RÉPUBLIQUE ALGÉRIENNE DÉMOCRATIQUE ET POPULAIRE

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université Echahid Hamma Lakhdar- EL OUED

Faculté des Sciences Exactes

Département d'informatique

Intitulé du cours :

Données semi-structurées (DSS)

Ce cours est destiné aux étudiants de :

3ème année Licence en Systèmes Informatiques

(SEMESTRE 6)

Préparé par : Dr. ABBAS MESSAOUD

Maître de Conférences -A-

messaoud-abbas@univ-eloued.dz

Course Outline

Preface

This booklet is a teaching aid primarily intended for third-year Computer Science undergraduate students, specialising in “Computer Systems (CS).” It contains lecture notes on the module “Semi-Structured Data” as well as practical exercises. The objective is to facilitate the study and mastery of models, operations, and techniques related to the use of semi-structured data, as an alternative to highly structured data models. This includes topics such as schema definition, XML document validation, query languages, XML document access management, etc.

I. XML

1.1 Introduction

XML may appear similar to HTML, but they are completely different. Both are successors of SGML, which itself originated from GML. The latter was developed by IBM to separate the appearance of text documents from their content, particularly for technical documentation. It is easy to change the appearance (layout, fonts, colors, etc.) of such a document without rewriting a single line of content. Other languages, like LaTeX, are similar: you write the text without worrying about the layout. This also makes it easier to translate the document into other languages.

In contrast, WYSIWYG software like Word mixes layout and content. Even with styles, it remains difficult to modify the appearance of a document without having to partially rewrite it.

XML can be seen as a generalization of SGML and HTML, allowing for the construction of all kinds of documents.

What is XML ("Extensible Markup Language")?

It is a language used to represent and structure information using tags. Anyone can define and use tags as they wish.

Example:

```
text ... <TAG> ... text … </TAG> ...
```

The XML format is at the core of many current processes:

- data exchange between servers and clients,
- programming tools and languages,
- native XML databases.

XML defines the syntax of documents, but applications define the tags, their meaning, and what they should contain. APIs exist in all programming languages to read and write XML documents.

Example of an XML file
An XML file represents structured (well-typed) information. The following example models a route composed of steps:

```
xml
CopyEdit
<?xml version="1.0" encoding="utf-8"?>
<!-- fictitious route -->
<itineraire>
   <etape distance="0km">départ</etape>
   <etape distance="13km">turn right</etape>
   <etape distance="22km">arrival</etape>
</itineraire>
```

You can choose tags and attributes as you wish. They just need to be consistent and regular so they can be processed by software (a parser).

You could do this, but good luck processing it:

```
xml
CopyEdit
<?xml version="1.0" encoding="utf-8"?>
<!-- fictitious route -->
<itineraire>
<etape distance="0km">départ</etape>
<Etape><distance>13km</distance>turn right</Etape>
<etape distance="22km"><infos>arrival</infos></etape>
</itineraire>
```

Example applications of XML

XML as a file format:

Many software tools directly use an XML format to save their files:

- Office software: LibreOffice uses the OpenDocument format.
- Vector drawing with Inkscape uses the SVG format.
- Mathematical equations use the MathML format,
- and more…

XML databases

Native XML databases: data is stored in XML format and queries are written in a language (XQuery) that allows performing the equivalent of SQL.

Data exchange between clients and servers

XML is the format used to represent volatile data in many protocols such as RSS, XML-RPC, SOAP, AJAX, and others.

1.2 Structure of an XML Document

An XML document is composed of several parts:

a) Document header specifying the version and encoding,

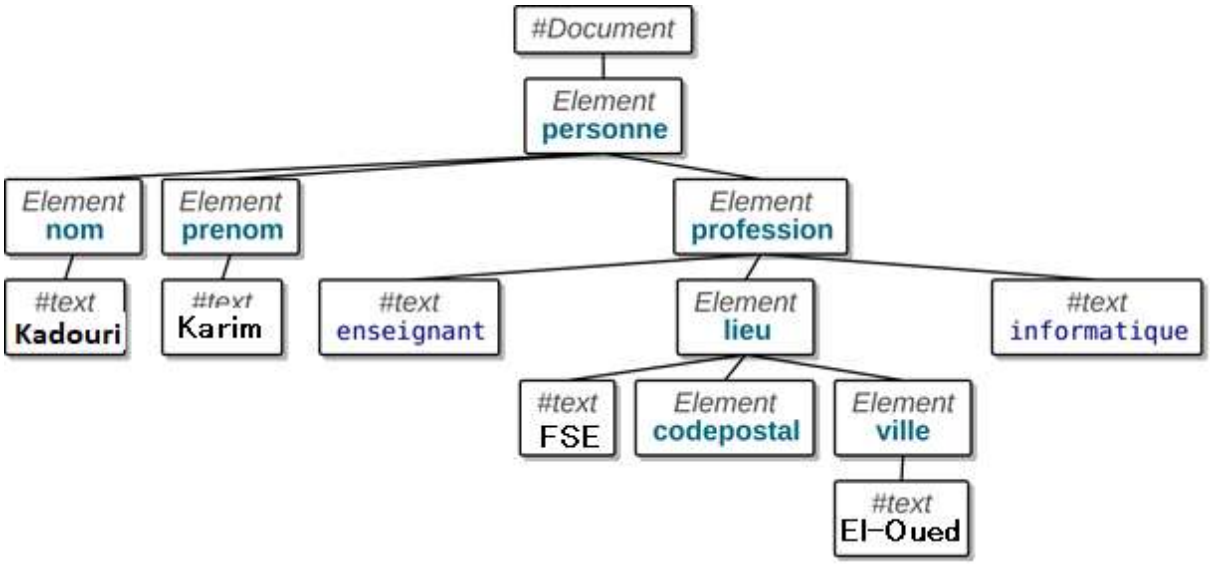
b) Optional rules to verify if the document is valid (“Document type”),

c) A tree of elements starting from an element called the root.

