XML introduction

Document number: 1006_09a.fm

7 September 2020

ABIS Training & Consulting
Diestsevest 32 / 4b
B-3000 Leuven
Belgium

Document O:\Courses\XML\1006\1006_09a.fm (7 September 2020)

Address comments concerning the contents of this publication to:

ABIS Training & Consulting
Diestsevest 32 / 4b, B-3000 Leuven, Belgium
Tel.: (+32)-16-245610
website: http://www.abis.be/
e-mail: training@abis.be

TABLE OF CONTENTS

PREFACE

XML is a worldwide standard for exchanging, storing and handling structured text and data.

During this course you will learn how to use XML, how to write XML documents, and how to validate them using an XML Schema.

Useful books are :

XML in a Nutshell 3rd ed. (E. Harold & W. Means), O'Reilly, 2004

Professional XML, (Bill Evjen, Kent Sharkey,...), Wrox Press Ltd, 2007

The XML Handbook 5th ed. (Charles F.Goldfarb & Paul Prescod), Prentice Hall, 2003

Interesting Web sites

- www.w3.org
- www.xml.com
- www.xml.org
- www.w3schools.com
- www.altova.com

CHAPTER 1. INTRODUCING XML

This chapter explains why XML was invented (within the Web context), and teaches the basic principles of structuring data with XML, such as elements, tags, encoding and well-formedness.

A little history

- 1970s: birth of the Internet
- 1990: Tim Berners-Lee (CERN) develops the World Wide Web (HTML + HTTP)
- 1994: World Wide Web Consortium (W3C) founded
- 1998: XML Version 1.0

>> www.w3.org

1.1 XML originated in the World Wide Web

1.1.1 The W3C

Since its conception in the 1970s, the Internet is constantly growing as a universal network for interconnecting people, machines and applications. At the beginning of its success story were clever communication protocols, such as **TCP/IP**, and useful applications such as Telnet, FTP, newsgroups and e-mail.

But the big breakthrough was caused in the beginning 90s by Tim Berners-Lee & Co, at the CERN (Genève), with the development of the World Wide Web application: with one single browser application, a user could now view and use hyperlinked pages, containing very diverse content, distributed over many servers on the Internet. The two underlying standards are:

- The Hypertext Markup Language (**HTML**): basically a way to indicate for a given page how its text should be displayed, combined with images etc, hyperlinked to other pages, and how to make it interactive.

- The Hypertext Transfer Protocol (**HTTP**): a communication protocol, ruling how a browser gets a web page (and its components) from a server and how user input is sent to a server.

In October 1994, Tim Berners-Lee founded the **World Wide Web Consortium** (W3C) at the Massachusetts Institute of Technology in collaboration with CERN, with support from DARPA and the European Commission. The W3C describes its mission and activities as follows (see **www.w3.org**):

> *The W3C develops interoperable technologies (specifications, guidelines, software, and tools) to lead the Web to its full potential. W3C is a forum for information, commerce, communication, and collective understanding.*

The W3C (..)

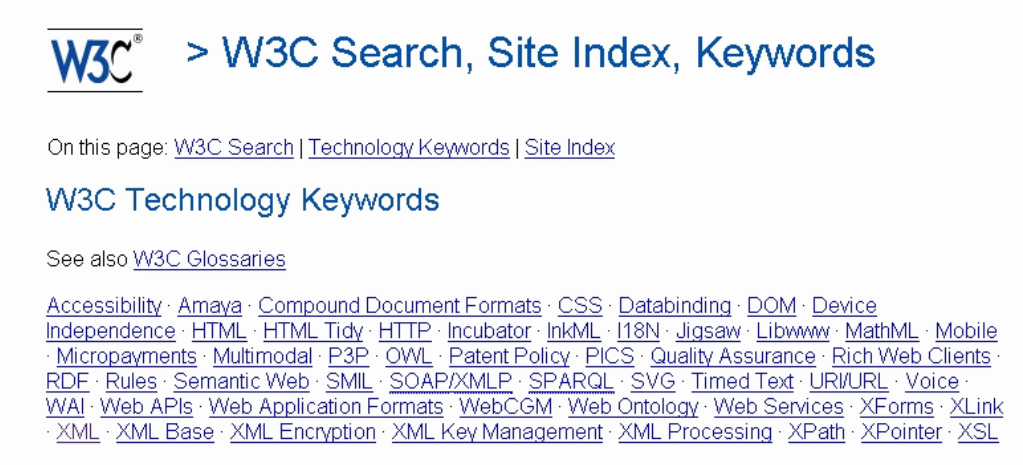
www.w3.org

W3C goals:

- **Universal Access**
- **Semantic Web**
- **Web of Trust**

Specifying standards for several technologies

Figure 1: Some W3C technologies



Many of these are XML-based ...

W3C's long term goals for the Web are:

- **Universal Access**: the Web as the universe of network-accessible information (available through your computer, phone, television, or networked refrigerator...). Today this universe benefits society by enabling new forms of human communication and opportunities to share knowledge. One of W3C's primary goals is to make these benefits available to all people, whatever their hardware, software, network infrastructure, native language, culture, geographical location, or physical or mental ability. W3C's Internationalization Activity, Device Independence Activity, Voice Browser Activity, and Web Accessibility Initiative all illustrate our commitment to universal access.

- **Semantic Web**: People currently share their knowledge on the Web in a language intended for other *people*. On the Semantic Web ("semantic" means "having to do with meaning"), we will be able to express ourselves in terms that our *computers* can interpret and exchange. By doing so, we will enable them to help us find quickly what we're looking for: medical information, a movie review, a book purchase order, etc. The W3C languages RDF, XML, XML Schema, and XML signatures are the building blocks of the Semantic Web.

- **Web of Trust**: The Web is a collaborative medium, not read-only like a magazine. In fact, the first Web browser was also an editor, though most people today think of browsing as primarily viewing, not interacting. To promote a more collaborative environment, we must build a "Web of Trust" that offers confidentiality, instills confidence, and makes it possible for people to take responsibility for (or be accountable for) what they publish on the Web. These goals drive much of W3C's work around XML signatures, annotation mechanisms, group authoring, versioning, etc.

It is obvious that not all of the above goals can be achieved with the basic HTML standard, since that was typically developed for presentation and human interaction. Instead we now require a more universal standard for expressing structure and content, so that machines can understand it and interact without human intervention and human interpretation: this has lead to the specification of **XML (Extensible Markup Language)**, with a first official release in 1998.

In fact the W3C develops standards for a lot of related technologies, e.g. images on the Web can be represented with Scalable Vector Graphics (SVG), which uses XML as an underlying standard.

More in particular, several standards explicitly extend XML's usability, e.g. XPath, XML Encryption, XQuery, XLink, XSLT.

Design goals for XML

To store the data platform independently

- **XML shall be straightforwardly usable over the Internet**
- **XML shall support a wide variety of applications**
- **It shall be easy to write programs which process XML documents**

To store the data in a clear and logical design

- **XML documents should be human-legible and reasonably clear**
- **The XML design should be prepared quickly**
- **The design of XML shall be formal and concise**
- **XML documents shall be easy to create**
- **Terseness in XML markup is of minimal importance**
- **The number of optional features in XML is to be kept to the absolute minimum, ideally zero**

To store the data in a independent format

- **XML shall be compatible with SGML**

1.1.2 Design goals of XML

The five I's of XML are Information, Interoperability, Integration, Independence, International.

In the first place, XML can contain diverse forms of (mostly textual) **Information**, ranging from very structured table-like data to semi-structured document-like content. The purpose of XML is to open up that information in such a way that it can be shared between different platforms and applications (**Interoperability**). This last feature is also the reason why XML plays an important roll in the **Integration**, which is the key word in today's technology environment. The fact that XML was created by the World Wide Web Consortium (W3C), and not by a private enterprise, should insure its **Independency**. Besides that, it is also independent from any platform, application, programming language, data model or delivery device. The last feature of XML is the fact that it is an **International** language, so it does not only work with ASCII but also with the Chinese characters or even Klingon's, because it uses Unicode as the digital representation of the characters.

The Goals of XML

The original design goals were formulated by the W3C as follows:

- To store the data platform independently
  - XML shall be straightforwardly usable over the Internet.
  - XML shall support a wide variety of applications.
  - It shall be easy to write programs which process XML documents.
- To store the data in a clear and logical design
  - XML documents should be human-legible and reasonably clear.
  - The XML design should be prepared quickly.
  - The design of XML shall be formal and concise.
  - XML documents shall be easy to create.
  - Terseness in XML markup is of minimal importance.
  - The number of optional features in XML is to be kept to the absolute minimum, ideally zero.
- To store the data in a independent format
  - XML shall be compatible with SGML.

An example: storing/exchanging data

(the ‘old’ way, e.g.:)

Records with field delimiter (e.g. ; , blank)

```
Jean Dupont  1200  3000 Brussel Belgium
Ann “Van Halen” 2400 2000 Antwerpen Belgium
Josette “‘s Meyers” 1800 1000-AX Amsterdam “The Netherlands”
```

Tabular text (fixed column width)

```
1.........11........21........31........41........51........61....
Jean     Dupont      1200   3000     Brussel      Belgium
Ann      Van Halen   2400   2000     Antwerpen    Belgium
Josette ‘s Meyers    1800   1000-AX  Amsterdam    The Netherlands
```

Some issues:

- separator sign as data (; or “)?
- record structure should be fixed
- field substructure?
- where do you put the field names?
- data records << >> lines in sequential text file (CRLF...)
- character set is presumed

1.2 XML contains structured data

1.2.1 The ‘old’ way

Text files have been used for long to **store** and especially to **exchange** structured information (e.g. between databases and spreadsheets). The information can be stored in records where every field is separated from the next by some separator or delimiter character (e.g. a blank, a comma, a tab, ...). This technique causes trouble when the separator character, e.g. the blank, is are part of the content of a column. Quotes can be used to indicate that the blank is included.

An other technique is the alignment of the fields in fixed width columns. This avoids problems with separator characters, but the structure of the data in the file is predefined and fixed, which can cause some trouble when you want to insert a new field or when a certain field is too short for a value.

Some other issues with these text files (and the many other variants):

- Where do you put the names of the fields (in general: the metadata)? This could be done e.g. with a header record (implying that all records have a similar structure...).
- In general the records in these files have a flat structure (fields do not have substructure).
- Most of the time a specific character set is assumed: if the processing application does not use the same set for decoding, character confusion may occur.

The XML way

```
<?xml version="1.0" encoding="UTF-8"?>
<PersonList>
    <Person>
        <FirstName>Jean</FirstName>
        <LastName>Dupont</LastName>
        <Salary>1200</Salary>
        <ZIPCode>3000</ZIPCode>
        <City>Leuven</City>
        <Country>Belgium</Country>
    </Person>
    <Person>
        <FirstName>Ann</FirstName>
        <LastName>Van Halen</LastName>
        <Salary>2400</Salary>
        <ZIPCode>2000</ZIPCode>
        <City>Antwerpen</City>
        <Country>Belgium</Country>
    </Person>
    <Person>
        <Name>
            <FirstName>Josette</FirstName>
            <LastName>'s Meyers</LastName>
        </Name>
        <Salary>1800</Salary>
        <ZIPCode>1000 AX</ZIPCode>
        <City>Amsterdam</City>
        <Country>The Netherlands</Country>
    </Person>
</PersonList>
```

Note:

- elements, tags, tree, root element
- indentation & whitespace

1.2.2 The XML tree

The slide shows how the text information from the previous data example can easily be represented in an XML document.

