOccurs> ne peut être réglé que sur 1 (les <minOccurs> et <maxOccurs> sont décrits plus loin).	<xs:element name="lastname" type="xs:string"/> </xs:all> </xs:complexType> </xs:element>
L'indicateur <choice> spécifie qu'un élément enfant ou un autre peut apparaître :	<xs:element name="person"> <xs:complexType> <xs:choice> <xs:element name="employee" type="employee"/> <xs:element name="member" type="member"/> </xs:choice> </xs:complexType> </xs:element>
L'indicateur <sequence> spécifie que les éléments enfants doivent apparaître dans un ordre spécifique :	<xs:element name="person"> <xs:complexType> <xs:sequence> <xs:element name="firstname" type="xs:string"/> <xs:element name="lastname" type="xs:string"/> </xs:sequence> </xs:complexType> </xs:element>

b. Indicateurs d'occurrence

Les indicateurs d'occurrence sont utilisés pour définir la fréquence à laquelle un élément peut se produire.

Remarque : Pour tous les indicateurs "Ordre" et "Groupe" (tous, tous, choix, séquence, nom de groupe et référence de groupe), la valeur par défaut pour maxOccurs et minOccurs est 1.

L'indicateur <maxOccurs> spécifie le nombre maximum de fois qu'un élément peut se produire :	<xs:element name="person"> <xs:complexType> <xs:sequence> <xs:element name="full_name" type="xs:string"/> <xs:element name="child_name" type="xs:string"

L'exemple ci indique que l'élément "child_name" peut apparaître au minimum une fois (la valeur par défaut de minOccurs est 1) et au maximum dix fois dans l'élément "person".	maxOccurs="10"/> </xs:sequence> </xs:complexType> </xs:element>
L'indicateur <minOccurs> spécifie le nombre minimum de fois qu'un élément peut se produire : L'exemple ci-dessus indique que l'élément "child_name" peut apparaître au minimum zéro fois et au maximum dix fois dans l'élément "person". **Astuce :** Pour permettre à un élément d'apparaître un nombre illimité de fois, utilisez l'instruction maxOccurs="unbounded" :	<xs:element name="person"> <xs:complexType> <xs:sequence> <xs:element name="full_name" type="xs:string"/> <xs:element name="child_name" type="xs:string" maxOccurs="10" minOccurs="0"/> </xs:sequence> </xs:complexType> </xs:element>

c. Indicateurs du groupe

Les indicateurs de groupe sont utilisés pour définir des ensembles d'éléments liés.

- *Groupes d'éléments*

Les groupes d'éléments sont définis avec la déclaration de groupe, comme ceci :

```
<xs:group name="groupname">
...
</xs:group>
```

Vous devez définir un élément all, choice ou sequence dans la déclaration de	<xs:group name="persongroup"> <xs:sequence> <xs:element name="firstname" type="xs:string"/>

groupe. L'exemple suivant définit un groupe nommé "persongroup", qui définit un groupe d'éléments qui doivent apparaître dans une séquence exacte :	<xs:element name="lastname" type="xs:string"/> <xs:element name="birthday" type="xs:date"/> </xs:sequence> </xs:group>
Après avoir défini un groupe, vous pouvez le référencer dans une autre définition, comme ceci :	<xs:group name="persongroup"> <xs:sequence> <xs:element name="firstname" type="xs:string"/> <xs:element name="lastname" type="xs:string"/> <xs:element name="birthday" type="xs:date"/> </xs:sequence> </xs:group> <xs:element name="person" type="personinfo"/> <xs:complexType name="personinfo"> <xs:sequence> <xs:group ref="persongroup"/> <xs:element name="country" type="xs:string"/> </xs:sequence> </xs:complexType>

- *Groupes d'attributs*

Les groupes d'attributs sont définis avec la déclaration attributeGroup, comme ceci :