- An element has a name, attributes, and content.
- The content of an element can be:
  - nothing: an empty element noted `<name/>` or `<name attributes.../>`
  - text
  - other elements (child elements).
- A non-empty element is delimited by an opening tag and a closing tag.
  - an opening tag is noted `<name attributes...>`
  - a closing tag is noted `</name>`

Here is an example of an XML document representing a person:

```
xml
CopyEdit
<?xml version="1.0" encoding="utf-8"?>
<personne>
        <nom>Kadouri</nom>
        <prenom>Karim</prenom>
        <profession>enseignan
                <lieu>
            Faculté Sciences Exactes
                <codepostal/>
                    <ville>El-Oud</ville>
                </lieu>
         informatique
        </profession>
</personne>
```



Prologue XML

La première ligne d'un document XML est appelée prologue XML. Elle spécifie la version de la norme XML utilisée (1.0 ou 1.1 qui sont très similaires) ainsi que l'encodage :

```
<?xml version="1.0" encoding="utf-8"?>
<?xml version="1.0" encoding="iso-8859-15"?>
```

Il y a plusieurs normes d'encodage :

• ASCII ne représente que les 128 premiers caractères Unicode.

• ISO 8859-1 ou Latin-1 représente 191 caractères de l'alphabet latin n°1 (ouest de l'Europe) à l'aide d'un seul octet. Les caractères € et œ n'en font pas partie, pourtant il y a æ.

• ISO 8859-15 est une norme mieux adaptée au français. Elle rajoute € et œ et supprime des caractères peu utiles comme ¤.

• UTF-8 représente les caractères à l'aide d'un nombre variable d'octets, de 1 à 4 selon le caractère.

Par exemple : A est codé par 0x41, é par 0xC3,0xA9 et € par 0xE2,0x82,0xAC.

Commentaire XML

Voici un exemple d'un commentaire XML

```
<!-- voici un commentaire -->

<!--
voici un autre commentaire
et ça -- , c'est une erreur
-->
```

La seule contrainte est de ne pas pouvoir employer les caractères -- dans le commentaire, même s'ils ne sont pas suivis de >.

Après le prologue, on peut trouver plusieurs parties optionnelles délimitées par <?...?> ou <!...>. Ce sont des instructions de traitement, des directives pour l'analyseur XML.
Par exemple :
• un *Document Type Definitions* (DTD) qui permet de valider le contenu du document.
• une feuille de style,

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE personne SYSTEM "personne.dtd">
<?xml-stylesheet href="personne.css" type="text/css"?>
<personne>
...
</personne>
```

XML Prologue

The first line of an XML document is called the XML prologue. It specifies the version of the XML standard used (1.0 or 1.1, which are very similar) as well as the encoding:

```
xml
CopyEdit
<?xml version="1.0" encoding="utf-8"?>
<?xml version="1.0" encoding="iso-8859-15"?>
```

There are several encoding standards:

- **ASCII** represents only the first 128 Unicode characters.

- **ISO 8859-1** or Latin-1 represents 191 characters of the Latin-1 alphabet (Western Europe) using a single byte. The characters € and œ are not included, although æ is.
- **ISO 8859-15** is a standard better suited to French. It adds € and œ and removes less useful characters like ¤.
- **UTF-8** represents characters using a variable number of bytes, from 1 to 4 depending on the character.
  For example: A is encoded as 0x41, é as 0xC3 0xA9, and € as 0xE2 0x82 0xAC.
-

XML Comments

Here is an example of an XML comment:

```
xml
CopyEdit
<!-- here is a comment -->
xml
CopyEdit
<!--
here is another comment
and this -- is an error
-->
```

The only restriction is that the characters `--` cannot be used within a comment, even if not followed by `>`.

After the prologue, there can be several optional parts delimited by `<?...?>` or `<!...>`. These are processing instructions or directives for the XML parser.
For example:

- A Document Type Definition (DTD) that allows validation of the document content.
- A stylesheet,

Example:

```
xml
CopyEdit
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE personne SYSTEM "personne.dtd">
```

```
<?xml-stylesheet href="personne.css" type="text/css"?>
<personne>
...
</personne>
```

XML Elements

An XML element is delimited by:

- an opening tag `<name attributes...>`
- a mandatory closing tag `</name>`.

The content of the element lies between the two tags and can include child elements and/or text.

Example:

```
xml
CopyEdit
<parent>
   text1
   <child>text2</child>
   text3
</parent>
```

If the element has no content (empty element), its opening and closing tags can be combined into a single tag:

```
xml
CopyEdit
<name attributes.../>
```

The same type of element can appear multiple times under the same parent, with identical or different contents:

```
xml
CopyEdit
<element1>
  <element2>text1</element2>
  <element2>text2</element2>
  <element2>text1</element2>
</element1>
```

Qualified Names

The colon : character is used to separate a name into two parts: prefix and local name. Together they form a qualified name.

For example, `iut:departement` is a qualified name prefixed by `iut`. This prefix defines a namespace.

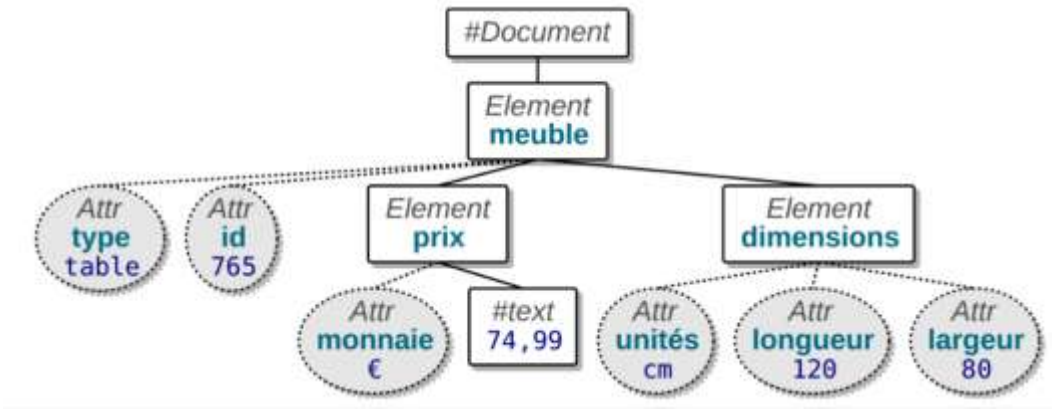
Qualified name = prefix:local name

Namespaces

A namespace defines a family of names to avoid confusion between elements that have the same name but different meanings. This happens when an XML document models information from multiple domains.

Example with ambiguity:

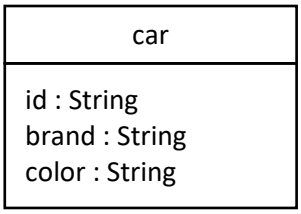
```
xml
CopyEdit
<meuble id="765">
  <table prix="74,99€">Product</table>
  <table border="1">
    <tr><th>length</th><th>width</th></tr>
    <tr><td>120cm</td><td>80cm</td></tr>
  </table>
</meuble>
```



Remarks:
There is no specific order for the attributes of an element. An attribute can only appear once.

1.3 Modeling of Tables (Entities)

Consider the following car table:



Here are possible XML representations for this table:

```
<?xml version="1.0" encoding="utf-8"?>
<cars>
<car id="125" brand="Renault" color="grise"/>
<car id="982" brand="Peugeot" color="noire"/>
</cars>
```

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

 <car>
    <id>125</id>
   <brand>Renault</brand>
   <color>gray</color>
 </car>

 <car>
   <id>982</id>
   <brand>Peugeot</brand>
   <color>black</color>
 </car>

</cars>
```

Attributes or sub-elements?