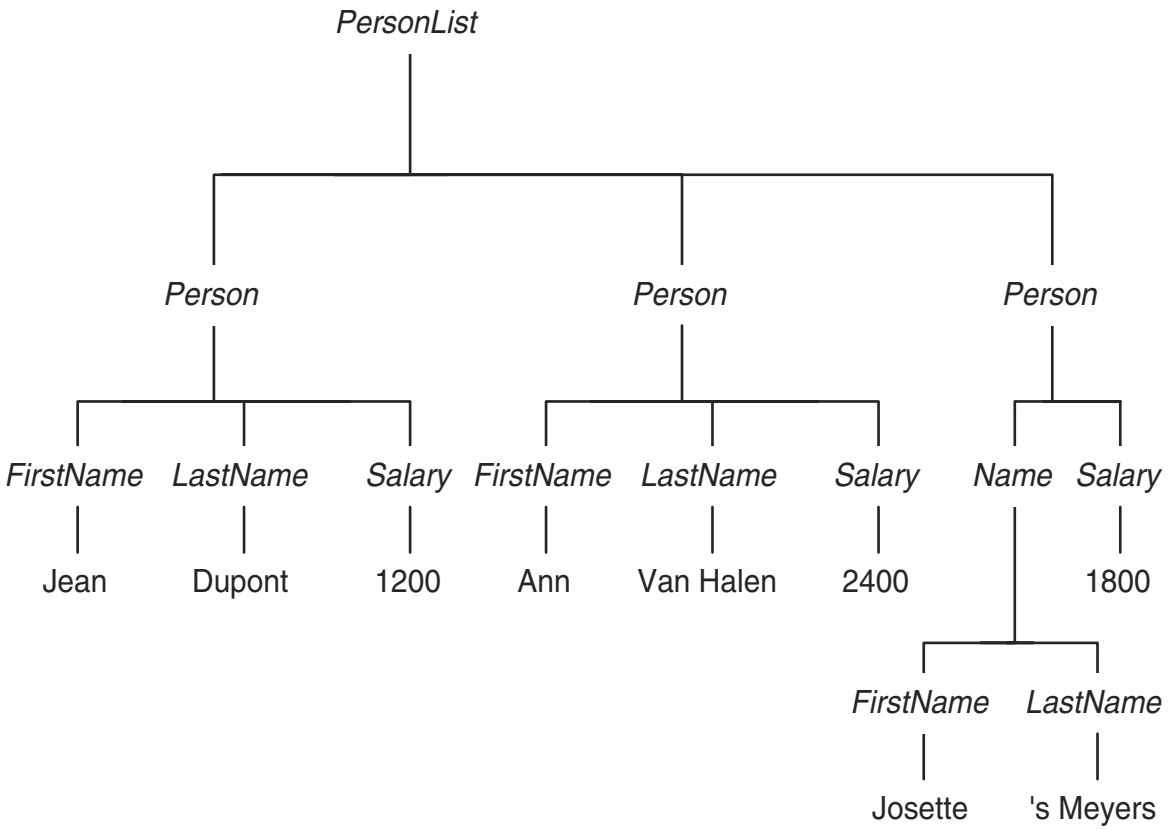
In XML, the information is organized in a **tree**-like structure, containing **elements**, which in their turn can contain more elements and/or text content. Elements are indicated through **markup** with start and end **tags**:

<elementName>content</elementName>

At the top of the tree is one single element: the root element or **document element** (<PersonList>). XML applies some typical terms for navigating trees:

- <Person> is the **parent** of <LastName>. There is only one parent per child;
- <LastName> is a **child** of <Person>;
- <LastName> and <Salary> are each other's **sibling**.

Figure 2: An XML Document Tree



Note that the XML tree doesn't have to be 'symmetrical', e.g. the Person elements in the above example may have different structures.

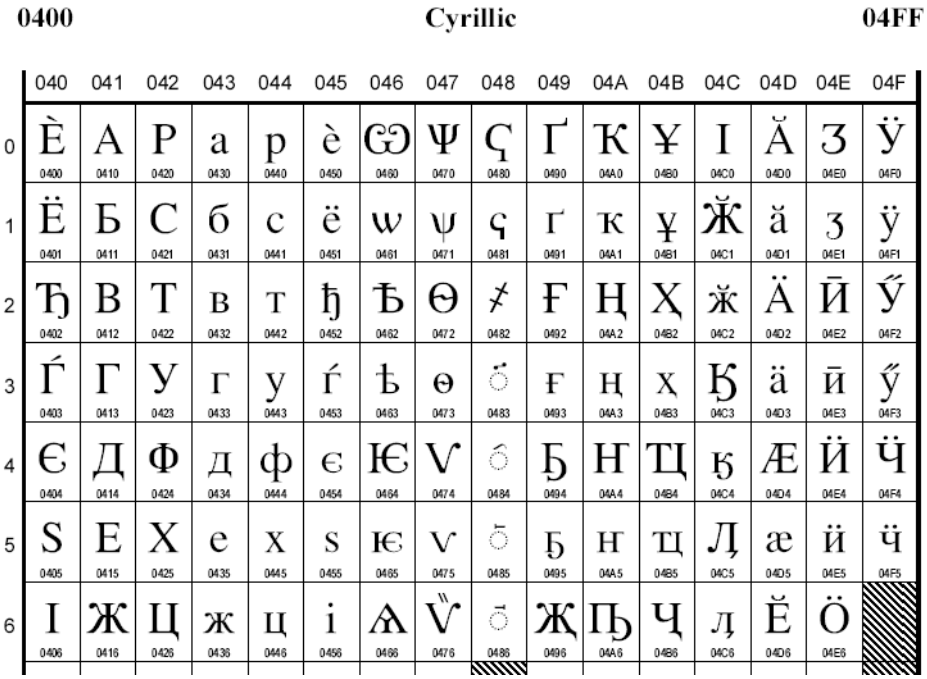
One universal character set: Unicode

- unique code for each character (more then 100 000!)

Several encodings:

- UTF-16 (2, 4, ... bytes)
- UTF-8 (1, 2, 3, ... bytes)
- ISO-8859-x (1-byte)
- etc...

See www.unicode.org



Encoding indicated in xml declaration:

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


1.2.3 Unicode

Unicode provides a unique number for every character, no matter what the platform, no matter what the program, no matter what the language.

Unicode was originally (1987) designed as a simple character set of 16-bit scalar values, since this would include most of the special symbols or characters needed for any modern language.
While 65.535 characters might be enough to support most current languages, the needs of scholars and historians, coupled with the dynamic nature of ideographic languages, demanded a larger number space. Since 1996, surrogate blocks allow to expand the character number space beyond 16-bit numbers.
To read more about Unicode, visit their website http://www.unicode.org.

Actually a distinction must be made between the **character set** and its **encoding**(s).

- The **UTF-8** encodes 7-bit ASCII normally (1 byte), but requires 2 to 5 bytes for everything else - this is great for predominantly ASCII text, but is not very nice for the rest of the world.

- The **UTF-16** encoding is the simplest, storing most characters as 2 bytes. Additional characters may be encoded using surrogate blocks, resulting in 4 bytes.

Besides the above encodings, which *should* be supported by all XML processors, many other encodings *may* be supported, some of them (such as the ISO-8859 series) being limited 1-byte encodings, which will not allow the representation of all Unicode characters: e.g. the ISO-8859-1 does not support the euro sign.

XML syntax

XML = text (= markup + data):

- **case sensitive**
- **Unicode character set**
- **special characters:**

<	<
>	>
&	&
‘	'
“	"

Names (e.g. elements & attributes):

- **start with standard letter or underscore _ or colon :**
- **no spaces**
- **no (xml..., XML..., xML...)**
- **case sensitive**

Well-formedness:

- **one root element**
- **properly nested elements**
- **the other syntax rules...**

1.2.4 Some XML syntax

Text

XML is case-sensitive (unlike HTML), including element tags and attribute values.

XML uses the characters that are defined in the Unicode character set and supports multiple encodings (at least UTF-8 and UTF-16).

5 characters have special meaning in XML, they must not appear within regular text data. All of these characters have an alternate representation in the form of character entity references. You may also define your own entity references.

Names

Most XML structures are named, e.g. elements and attributes.

Names can contain letters, numbers, and other characters (underscore, colon, hyphen and period. It can also contain non-latin characters.

Names must not start with a number or other punctuation characters (except underscore and colon).

Names must not contain spaces

Names must not begin with xml, XML or any other string that would match any variation of those three characters (reserved for W3C use only).

1.2.5 Well-formedness

A document with intelligible markup is called a **well-formed** document.

The basic recipe for constructing a well-formed document is rather simple: make sure all elements are properly nested and closed, have only one root element, use the proper characters, etc...

Note that well-formedness does *not* guarantee that the document contains the proper structure and data for a given application, i.e. that its content is **valid** (see further).

Diverse content

- **data oriented: e.g. PersonList**
- **document oriented: e.g.**

```
<PLAY>
  <TITLE>The Tragedy of Hamlet, Prince of Denmark</TITLE>
  <PERSONAE>
    <TITLE>Dramatis Personae</TITLE>
    <PERSONA>CLAUDIUS</PERSONA>
    <PERSONA>HAMLET</PERSONA>
    <PERSONA>POLONIUS</PERSONA>
    <PGROUP>
      <PERSONA>ROSENCRANTZ</PERSONA>
      <PERSONA>GUILDENSTERN</PERSONA>
      <GRPDESCR>courtiers</GRPDESCR>
    </PGROUP>
    <PERSONA>OPHELIA</PERSONA>
    <PERSONA>Ghost of Hamlet's Father</PERSONA>
  </PERSONAE>
  <ACT>
    <TITLE>ACT I</TITLE>
    <SCENE>
      <TITLE>SCENE I. Elsinore, before the castle.</TITLE>
      <STAGEDIR>FRANCISCO at his post. Enter BERNARDO</STAGEDIR>
      <SPEECH>
        <SPEAKER>BERNARDO</SPEAKER>
        <LINE>Who's there?</LINE>
      </SPEECH>
      <SPEECH>
        <SPEAKER>FRANCISCO</SPEAKER>
        <LINE>Nay, answer me: stand, and unfold yourself.</LINE>
      </SPEECH>
      ...
      <STAGEDIR>Enter HAMLET</STAGEDIR>
      <SPEECH>
        <SPEAKER>HAMLET</SPEAKER>
        <LINE>To be, or not to be: that is the question:</LINE>
        <LINE>Whether 'tis nobler in the mind to suffer</LINE>
        <LINE>The slings and arrows of outrageous fortune,</LINE>
        ...
```

- **even more document oriented:**

```
...
<STAGEDIR>It is night,<PERSONA>FRANCISCO</PERSONA> is at his post, when
<PERSONA>BERNARDO</PERSONA> enters.</STAGEDIR>
...
```

1.2.6 Data oriented vs. document oriented

XML documents can contain any text load.

Sometimes this is very structured data, such as you would typically find in a **relational database**. The document contains elements with a very predictable (repeating) structure, e.g. the PersonList of the previous example. It is said to be **data oriented**. A further distinction could be made:

- really flat single-table documents: they contain e.g. only Person data;
- rather composite documents, containing all the data for a given transaction (e.g. an enrolment for a course, containing details about the course, the enroller, the enrollee, the invoicing, etc...

On the other hand the content can be some semi-structured text like a book or a technical manual, containing elements like <paragraph> and <chapter> along with some more semantic things like <author> or <publicationDate>. Such an XML document is said to be **document oriented** or 'narrative'. Typical in this kind of documents is the occurrence of **mixed content**, i.e. flat text interspersed with elements (this is the case also for most HTML documents).

Note: it may seem somewhat confusing that officially we always speak about XML *documents*, regardless of their orientation (data or document)...

XML = Extensible *Markup* Language

Markup in general:

- **formatting markup: layout & presentation**

... to be, or not to be, <bold>that is the question</bold>: whether ...

- **generic markup: structure, content**

... <person><firstName>John</firstName>...

Discussion: separate content and presentation?

1.3 XML as a markup language

XML stands for Extensible *Markup Language*. This paragraph discusses the nature of markup languages and their evolution in the Word Wide Web context.

1.3.1 What is markup?

The term **markup** was originally used in the editing and printing world (>formatting markup):

> *Detailed stylistic instructions for typesetting something that is to be printed; manual markup is usually written on the copy (e.g. underlining words that are to be set in italics)*

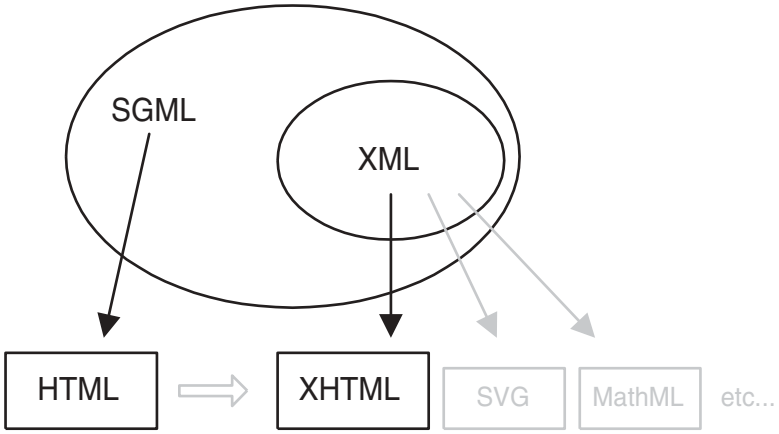
In the IT world it got a more generalized meaning (>generic markup):

> *In computerised document preparation, a method of adding in-place information to the text indicating the logical components of a document, such as paragraphs, headers or footnotes. SGML is an example of such a system. Specific instructions for layout of the text on the page do not appear in the markup.*

Extrapolated further, the markup can also contain semantics, such as a Person containing a FirstName etc.

On of the main issues in markup land is still the separation between content and presentation.

Figure 3: SGML-XML-HTML



SGML= Standard Generalized Markup Language (1986)

- a lot of features
- complicated
- hard to implement in a web context

1.3.2 Markup languages

HTML (the HyperText Markup Language) turned the Web into the world's library. Now its sibling, XML (the Extensible Markup Language) is making the Web the world's commercial and financial hub. How?

You can see why by comparing XML and HTML. Both are based on SGML, but have a different goal. The goal of HTML is presenting data in a web browser in such a way that it was easy for a web developer to write it. The goal of XML goes further; it's going to structure the information in such a way that it can be used and presented in any way you want it.