<xs:attributeGroup name="groupname">
...
</xs:attributeGroup>

L'exemple suivant définit un groupe d'attributs nommé "personattrgroup" :	<xs:attributeGroup name="personattrgroup"> <xs:attribute name="firstname" type="xs:string"/> <xs:attribute name="lastname" type="xs:string"/> <xs:attribute name="birthday" type="xs:date"/> </xs:attributeGroup>
Après avoir défini un groupe d'attributs, vous pouvez le référencer dans une autre définition, comme ceci :	<xs:attributeGroup name="personattrgroup"> <xs:attribute name="firstname" type="xs:string"/> <xs:attribute name="lastname" type="xs:string"/> <xs:attribute name="birthday" type="xs:date"/> </xs:attributeGroup> <xs:element name="person"> <xs:complexType>

```
  <xs:attributeGroup ref="personattrgroup"/>
 </xs:complexType>
</xs:element>
```

13. Restrictions/facettes XSD

Les restrictions sont utilisées pour définir des valeurs acceptables pour les éléments ou attributs XML. Les restrictions sur les éléments XML sont appelées facettes.

a. Restrictions sur les valeurs

L'exemple suivant définit un élément appelé "age" avec une restriction. La valeur de l'âge ne peut pas être inférieure à 0 ou supérieure à 120 :

```
<xs:element name="age">
 <xs:simpleType>
  <xs:restriction base="xs:integer">
   <xs:minInclusive value="0"/>
   <xs:maxInclusive value="120"/>
  </xs:restriction>
 </xs:simpleType>
</xs:element>
```

b. Restrictions sur un ensemble de valeurs

Pour limiter le contenu d'un élément XML à un ensemble de valeurs acceptables, nous utiliserions la contrainte d'énumération.

L'exemple ci-dessous définit un élément appelé "voiture" avec une restriction. Les seules valeurs acceptables sont : Audi, Golf, BMW :

```
<xs:element name="car">
 <xs:simpleType>
  <xs:restriction base="xs:string">
   <xs:enumeration value="Audi"/>
   <xs:enumeration value="Golf"/>
   <xs:enumeration value="BMW"/>
  </xs:restriction>
 </xs:simpleType>
</xs:element>
```

OU

```
<xs:element name="car" type="carType"/>

<xs:simpleType name="carType">
 <xs:restriction base="xs:string">
  <xs:enumeration value="Audi"/>
  <xs:enumeration value="Golf"/>
  <xs:enumeration value="BMW"/>
 </xs:restriction>
</xs:simpleType>
```

Remarque : Dans ce cas, le type "carType" peut être utilisé par d'autres éléments car il ne fait pas partie de l'élément "car"

c. Restrictions sur une série de valeurs

Pour limiter le contenu d'un élément XML afin de définir une série de chiffres ou de lettres pouvant être utilisés, nous utiliserions la contrainte de modèle.

L'exemple ci-dessous définit un élément appelé "lettre" avec une restriction. La seule valeur acceptable est UNE des lettres MINUSCULES de a à z :

```
<xs:element name="letter">
 <xs:simpleType>
  <xs:restriction base="xs:string">
   <xs:pattern value="[a-z]"/>
  </xs:restriction>
 </xs:simpleType>
</xs:element>
```

L'exemple suivant définit un élément appelé "initials" avec une restriction. La seule valeur acceptable est TROIS des lettres MAJUSCULES de a à z :

```
<xs:element name="initials">
 <xs:simpleType>
  <xs:restriction base="xs:string">
   <xs:pattern value="[A-Z][A-Z][A-Z]"/>
  </xs:restriction>
 </xs:simpleType>
</xs:element>
```

L'exemple suivant définit également un élément appelé "initials" avec une restriction. La seule valeur acceptable est TROIS des lettres MINUSCULES OU MAJUSCULES de a à z :

```
<xs:element name="initials">
 <xs:simpleType>
  <xs:restriction base="xs:string">
   <xs:pattern value="[a-zA-Z][a-zA-Z][a-zA-Z]"/>
  </xs:restriction>
 </xs:simpleType>
</xs:element>
```

L'exemple suivant définit un élément appelé "choix" avec une restriction. La seule valeur acceptable est UNE des lettres suivantes : x, y, OU z :

```
<xs:element name="choice">
 <xs:simpleType>
  <xs:restriction base="xs:string">
   <xs:pattern value="[xyz]"/>
  </xs:restriction>
 </xs:simpleType>
</xs:element>
```