When deciding whether to represent information as attributes or as sub-elements, you need to consider the following criteria:

- Attributes cannot contain multiple values, whereas you can have multiple sub-elements with the same name but different contents,
- Attributes cannot contain tree structures, unlike elements,

- Attributes are not easily extensible, whereas you can always add new sub-elements in a hierarchy without changing existing software,
- It is not more complicated to access sub-elements than attributes because both are nodes in the underlying tree.

2. Validation d'un document XML

Why validation is necessary:

Data exchange — imagine that a person X wants to exchange data about students: last name, first name, date of birth, average grade, etc. Both parties must agree on a data format.

DTDs (Document Type Definitions), being the oldest form, are supported by most tools. Then come what are called W3C schemas, a more modern but also more complex type of grammar.

2.1 Validation by DTD

A DTD (Document Type Definition) is a relatively old form of grammar because it originates from the SGML (Standard Generalized Markup Language) universe.

For example, an XML document having an external DTD `cours.dtd`, located in the same directory as our XML document (relative access), appears as follows:

```
xml
CopyEdit
<?xml version="1.0"?>
<!DOCTYPE cours SYSTEM "cours.dtd">
<cours>
…
</cours>
```

In this example, the DTD `cours.dtd` is located relative to our XML document.
The keyword SYSTEM is important and indicates that this is a DTD specific to you.
The alternative keyword is PUBLIC.

Definition of an element

```
xml
CopyEdit
<!ELEMENT ElementName DEF_CONTENT>
```

`DEF_CONTENT` can contain:

- `EMPTY`: the element has no content; it is therefore empty. However, it may have attributes.

- `ANY`: the element can contain any element present in the DTD.
- `(#PCDATA)`: the element contains text. The `#` character is there to avoid any ambiguity with a tag and indicates to the parser that it is a keyword. PCDATA means Parsable Character DATA.
- An element placed between parentheses like `(element_name)`. The element name refers to a reference to an element described elsewhere in the DTD.
- A set of elements separated by operators, all enclosed in parentheses.
    - The choice operator, represented by the character `"|"`, indicates that one or the other of two elements (or two sets of elements) must be present.
    - The sequence operator, represented by the character `","`, indicates that both elements (or sets of elements) must be present. Additional parentheses can be used to remove ambiguity.

Some examples:

```
xml
CopyEdit
<!ELEMENT personne (nom_prénom | nom)>
<!ELEMENT nom_prenom (#PCDATA)>
<!ELEMENT nom (#PCDATA)>
```

This is the form of a grammar:
Personne → nom_prenom | nom
nom_prenom → text
nom → text

This allows us two XML documents, either:

```
xml
CopyEdit
<personne>
  <nom_prenom>Karim Kadouri</nom_prenom>
</personne>
```

or:

```
xml
CopyEdit
<personne>
  <nom>Kadouri</nom>
</personne>
```

Another case with the sequence operator:

```
xml
CopyEdit
<!ELEMENT personne(prenom,nom)>
<!ELEMENT prenom (#PCDATA)>
<!ELEMENT nom (#PCDATA)>
```

It is like a grammar:

Personne → prenom nom
Prenom → text
Nom → text

Here, the sequence operator limits the possibilities to only one valid XML document:

```
xml
CopyEdit
<personne>
  <prenom>Karim</prenom>
  <nom>Kadouri</nom>
</personne>
xml
CopyEdit
<personne>
  <nom>Kadouri</prenom>
  <prenom>Karim</nom>
</personne>
```

Contents (element or group of elements) can be quantified by the operators `*`, `+`, and `?`. These operators are linked to the concept of cardinality. When no operator is present, the quantification is 1 (i.e., always present).
Here are the details of these operators:

- `*` : 0 to n times;
- `+` : 1 to n times;
- `?` : 0 or 1 time.

Some examples:

```
xml
CopyEdit
<!ELEMENT plan (introduction?,chapitre+,conclusion?)>
```

The `plan` element contains an optional `introduction` element, followed by at least one `chapitre` element, and ends with an optional `conclusion` element.

```
xml
CopyEdit
<!ELEMENT chapitre (auteur*,paragraphe+)>
```

The `chapitre` element contains 0 to n `auteur` elements followed by at least one `paragraphe` element.

```
xml
CopyEdit
<!ELEMENT livre (auteur?,chapitre)+>
```

The `livre` element contains at least one element, each being a group of elements where the `auteur` element is optional and the `chapitre` element is present exactly once.

Definition of attributes

Attributes are specified in the `ATTLIST` declaration. This is independent of the `ELEMENT` declaration, so the element name to which the attributes apply is repeated. The syntactic form can be considered as:

```
xml
CopyEdit
<!ATTLIST ELEMENT_NAME
  Attr_def* … >
```

`Attr_def` is detailed as follows:

```
pgsql
CopyEdit
name TYPE OBLIGATION DEFAULT_VALUE
```

The TYPE can mainly be:

- `CDATA`: text (Character Data);
- `ID`: a unique identifier (combination of letters and digits);
- `IDREF`: a reference to an ID;
- `IDREFS`: a list of references to IDs (separated by spaces);
- `NMTOKEN`: a token (no spaces allowed);
- `NMTOKENS`: a list of tokens (space-separated);
- An enumeration of values: each value separated by the character `|`.

OBLIGATION does not apply to enumerations, which are followed by a default value. Otherwise, it is expressed as:

- `#REQUIRED`: required attribute.
- `#IMPLIED`: optional attribute.
- `#FIXED`: attribute always present with a fixed value. For example, to impose the presence of a namespace.

DEFAULT_VALUE is present for enumerations or when the value is typed with `#IMPLIED` or `#FIXED`.