SGML

What is XML's relationship to SGML and HTML? They obviously have one thing in common: the "ML" in their names. It stands for **markup language**, a collection of markup codes and rules for adding those codes to documents in order to indicate the structure or appearance of those documents.

Early markup languages were usually invented by companies that sold document processing software. As an alternative for these proprietary systems, SGML - Standard Generalized Markup Language - was issued as an International Standard in 1986. The "Standard Generalized" part meant that document developers were no longer tied to a particular vendor's markup language, but could instead develop their own markup, ideally in such a way that their documents would easily convert into other formats.

The problem with SGML was that it was very complicated to use. In 1989, a researcher of CERN (Centre Européen pour la Recherche Nucléaire) started from a simple example document type in the SGML standard and developed a hypertext version called the Hypertext Markup Language. The simplicity of HTML and the other Web specifications allowed programmers around the world to quickly build systems and tools to work with the Web.

HTML (1990)

- defines layout & presentation, not structure
- is not extensible (predefined tags)
- can not be validated

The reaction of the W3C:

- Cascading Style Sheets (CSS)
- Extensible Markup Language (XML)

XML (1998):

- defines structure & semantics
- extensible (invent your own tags!)
- can be validated (DTD, Schema, ...)

HTML

There are several problems with HTML. It only uses a fixed set of element types. It is not extensible and therefore cannot be tailored for particular document types. There was also no way to check the structure or do any kind of validation on HTML documents.

As the Web grew in popularity many people started to chafe under HTML's fixed document type. Browser vendors saw an opportunity to gain market share by making incompatible extensions to HTML. Most of the extensions were formatting commands and thus damaged the Web's interoperability.

It is clear now that HTML is certainly not perfect. What about this SGML thing then? Why can't we just use that? It is true that SGML makes it possible to define your own formats for your own documents, to handle large, complex documents, and to manage large information repositories.

Full SGML contains many options not needed for web applications, and has a cost/benefit ratio that is unattractive to current web-browser vendors

Improving on HTML

As the interoperability and scalability of the Web became more and more endangered by proprietary formatting markup, the World Wide Web Consortium decided to act. They attacked the problem in several ways:

- They invented a simple HTML-specific stylesheet language called Cascading Style Sheets (CSS) that allowed people to attach formatting to HTML documents without filling the HTML itself with proprietary, rendition-oriented markup.

- The CSS was only a stopgap. For the Web to move to a new level, it had to incorporate the third of SGML's important ideas, that document types should be formally defined so that documents can be checked for validity against them.

- Therefore, the World Wide Web Consortium (http://www.w3c.org) decided to develop a subset of SGML that would retain SGML's major virtues but also embrace the Web ethic of minimalist simplicity. They decided to give the new language the catchy name **Extensible Markup Language** (XML). This XML offered the things we were looking for: it was about content, you could invent your own tags (hence: extensible), and it has a mechanism for validating documents (originally through DTD, later on also through XML Schema and other means).

CHAPTER 2. XML DOCUMENT

This chapter will explain the concepts of how to create an XML document. This will give us the basic ideas we will need in the following chapters when we are imposing structure through DTD and XML Schema.

Syntactic Constructs

Text strings (Unicode)

Mark-up and data

Case-sensitivity

White space: tab, spaces, CR, LF, ...

Names

- **unicode**
- **no space**
- **start with A-Z, a-z, _, :**

2.1 Syntactic constructs

2.1.1 String

XML documents are composed of characters from the **UNICODE** character set. Any such sequence of characters is called a string.

2.1.2 Case-sensitivity

XML is case-sensitive. That means that if the XML specification says to insert the word “ELEMENT”, it means that you should insert “ELEMENT” and not “element” or “Element” or “EleMeNt”.

2.1.3 Mark-up and data

The constructs such as tags, entity references, and declarations are called **mark-up**. These are the parts of your document that are supposed to be understood by the XML processor. The parts that are between the mark-up are typically supposed to be understood only by other human beings. That is the **character data**.

2.1.4 White space

White space characters (like tab, spaces, carriage return, line feed) often have no significance for XML, e.g. in between elements (unless mixed content). This allows for example for ‘pretty printing’ an XML document.

2.1.5 Names

For names of elements, entities and attributes you can use anything with a few restrictions:

- must start with a standard letter (A-Z, a-z), underscore(_) or colon (:);
- must not start with any variant of the xml string (xml XML Xml etc.);
- cannot contain spaces.

Document and Document fragment

Document

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

<personList>
    <person>
        <name>
            <firstName>Mary</firstName>
            <lastName>Jones</lastName>
        </name>
    </person>
    <person>
        <name>
            <firstName>John</firstName>
            <lastName>Williams</lastName>
        </name>
    </person>
</personList>
```

Document Fragment

```
        <name>
            <firstName>John</firstName>
            <lastName>Williams</lastName>
        </name>
    </person>
    <person>
        <name>
            <firstName>Mary</firstName>
            <lastName>Jones</lastName>
        </name>
```


2.2 Nodes

2.2.1 Documents

A Document node is a complete document, including the XML Declaration. All XML documents may (and should!) begin with a single XML declaration.

Well-formed XML documents are defined as being in the form of a simple hierarchical tree, each document having one, and only one, root node, called the document entity or the document root.

The document root of each XML document is also the point of attachment for the document's description using a DTD or Schema.

2.2.1.1 The XML Declaration

The attributes have been defined by the XML 1.0 specification:

- **version** -compulsory; value must be "1.0"; this attribute enables support of future versions of XML.
- **encoding** -optional; value must be legal character encoding, such as "UTF-8", "UTF-16", or "ISO-8859-1" (the Latin-1 character encoding). All XML parsers are required to support at least UTF-8 and UTF-16. If this attribute is not included, "UTF-8" or "UTF-16" encoding is assumed.
- **standalone** -optional; value must be either "yes" or "no"; where "yes" means that all required entity declarations are contained within the document, and "no" means that an external DTD is required.

2.2.1.2 Character Encodings

Unicode provides a unique number for every character, no matter what the platform, no matter what the program, no matter what the language.

Unicode was originally designed as a simple set of 16-bit scalar values, since this would include most of the special symbols or characters needed for any modern language, but was later expanded to allow for much more than 65 535 characters with 'surrogate blocks'.

The UTF-8 encoding encodes 7-bit ASCII 'normally'(as 1 byte), but requires 2 to 5 bytes for everything else - this is great for predominately ASCII text, but is not very nice for the rest of the world.

The UTF-16 encoding is the simplest, storing most characters as 2 bytes, very 'special' characters require 4 bytes.

2.2.2 Document fragments

A document fragment is a part of a document that might or might not be well formed. It can be a group of elements with no root element.

Document Types

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

<!DOCTYPE  personList [<!ELEMENT personList (person)+ > <!ELEMENT
person (name)> <!ELEMENT name (firstName, lastName)> <!ELEMENT
 firstName (#PCDATA)><!ELEMENT lastName (#PCDATA)>]>

<personList><person><name><firstName>John</firstName><lastName>
Williams</lastName></name></person><person><name>
<firstName>Mary</firstName><lastName>Jones</lastName></name>
</person></personList>
```

SYSTEM

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE personList SYSTEM "person.dtd">
<personList><person><name><firstName>John</firstName><lastName>
Williams</lastName></name></person><person><name><firstName>
Mary</firstName><lastName>Jones</lastName></name></person>
</personList>
```

PUBLIC

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE personList PUBLIC "-//Abis Training and Consulting"
"http://www.abis.be/person.dtd">
<personList><person><name><firstName>JohnJohn</firstName>
<lastName>Williams</lastName></name></person><person><name>
<firstName>Mary</firstName><lastName>Jones</lastName></name>
</person></personList>
```

2.2.3 Document types

An XML document may contain a DOCTYPE declaration. It contains information that tells the processor about the Document Type Definition (or DTD). The first argument refers to the root element of the XML document. These two must match because the DTD can describe only specific structures.

Public

The PUBLIC identifier is used to reference a DTD that is catalogued and referenced using some method that is not defined in XML, but rather is the result of a standard within an organization, or an agreement between parties exchanging XML data.

System

An XML parser may try to generate a URI using the PUBLIC identifier. If it is unable to do so, the SYSTEM identifier may be used.

Elements

- **Root Element (Document Element)**
- **Child Elements (descendants)**

```
<personList>
        <person>
                <name>
                    <firstName>John</firstName>
                    <lastName>Williams</lastName>
                </name>
        </person>
        <person>
                <name>
                    <firstName>Mary</firstName>
                    <lastName>Jones</lastName>
                </name>
        </person>
</personList>
```

Empty element:

```
<middleName></middleName>
<middleName/>
```


2.2.4 Elements

Elements are the basic building blocks of an XML mark-up. They may contain other elements, character data, character references, entity references, PI's (processing instructions), comments and/or CDATA sections- these are collectively known as element content.

Elements are delimited by tags, which consist of the element type name enclosed with a pair of angle brackets ("< >"). Every element must be delimited with a **start-tag** and an **end-tag**, unlike loose HTML the end tag cannot be missed. A special case is the shorthand notation for an **empty element**, e.g. <middleName/>.

Root Element

The root element is the topmost element in a document. It is the child of the document itself and contains all the other elements in the document as descendants.

Child Elements

All other elements in an XML document are descendants ("children") of the document element. A document consists of the element tree and its parent-child relationships. XML imposes a key constraint upon elements: they must be properly nested.

Attributes

- ‘extra’ information
- (metadata)

```
<person>
        <age>47</age>
        <income valuta="EUR">2000</income>
        <name firstName="Mary" lastName=”Jones"></name>
</person>
```


2.2.5 Attributes

If elements are the “nouns” of XML, then attributes are its “adjectives”. Often there is some extra information that is to be added to the element. This can be done using attributes, each of which is comprised of a name-value pair.

In fact attributes are mainly intended to represent **metadata** (such as keys, units, file names, types) rather than data (such as “John” or “Brussels”).

Attributes have no specific order and must be unique for each element.

Text

```
<personList>
        <person>
                <name>
                    <firstName>John</firstName>
                    <lastName>Williams</lastName>
                </name>
        </person>
        <person>
                <name>
                    <firstName>Mary</firstName>
                    <lastName>Jones</lastName>
                </name>
        </person>
</personList>
```

```
<personList><person><name><firstName>John</firstName><lastName>
Williams</lastName></name></person><person><name><firstName>
Mary</firstName><lastName>Jones</lastName></name></person>
</personList>
```


2.2.6 Text

Any text within another node is a text node.

Note that white space characters, such as spaces, tabs, carriage return and line feed, have no significance in certain situations, e.g. in between elements (unless for mixed-content elements), so that in principle an entire XML document could be on a single line.

Entities, Entity references and Notations

Character entity references

©
©

Entity references

&someText;

Table 1: Character Entity References

Entity	Usage
`&amp;`	used to escape the & character
`&lt;`	used to escape the < character
`&gt;`	used to escape the > character
`&apos;`	used to escape the ‘ character
`&quot;`	used to escape the “ character

Notation

2.2.7 Entities, entity references and notations

XML provides two simple methods of representing characters that don't exist in the ASCII character set.

Character entity references

Represent a displayable character, and are comprised of a decimal or hexadecimal number, preceded by "&#" and followed by a semi-colon(;)

Example
©
©

€
euro sign

Entity references

Entity references are legal XML names, preceded by an ampersand (&) and followed by a semi-colon (;) character.

Notations

Notations are a (complex) technique, inherited from SGML, to include and treat non-XML information in an XML context.

PI's, Comments and CDATA

Processing Instruction (PI's)

```
<?Dumb-Formatter FONT="16pt"?>
```

Comments

```
<!-- This is the way to have comments in your XML documents or DTDs -->
```

CDATA

```
<![CDATA[ John&Mary's kids: <name>Rick</name> &
                <name>Jane</name>  ]]>
```


2.2.8 Processing Instructions, Comments and CDATA Sections

Processing Instruction (PI's)

Since XML, like SGML before it, is a descriptive language, it does not presume to try to explain how to handle an element, or its contents, but a PI allows us to leave specific instructions for an specific processor/program.

```
<?target ... instruction ...?>
```

Comments

Can be useful to insert notes, or comments, into a document.

```
<!-- ... comment text ... -->
```

CDATA sections

CDATA sections are a method of including text that contains characters that would otherwise be recognized as mark-up.

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

CHAPTER 3. XML SCHEMA

In the early days of XML, the Document Type Definition (DTD), as borrowed from the SGML, was the only way of constraining an XML document. From the 21st century on, a more sophisticated standard emerged: the XML Schema.

These days, XML Schema is the "official" standard for validating XML documents.

