# CHAPTER 3

**Data Mapping and Exchange:** Metadata; Data representation and encoding; XML, DTD, XML schemas

## 3.1 Data Mapping and Exchange

**Data Mapping:**

In computing and data management, **data mapping** is the process of creating data element mappings between two distinct data models

In metadata, the term **data element** is an atomic unit of data that has precise meaning or precise semantics. A data element has:

1. An identification such as a data element name
2. A clear data element definition
3. One or more representation terms
4. Optional enumerated values Code (metadata)
5. A list of synonyms to data elements

A **data model** organizes data elements and standardizes how the data elements relate to one another. Since data elements document real life people, places and things and the events between them, the data model represents reality.

***Why Need Data Mapping***

Data mapping is used as a first step for a wide **variety of data integration** tasks including:

- Data transformation or data mediation between data source and destination
- Identification of data relationships
- Discovery of hidden sensitive data
- Consolidation of multiple databases into a single data base and identifying redundant columns of data for consolidation or elimination

**Data integration** involves combining data residing in different sources and providing users with a unified view of these data. This process becomes significant in a variety of situations, which include both commercial (when two similar companies need to merge their databases) and scientific (combining research results from different bioinformatics repositories) domains.

**Data Exchange:**

**Data exchange** is the process of taking data structured under a *source* schema and actually transforming it into data structured under a *target* schema, so that the target data is an accurate representation of the source data.

***Data Exchange Format***

Often there are a few dozen different source and target schema (proprietary data formats) in some specific domain. Often people develop a **exchange format** or **interchange format** for some single domain, and then write a few dozen different routines to (indirectly) translate each and every source schema to each and every target schema by using the interchange format as an intermediate step. That requires a lot less work than writing and debugging the hundreds of different routines that would be required to directly translate each and every source schema directly to each and every target schema.

**Example**

**Standard Interchange Format** for geospatial data, **Data Interchange Format** for spreadsheet data, **Quicken Interchange Format** for financial data etc.

***Data Exchange Language***

A data exchange language is a language that is domain-independent and can be used for any kind of data. The term is also applied to any **file format** that can be read by more than one program, including proprietary formats such as **Microsoft Office** documents. Some of the **formal languages** are better suited for this task than others, since their specification is driven by a formal process instead of a particular software implementation needs.

**Example**

Resource Description Framework (RDF), JSON (JavaScript Object Notation), Rebol, YAML, Gellish, XML

## 3.2 Metadata

Metadata (metacontent) is defined as the data providing information about one or more aspects of the data, such as:

- Means of creation of the data
- Purpose of the data
- Time and date of creation
- Creator or author of the data
- Location on a computer network where the data were created

**Example**

Digital image may include metadata that describe the picture size, the color depth, the image resolution, time and date of image creation.

A text document's metadata may contain information about how long the document is, who the author is, when the document was written, and a short summary of the document.

### 3.3 Introduction to XML

- XML stands for Extensible Markup Language
- XML is a markup language much like HTML
- XML was designed to describe data, not to display data
- XML tags are not predefined. You must define your own tags
- XML is designed to be self-descriptive
- XML is a W3C Recommendation
- XML does not DO anything

**Difference Between XML and HTML**

- XML is not a replacement for HTML; XML is a complement to HTML.
- ***XML is a software- and hardware-independent tool for carrying information.***

- XML was designed to describe data, with focus on what data is
- HTML was designed to display data, with focus on how data looks

**XML Does Not DO Anything:**

The following example is a note to Tove, from Jani, stored as XML:

```
<note>
 <to>Tove</to>
 <from>Jani</from>
 <heading>Reminder</heading>
 <body>Don'tforget me this weekend!</body>
</note>
```

The note above is quite self descriptive. It has sender and receiver information, it also has a heading and a message body.

But still, this XML document does not DO anything. It is just information wrapped in tags. ***Someone must write a piece of software to send, receive or display it.***

***How Can XML be used?***

XML is used in many aspects of web development, often to simplify data storage and sharing.