Some examples:

```
xml
CopyEdit
<!ATTLIST chapitre
  titre CDATA #REQUIRED
  auteur CDATA #IMPLIED >
```

The element `chapitre` here has a required attribute `titre` and an optional attribute `auteur`.

```
xml
CopyEdit
<!ATTLIST crayon
```

```
  couleur (rouge|vert|bleu) "bleu">
```

The element `crayon` has an attribute `couleur` whose values belong to the set {rouge, vert, bleu}.

Definition of entities

Entities are declared by the `ENTITY` declaration. As introduced in the previous chapter, an entity associates a name with a value. This name is used in the XML document as an alias or shortcut to the value, using the syntax `&name;`. The value of an entity can be internal or external.

The syntax to declare an entity is simply:

```
xml
CopyEdit
<!ENTITY name "VALUE">                <!-- internal case -->
<!ENTITY name SYSTEM "someText.txt"> <!-- external case -->
```

Examples:

Example 1

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE itineraire [
  <!ELEMENT itineraire (boucle?, etape+, variante*)>
  <!ELEMENT boucle EMPTY>
  <!ELEMENT etape (#PCDATA)>
  <!ELEMENT variante ANY>
]>
<itineraire>
  <boucle/>
  <etape>départ</etape>
  <etape>tourner à droite</etape>
  <variante>
    <etape>départ</etape><etape>tourner à gauche</etape>
  </variante>
</itineraire>
```

Example 2

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

```
<!DOCTYPE cours [
  <!ENTITY auteur "Alexandre Brillant">
  <!ELEMENT cours (#PCDATA)>
]>
<cours>
  Cours réalisé par &auteur;
</cours>
```

Example 3

```
xml
CopyEdit
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE itineraire [
  <!ELEMENT itineraire (etape+)>
  <!ATTLIST itineraire nom CDATA #IMPLIED>
  <!ELEMENT etape (#PCDATA)>
  <!ATTLIST etape distance CDATA #REQUIRED>
]>
<itineraire nom="essai">
  <etape distance="0km">départ</etape>
  <etape distance="1km">tourner à droite</etape>
</itineraire>
```

Example 4

```
xml
CopyEdit
<!ELEMENT agence (nom, bureau+, periode, voyage*) >
<!ATTLIST agence niveau CDATA #IMPLIED>
<!ELEMENT nom (#PCDATA)>
<!ELEMENT bureau (#PCDATA)>
<!ATTLIST bureau code_bureau ID #REQUIRED>
<!ELEMENT periode (#PCDATA)>
<!ELEMENT voyage (pays, duree, descriptif, prix)>
<!ATTLIST voyage
  niveau (1 | 2 | 3 | 4 | 5) #IMPLIED
  responsable IDREF #REQUIRED
  code_voyage ID #REQUIRED>
<!ELEMENT descriptif (#PCDATA | remarque | voir)* >
```

```
<!ATTLIST descriptif langue CDATA "fr">
<!ELEMENT remarque (#PCDATA)>
<!ELEMENT voir EMPTY>
<!ATTLIST voir autres IDREFS #REQUIRED>
<!ELEMENT prix (#PCDATA)>
<!ELEMENT pays (#PCDATA)>
<!ELEMENT duree (#PCDATA)>
```

2.2 Validation by XML Schema

What is an XML Schema?

An XML Schema describes the structure of an XML document.

The XML Schema language is also referred to as XML Schema Definition (XSD).

XSD Example

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

The purpose of an XML Schema is to define the legal building blocks of an XML document:

- the elements and attributes that can appear in a document
- the number of (and order of) child elements
- data types for elements and attributes
- default and fixed values for elements and attributes
-

Why Learn XML Schema?

In the XML world, hundreds of standardized XML formats are in daily use.

Many of these XML standards are defined by XML Schemas.

XML Schema is an XML-based (and more powerful) alternative to DTD.

XML Schemas use XML Syntax

Another great strength about XML Schemas is that they are written in XML.

- You don't have to learn a new language
- You can use your XML editor to edit your Schema files
- You can use your XML parser to parse your Schema files
- You can manipulate your Schema with the XML DOM
- You can transform your Schema with XSLT

XML Schemas are extensible, because they are written in XML.

With an extensible Schema definition you can:

- Reuse your Schema in other Schemas
- Create your own data types derived from the standard types
- Reference multiple schemas in the same document

XML Schemas Secure Data

Communication

When sending data from a sender to a receiver, it is essential that both parts have the same "expectations" about the content.

With XML Schemas, the sender can describe the data in a way that the receiver will understand.

A date like: "03-11-2004" will, in some countries, be interpreted as 3.November and in other countries as 11.March.

However, an XML element with a data type like this:

<date type="date">2004-03-11</date>

ensures a mutual understanding of the content, because the XML data type "date" requires the format "YYYY-MM-DD".

Well-Formed is Not Enough

A well-formed XML document is a document that conforms to the XML syntax rules, like:

- it must begin with the XML declaration

- it must have one unique root element
- start-tags must have matching end-tags
- elements are case sensitive
- all elements must be closed
- all elements must be properly nested
- all attribute values must be quoted
- entities must be used for special characters

Even if documents are well-formed they can still contain errors, and those errors can have serious consequences.

Think of the following situation: you order 5 gross of laser printers, instead of 5 laser printers. With XML Schemas, most of these errors can be caught by your validating software.

XSD How To?

XML documents can have a reference to a DTD or to an XML Schema.

A Simple XML Document

Look at this simple XML document called "note.xml":

```
<?xml version="1.0"?>
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

Example of an XML Schema

The following example is an XML Schema file called "note.xsd" that defines the elements of the XML document above ("note.xml"):

```
<?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: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>
```

The note element is a **complex type** because it contains other elements. The other elements (to, from, heading, body) are **simple types** because they do not contain other elements. You will learn more about simple and complex types in the following chapters.

A Reference to an XML Schema

This XML document has a reference to an XML Schema:

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

<note
xmlns="https://www.w3schools.com"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="https://www.w3schools.com/xml note.xsd">
 <to>Tove</to>
 <from>Jani</from>
 <heading>Reminder</heading>
 <body>Don't forget me this weekend!</body>
</note>
```

XSD - The <schema> Element

The <schema> element is the root element of every XML Schema.

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

<xs:schema>
  ...
  ...
</xs:schema>
```

The <schema> element may contain some attributes. A schema declaration often looks something like this:

```
<?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>
```

The fragment:

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

indicates that the elements and data types used in the schema come from the "http://www.w3.org/2001/XMLSchema" namespace. It also specifies that the elements and data types that come from the "http://www.w3.org/2001/XMLSchema" namespace should be prefixed with **xs:**

This fragment:

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

indicates that the elements defined by this schema (note, to, from, heading, body.) come from the "https://www.w3schools.com" namespace.