Comparing XML Schema and DTD

Problems with DTD:

- **they use a ‘different’ syntax (from SGML)**
- **programmatic processing of their metadata is difficult**
  - **no real API**
- **DTDs are not extensible**
- **Do not support namespaces**
- **Do not support datatypes**
- **DTDs do not support inheritance**

3.1 Introduction to XML Schema

3.1.1 Problems with DTD

DTDs are difficult to write and to understand

DTDs are inherited from the SGML era, and use a syntax other than XML, namely Extended Backus Naur Form (EBNF), which many people find difficult to read and use. XML Schema, however, actually use XML to describe the XML documents they define.

Programmatic processing of metadata is difficult

We have programming interfaces for XML documents (DOM, SAX), but you can't use these programming interfaces for documents written in EBNF.

DTDs are not extensible and do not provide support for namespaces

DTDs do not really support the whole namespaces concept. On the other hand DTDs are not extensible: we have to retype everything if we want to add a new element, we can't say this is also a person but with some extra features.

DTDs do not support datatypes

DTDs check structures but don't look at the contents of the elements.

DTDs do not support inheritance

The idea comes from OO where you can define objects who inherit certain characteristics from other objects. A book is a publication, so why can't we define books and inherit the already defined characteristics of a publication.

Comparing XML Schema and DTD

Advantages of XML Schema:

- **uses XML syntax**
- **element and attribute definitions are possible**
- **open content models can be defined**
- **greater flexibility for content models**
- **provide a richer set of datatypes**
- **allow built-in and user-defined datatypes to be created**

Other Schema Technologies

- **XDR (Microsoft) (deprecated)**
- **Schematron**
- **Relax NG**

XML Schema 1.0 specification (2001)

- **Part 0: Primer**
- **Part 1: Structures**
- **Part 2: Datatypes**

XML Schema 1.1 (2012)

3.1.2 Advantages of XML Schema

- XML Schema uses **XML Syntax**

  This gives us the following advantages:

  - No need to learn another language

  - XML Schema can be used like XML: for example, parsers can parse Schema files, the Schema files can be transformed with XSLT and it is extensible

- XML Schema supports a rich set of **data types** and it is possible to define our proper data types. This makes it easier to work with data from/for a database.

- In XML Schema the primary key - foreign key pair that we are familiar with in the database system can be simulated.

- XML Schema provides a better **content modelling** mechanism. It has the ability to constrain the order and number of child elements in mixed content models. For child element a minimum and/or maximum number of occurrences can be defined

- XML Schema supports **namespaces**: the different elements from the document that has to be validated can have the same name without causing conflict, because every name is associated with a namespace. This is also possible because elements can be defined in their context.

3.1.3 Other Schema Technologies

Because the ability to validate document instances and describe vocabularies is so important to XML, during the wait for schemas to become a W3C standard a number of other proposed schema languages have arisen. The most important one was Microsoft's XML-Data Reduced (XDR), which was implemented in MSXML and IE5, and used in strategies such as BizTalk. It was based on a submission to W3C called XML-Data, which has greatly influenced the development of XML Schemas. With the introduction of the official XML Schema standard (2001), XDR gradually disappeared.

Other techniques for validation (more powerful and/or simpler than XML Schema) include Schematron and Relax NG.

3.1.4 XML Schema specification

There are three parts of the XML Schema specification. There is the primer, which introduces the key topics in schemas, and two reference sections: structures and datatypes.

- Datatypes are the basic building blocks of XML Schemas, which are used as the basis of all the larger components of a schema.

- Structures are more like the compounds: these can be constructed from the datatypes, and are used to describe the element, attribute, and validation structure of a document type.

The XML Schema 1.0 (2001) is still the most used. The 1.1 version (2012) added a.o. assertions (as already present in Schematron).

Schemas begin with the <schema> tag

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

Elements and attributes must be defined in a structure

Defining an element:

- name
- type
- [cardinality]

```
<xsd:element name="title" type="xsd:string" minOccurs="0"/>
```

Defining an attribute

- name
- type
- [use]

```
<xsd:attribute name = "countrycode"
type="xsd:string" use="required" />
```


3.2 Basic syntax elements

3.2.1 The schema tag

The `xmlns` attribute defines the namespace for schemas. This means that the language for defining schemas uses the predefined tags of that namespace.

3.2.2 Elements

Elements are defined with attributes on the element tag: (list is not complete)

name	Element name
type	Specifies the element type
minOccurs	Specifies the number of occurrences of the element in this content model. 0 makes the element optional default: 1
maxOccurs	The maximum number of occurrences of the element in this content model. ‘unbounded’ means any number of times. default: 1

It’s possible to define the number of occurrences of an element (= cardinality). It’s possible to specify the attributes minOccurs (the minimum number of occurrences) and maxOccurs (the maximum number of occurrences). The value “unbounded” on maxOccurs means there is no maximum. Both attributes have a default value of one.

3.2.3 Attributes

Attributes are defined with attributes on the attribute tag: (list is not complete)

name	Attribute name
default	default value
use	whether this is a ‘required’ or ‘optional’ attribute

Example of a Schema

country.xml

```
<country     countryCode="BE"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:noNamespaceSchemaLocation="country.xsd">
    <countryName>Belgium</countryName>
    <continent>Europe</continent>
    <countryLeader>
        <title>King</title>
        <name>Philippe/Filip</name>
        <since>2013-07-21</since>
    </countryLeader>
</country>
```

country.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:element name="country">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="countryName"  type="xsd:string"/>
                <xsd:element name="continent" type="xsd:string"/>
                <xsd:element name="countryLeader">
                    <xsd:complexType>
                        <xsd:sequence>
                            <xsd:element name="title" type="xsd:string"/>
                            <xsd:element name="name" type="xsd:string" />
                            <xsd:element name="since" type="xsd:date"/>
                        </xsd:sequence>
                    </xsd:complexType>
                </xsd:element>
            </xsd:sequence>
            <xsd:attribute name ="countryCode" type="xsd:string"/>
        </xsd:complexType>
    </xsd:element>
</xsd:schema>
```


3.2.4 Example of a Schema

The www.w3.org/2001/XMLSchema namespace is used for identifying the Schema itself.

The www.w3.org/2001/XMLSchema-instance namespace is used for Schema extensions in documents (documents are to be considered as ‘instances’ of a schema).

The element `xsi:noNamespaceSchemaLocation` defines the location of a W3C Schema without a target namespace.

Defining structure (I)

```
<?xml version="1.0" encoding="UTF-8"?>
<CompanyList>
    <Company>
        <CompanyName>ABIS N.V.</CompanyName>
        <CompanyAddress>LEUVEN</CompanyAddress>
        <Boss>
            <Name>
                <FirstName>JAN</FirstName>
                <LastName>SMITHS</LastName>
            </Name>
            <Salary>2450</Salary>
            <Birthday>1947-03-24</Birthday>
        </Boss>
        <Employees>
            <Employee>
                <Name>
                    <FirstName>JOS</FirstName>
                    <LastName>BOSMANS</LastName>
                </Name>
                <Salary>1450</Salary>
                <Birthday>1975-09-02</Birthday>
            </Employee>
            <Employee>
                <Name>
                    <FirstName>ANN</FirstName>
                    <LastName>DE KEYSER</LastName>
                </Name>
                <Salary>1550</Salary>
                <Birthday>1952-05-24</Birthday>
            </Employee>
            <Employee>
                <Name>
                    <FirstName>PETER</FirstName>
                    <LastName>TAVERNIER</LastName>
                </Name>
                <Salary>2445</Salary>
                <Birthday>1953-04-24</Birthday>
            </Employee>
        </Employees>
    </Company>
    <Company>
            ...
    </Company>
</CompanyList>
```


3.3 Defining structure

The XML document on the opposite page represents a CompanyList.

A CompanyList consists of one or more Company elements.

Each Company has got a CompanyName, a CompanyAddress, a Boss and an Employees element.

Figure 4: The CompanyList Structure

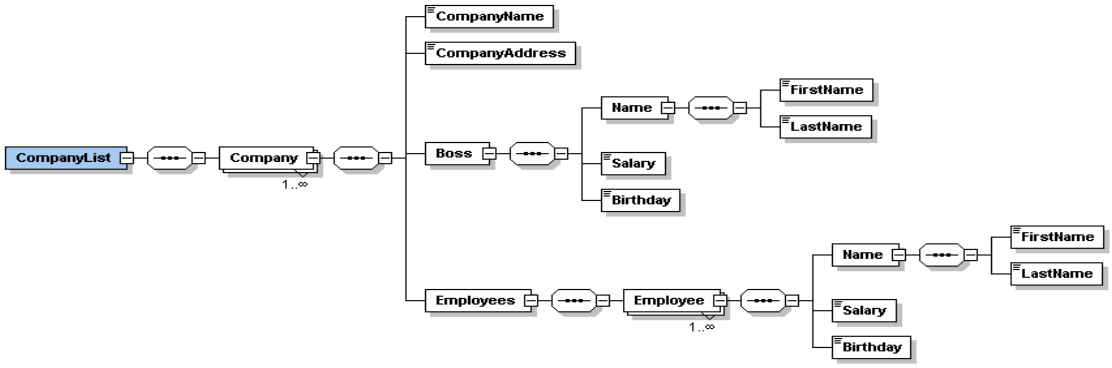


Figure 5: The CompanyList Document

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
- <CompanyList xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="VenetianBlindCompanyList.xsd">
  - <Company>
      <CompanyName>ABIS N.V.</CompanyName>
      <CompanyAddress>LEUVEN</CompanyAddress>
    - <Boss>
      - <Name>
          <FirstName>JAN</FirstName>
          <LastName>SMITHS</LastName>
        </Name>
        <Salary>2450</Salary>
        <Birthday>1947-03-24</Birthday>
      </Boss>
    - <Employees>
      - <Employee>
        - <Name>
            <FirstName>JOS</FirstName>
            <LastName>BOSMANS</LastName>
          </Name>
          <Salary>1450</Salary>
          <Birthday>1975-09-02</Birthday>
        </Employee>
      + <Employee>
      + <Employee>
      </Employees>
    </Company>
  + <Company>
  </CompanyList>
```

The first picture on the left shows the general structure of the document. The way of representing a structure in a schematic overview is the one used in XMLSpy.

The second picture on the left represents a particular document (=instance, created according to a schema) in browser view.

The element and attribute tags must be combined to define the structure. There are three well known schema design ‘patterns’, which are often combined:

1. Russian Doll Design or local components/definitions

   The Russian Doll Design tightly follows the structure of an XML document. One of the key features is to define each element and attribute within its context.

2. Salami Slice or global components

   The elements and attributes are defined separately and are referred to, to build up the structure.

3. Venetian Blind or global types

   Complex and simple (data) types are made and used to define the elements.

The Russian Doll Design

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="CompanyList">
        <xs:complexType><xs:sequence>
            <xs:element name="Company" maxOccurs="unbounded">
                <xs:complexType><xs:sequence>
                    <xs:element name="CompanyName"/>
                    <xs:element name="CompanyAddress"/>
                    <xs:element name="Boss">
                        <xs:complexType>
                            <xs:sequence>
                                <xs:element name="Name">
                                    <xs:complexType>
                                        <xs:sequence>
                                        <xs:element name="FirstName" />
                                        <xs:element name="LastName"/>
                                    </xs:sequence>
                                    </xs:complexType>
                                </xs:element>
                                <xs:element name="Salary"/>
                                <xs:element name="Birthday"/>
                            </xs:sequence>
                        </xs:complexType>
                    </xs:element>
                    <xs:element name="Employees">
                        <xs:complexType><xs:sequence>
                                <xs:element name="Employee"
                                    maxOccurs="unbounded">
                                    <xs:complexType><xs:sequence>
                                            <xs:element name="Name">
                                            <xs:complexType>
                                            <xs:sequence>
                                                <xs:element name="FirstName"/>
                                                <xs:element name="LastName"/>
                                            </xs:sequence>
                                            </xs:complexType>
                                            </xs:element>
                                            <xs:element name="Salary"/>
                                            <xs:element name="Birthday"/>
                                    </xs:sequence></xs:complexType>
                                </xs:element>
                        </xs:sequence></xs:complexType>
                    </xs:element>
                </xs:sequence></xs:complexType>
            </xs:element>
        </xs:sequence></xs:complexType>
    </xs:element>
</xs:schema>
```

3.3.1 The Russian Doll Design

A Russian Doll design schema mirrors the XML document. Each component (element or attribute) is defined within its context. This means that a child element is defined within the definition of the parent element: the child element is defined ‘locally’.

In the example the elements ‘Name’, ‘Salary’ and ‘Birthday’ are defined within the definition of the elements ’Boss’ and ‘Employee’.

This method has the advantage that it allows multiple occurrences of a same element name with a different meaning or definition. In the example ‘Name’ makes part of the definition of ‘Employee’. ‘Name’ could also make part of the definition of a company element in the same XML document.

It has the disadvantage that the definitions (element, attribute, type definitions) are not reusable.

The attribute types of an element must be specified last. There appears to be no special reason for this, but the W3C XML Schema Working Group thought it simpler to impose a relative order to the definitions of the list of elements and attributes within a complex type.

Types

A complexType is a datatype for elements holding attributes and non-text children. The definition of a complexType must contain a compositor to define the order of the sub-elements. In this example the sequence compositor was used.
A simpleType datatype is valid for an element that holds only values and no element or attribute sub-nodes.

Salami Slice - Global definition

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="CompanyList">
        <xs:complexType><xs:sequence>
            <xs:element name="Company" maxOccurs="unbounded">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name="CompanyName"/>
                        <xs:element name="CompanyAddress"/>
                        <xs:element name="Boss">
                            <xs:complexType>
                                <xs:sequence>
                                    <xs:element ref="Name"/>
                                    <xs:element ref="Salary"/>
                                    <xs:element ref="Birthday"/>
                                </xs:sequence>
                            </xs:complexType>
                        </xs:element>
                        <xs:element name="Employees">
                            <xs:complexType>
                                <xs:sequence>
                                    <xs:element name="Employee"
                                                maxOccurs="unbounded">
                                        <xs:complexType>
                                            <xs:sequence>
                                                <xs:element ref="Name"/>
                                                <xs:element ref="Salary"/>
                                                <xs:element ref="Birthday"/>
                                            </xs:sequence>
                                        </xs:complexType>
                                    </xs:element>
                                </xs:sequence>
                            </xs:complexType>
                        </xs:element>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
        </xs:sequence></xs:complexType>
    </xs:element>
    <xs:element name="Name">
        <xs:complexType><xs:sequence>
            <xs:element name="FirstName" />
            <xs:element name="LastName"/>
        </xs:sequence></xs:complexType>
    </xs:element>
    <xs:element name="Salary" />
    <xs:element name="Birthday" />
</xs:schema>
```


3.3.2 Salami Slice

While the previous design method is very simple, it can lead to significant depth in the embedded definitions, making it hardly readable and difficult to maintain when documents are complex. It also has the drawback of being very different from a DTD structure, an obstacle for human or machine agents wishing to transform DTDs into XML Schemas, or even just use the same design guides for both technologies.