1. XML Separates Data from HTML
2. XML Simplifies Data Sharing
3. XML Simplifies Data Transport
4. XML Simplifies Platform Changes

***Internet Languages Written in XML***

Several Internet languages are written in XML. Here are some examples: XHTML, XML Schema, SVG, WSDL and RSS

## 3.4 XML Tree

***XML Documents Form a Tree Structure***

- XML documents must contain a **root element**. This element is "the parent" of all other elements.
- The elements in an XML document form a document tree. The tree starts at the root and branches to the lowest level of the tree.
- All elements can have sub elements (child elements):

-
  ```
  <root>
    <child>
      <subchild>.... </subchild>
    </child>
  </root>
  ```
- The terms parent, child, and sibling are used to describe the relationships between elements. Parent elements have children. Children on the same level are called siblings (brothers or sisters).
- All elements can have text content and attributes (just like in HTML).

***XML element***

- An XML document contains XML Elements.
- An XML element is everything from (including) the element's start tag to (including) the element's end tag.
- An element can contain:

- other elements
- text
- attributes
- or a mix of all of the above...

***Empty XML Elements***

An alternative syntax can be used for XML elements with no content: Instead of writing a book element (with no content) like this:

<book></book>

It can be written like this:

<book />

This sort of element syntax is called self-closing.

***XML Naming Rules***

XML elements must follow these naming rules:

- Names can contain letters, numbers, and other characters
- Names cannot start with a number or punctuation character
- Names cannot start with the letters xml (or XML, or Xml, etc)
- Names cannot contain spaces

Any name can be used, no words are reserved.

***XML Attributes***

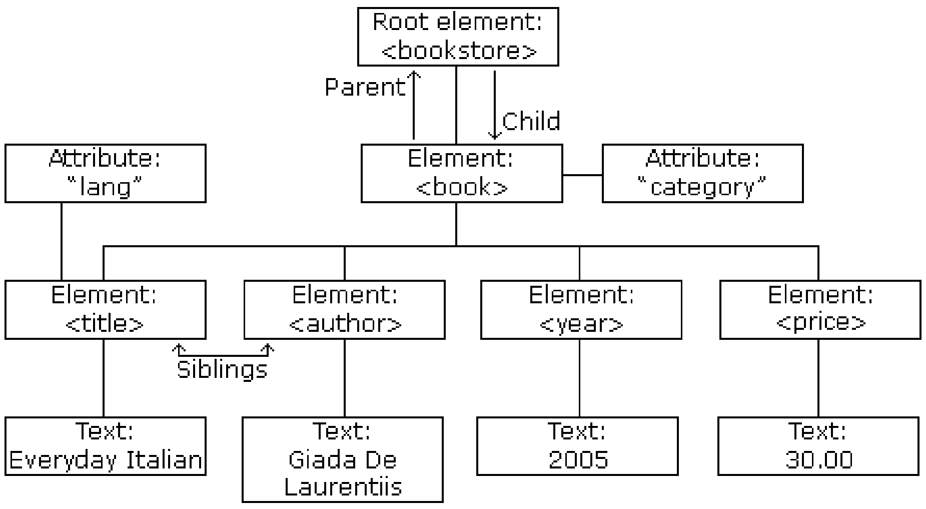
XML elements can have attributes, just like HTML. Attributes provide additional information about an element. Attributes often provide information that is not a part of the data. In the example below, the file type is irrelevant to the data, but can be important to the software that wants to manipulate the element:

<file type="gif">computer.gif</file>

<img src="computer.gif">

<a href="demo.asp">

Example:



- The image above represents one book in the XML below:

-
  ```
  <bookstore>
    <book category="COOKING">
      <title lang="en">Everyday Italian</title>
      <author>Giada De Laurentiis</author>
      <year>2005</year>
      <price>30.00</price>
    </book>
    <book category="Programming Language">
      <title lang="en">XML</title>
      <author>Dr.J.VijiPriya </author>
      <year>2014</year>
      <price>150</price>
    </book>
   </bookstore>
  ```
- The root element in the example is <bookstore>. All <book> elements in the document are contained within <bookstore>.
- The <book> element has 4 children: <title>,< author>, <year>, <price>.

## 3.5 XML Syntax Rules

The syntax rules of XML are very simple and logical.

**1. All XML Elements Must Have a Closing Tag**

- In HTML, some elements do not have to have a closing tag:

-
  ```
  <p>This is a paragraph.
  <br>
  ```
- In XML, it is illegal to omit the closing tag. All elements **must** have a closing tag:

-
  ```
  <p>This is a paragraph.</p>
  <br />
  ```

**2. XML Tags are Case Sensitive**

- XML tags are case sensitive. The tag <Letter> is different from the tag <letter>.
- Opening and closing tags must be written with the same case:

-
  ```
  <Message>This is incorrect</message>
   <message>This is correct</message>
  ```

**3. XML Elements Must be Properly Nested**

- In HTML, you might see improperly nested elements:
- <b><i>This text is bold and italic</b></i>
- In XML, all elements **must** be properly nested within each other:
- <b><i>This text is bold and italic</i></b>
- In the example above, "Properly nested" simply means that since the <i> element is opened inside the <b> element, it must be closed inside the <b> element.