L'exemple suivant définit un élément appelé "prodid" avec une restriction. La seule valeur acceptable est CINQ chiffres dans une séquence, et chaque chiffre doit être compris entre 0 et 9 :

```
<xs:element name="prodid">
 <xs:simpleType>
  <xs:restriction base="xs:integer">
   <xs:pattern value="[0-9][0-9][0-9][0-9][0-9]"/>
  </xs:restriction>
 </xs:simpleType>
</xs:element>
```

d. Autres restrictions sur une série de valeurs

L'exemple ci-dessous définit un élément appelé "lettre" avec une restriction. La valeur acceptable est zéro ou plusieurs occurrences de lettres minuscules de a à z :

```
<xs:element name="letter">
 <xs:simpleType>
  <xs:restriction base="xs:string">
   <xs:pattern value="([a-z])*"/>
  </xs:restriction>
 </xs:simpleType>
</xs:element>
```

L'exemple suivant définit également un élément appelé "lettre" avec une restriction. La valeur acceptable est une ou plusieurs paires de lettres, chaque paire étant constituée d'une lettre minuscule suivie d'une lettre majuscule. Par exemple, "sToP" sera validé par ce pattern, mais pas "Stop" ou "STOP" ou "stop" :

```
<xs:element name="letter">
 <xs:simpleType>
  <xs:restriction base="xs:string">
   <xs:pattern value="([a-z][A-Z])+"/>
```

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

L'exemple suivant définit un élément appelé "gender" avec une restriction. La seule valeur acceptable est masculin OU féminin :

```
<xs:element name="gender">
 <xs:simpleType>
  <xs:restriction base="xs:string">
   <xs:pattern value="male|female"/>
  </xs:restriction>
 </xs:simpleType>
</xs:element>
```

L'exemple suivant définit un élément appelé "password" avec une restriction. Il doit y avoir exactement huit caractères dans une ligne et ces caractères doivent être des lettres minuscules ou majuscules de a à z, ou un nombre de 0 à 9 :

```
<xs:element name="password">
 <xs:simpleType>
  <xs:restriction base="xs:string">
   <xs:pattern value="[a-zA-Z0-9]{8}"/>
  </xs:restriction>
 </xs:simpleType>
</xs:element>
```

e. Restrictions sur les caractères d'espacement

Pour spécifier comment les caractères d'espacement doivent être gérés, nous utiliserons la contrainte whiteSpace.

Cet exemple définit un élément appelé "adresse" avec une restriction. La contrainte whiteSpace est définie sur "preserve", ce qui signifie que le processeur XML NE SUPPRIMERA AUCUN espace blanc :

```
<xs:element name="address">
 <xs:simpleType>
  <xs:restriction base="xs:string">
   <xs:whiteSpace value="preserve"/>
  </xs:restriction>
```

```
  </xs:simpleType>
</xs:element>
```

Cet exemple définit également un élément appelé "adresse" avec une restriction. La contrainte whiteSpace est définie sur "replace", ce qui signifie que le processeur XML REMPLACERA tous les caractères d'espace blanc (sauts de ligne, tabulations, espaces et retours chariot) par des espaces :

```
<xs:element name="address">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:whiteSpace value="replace"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

Cet exemple définit également un élément appelé "adresse" avec une restriction. La contrainte whiteSpace est définie sur "collapse", ce qui signifie que le processeur XML SUPPRIMERA tous les espaces blancs (les sauts de ligne, les tabulations, les espaces, les retours chariot sont remplacés par des espaces, les espaces de début et de fin sont supprimés et plusieurs espaces sont réduits à un seul espace):

```
<xs:element name="address">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:whiteSpace value="collapse"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

f. Restrictions de longueur

Pour limiter la longueur d'une valeur dans un élément, nous utiliserions les contraintes length, maxLength et minLength.