The second design is based on a flat catalog of all the elements available in the sample document and, for each of them, lists of child elements and attributes. This is achieved through using references to element and attribute definitions that need to be within the scope of the reference, leading to a flat design.

Thus the Salami Slice patterns uses **global component** definitions.

Using a reference to an element or an attribute is somewhat comparable to cloning an object. The element or attribute is defined first, and it can be duplicated at another place in the document structure by the reference mechanism, in the same way an object can be cloned. The two elements (or attributes) are then two instances of the same class.

Venetian Blind

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:complexType name="PersonType">
        <xs:sequence>
            <xs:element ref="Name"/>
            <xs:element ref="Salary"/>
            <xs:element ref="Birthday"/>
        </xs:sequence>
    </xs:complexType>
    <xs:element name="CompanyList">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="Company" maxOccurs="unbounded">
            <xs:complexType>
              <xs:sequence>
                <xs:element name="CompanyName" type="xs:string"/>
                <xs:element name="CompanyAddress" type="xs:string"/>
                <xs:element name="Boss" type="PersonType"/>
                <xs:element name="Employees">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element name="Employee"
                                        type="PersonType"
                                        maxOccurs="unbounded"/>
                        </xs:sequence>
                  </xs:complexType>
                </xs:element>
              </xs:sequence>
            </xs:complexType>
          </xs:element>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
    <xs:element name="Name">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="FirstName" type="xs:string"/>
                <xs:element name="LastName" type="xs:string"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="Salary" type="xs:string"/>
    <xs:element name="Birthday" type="xs:string"/>
</xs:schema>
```


3.3.3 Venetian Blind

W3C XML Schema gives us a third mechanism, which is to define data types (either simple types that will be used for PCDATA elements or attributes, or complex types that will be used only for elements) and to use these types to define our attributes and elements.

This is achieved by giving a name to the simpleType and complexType elements, and locating them outside of the definitions of elements and attributes.

In this way the Venetian Blind offers **global type** definitions.

Compositors: sequence

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

    <xsd:element name="countryName" type="xsd:string"/>
    <xsd:element name="continent" type="xsd:string"/>
    <xsd:element name="title" type="xsd:string"/>
    <xsd:element name="name" type="xsd:string"/>
    <xsd:element name="since" type="xsd:date"/>

    <xsd:attribute name="countryCode" type="xsd:string"/>

    <xsd:complexType name="countryLeaderType">
        <xsd:sequence>
            <xsd:element ref="title"/>
            <xsd:element ref="name"/>
            <xsd:element ref="since"/>
        </xsd:sequence>
    </xsd:complexType>

    <xsd:element name="countryLeader" type="countryLeaderType"/>

    <xsd:element name="country">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element ref="countryName"/>
                <xsd:element ref="continent"/>
                <xsd:element    ref="countryLeader"
                                minOccurs="2" maxOccurs="2"/>
            </xsd:sequence>
            <xsd:attribute ref="countryCode"/>
        </xsd:complexType>
    </xsd:element>

</xsd:schema>
```


3.4 Compositors

3.4.1 Sequence

The `xsd:sequence` compositor defines *ordered* groups of elements. The order in the XML document must be the same as the order in which the elements appear in the schema.

Note that a complexType + sequence is needed, even if an element contains only one single child element ...

Compositors: Choice / All

country.xml

```
<country countryCode="BE"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="country.xsd">
    <countryName>Belgium</countryName>
    <continent>Europe</continent>
    <countryLeader>
        <title>King</title>
        <name>Philippe/Filip</name>
        <since>2013-07-21</since>
    </countryLeader>
    <countryLeader>
        <title>Primeminister</title>
        <firstName>Charles</firstName>
        <lastName>Michel</lastName>
        <since>2014-10-11</since>
    </countryLeader>
</country>
```

country.xsd

```
<xsd:element name="countryLeader">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element ref="title"/>
            <xsd:choice>
                <xsd:element ref="name"/>
                <xsd:sequence>
                    <xsd:element name="firstName" type="xsd:string"/>
                    <xsd:element name="lastName" type="xsd:string"/>
                </xsd:sequence>
            </xsd:choice>
            <xsd:element ref="since"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
```


3.4.2 Choice

The choice compositor describes a choice between several possible elements or groups of elements. Exactly one of the alternatives is to be chosen (as with the logical XOR operation).

3.4.3 All

All defines an unordered set of elements. The elements can appear in any order but there are a few restrictions (which make the All compositor rather unpopular):

- The elements can appear only as unique children at the top of a content model.
- Their children can be only `xsd:element` definitions or references and cannot have a cardinality greater than one.

Groups

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

    <xsd:group name="location">
        <xsd:sequence>
            <xsd:element name="countryName" type="xsd:string"/>
            <xsd:element name="continent" type="xsd:string"/>
        </xsd:sequence>
    </xsd:group>

    <xsd:element name="title" type="xsd:string"/>
    <xsd:element name="name" type="xsd:string"/>
    <xsd:element name="since" type="xsd:date"/>

    <xsd:attributeGroup name="countryAttributes">
        <xsd:attribute   name="countryCode" type="xsd:string"/>
        <xsd:attribute   name="countryTelephoneCode"
                         type="xsd:string"/>
    </xsd:attributeGroup>

    <xsd:element name="countryLeader">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element ref="title"/>
                <xsd:element ref="name"/>
                <xsd:element ref="since"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>

    <xsd:element name="country">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:group ref="location"/>
                <xsd:element ref="countryLeader"/>
            </xsd:sequence>
            <xsd:attributeGroup ref="countryAttributes"/>
        </xsd:complexType>
    </xsd:element>

</xsd:schema>
```

3.5 Groups

A group is a definition that contains elements or attributes. These groups are not datatypes, but are containers holding a set of elements or attributes that can be used to describe complex types.

A group is an “invisible” ingredient as far as the XML document is concerned: it certainly is *not* an “element”. Let’s say it’s an “alias” which is just locally used in the XML Schema.

Types

ComplexTypes
= with elements and/or attributes:

- **simple content**
  - **only text nodes (no children)**
  - **can be extended (with attributes)**
- **mixed content: elements and text**

SimpleTypes
= no elements, no attributes

- **primitive data types**
  - **xs:string, xs:date, ...**
- **derived data types**
  - **derived from the primitive data types by restriction, union, list**
  - **facets: maxLength, enumeration, pattern**

Warning:

- **elements default type = xs:anyType**
- **attributes default type = xs:anySimpleType**

3.6 Types

Whereas complexTypes describe the structure (of elements), the XML Schema also allows to define the data types ('simple types') of elements and attributes. Like most modern programming languages, XML Schemas provide two basic kinds of datatypes: primitive data types and derived data types (which can be derived from existing data types by mechanisms such as restriction, list and union -- see further)

An important warning: if the type of an element is not specified, it is by default an xs:anyType, which in practice means it can contain any well-formed content (even if not defined elsewhere in the schema).
The default for attributes is xs:anySimpleType.

Empty content elements

country.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<country xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="Empty.xsd"
  countryCode="BE">
</country>
```

Empty.xsd

```
<xsd:element name="country">
    <xsd:complexType>
        <xsd:attribute name="countryCode" type="xsd:string"/>
    </xsd:complexType>
</xsd:element>
```

Simple content elements

country.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<country xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="SimpleContent.xsd"
  countryCode="BE">
    Belgium
</country>
```

SimpleContent.xsd

```
<xsd:element name="country">
    <xsd:complexType>
        <xsd:simpleContent>
            <xsd:extension base="xsd:string">
                <xsd:attribute name="countryCode" type="xsd:string"/>
            </xsd:extension>
        </xsd:simpleContent>
    </xsd:complexType>
</xsd:element>
```


3.7 ComplexTypes

3.7.1 Empty content elements

Empty content elements are defined using a regular `xsd:complexType` construct and by purposefully omitting the definition of a child element.

3.7.2 Simple content elements

Simple content elements, i.e. character data elements with attributes, can be derived from simple types using `xsd:simpleContent`.

Mixed content elements

country.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<country xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="MixedContent.xsd"
countryCode="BE">
"To be or not to be" said king <king>Philippe</king> to king
<king>Lear</king> while taking a bite out of a juicy <fruit>apple</fruit>
</country>
```

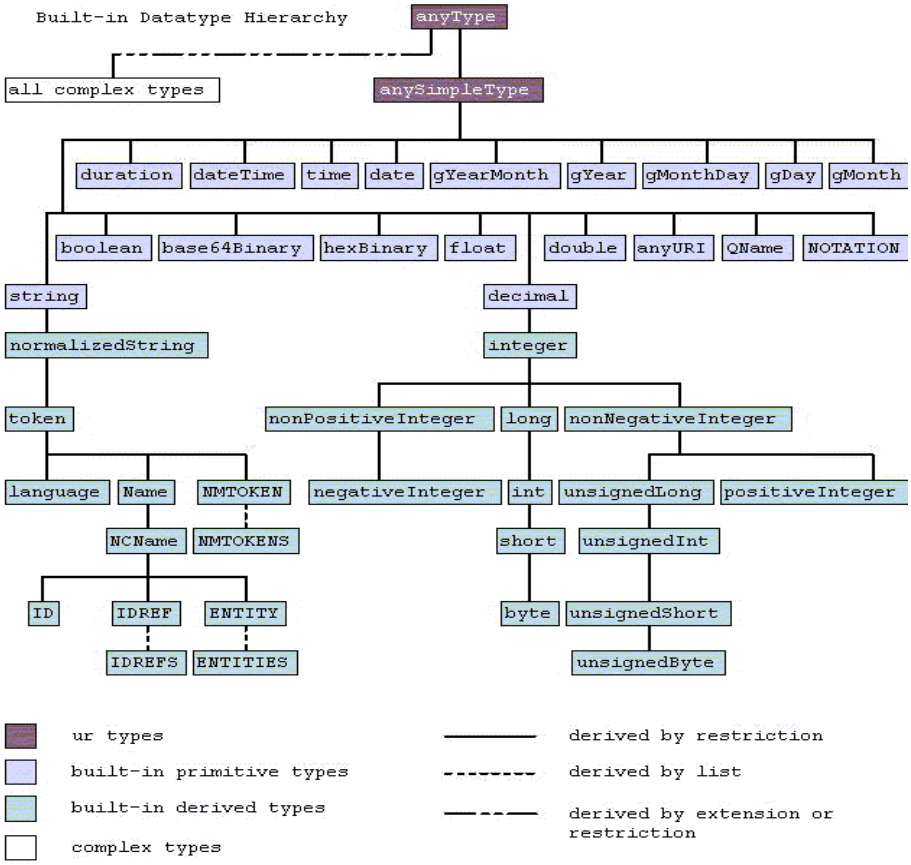
MixedContent.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:element name="country">
        <xsd:complexType mixed="true">
            <xsd:choice maxOccurs="unbounded">
                <xsd:element name="king" type="xsd:string"/>
                <xsd:element name="fruit" type="xsd:string"/>
            </xsd:choice>
            <xsd:attribute name="countryCode" type="xsd:string"/>
        </xsd:complexType>
    </xsd:element>
</xsd:schema>
```


3.7.3 Mixed content elements

W3C XML Schema supports mixed content though the mixed attribute in the `xsd:complexType` element.

Figure 6: Primitive and Derived Datatypes



3.8 SimpleTypes

Primitive datatypes are not defined in terms of other datatypes. They are like the most basic building blocks of XML schemas, rather like atoms, and indeed are the basis for all other types. For example string, Boolean, Float, Double, ID, IDREF and Entity are examples of primitive dataypes in XML Schemas.

Figure 7: Primitive Types

Table 2. Simple Types Built In to XML Schema	
Simple Type	**Examples (delimited by commas)**
string	Confirm this is electric
CDATA	Confirm this is electric
token	Confirm this is electric
byte	-1, 126
unsignedByte	0, 126
binary	62696E617279
integer	-126789, -1, 0, 1, 126789
positiveInteger	1, 126789
negativeInteger	-126789, -1
nonNegativeInteger	0, 1, 126789
nonPositiveInteger	-126789, -1, 0
int	-1, 126789675
unsignedInt	0, 1267896754
long	-1, 12678967543233
unsignedLong	0, 12678967543233
short	-1, 12678
unsignedShort	0, 12678
decimal	-1.23, 0, 123.4, 1000.00
float	-INF, -1E4, -0, 0, 12.78E-2, 12, INF, NaN
double	-INF, -1E4, -0, 0, 12.78E-2, 12, INF, NaN
boolean	true, false
time	13:20:00.000, 13:20:00.000-05:00

SimpleTypes (II)