**4. XML Documents Must Have a Root Element**

- XML documents must contain one element that is the **parent** of all other elements. This element is called the **root** element.

-
  ```
  <root>
    <child>
      <subchild>.....</subchild>
    </child>
  </root>
  ```

5. **XML Attribute Values Must be Quoted**

- XML elements can have attributes in name/value pairs just like in HTML.
- In XML, the attribute values must always be quoted.

-
  ```
  <note date=12/11/2007>
    <to>Tove</to>
    <from>Jani</from>
  </note>
  ```

-
  ```
  <note date="12/11/2007">
    <to>Tove</to>
    <from>Jani</from>
  </note>
  ```
- The error in the first document is that the date attribute in the note element is not quoted.

6. **Entity References**

- Some characters have a special meaning in XML.
- If you place a character like "<" inside an XML element, it will generate an error because the parser interprets it as the start of a new element.
- This will generate an XML error:
- `<message>if salary < 1000 then</message>`
- To avoid this error, replace the "<" character with an **entity reference**:
- `<message>if salary &lt; 1000 then</message>`
- There are 5 predefined entity references in XML:

| | | |
|---|---|---|
| &lt; | < | less than |
| &gt; | > | greater than |
| &amp; | & | ampersand |
| &apos; | ' | Apostrophe |
| &quot; | " | quotation mark |

**Note:** Only the characters "<" and "&" are strictly illegal in XML. The greater than character is legal, but it is a good habit to replace it.

7. **Comments in XML**

- The syntax for writing comments in XML is similar to that of HTML.
- `<!-- This is a comment -->`

8. **White-space is preserved in XML**

- HTML truncates multiple white-space characters to one single white-space:

| | |
|---|---|
| HTML: | Hello        Tove |
| Output: | Hello Tove |

- With XML, the white-space in a document is not truncated.

**9. XML Stores New Line as LF**

- Windows applications store a new line as: carriage return and line feed (CR+LF).
- Unix and Mac OSX uses LF.
- Old Mac systems uses CR.
- XML stores a new line as LF.

**10. Well Formed XML**

- XML documents that conform to the syntax rules above are said to be "Well Formed" XML documents.

## 3.6 XML Declaration or XML Prolog

**Example 1: A Sample XML Document (example1.xml)**

```
<?xml version="1.0" encoding="UTF-8"?>
<document>
   <heading> Hello From XML    </heading>
   <message> This is an XML document! </message>
</document>
```

Like all XML documents, this one starts with an XML declaration, <?xml version="1.0" encoding="UTF-8"?>. This XML declaration indicates that we're using XML version 1.0, and using the UTF-8 character encoding,

This XML declaration, <?xml?>, uses two attributes, **version** and **encoding**, to set the version of XML and the character set we're using. Next we create a new XML element named <document>. XML tags th emselves always start with < and end with >.Then we store other elements in our <document> element, or text data, as we wish.

***Character Encodings: ASCII, Unicode, and UCS***

The characters in an XML document are stored using numeric codes. That can be an issue, because different character sets use different codes, which means an XML processor might have problems trying to read an XML document that uses a character set called a character encoding

***Which character sets are supported in XML? ASCII? Unicode? UCS?***

There are many character encodings that an XML processor can support, such as the following:

- US-ASCII— U.S. ASCII
- UTF-8— Compressed Unicode
- UTF-16— Compressed UCS
- ISO-10646-UCS-2— Unicode
- ISO-10646-UCS-4— UCS
- ISO-2022-JP— Japanese
- ISO-2022-CN— Chinese
- ISO-8859-5— ASCII and Cyrillic

## 3.7 Cascading Style Sheets (CSS)

One of the most popular reasons for using style sheets with XML is that you store your data in an XML document, and specify how to display that data using a separate document, the style sheet shown in Figure 3.1. By separating the presentation details from the data, you can change the entire presentation with a few changes in the style sheet, instead of making multiple changes in your data itself.