Cet exemple définit un élément appelé "password" avec une restriction. La valeur doit comporter exactement huit caractères :

```
<xs:element name="password">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:length value="8"/>
```

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

Cet exemple définit un autre élément appelé "password" avec une restriction. La valeur doit comporter au moins cinq caractères et au maximum huit caractères :

```
<xs:element name="password">
 <xs:simpleType>
  <xs:restriction base="xs:string">
   <xs:minLength value="5"/>
   <xs:maxLength value="8"/>
  </xs:restriction>
 </xs:simpleType>
</xs:element>
```

g. Restrictions pour les types de données

Constraint	Description
enumeration	Defines a list of acceptable values
fractionDigits	Specifies the maximum number of decimal places allowed. Must be equal to or greater than zero
length	Specifies the exact number of characters or list items allowed. Must be equal to or greater than zero
maxExclusive	Specifies the upper bounds for numeric values (the value must be less than this value)
maxInclusive	Specifies the upper bounds for numeric values (the value must be less than or equal to this value)
maxLength	Specifies the maximum number of characters or list items allowed. Must be equal to or greater than zero
minExclusive	Specifies the lower bounds for numeric values (the value must be greater than this value)
minInclusive	Specifies the lower bounds for numeric values (the value must be greater than or equal to this value)
minLength	Specifies the minimum number of characters or list items allowed. Must be equal to or greater than zero
pattern	Defines the exact sequence of characters that are acceptable

totalDigits	Specifies the exact number of digits allowed. Must be greater than zero
whiteSpace	Specifies how white space (line feeds, tabs, spaces, and carriage returns) is handled

IV. Validation des documents XML : DTD

Un document XML avec une syntaxe correcte est appelé « document bien formé ». Un document XML avec une syntaxe correcte et validé par une DTD est appelé « document bien formé » et « valide ».

L'extension du fichier DTD est « .dtd » qui est l'acronyme de « Document Type Definition ». Il définit la structure et les éléments légaux et attributs d'un document XML

1. Syntaxes

Le mot-clé SYSTEM est important et indique qu'il s'agit d'une DTD qui vous est propre. L'alternative est le mot-clé PUBLIC

a. Externe au fichier XML

```
<!DOCTYPE element_racine SYSTEM "URI vers la DTD">
```

Exemple

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

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

b. Interne au fichier XML

```
<!DOCTYPE element_ racine [

    Corps du dtd

]>
```