```
<xsd:element name="PersonName">
    <xsd:simpleType>
        <xsd:restriction base="xsd:string">
            <xsd:minLength value="1"/>
            <xsd:maxLength value="10"/>
        </xsd:restriction>
    </xsd:simpleType>
</xsd:element>

<xsd:simpleType name="bankAccountType">
    <xsd:union>
        <xsd:simpleType>
            <xsd:restriction base="xsd:string">
                <xsd:pattern value="[A-Z]{2}[0-9]{16}"/>
            </xsd:restriction>
        </xsd:simpleType>
        <xsd:simpleType>
            <xsd:restriction base="xsd:string">
                <xsd:enumeration value="Onbekend"/>
                <xsd:enumeration value="Inconnu"/>
            </xsd:restriction>
        </xsd:simpleType>
    </xsd:union>
</xsd:simpleType>

<xsd:element name="BankAccount" type="bankAccountType"/>

<xsd:simpleType name="LeuvenTelephoneType">
    <xsd:restriction base="xsd:string">
        <xsd:pattern value="016/[0-9]{6}"/>
    </xsd:restriction>
</xsd:simpleType>
```


3.8.1 Derivation of simple types

Derived types are defined in terms of existing types. New types may be derived from either primitive type or another derived type. For example, the XML Schema CDATA type is derived from the `string` type, which is its base type, represents white space normalized strings of characters and is derived from the primitive type string.

Simple derived datatypes are defined by derivation from other datatypes, either predefined and identified by the W3C XML Schema namespace, or defined elsewhere in your schema.

The different kind of **restriction**s that can be applied on a datatype are called facets. E.g. the `xsd:pattern` uses a regular expression syntax to show possible values, and other facets allow constraints on the length of a value, an enumeration of the possible values, the minimal and maximal values, precision and scale, period and duration, etc.

Two other derivation methods are available that allow the definition of whitespace separated **list**s and **union** of datatypes. The example definition uses `xsd:union`, and extends the definition of our type for bank account numbers to accept the values `Onbekend` and `Inconnu`.

Constraints (I)

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<xsd:element name="country">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="countryName"/>
                <xsd:element    name="countryLeader"
                                maxOccurs="unbounded">
                    <xsd:complexType>
                        <xsd:sequence>
                            <xsd:element name="firstName"/>
                            <xsd:element name="number"/>
                        </xsd:sequence>
                    </xsd:complexType>
                </xsd:element>
            </xsd:sequence>
        </xsd:complexType>
        <xsd:unique name="oneLeader">
            <xsd:selector xpath="countryLeader"/>
            <xsd:field xpath="firstName"/>
            <xsd:field xpath="number"/>
        </xsd:unique>
</xsd:element>
</xsd:schema>
```


3.9 Constraints

3.9.1 Unique constraints

W3C XML Schema provides several flexible XPath-based features for describing uniqueness constraints and corresponding references constraints. The first of these, a simple uniqueness declaration, is declared with the xs:unique element.

The two XPaths defined in the uniqueness constraint are evaluated relative to the context node.

The first of these paths is defined by the selector element. The purpose is to define the element which has the uniqueness constraint -- the node to which the selector points must be an element node.

The second path, specified in the xs:field element is evaluated relative to the element identified by the xs:selector, and can be an element or an attribute node. This is the node whose value will be checked for uniqueness. Combinations of values can be specified by adding other xs:field elements within xs:unique.

```
<country xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="Constraints.xsd">
   <countryName>Belgie</countryName>
   <countryLeader>
      <firstName>Albert</firstName>
      <number>II</number>
   </countryLeader>
   <countryLeader>
      <firstName>Albert</firstName>
      <number>I</number>
   </countryLeader>
   <countryLeader>
      <firstName>Boudewijn</firstName>
      <number>I</number>
   </countryLeader>
</country>
```

Constraints (II)

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<xsd:element name="continent">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element name="continentName" type="xsd:string"/>
            <xsd:element name="country" maxOccurs="12">
                    <xsd:complexType>
                        <xsd:sequence>
                            <xsd:element    name="countryName"
                                            type="xsd:string"/>
                        </xsd:sequence>
                    </xsd:complexType>
            </xsd:element>
            <xsd:element name="countryLeader" maxOccurs="12">
                    <xsd:complexType>
                        <xsd:sequence>
                            <xsd:element name="countryLeaderName"/>
                            <xsd:element name="countryLeaderCountry"/>
                        </xsd:sequence>
                    </xsd:complexType>
            </xsd:element>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:key name="oneCountry">
            <xsd:selector xpath="country"/>
            <xsd:field xpath="countryName"/>
    </xsd:key>
    <xsd:keyref name="countryRef" refer="oneCountry">
            <xsd:selector xpath="countryLeader"/>
            <xsd:field xpath="countryLeaderCountry"/>
    </xsd:keyref>
</xsd:element>
</xsd:schema>
```

3.9.2 Key constraints

The second construct, xs:key, is similar to xs:unique except that the value has to be non null (note that xs:unique and xs:key can both be referenced)

These capabilities are almost independent of the other features in a schema. They are disconnected from the definition of the datatypes. The only point anchoring them to the schema is the place where they are defined, which establishes the scope of the uniqueness constraints.

```
<?xml version="1.0" encoding="UTF-8"?>
<continent xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="Keys.xsd">
  <continentName>Europe</continentName>
  <country>
    <countryName>Belgium</countryName>
  </country>
  <country>
     <countryName>The Netherlands</countryName>
  </country>
  <country>
      <countryName>Luxemburg</countryName>
  </country>
  <countryLeader>
      <countryLeaderName>Willem II</countryLeaderName>
      <countryLeaderCountry>The Netherlands</countryLeaderCountry>
  </countryLeader>
  <countryLeader>
      <countryLeaderName>Albert II</countryLeaderName>
      <countryLeaderCountry>Belgium</countryLeaderCountry>
  </countryLeader>
  <countryLeader>
     <countryLeaderName>Boudewijn</countryLeaderName>
      <countryLeaderCountry>Belgium</countryLeaderCountry>
 </countryLeader>
</continent>
```

Documenting Schemas

```
<xsd:element name="book">
    <xsd:annotation>
    <xsd:documentation xml:lang="en">
        Root element
    </xsd:documentation>
    <xsd:documentation xml:lang="fr">
        Element racine
    </xsd:documentation>
    <xsd:appinfo source="http://example.com/foo/">
        <bind xmlns="http://example.com/bar/">
            <class name="Book"/>
        </bind>
    </xsd:appinfo>
    </xsd:annotation>
...
```


3.10 Documenting Schemas

Perhaps the first step in writing reusable schemas is to document them. W3C XML Schema provides an alternative to XML comments (for humans) and processing instructions (for machines) that might be easier to handle for supporting tools. Human readable documentation can be defined by xs:documentation elements, while information targeted to applications should be included in xs:appinfo elements. Both elements need to be included in an xs:annotation element and accept optional xml:lang and source attributes and any content type. The source attribute is a URI reference that can be used to indicate the purpose of the comment documentation or application information.

Including schemas in other schemas

country.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:include schemaLocation="country2.xsd"/>
    <xsd:element name="country">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="countryName"  type="xsd:string"/>
                <xsd:element name="continent" type="xsd:string"/>
                <xsd:element ref="countryLeader"/>
            </xsd:sequence>
            <xsd:attribute name ="countryCode" type="xsd:string"/>
        </xsd:complexType>
    </xsd:element>
</xsd:schema>
```

country2.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:element name="countryLeader">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="title" type="xsd:string"/>
                <xsd:element name="name" type="xsd:string" />
                <xsd:element name="since" type="xsd:date"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
</xsd:schema>
```


3.11 Including schemas

Schemas can be reused in other schemas with the xs:include, which should be positioned before the other definitions. In this way, all global definitions of the included schema become available in the including schema.

One schema can include many others, and includes can be daisy chained. This allows for modularity and reusability, and becomes even more powerful when combined with namespaces (see further).

CHAPTER 4. NAMESPACES

An application may want to process documents from many parties, and these parties may want to represent their data elements in many different ways. Moreover, in a single document, an application may need to separately refer to elements with the same name that are created by different parties. How can you distinguish between such different definitions with the same name? This is where XML Namespaces come to the rescue.

Why namespaces?

- support multiple vocabularies
- support various implementations / versions / ...

XML Namespaces (V1.0/1.1) (1999/2004)

- a structured collection of names to identify element and attribute names and types
- namespace identified by URI

  `xmlns:abis="http://www.abis.be"`
- qualified names (prefix + local name)

  `<abis:Course>XML Concepts</abis:Course>`

4.1 XML Namespaces

Documents that contain multiple mark-up vocabularies, pose problems of recognition and collision. Different vocabularies can contain elements and attributes which can also be used in other vocabularies, this is what we call collision. The namespace specification describes a mechanism which will help you avoid these problems.

An **XML namespace** is a collection of names, identified by a URI reference, which are used in XML documents for naming elements, types and attributes. URI references which identify namespaces are considered identical when they are exactly the same character-for-character.

Names for XML namespaces may appear as qualified names, which contain a single colon, separating the name into a namespace prefix and a logical part. URI references can contain characters not allowed in names, so cannot be used directly as namespace prefixes. Therefore, the namespace prefix serves as a proxy for URI references.

XML Namespaces can be used to distinguish between various implementations of a standard. E.g. xmlns:env="http://schemas.xmlsoap.org/soap/envelope/" and xmlns:env="http://www.w3.org/2003/05/soap-envelope" respectively indicate SOAP V1.1 and SOAP V1.2.

No Namespace

```
<?xml version="1.0" encoding="UTF-8"?>
<enrolmentInfo>
    <enrolmentList>
        <enrolment enrolmentNumber="25752">
            <reduction>0.85</reduction>
            <course>COBOL</course>
        </enrolment>
    </enrolmentList>
    <courseList>
        <company companyNumber="38811">
            <companyName>"Abis"</companyName>
            <courseList>
                <course nr="424">Programming in COBOL</course>
                <course nr="455">Structured Programming</course>
            </courseList>
        </company>
    </courseList>
</enrolmentInfo>
```

4.1.1 No Namespace

When there is no namespace declaration, the namespace of the element does not exist. So the element's namespace is null. The element is 'in no namespace'.

EXPLICIT NAMESPACES (I)

```
<?xml version="1.0" encoding="UTF-8"?>
<enrolmentInfo>
    <enrol:enrolmentList  xmlns:enrol="http://www.abis.be/enrol/">
        <enrol:enrolment enrolmentNumber="25752">
            <enrol:reduction>0.85</enrol:reduction>
            <enrol:course>XML basics</enrol:course>
        </enrol:enrolment>
    </enrol:enrolmentList>
    <course:courseList xmlns:course="http://www.abis.be/course/">
        <course:company companyNumber="38811">
            <course:companyName>Abis</course:companyName>
            <course:courses>
                <course:course>XML basics</course:course>
                <course:course>XML Java</course:course>
            </course:courses>
        </course:company>
    </course:courseList>
</enrolmentInfo>
```


4.1.2 Explicit Namespaces

Elements or attributes can explicitly be placed in a namespace. The namespace must be declared and the prefix is used in each element.

EXPLICIT NAMESPACES (II)