This fragment:

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

indicates that the default namespace is "https://www.w3schools.com".

This fragment:

```
elementFormDefault="qualified"
```

indicates that any elements used by the XML instance document which were declared in this schema must be namespace qualified.

Referencing a Schema in an XML Document

This XML document has a reference to an XML Schema:

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

<note xmlns="https://www.w3schools.com"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="https://www.w3schools.com note.xsd">

<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>
```

What is a Simple Element?

A simple element is an XML element that can contain only text. It cannot contain any other elements or attributes.

However, the "only text" restriction is quite misleading. The text can be of many different types. It can be one of the types included in the XML Schema definition (boolean, string, date, etc.), or it can be a custom type that you can define yourself.

You can also add restrictions (facets) to a data type in order to limit its content, or you can require the data to match a specific pattern.

Defining a Simple Element

The syntax for defining a simple element is:

```
<xs:element name="xxx" type="yyy"/>
```

where xxx is the name of the element and yyy is the data type of the element.

XML Schema has a lot of built-in data types. The most common types are:

- xs:string
- xs:decimal
- xs:integer
- xs:boolean
- xs:date
- xs:time

- Here are some XML elements:

```
<lastname>Refsnes</lastname>
<age>36</age>
<dateborn>1970-03-27</dateborn>
```

- And here are the corresponding simple element definitions:

```
<xs:element name="lastname" type="xs:string"/>
<xs:element name="age" type="xs:integer"/>
<xs:element name="dateborn" type="xs:date"/>
```

Default and Fixed Values for Simple Elements

Simple elements may have a default value OR a fixed value specified.

A default value is automatically assigned to the element when no other value is specified.

In the following example the default value is "red":

```
<xs:element name="color" type="xs:string" default="red"/>
```

A fixed value is also automatically assigned to the element, and you cannot specify another value.

In the following example the fixed value is "red":

```
<xs:element name="color" type="xs:string" fixed="red"/>
```

Attributes

Simple elements cannot have attributes. If an element has attributes, it is considered to be of a complex type. But the attribute itself is always declared as a simple type.

How to Define an Attribute?

The syntax for defining an attribute is:

```
<xs:attribute name="xxx" type="yyy"/>
```

where xxx is the name of the attribute and yyy specifies the data type of the attribute.

XML Schema has a lot of built-in data types. The most common types are:

- xs:string
- xs:decimal
- xs:integer
- xs:boolean
- xs:date
- xs:time

Example

Here is an XML element with an attribute:

```
<lastname lang="EN">Smith</lastname>
```

And here is the corresponding attribute definition:

```
<xs:attribute name="lang" type="xs:string"/>
```

Default and Fixed Values for Attributes

Attributes may have a default value OR a fixed value specified.

A default value is automatically assigned to the attribute when no other value is specified.

In the following example the default value is "EN":

```
<xs:attribute name="lang" type="xs:string" default="EN"/>
```

A fixed value is also automatically assigned to the attribute, and you cannot specify another value.

In the following example the fixed value is "EN":

```
<xs:attribute name="lang" type="xs:string" fixed="EN"/>
```

Optional and Required Attributes

Attributes are optional by default. To specify that the attribute is required, use the "use" attribute:

```
<xs:attribute name="lang" type="xs:string" use="required"/>
```

Restrictions on Content

When an XML element or attribute has a data type defined, it puts restrictions on the element's or attribute's content.

If an XML element is of type "xs:date" and contains a string like "Hello World", the element will not validate.

With XML Schemas, you can also add your own restrictions to your XML elements and attributes. These restrictions are called facets. You can read more about facets in the next chapter.

Restrictions on Values

The following example defines an element called "age" with a restriction. The value of age cannot be lower than 0 or greater than 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>--
```

Restrictions on a Set of Values

To limit the content of an XML element to a set of acceptable values, we would use the enumeration constraint.

The example below defines an element called "car" with a restriction. The only acceptable values are: 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>
```

The example above could also have been written like this:

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

Definition of the 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>
```

Note: In this case the type "carType" can be used by other elements because it is not a part of the "car" element.

Restrictions on a Series of Values

To limit the content of an XML element to define a series of numbers or letters that can be used, we would use the **pattern** constraint.

The example below defines an element called "letter" with a restriction. The only acceptable value is ONE of the LOWERCASE letters from a to z:

```
<xs:element name="letter">
 <xs:simpleType>
```

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

The next example defines an element called "initials" with a restriction. The only acceptable value is THREE of the UPPERCASE letters from a to 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>
```

The next example also defines an element called "initials" with a restriction. The only acceptable value is THREE of the LOWERCASE OR UPPERCASE letters from a to 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>
```

The next example defines an element called "choice" with a restriction. The only acceptable value is ONE of the following letters: x, y, OR z:

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

The next example defines an element called "prodid" with a restriction. The only acceptable value is FIVE digits in a sequence, and each digit must be in a range from 0 to 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>
```

Other Restrictions on a Series of Values

The example below defines an element called "letter" with a restriction. The acceptable value is zero or more occurrences of lowercase letters from a to z:

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

The next example also defines an element called "letter" with a restriction. The acceptable value is one or more pairs of letters, each pair consisting of a lower case letter followed by an upper case letter. For example, "sToP" will be validated by this pattern, but not "Stop" or "STOP" or "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>
```

The next example defines an element called "gender" with a restriction. The only acceptable value is male OR female:

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

The next example defines an element called "password" with a restriction. There must be exactly eight characters in a row and those characters must be lowercase or uppercase letters from a to z, or a number from 0 to 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>
```

Restrictions on Whitespace Characters

To specify how whitespace characters should be handled, we would use the whiteSpace constraint.

This example defines an element called "address" with a restriction. The whiteSpace constraint is set to "preserve", which means that the XML processor WILL NOT remove any white space characters:

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

This example also defines an element called "address" with a restriction. The whiteSpace constraint is set to "replace", which means that the XML processor WILL REPLACE all white space characters (line feeds, tabs, spaces, and carriage returns) with spaces:

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

This example also defines an element called "address" with a restriction. The whiteSpace constraint is set to "collapse", which means that the XML processor WILL REMOVE all white space characters (line feeds, tabs, spaces, carriage returns are replaced with spaces, leading and trailing spaces are removed, and multiple spaces are reduced to a single space):

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

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

Restrictions on Length

To limit the length of a value in an element, we would use the length, maxLength, and minLength constraints.

This example defines an element called "password" with a restriction. The value must be exactly eight characters:

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

This example defines another element called "password" with a restriction. The value must be minimum five characters and maximum eight characters:

```
<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>
```

Restrictions for Datatypes

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

XSD Complex Elements

A complex element is an XML element that contains other elements and/or attributes.

Examples of Complex Elements

A complex XML element, "product", which is empty:

```
<product pid="1345"/>
```

A complex XML element, "employee", which contains only other elements:

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

A complex XML element, "food", which contains only text:

```
<food type="dessert">Ice cream</food>
```

A complex XML element, "description", which contains both elements and text:

```
<description>
   It happened on     <date lang="norwegian">  03.03.99   </date> ....
</description>
```

How to Define a Complex Element

Look at this complex XML element, "employee", which contains only other elements:

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

We can define a complex element in an XML Schema two different ways:

1. The "employee" element can be declared directly by naming the element, like this:

```
<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>
```

If you use the method described above, only the "employee" element can use the specified complex type. Note that the child elements, "firstname" and "lastname", are surrounded by the <sequence> indicator. This means that the child elements must appear in the same order as they are declared. You will learn more about indicators in the XSD Indicators chapter.

2. The "employee" element can have a type attribute that refers to the name of the complex type to use:

```
<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>
```

You can also base a complex type on an existing complex type and add some elements, like this:

```
<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>
```

Complex Empty Elements

An empty XML element:

```
<product prodid="1345" />
```

The "product" element above has no content at all. To define a type with no content, we must define a type that allows elements in its content, but we do not actually declare any elements, like this:

```
<xs:element name="product">
 <xs:complexType>
  <xs:complexContent>
   <xs:restriction base="xs:integer">
    <xs:attribute name="prodid" type="xs:positiveInteger"/>
   </xs:restriction>
  </xs:complexContent>
```

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

In the example above, we define a complex type with a complex content. The complexContent element signals that we intend to restrict or extend the content model of a complex type, and the restriction of integer declares one attribute but does not introduce any element content.

However, it is possible to declare the "product" element more compactly, like this:

```
<xs:element name="product">
 <xs:complexType>
  <xs:attribute name="prodid" type="xs:positiveInteger"/>
 </xs:complexType>
</xs:element>
```

Or you can give the complexType element a name, and let the "product" element have a type attribute that refers to the name of the complexType (if you use this method, several elements can refer to the same complex type):

```
<xs:element name="product" type="prodtype"/>

<xs:complexType name="prodtype">
 <xs:attribute name="prodid" type="xs:positiveInteger"/>
</xs:complexType>
```

Complex Types Containing Elements Only

An XML element, "person", that contains only other elements:

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

You can define the "person" element in a schema, like this:

```
<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>
```

Notice the <xs:sequence> tag. It means that the elements defined ("firstname" and "lastname") must appear in that order inside a "person" element.

Or you can give the complexType element a name, and let the "person" element have a type attribute that refers to the name of the complexType (if you use this method, several elements can refer to the same complex type):

```
<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>
```

Complex Text-Only Elements

This type contains only simple content (text and attributes), therefore we add a simpleContent element around the content. When using simple content, you must define an extension OR a restriction within the simpleContent element, like this:

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

OR

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

Tip: Use the extension/restriction element to expand or to limit the base simple type for the element.

Here is an example of an XML element, "shoesize", that contains text-only:

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

The following example declares a complexType, "shoesize". The content is defined as an integer value, and the "shoesize" element also contains an attribute named "country":

```
<xs:element name="shoesize">
 <xs:complexType>
  <xs:simpleContent>
   <xs:extension base="xs:integer">
    <xs:attribute name="country" type="xs:string" />
   </xs:extension>
  </xs:simpleContent>
 </xs:complexType>
</xs:element>
```

We could also give the complexType element a name, and let the "shoesize" element have a type attribute that refers to the name of the complexType (if you use this method, several elements can refer to the same complex type):

```
<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>
```

Complex Types with Mixed Content

An XML element, "letter", that contains both text and other elements:

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

The following schema declares the "letter" element:

```
<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>
```

Note: To enable character data to appear between the child-elements of "letter", the mixed attribute must be set to "true". The <xs:sequence> tag means that the elements defined (name, orderid and shipdate) must appear in that order inside a "letter" element.

We could also give the complexType element a name, and let the "letter" element have a type attribute that refers to the name of the complexType (if you use this method, several elements can refer to the same complex type):

```
<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>
```

Indicators

There are seven indicators:

Order indicators:

- All
- Choice
- Sequence

Occurrence indicators:

- maxOccurs
- minOccurs

Group indicators:

- Group name
- attributeGroup name

Order Indicators

Order indicators are used to define the order of the elements.

All Indicator

The <all> indicator specifies that the child elements can appear in any order, and that each child element must occur only once:

```
<xs:element name="person">
 <xs:complexType>
  <xs:all>
   <xs:element name="firstname" type="xs:string"/>
   <xs:element name="lastname" type="xs:string"/>
  </xs:all>
 </xs:complexType>
</xs:element>
```

Note: When using the <all> indicator you can set the <minOccurs> indicator to 0 or 1 and the <maxOccurs> indicator can only be set to 1 (the <minOccurs> and <maxOccurs> are described later).

Choice Indicator

The <choice> indicator specifies that either one child element or another can occur:

```
<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>
```

Sequence Indicator

The <sequence> indicator specifies that the child elements must appear in a specific order:

```
<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>
```

Occurrence Indicators

Occurrence indicators are used to define how often an element can occur.

Note: For all "Order" and "Group" indicators (any, all, choice, sequence, group name, and group reference) the default value for maxOccurs and minOccurs is 1.

maxOccurs Indicator

The <maxOccurs> indicator specifies the maximum number of times an element can occur:

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

The example above indicates that the "child_name" element can occur a minimum of one time (the default value for minOccurs is 1) and a maximum of ten times in the "person" element.

minOccurs Indicator

The <minOccurs> indicator specifies the minimum number of times an element can occur:

```
<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>
```

The example above indicates that the "child_name" element can occur a minimum of zero times and a maximum of ten times in the "person" element.

Tip: To allow an element to appear an unlimited number of times, use the maxOccurs="unbounded" statement:

A working example:

An XML file called "Myfamily.xml":

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

<persons xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="family.xsd">

<person>
 <full_name>Hege Refsnes</full_name>
 <child_name>Cecilie</child_name>
</person>

<person>
 <full_name>Tove Refsnes</full_name>
 <child_name>Hege</child_name>
 <child_name>Stale</child_name>
 <child_name>Jim</child_name>
 <child_name>Borge</child_name>
</person>

<person>
 <full_name>Stale Refsnes</full_name>
</person>

</persons>
```

The XML file above contains a root element named "persons". Inside this root element we have defined three "person" elements. Each "person" element must contain a "full_name" element and it can contain up to five "child_name" elements.