Exemple

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

<!DOCTYPE etudiants [

    <!ELEMENT etudiants ( matricule, nom )>
```

<!ELEMENT matricule (#PCDATA)>

<!ELEMENT nom (#PCDATA)>

]>

c. *Mixte (Internet et externe) au fichier* XML

Enfin, il est possible de mélanger les deux notations pour avoir une partie de la DTD dans un fichier séparé et une autre partie embarquée dans le document XML :

Syntaxe :

<!DOCTYPE element_racine SYSTEM "URI vers la DTD"> [

Corps du dtd

]>

Exemple

<?xml version="1.0" encoding="utf-8" ?>

< !DOCTYPE racine SYSTEM " racine.dtd" [

<!ELEMENT matricule (#PCDATA)>

<!ELEMENT nom (#PCDATA)>

]>

<racine>

2. Définition d'un élément

Voici une synthèse de cette syntaxe :

<!ELEMENT nom_element DEF_CONTENU>

DEF_CONTENU peut contenir :

- EMPTY : l'élément n'a pas de contenu ; il est donc vide. Il peut cependant avoir des attributs.
- ANY : l'élément peut contenir n'importe quel élément présent dans la DTD.
- (#PCDATA) : l'élément contient du texte. Le caractère # est là pour éviter toute ambiguïté avec une balise et indique au parseur qu'il s'agit d'un mot-clé. PCDATA signifie Parsable Character DATA.

- Un élément placé entre parenthèses comme (nom_element). Le nom d'un élément désigne une référence vers un élément décrit dans une autre partie de la DTD
- Un ensemble d'éléments séparés par des opérateurs, le tout placé entre parenthèses. L'opérateur de choix, représenté par le caractère |, indique que l'un ou l'autre de deux éléments (ou deux ensembles d'éléments) doit être présent. L'opérateur de suite (ou séquence), représenté par le caractère virgule (,) indique que les deux éléments (ou les deux ensembles d'éléments) doivent être présents. Des parenthèses supplémentaires peuvent être utilisées pour lever les ambiguïtés

Tableau récapitulatif

DEF_CONTENU	**Description**
<!ELEMENT element (#PCDATA)>	l'élément pair contient du texte
<!ELEMENT element EMPTY>	l'élément n'a pas de contenu
<!ELEMENT element ANY>	l'élément peut contenir n'importe quel élément présent dans la DTD
<!ELEMENT element a*>	l'élément peut contenir l'élément a zéro ou plusieurs fois
<!ELEMENT element a+>	l'élément peut contenir l'élément a un ou plusieurs fois
<!ELEMENT element (a, b, c)>	l'élément doit contenir les éléments a, b et c dans cet ordre
<!ELEMENT element (a|#PCDATA)>	l'élément peut contenir l'élément a ou du texte
<!ELEMENT element (a|b|c)*>	l'élément peut contenir zéro ou plusieurs fois l'élément a, b ou c

3. Quelques exemples :

XML	DTD
<personne> <nom_prenom>Brillant Alexandre</nom_prenom> </personne> OU <personne> <nom>Brillant</nom> </personne>	<!ELEMENT personne (nom_prenom | nom)> <!ELEMENT nom_prenom (#PCDATA)> <!ELEMENT nom (#PCDATA)>

XMl	DTD
<personne> <prenom>Alexandre</prenom> <nom>Brillant</nom> </personne>	<!ELEMENT personne(prenom,nom)> <!ELEMENT prenom (#PCDATA)> <!ELEMENT nom (#PCDATA)

XML	DTD
	<!ELEMENT account (name, sortcode?, accnumber, transactions)> <!ELEMENT name #PCDATA> <!ELEMENT sortcode #PCDATA> <!ELEMENT accnumber #PCDATA> <!ELEMENT transactions (deposit, credit, adjustment)*> <!ELEMENT deposit (date, amount)> <!ELEMENT credit (date, amount)> <!ELEMENT adjustment (date, amount)> <!ELEMENT date #PCDATA> <!ELEMENT amount #PCDATA>

4. Définition d'un attribut

Les attributs sont précisés dans l'instruction ATTLIST. Cette dernière, étant indépendante de l'instruction ELEMENT, on précise à nouveau le nom de l'élément sur lequel s'applique le ou les attributs.

a. Syntaxe

<!ATTLIST element
Nom_attribut TYPE OBLIGATION VALEUR_PAR_DEFAUT >

Types	Explication
CDATA	du texte (Character Data) ;
ID	un identifiant unique (combinaison de chiffres et de lettres) ;
IDREF	une référence vers un ID ;
IDREFS	une liste de références vers des ID (séparation par un blanc) ;
NMTOKEN	un mot (donc pas de blanc) ;
NMTOKENS	une liste de mots (séparation par un blanc) ;
Une énumération de valeurs	chaque valeur est séparée par le caractère |

L'OBLIGATION ne concerne pas les énumérations qui sont suivies d'une valeur par défaut. Dans les autres cas, on l'exprime ainsi :

Obligations	explication
#REQUIRED	attribut obligatoire.
#IMPLIED	attribut optionnel.
#FIXED	attribut toujours présent avec une valeur. Cela peut servir, par exemple, à imposer la présence d'un espace de noms

La VALEUR_PAR_DEFAUT est présente pour l'énumération ou lorsque la valeur est typée avec #IMPLIED ou #FIXED.

b. Exemple

L'élément chapitre possède ici un attribut titre obligatoire et un attribut auteur optionnel.

<!ATTLIST crayon couleur (rouge|vert|bleu) "bleu">

L'élément crayon possède un attribut couleur dont les valeurs font partie de l'ensemble rouge, vert, bleu.

5. Définition d'une entité

Les entités sont déclarées par l'instruction ENTITY.

a. Syntaxe interne

Syntaxe	Exemple
<!ENTITY nom "VALEUR">	

b. Syntaxe externe

Les mots-clés SYSTEM et PUBLIC servent donc à réaliser un lien vers une valeur présente dans un fichier.

Syntaxe	Exemple
<!ENTITY nom SYSTEM "URI vers le document">	<!ENTITY nom SYSTEM "unTexte.txt">