```
<?xml version="1.0" encoding="UTF-8" ?>
<enrolmentInfo          xmlns:enrol="http://www.abis.be/enrol/"
                        xmlns:course="http://www.abis.be/course/">
    <enrol:enrolmentList  >
        <enrol:enrolment enrolmentNumber="25752">
            <enrol:reduction>0.85</enrol:reduction>
            <enrol:course>XML basics</enrol:course>
        </enrol:enrolment>
    </enrol:enrolmentList>
    <course:courseList >
        <course:company companyNumber="38811">
            <course:companyName>Abis</course:companyName>
            <course:courses>
                <course:course>XML basics</course:course>
                <course:course>XML Java</course:course>
            </course:courses>
        </course:company>
    </course:courseList>
</enrolmentInfo>
```


In this example, both namespaces are declared within the root Element. It is important to note that the root Element itself is not part of any namespace, because it doesn't have a prefix, and no default namespace is declared for it.

Default Namespaces

```
<?xml version="1.0" encoding="UTF-8" ?>
<enrolmentInfo      xmlns="http://www.abis.be/enrol/"
                    xmlns:course="http://www.abis.be/course/">
    <enrolmentList>
        <enrolment enrolmentNumber="25752">
            <reduction>0.85</reduction>
            <course>XML basics</course>
        </enrolment>
    </enrolmentList>
    <course:courseList >
        <course:company companyNumber="38811">
            <course:companyName>Abis</course:companyName>
            <course:courses>
                <course:course>XML basics</course:course>
                <course:course>XML Java</course:course>
            </course:courses>
        </course:company>
    </course:courseList>
</enrolmentInfo>
```


4.1.3 Default Namespaces

When a default namespace is specified, all the elements without prefix will be part of that namespace. In the example, the root element will also be part of the namespace.

Namespaces and Attributes

```
<?xml version="1.0" encoding="UTF-8" ?>
<enrolmentInfo          xmlns="http://www.abis.be/enrol/"
                        xmlns:course="http://www.abis.be/course/">
    <course:courseList >
        <course:company course:companyNumber="38811">
            <course:companyName>Abis</course:companyName>
            <course:courses>
                <course:course>XML basics</course:course>
                <course:course>XML Java</course:course>
            </course:courses>
        </course:company>
    </course:courseList>
    <enrolmentList>
        <enrolment enrolmentNumber="25752"
                        course:title="XML basics">
                <reduction>0.85</reduction>
        </enrolment>
    </enrolmentList>
</enrolmentInfo>
```


4.1.4 Namespaces and Attributes

Attributes can either belong to the same namespace of the element for which the attribute is defined, or belong to a completely different namespace than that to which the element belongs. The definition for an attribute that belongs to a namespace is the same as that for elements, with the "prefix:" notation coming before the attribute name. If no namespace prefix is provided for the attribute name, the attribute will not belong to any namespace.

It's important to notice that **attributes are never part of the default namespace**. Any attribute that doesn't have a prefix - whether or not its element is any namespace - is in no namespace at all.

Namespace Scope

```
<?xml version="1.0" encoding="UTF-8" ?>
<courseList xmlns= "www.abis.be">
    <course>
        <table xmlns="http://www.w3.org/TR/REC/REC-html140">
            <tr>
                <td>Course</td><td>Number of Days</td>
            </tr>
            <tr>
                <td>xml base</td><td>4</td>
            </tr>
            <tr>
                <td>SQL</td><td>2</td>
            </tr>
        </table>
    </course>
</courseList>
```

```
<?xml version="1.0" encoding="UTF-8" ?>
<courseList xmlns:course= "www.abis.be">
    <course:course>
        <course:name>XML basics</course:name>
    </course:course>
    <course:course>
        <course:name>XML java </course:name>
    </course:course>
    <course:course xmlns:course= "www.sas.com" >
        <course:title>XML SAS</course:title>
    </course:course>
    <course:course>
        <course:name>JAVA</course:name>
    </course:course>
</courseList>
```

4.1.5 Namespace Scope

The namespace declaration is considered to apply to the element where it is specified and to all elements within the content of that element, unless overridden by another namespace declaration with the same namespace attribute part.

A default namespace is considered to apply to the element where it is declared (if that element has no namespace prefix), and to all elements with no prefix within the content of that element. If the URI reference in a default namespace declaration is empty, then unprefixed elements in the scope of the declaration are not considered to be in any namespace. Note that a default namespace does not apply directly to attributes.

Namespaces and Schemas (I)

course.xsd

```
<xsd:schema  targetNamespace="www.abis.be"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:abis="www.abis.be">
    <xsd:element name="Course">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element ref="abis:CourseName"/>
                <xsd:element ref="abis:CourseId" />
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="CourseName" type="xsd:string" />
    <xsd:element name="CourseId" type="abis:CourseCodeType" />
    <xsd:simpleType name="CourseCodeType">
        <xsd:restriction base="xsd:string">
            <xsd:pattern value="[A-Z]{1}\d{4}"/>
        </xsd:restriction>
    </xsd:simpleType>
</xsd:schema>
```

Elements defined:

abis:CourseName abis:CourseId abis:Course

ElementTypes defined:

abis:CourseCodeType

course.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<abis:Course xmlns:abis="www.abis.be"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:schemaLocation="www.abis.be  course.xsd">
    <abis:CourseName>XML</abis:CourseName>
    <abis:CourseId>Z1006</abis:CourseId>
</abis:Course>
```


4.2 Namespaces and Schemas

4.2.1 The Target Namespace

Definitions and declarations in a schema can refer to names that may belong to other namespaces (so called source namespaces). The targetNamespace is the namespace which includes the defined elements. A schema has one target namespace and possibly many source namespaces.

In this example the target namespace also happens to be one of the source namespaces because the definitions are using other elements and types from that target namespace.

Namespaces and Schemas (II)

```
<CourseList xmlns="www.abis.be"
            xmlns:kuleuven="www.kuleuven.be"
            xmlns:ucll="www.ucll.be"
            xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xsi:schemaLocation=" www.abis.be abiscourse.xsd
                                 www.kuleuven.be kuleuvencourse.xsd
                                 www.ucll.be ucllcourse.xsd">
    <Course>
        <CourseName>XML</CourseName>
        <CourseId>Z1006</CourseId>
    </Course>
    <kuleuven:Course>
        <kuleuven:CourseName>Taalkunde</kuleuven:CourseName>
        <kuleuven:CourseNumber>2465</kuleuven:CourseNumber>
        <kuleuven:Faculty>Letteren</kuleuven:Faculty>
    </kuleuven:Course>
    <ucll:Course>
        <ucll:CourseName>Electrotechniek</ucll:CourseName>
    </ucll:Course>
</CourseList>
```


4.2.2 Using multiple namespace names from multiple schemas

An XML document may refer to names of elements from multiple namespaces that are defined in multiple schemas. To refer to and abbreviate the name of a namespace, again use xmlns declarations. To refer to the locations of the respective schemas, use the xsi:schemaLocation attribute.

Importing Schemas

1) kuleuven.xsd

```
<xsd:schema targetNamespace="www.kuleuven.be"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        elementFormDefault="qualified">
    <xsd:element name="Course">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="CourseName" type="xsd:string"/>
                <xsd:element name="CourseNumber" type="xsd:integer"/>
                <xsd:element name="Faculty" type="xsd:string"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
</xsd:schema>
```

2) courseList.xsd

```
<xsd:schema  targetNamespace="www.abis.be"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:kuleuven="www.kuleuven.be" xmlns="www.abis.be">

    <xsd:import namespace="www.kuleuven.be"
                    schemaLocation="kuleuven.xsd" />

    <xsd:element name="CourseList">
        <xsd:complexType>
            <xsd:choice maxOccurs="unbounded">
                <xsd:element ref="Course" />
                <xsd:element ref="kuleuven:Course" />
            </xsd:choice>
        </xsd:complexType>
    </xsd:element>
    ...
</xsd:schema>
```

3) courseList.xml

```
<CourseList xmlns="www.abis.be"
                xmlns:kuleuven="www.kuleuven.ac.be"
                xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                xsi:schemaLocation="www.abis.be  courseList.xsd">

    <Course>
        <CourseName>XML</CourseName>
        <CourseId>Z1006</CourseId>
    </Course>
    <kuleuven:Course>
        <kuleuven:CourseName>Europese Lit</kuleuven:CourseName>
        <kuleuven:CourseNumber>5423</kuleuven:CourseNumber>
        <kuleuven:Faculty>Letteren</kuleuven:Faculty>
    </kuleuven:Course>

</CourseList>
```


4.2.3 Importing a namespace in the schema

If another namespace than the targetNamespace is used, it must be imported using the import declaration element whose schemaLocation attribute specifies the location of the file that contains the schema.

APPENDIX A. EXERCISES

A.1 XML design

Consider the following data. As you already guessed, the columns contain a person's name, first name, function, company and company's town.

SMITHS	JAN	TRAINING CONSULT	ABIS N.V.	LEUVEN
TAVERNIER	PETER	PROGRAMMER	ABIS N.V.	LEUVEN
DE KEYSER	ANN	PROGRAMMER	ABIS N.V.	LEUVEN
NIEHOF	RUUD	EDP-MANAGER	COMPUTRAIN	ROTTERDAM
VAN HEIJKOOP	GERT	ANALYST	COMPUTRAIN	ROTTERDAM
PEREZ	MARIA	MANAGER	DATAWISHES N.V.	BRUSSEL
LIVIER	F.	ANALYST	DATAWISHES N.V.	BRUSSEL
LOOSE	K.	CONSULTANT	-	-
BENOIT	PHILIP	SOFTWARE ENGINEER	LOC COMPUTER CORP.	BRUSSEL

1. Create a xml file containing the entire information as a list of persons. Save the results in PersonList.xml

2. Create an xml file containing the same information structured as a list of companies. Save the results in CompanyList.xml

3. Create an xml file Relational.xml containing the same information on a relational way. Define first all the company information and then all the person information. Each Company is a new element in a CompanyList element, each Person is a new element in the PersonList element. Use elements to represent the data, and attributes for the metadata (e.g. for expressing the relation between a person and his employer.

Note: all three solutions should contain the entire information.

A.2 Schemas

1. Copy the XML document PersonList.xml to a new XML file called SchemaPersonList.xml.

   Write a schema to validate this file using the Russian Doll technique.

2. Copy the XML document Relational.xml to a new XML file called SchemaRelational.xml.

   Create a schema to validate this file. Make sure that:

   - Name becomes a global element.
   - The complex type inside the Company element becomes a global complex type.

3. Change the solution of the previous exercise, such that the Name element cannot just contain 1 firstName and 1 lastName, but that a user can choose between

   - 1-3 first names followed by a last name, or
   - a last name followed by 1-3 first names.

   Example:

```
<Name>
   <FirstName>JAN</FirstName>
   <FirstName>PETER</FirstName>
   <FirstName>WILLIAM</FirstName>
   <LastName>SMITHS</LastName>
</Name>

<Name>
   <LastName>DE KEYSER</LastName>
   <FirstName>ANN</FirstName>
   <FirstName>SOPHIE</FirstName>
</Name>
```

4. (optional) Copy the file CompanyList.xml to IncludeCompanyList.xml. Copy the SimpleTypeCompanyList.xsd schema to IncludeCompanyList.xsd, and adapt this so that the postcode type is included from a second schema (postcode.xsd).

A.3 Namespaces

1. Put all the elements of previous schema in a namespace called “www.abis.be”. Try to validate the corresponding XML file again.

2. Move the Name element and the simple type for the zipcode (as created above) to a new schema called abisbase.xsd. The namespace of this schema should be “www.abisbase.be”.

   Import this xsd back into the original one, and validate the corresponding XML file.