Here is the schema file "family.xsd":

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

```
elementFormDefault="qualified">

<xs:element name="persons">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="person" maxOccurs="unbounded">
    <xs:complexType>
     <xs:sequence>
      <xs:element name="full_name" type="xs:string"/>
      <xs:element name="child_name" type="xs:string"
      minOccurs="0" maxOccurs="5"/>
     </xs:sequence>
    </xs:complexType>
   </xs:element>
  </xs:sequence>
 </xs:complexType>
</xs:element>

</xs:schema>
```

Group Indicators

Group indicators are used to define related sets of elements.

Element Groups

Element groups are defined with the group declaration, like this:

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

You must define an all, choice, or sequence element inside the group declaration. The following example defines a group named "persongroup", that defines a group of elements that must occur in an exact sequence:

```
<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>
```

After you have defined a group, you can reference it in another definition, like this:

```
<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>
```

Attribute Groups

Attribute groups are defined with the attributeGroup declaration, like this:

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

The following example defines an attribute group named "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>
```

After you have defined an attribute group, you can reference it in another definition, like this:

```
<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>
```

3. XPath

XPath can be used to navigate through elements and attributes in an XML document.

XPath Path Expressions

XPath uses path expressions to select nodes or node-sets in an XML document.

These path expressions look very much like the path expressions you use with traditional computer file systems:

Let's examine Xpath expressions through examples, using the following sample of an XML document :

```
<?xml version="1.0" encoding="UTF-8"?>
<AAA>
    <BBB>
         <CCC lang="fr">
         XXXXXXX
         </CCC>
    </BBB>
    <BBB>
         <CCC lang="ar">
         XXXXXXX
         </CCC>
    </BBB>

    <DDD>
       <BBB lang="en">fffff</BBB>
        <BBB lang="Ge">kkkkkk</BBB>
    </DDD>
</AAA>
```

Let’s exmine the following X-path queries (the second column shows description of the each query):

Expression	Signification
/AAA	Selects the root element AAA
AAA/BBB	Returns BBB elements inside AAA
AAA/BBB/CCC	Returns CCC elements inside BBB children of AAA
AAA/BBB[1]	Returns the first BBB element inside AAA
AAA/BBB[last()]	Returns the last BBB element inside AAA
//BBB	Selects all BBB elements in the document
//@lang	Selects all attributes lang
//BBB[@lang]	Selects all BBB elements having attribute lang
//CCC[@*]	All CCC elements having attributes
//*[not(@*)]	All elements not having attributes
//BBB[not(@*)]	All elements of BBB not having attributes
//BBB[@lang=”en”]	All elements BBB having attributes lang=“en”
//*[count(BBB)=2]	All elements having 2 sub-elements BBB
//*[count(*)=2]	All elements have 2 children
//*[name()='BBB']	All elements named BBB
//*[starts-with(name(),'BB')]	All elements their names start with BB
//*[contains(name(),'C')]	All elements of their names contain the letter C
//*[string-length(name())=3] We may use > < <= >= != ….	All elements with names length =3
/AAA/BBB/CCC/parent::*	All parents of CCC (AAA/BBB/CCC)
/AAA/BBB | //CC	Collection of 2 results together
/AAA/BBB/CCC/ancestor::*	All ancestors of CCC
AAA/BBB[last()-1]	Selects the last but one book element
AAA/BBB[2]	Selects the second BBB element that is children of AAA
AAA/BBB[position()<3]	Selects the first two BBB elements that are children of AAA
//BBB[CCC>1000]	Selects BBB elements where the value of the children CCC > 1000.
//BBB[@ID='2']/CCC	Selects CCC children of BBB with ID=2

4. Projects/Exercises

Exercise 01

We want to register the students of the Exact sciences Faculty (an arbitrary number of students), using an XML document. Each student is identified by its registration number (**RegistrationNumber**) and should be specified with the following data:

- **FirstName**: text value, max length 20 characters, can contain blanks. (required).
- **FamilyName**: text value, max length 20 characters, can contain blanks. (required).
- **Age**: an integer value between 16 and 150 (required).
- **Address**: described below (not mandatory)
- **Level**: one of the following values: 1L, 2L, 3L, 1M, 2M (required)
- **speciality**: one of the following values: ComputerScience, Physics, Chemistry, Mathematics (required).
- **Average**: a Real value between 0.00 and 20.00, its default value is 0.00 (required).

An **address** is composed of four sub elements:

- **Number**: an integer value from 0 to 999. (required)
- **City**: text value, max length 30 characters, can contain blanks. (required).
- **PostalCode**: five digits from the list {0,1,2,3,4,5,6,7,8,9}, starting with Wilaya code (2 digits), followed by sequence number.(required)

The **registratioNnumber** is composed of 8 digits from 0—9. It must be presented as attributes in the XML document.

Required Tasks:

1- **Show a sample of XML document reflecting this description.**
2- **Propose a DTD for this project.**
3- **Propose a precise XML schema of El Oued University student documents.**

Exercise 02

Convert the following DTD into an XML schema.