APPENDIX B. SOLUTIONS

B.1 XML design

Consider the following data. As you already guessed, the columns contain a person's name, first name, function, company and company's town.

```
SMITHS          JAN         TRAINING CONSULT     ABIS N.V.            LEUVEN
TAVERNIER       PETER       PROGRAMMER           ABIS N.V.            LEUVEN
DE KEYSER       ANN         PROGRAMMER           ABIS N.V.            LEUVEN
NIEHOF          RUUD        EDP-MANAGER          COMPUTRAIN           ROTTERDAM
VAN HEIJKOOP    GERT        ANALYST              COMPUTRAIN           ROTTERDAM
PEREZ           MARIA       MANAGER              DATAWISHES N.V.      BRUSSEL
LIVIER          F.          ANALYST              DATAWISHES N.V.      BRUSSEL
LOOSE           K.          CONSULTANT           -                    -
BENOIT          PHILIP      SOFTWARE ENGINEER    LOC COMPUTER CORP.   BRUSSEL
```

1. Create a XML file containing the information on a person-structured base. Save the results in PersonList.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<PersonList>
  <Person>
    <Name>
       <FirstName>JAN</FirstName>
       <LastName>SMITHS</LastName>
    </Name>
    <Function>TRAINING CONSULT</Function>
    <Company>
      <CompanyName>ABIS N.V.</CompanyName>
      <CompanyAddress>LEUVEN</CompanyAddress>
    </Company>
  </Person>
  <Person>
    <Name>
       <FirstName>PETER</FirstName>
       <LastName>TAVERNIER</LastName>
    </Name>
    <Function>PROGRAMMER</Function>
    <Company>
      <CompanyName>ABIS N.V.</CompanyName>
      <CompanyAddress>LEUVEN</CompanyAddress>
    </Company>
  </Person>
  ...
</PersonList>
```

2. Create an XML file containing the same information on a company-structured base. Save the results in CompanyList.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<CompanyList>
  <Company>
     <CompanyName/>
     <CompanyAddress/>
     <PersonList>
        <Person>
          <Name>
            <FirstName>K.</FirstName>
            <LastName>LOOSE</LastName>
          </Name>
          <Function>CONSULTANT</Function>
       </Person>
     </PersonList>
   </Company>
   <Company>
     <CompanyName>ABIS N.V.</CompanyName>
     <CompanyAddress>LEUVEN</CompanyAddress>
     <PersonList>
       <Person>
         <Name>
            <FirstName>JAN</FirstName>
            <LastName>SMITHS</LastName>
         </Name>
         <Function>TRAINING CONSULT</Function>
       </Person>
       <Person>
         <Name>
            <FirstName>ANN</FirstName>
            <LastName>DE KEYSER</LastName>
         </Name>
         <Function>PROGRAMMER</Function>
       </Person>
       ...
     </PersonList>
   </Company>
   <Company>
     <CompanyName>COMPUTRAIN</CompanyName>
     <CompanyAddress>ROTTERDAM</CompanyAddress>
     <PersonList>
        <Person>
           <Name>
              <FirstName>GERT</FirstName>
              <LastName>VAN HEIJKOOP</LastName>
           </Name>
           <Function>ANALYST</Function>
        </Person>
        <Person>
           <Name>
              <FirstName>RUUD</FirstName>
              <LastName>NIEHOF</LastName>
           </Name>
           <Function>EDP-MANAGER</Function>
        </Person>
     </PersonList>
   </Company>
</CompanyList>
```

3. Create an XML file Relational.xml containing the same information in a relational way. Define first all the company information and then all the person information. Each Company is a new element in a CompanyList element, each Person is a new element in the PersonList element. Use elements to represent the data, and attributes for the metadata (e.g. for expressing the relation between a person and his employer.

```
<CompaniesAndPersons>
   <CompanyList>
      <Company Nr="1">
         <Name>ABIS</Name>
         <Address>LEUVEN</Address>
      </Company>
      <Company Nr="2">
         <Name>COMPUTRAIN</Name>
         <Address>ROTTERDAM</Address>
      </Company>
   </CompanyList>
   <PersonList>
      <Person Nr="0">
         <FirstName>K.</FirstName>
         <LastName>LOOZE</LastName>
      </Person>
      <Person Nr="1" Employer="1">
         <FirstName>JAN</FirstName>
         <LastName>SMITHS</LastName>
      </Person>
      <Person Nr="2" Employer="1">
         <FirstName>ANN</FirstName>
         <LastName>DE KEYSER</LastName>
      </Person>
      <Person Nr="3" Employer="1">
         <FirstName>PETER</FirstName>
         <LastName>TAVERNIER</LastName>
      </Person>
      <Person Nr="4" Employer="2">
         <FirstName>GERT</FirstName>
         <LastName>VAN HEIJKOOP</LastName>
      </Person>
      <Person Nr="5" Employer="2">
         <FirstName>RUUD</FirstName>
         <LastName>NIEHOF</LastName>
      </Person>
   </PersonList>
</CompaniesAndPersons>
```

```
<CP>
   <CompanyList>
     <Company Nr="1" Name="ABIS" place="LEUVEN"/>
     <Company Nr="2" Name="COMPUTRAIN" place="ROTTERDAM"/>
   </CompanyList>
   <PersonList>
     <Person Nr="1" Name="K." LastName="LOOZE" Function="CONSULTANT"/>
     <Person Nr="1" Name="JAN" LastName="SMITHS" Function="TRAINER CONSULTANT"
          Employer="1"/>
     <Person Nr="2" Name="ANN" LastName="DE KEYSER" Function="PROGRAMMER"
          Employer="1"/>
     <Person Nr="3" Name="PETER" LastName="TAVERNIER" Function="PROGRAMMER"
          Employer="1"/>
     <Person Nr="4" Name="GERT" LastName="VAN HEIJKOOP" Function="ANALYST"
          Employer="2"/>
     <Person Nr="5" Name="RUUD" LastName="NIEHOF" Function="EDP-MANAGER"
          Employer="2"/>
   </PersonList>
</CP>
```

B.2 Schemas

1. Copy the file CompanyList.xml to RussianDollCompanyList.xml. Write the schema to validate this file using the Russian Doll technique.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" >
 <xs:element name="CompanyList">
  <xs:complexType>
   <xs:sequence>
    <xs:element name="Company" maxOccurs="unbounded">
     <xs:complexType>
      <xs:sequence>
       <xs:element name="CompanyName" type="xs:string"/>
       <xs:element name="CompanyAddress" type="xs:string"/>
       <xs:element name="PersonList">
        <xs:complexType>
         <xs:sequence>
          <xs:element name="Person" maxOccurs="unbounded">
           <xs:complexType>
            <xs:sequence>
             <xs:element name="Name">
              <xs:complexType>
               <xs:sequence>
                <xs:element name="FirstName" type="xs:string"/>
                <xs:element name="LastName" type="xs:string"/>
               </xs:sequence>
              </xs:complexType>
             </xs:element>
             <xs:element name="Function" type="xs:string"/>
            </xs:sequence>
           </xs:complexType>
          </xs:element>
         </xs:sequence>
        </xs:complexType>
       </xs:element>
      </xs:sequence>
     </xs:complexType>
    </xs:element>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

2. Copy the file CompanyList.xml to SimpleTypeCompanyList.xml. Add a new attribute postcode on the element that stores the name of the companylocation. Try now to write a schema and check the pattern of the postcode.

```
<CompanyList xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:noNamespaceSchemaLocation="SimpleTypecompanyList.xsd">
 <Company>
  <CompanyName/>
   <CompanyAddress/>
   <PersonList>
    <Person>
     <Name>
      <FirstName>K.</FirstName>
      <LastName>LOOSE</LastName>
     </Name>
     <Function>CONSULTANT</Function>
    </Person>
   </PersonList>
  </Company>
  <Company>
   <CompanyName>ABIS N.V.</CompanyName>
   <CompanyAddress postcode="3000">LEUVEN</CompanyAddress>
   <PersonList>
    <Person>
     <Name>
      <FirstName>JAN</FirstName>
      <LastName>SMITHS</LastName>
     </Name>
     <Function>TRAINING CONSULT</Function>
    </Person>
    <Person>
     <Name>
      <FirstName>ANN</FirstName>
      <LastName>DE KEYSER</LastName>
     </Name>
     <Function>PROGRAMMER</Function>
    </Person>
    <Person>
     <Name>
      <FirstName>PETER</FirstName>
      <LastName>TAVERNIER</LastName>
     </Name>
     <Function>PROGRAMMER</Function>
    </Person>
   </PersonList>
  </Company>
  <Company>
   <CompanyName>COMPUTRAIN</CompanyName>
   <CompanyAddress postcode="2000 AC">ROTTERDAM</CompanyAddress>
   <PersonList>
    <Person>
     <Name>
      <FirstName>GERT</FirstName>
      <LastName>VAN HEIJKOOP</LastName>
     </Name>
     <Function>ANALYST</Function>
    </Person>
    <Person>
     <Name>
      <FirstName>RUUD</FirstName>
      <LastName>NIEHOF</LastName>
     </Name>
     <Function>EDP-MANAGER</Function>
    </Person>
   </PersonList>
  </Company>
</CompanyList>
```

```
<?xml version="1.0" encoding="UTF-8"?>
xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:simpleType name="postcodeType">
    <xs:restriction base="xs:string">
      <xs:pattern value="[1-9]{1}[0-9]{3}"/>
      <xs:pattern value="[1-9]{1}[0-9]{3} [A-Z]{2}"/>
   </xs:restriction>
 </xs:simpleType>
 <xs:element name="CompanyList">
  <xs:complexType>
   <xs:sequence>
    <xs:element name="Company" maxOccurs="unbounded">
     <xs:complexType>
      <xs:sequence>
       <xs:element name="CompanyName" type="xs:string"/>
       <xs:element name="CompanyAddress">
        <xs:complexType>
         <xs:simpleContent>
          <xs:extension base="xs:string">
           <xs:attribute name="postcode" type="postcodeType"/>
          </xs:extension>
         </xs:simpleContent>
        </xs:complexType>
       </xs:element>
       <xs:element name="PersonList">
        <xs:complexType>
         <xs:sequence>
          <xs:element name="Person" maxOccurs="unbounded">
           <xs:complexType>
            <xs:sequence>
             <xs:element name="Name">
              <xs:complexType>
               <xs:sequence>
                <xs:element name="FirstName" type="xs:string"/>
                <xs:element name="LastName" type="xs:string"/>
               </xs:sequence>
              </xs:complexType>
             </xs:element>
             <xs:element name="Function" type="xs:string"/>
            </xs:sequence>
           </xs:complexType>
          </xs:element>
         </xs:sequence>
        </xs:complexType>
       </xs:element>
      </xs:sequence>
     </xs:complexType>
    </xs:element>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

B.3 Namespaces

1. Copy the file SimpleTypeCompanyList.xml to NamespaceCompanyList.xml. Put all the elements in a namespace “www.abis.be”. Copy the schema SimpleTypeCompanyList.xsd to NamespaceCompanyList.xsd and try to validate NamespaceCompanyList.xml

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<CompanyList xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="NamespacecompanyList.xsd"
xmlns="www.abis.be"
>
 <Company>
  <CompanyName>ABIS N.V.</CompanyName>
    <CompanyAddress postcode="3000">LEUVEN</CompanyAddress>
     <PersonList>
      <Person>
       <Name>
        <FirstName>JAN</FirstName>
        <LastName>SMITHS</LastName>
       </Name>
       <Function>TRAINING CONSULT</Function>
       </Person>
       <Person>
     ...
 </Company>
  ...
</CompanyList>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="www.abis.be" xmlns:abis="www.abis.be"
   xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
 <xs:simpleType name="postcodeType">
       ...
 </xs:simpleType>
 <xs:element name="CompanyList">
  <xs:complexType>
   <xs:sequence>
    <xs:element name="Company" maxOccurs="unbounded">
     <xs:complexType>
      <xs:sequence>
       <xs:element name="CompanyName" type="xs:string"/>
       <xs:element name="CompanyAddress">
        <xs:complexType>
         <xs:simpleContent>
          <xs:extension base="xs:string">
           <xs:attribute name="postcode" type="abis:postcodeType"/>
          </xs:extension>
         </xs:simpleContent>
        </xs:complexType>
       </xs:element>
       <xs:element name="PersonList">
           ...
       </xs:element>
      </xs:sequence>
     </xs:complexType>
    </xs:element>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

2. Copy the file NamespaceCompanyList.xml to ImportNamespaceCompanyList.xml and copy the file NamespaceCompanyList.xsd to ImportNamespaceCompanyList.xsd. Create a new file ImportNamespacePost.xsd defining the namespace “www.post.be”. Delete the simple type PostcodeType in ImportNamepaceCompanyList.xsd and put it in the ImportNamespacePost.xsd. Import the schema ImportNamespacePost.xsd into ImportNamespaceCompanyList.xsd and try to validate again.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="www.abis.be" xmlns:post="www.post.be"
   xmlns:abis="www.abis.be"
   xmlns:xs="http://www.w3.org/2001/XMLSchema"
   elementFormDefault="qualified">
 <xs:import namespace="www.post.be"
                        schemaLocation="ImportNamespacePost.xsd"/>
 <xs:element name="CompanyList">
  <xs:complexType>
   <xs:sequence>
    <xs:element name="Company" maxOccurs="unbounded">
     <xs:complexType>
      <xs:sequence>
       <xs:element name="CompanyName" type="xs:string"/>
       <xs:element name="CompanyAddress">
        <xs:complexType>
         <xs:simpleContent>
          <xs:extension base="xs:string">
           <xs:attribute name="postcode" type="post:postcodeType"/>
          </xs:extension>
         </xs:simpleContent>
        </xs:complexType>
       </xs:element>
       <xs:element name="PersonList">
        <xs:complexType>
         <xs:sequence>
          <xs:element name="Person" maxOccurs="unbounded">
           <xs:complexType>
            <xs:sequence>
             <xs:element name="Name">
              <xs:complexType>
               <xs:sequence>
                <xs:element name="FirstName" type="xs:string"/>
                <xs:element name="LastName" type="xs:string"/>
               </xs:sequence>
              </xs:complexType>
             </xs:element>
             <xs:element name="Function" type="xs:string"/>
            </xs:sequence>
           </xs:complexType>
          </xs:element>
          </xs:sequence>
         </xs:complexType>
        </xs:element>
       </xs:sequence>
      </xs:complexType>
     </xs:element>
    </xs:sequence>
   </xs:complexType>
 </xs:element>
</xs:schema>
```