```
<!ELEMENT Car (Brand, power, Color)>
 <!ELEMENT Brand (Pegeot | Renault | BMW | Hyundai)>
 <!ELEMENT power (07 | 08 | 09 | 10 | 11)>
 <!ELEMENT Color (Red | Green | Blue) "Blue">
 <!ATTLIST Car Matricule CDATA #REQUIRED; >
```

Exercise 03:

Let's consider the following XML document

```
<cinema>
    <film id="lb78" type=" Documentary">
        <title>Les Bronzés</title>
        <year>1978</year>
        <director>Patrice Leconte</director>
        <role>
            <name>Popeye</name>
            <actor>Thierry Lhermite</actor>
        </role>
        <role>
            <name>Jean-Claude Dusse</name>
            <actor>Michel Blanc</actor>
        </role>
    </film>
    <film id="gf94" type=" Comedy">
        <title>Grosse fatigue</title>
        <year>1994</year>
        <director>Michel Blanc</director>
        <role>
            <name>Patrick Olivier</name>
            <actor>Michel Aoun</actor>
        </role>
        <role>
            <name>Carole Bouquet</name>
            <actor>Carole Bouquet</actor>
        </role>
        <price>Cannes meilleur scénario</price>
    </film>
    <producer>
        <name>Daniel Toscan du Plantier</name>
        <film ref="gf94" />
    </producer>
    <producer>
```

```
        <name>Yves Rousset-Rouard</name>
        <film ref="lb78" />
    </producer>
</cinema>
```

Description:

This document presents information on films. Each **film** has **title**, **year** of production, **director** and unlimited number of **roles**.

Each **role** identifies one **actor** and his role **name**.

The attributes '**Id'** and '**type'** are mandatory.

The production year should be between 1950 and 2023. The element **price** is optional. The element **producer** shows the name of the producer of each film.

Questions:

1- Give the XML schema and the DTD of the above XML document.
2- Write the following XPath queries on the above XML document:

A. Directors of all films. المخرجون لجميع الأفلام.

B. Titles of films whose ids begin with "lb". عناوين الأفلام التى تبدأ معرفاتها (ids) بحرف "lb"

C. Titles of Comedy films produced before 2000. عناوين الأفلام الكوميدية المنتجة قبل عام 2000.

D. Producer name of the film with id='lb78'. اسم منتج الفيلم الذى معرفه."lb78"

E. Titles of films where "Michel Blanc" doesn't act. عناوين الأفلام التى لم يمثل فيها "ميشيل بلان".

Exercise 04:

Published scientific articles are characterized by the following description:

- **authors**: Lists the author's names. Each name is a string between 3 and 30 characters. The number of authors is between 1 and 5.
- **title:** The title of the article. It is a string between 20 and 100 characters.
- **journal**: The name of the journal where the article was published. It is a string
- between 30 and 100 characters.
- **year**: The year of publication, between 1950 and 2050.
- **volume**: The volume number of the journal. It is a number between 1 and 999
- **number**: The issue number of the journal, a number between 1 and 12.
- **pages**: The page range where the article appears in the journal. It is written as fstPage-lstPage, where fstPage and lstPage are page numbers between 1 and 9999
- **month**: The month of publication. A value from; January, February, March, April, May, June, July, August, September, October, November, December.
- **keywords**: list of exactly 5 keywords associated with the article for indexing and search purposes. Each keyword is a text of a length between 3 and 30 characters.

A sample of an XML document describing articles is presented as follows:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<articles>
     <article>
          <authors>
               <author>John Smith</author>
               <author>Jane Doe</author>
          </authors>
          <title>Innovative In AI</title>
          <journal>Computers</journal>
          <year>2024</year>
          <volume>58</volume>
          <number>4</number>
          <Pages>
               <FirstPage>123</FirstPage>
               <LastPage>145</LastPage>
          </Pages>
          <month>May</month>
          <keywords>
            <keyword>AI</keyword>
            <keyword>ML</keyword>
            <keyword>Security</keyword>
            <keyword>IoT</keyword>
            <keyword>Algorithms</keyword>
          </keywords>
     </article>
</articles>
```

Questions:

1- **Propose** an XML schema for the published article description.

2- **Give** the **XPath** expressions researching:

- Titles of articles published in 2023
- Titles of articles where “John Smith” is an author.
- Titles of articles having more than 20 pages.
- Titles of articles working on IoT.

Exercise 05 (TP XPATH)

Consider the following XML file “cinema.xml”:

```xml
<cinema>
    <film id="lb78" type="comédie">
        <titre>Les Bronzés</titre>
        <annee>1978</annee>
        <realisateur>Patrice Leconte</realisateur>
        <role>
            <nom>Popeye</nom>
            <acteur>Thierry Lhermite</acteur>
        </role>
        <role>
            <nom>Jean-Claude Dusse</nom>
            <acteur>Michel Blanc</acteur>
        </role>
    </film>
    <film id="gf94" type="comédie">
        <titre>Grosse fatigue</titre>
```

```
            <annee>1994</annee>
            <realisateur>Michel Blanc</realisateur>
            <role>
                <nom>Patrick Olivier</nom>
                <acteur>Michel Blanc</acteur>
            </role>
            <role>
                <nom>Carole Bouquet</nom>
                <acteur>Carole Bouquet</acteur>
            </role>
            <prix>Cannes, meilleur scénario</prix>
        </film>
        <producteur>
            <nom>Daniel Toscan du Plantier</nom>
            <film ref="gf94" />
        </producteur>
        <producteur>
            <nom>Yves Rousset-Rouard</nom>
            <film ref="lb78" />
        </producteur>
    </cinema>
```

Express the following Xpath queries:

1. Titles of all films

2. Titles of films where Michel Blanc acts.

3. Types of films

4. Roles played by Michel Blanc.

5. The principle actor (the first) of each film.

6. The name of the producer that appears right after Yves Rousset-Rouard.

7. Actors who appear before Michel Blanc in the realization of Patrice Leconte films.

8. The directors (developers) of the films where played Michel Blanc and Thierry Lhermite.

9. Les titres des films avec plus de 5 acteurs dans la distribution

10. Directors who act in at least one of their films.
    //film[realisateur=role/acteur]/realisateur

11. The names of comedy producers.

12. Title of films which did not obtain a price

13. Actors in roles with name containing 'Morin'.

Bibliographie

[1]Hainaut, Jean-Luc. "Bases de données : Concepts, utilisation et développement-Collection

Sciences Sup, DUNOD, Paris, 2018.

[2] The XML Guild, 2007. *Advanced XML applications from the experts at The XML Guild*.

Thomson Course Technology.

[3] Abiteboul S, Buneman P, Suciu D. Data on the web: from relations to semistructured data and

XML. Morgan Kaufmann; 2000.

[4] Ben Lagha, Sarra, Walid Sadfi, and Mohammed Ben Ahmed. "Une comparaison SGML-

XML." Cahiers GUTenberg 33-34 (1999): 127-154.

[5] Christian Grun, Sebastian Gath, Alexander Holupirek, and Marc H. Scholl. INEX ¨

Efficiency Track meets XQuery Full Text in BaseX. In Pre-Proceedings of the 8th INEX

Workshop, pages 192–197, 2009

[6] Grün, C., 2010. Storing and querying large XML instances.