There's plenty of support for working with XML documents and style sheets in both **Internet Explorer and Netscape Navigator**. There are **two kinds of style sheets** you can use with XML document:

1. **Cascading Style Sheets (CSS)**, which you can also use with HTML documents
2. **Extensible Style sheet Language** style sheets **(XSL)**, designed to be used only with XML documents

***Example 2: An XML Document Using a Style Sheet (example2.xml)***

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/css" href="css1.css"?>
<document>
   <heading>  Hello From XML    </heading>
   <message> This is an XML document!   </message>
</document>
```

***A CSS Style Sheet (css1.css)***

```
heading {display: block; font-size: 24pt; color: #ff0000; text-align: center}
message {display: block; font-size: 18pt; color: #0000ff; text-align: center}
```

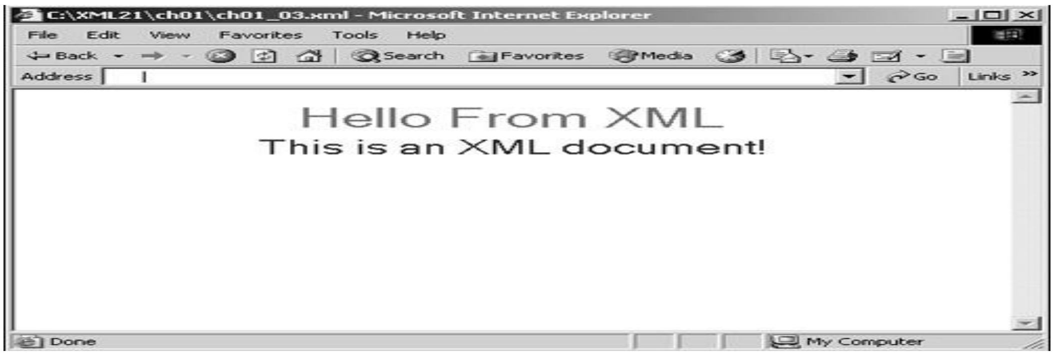

C:\XML21\ch01\ch01_03.xml - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address
Go Links
Done
My Computer
Hello From XML
This is an XML document!

**Figure 3.1: Viewing an XML document in Internet Explorer viewing**

***Extracting Data from an XML Document***

You can extract data from an XML document yourself using Scripting language like java script and to work with that data, rather than simply telling a browser how to display it. For example, suppose you want to extract the text from our <heading> element shown in Figure 3.2:

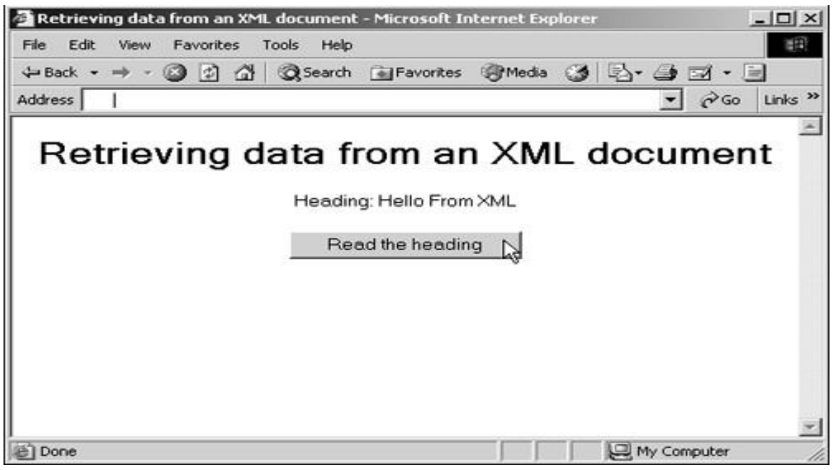
**Example 3: (example3.xml)**

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/css" href="css1.css"?>
<document>
   <heading>
      Hello From XML
   </heading>
   <message>
      This is an XML document!
    </message>
</document>
```

***Extracting Data from an XML Document Using JavaScript (example3.html)***

```
<HTML>
   <HEAD>
       <TITLE>
          Retrieving data from an XML document
       </TITLE>

       <XML ID="firstXML" SRC="example3.xml"></XML>

       <SCRIPT LANGUAGE="JavaScript">
          function getData()
          {
              xmldoc= document.all("firstXML").XMLDocument;

              nodeDoc = xmldoc.documentElement;
              nodeHeading = nodeDoc.firstChild;

              outputMessage = "Heading: " +
                   nodeHeading.firstChild.nodeValue;
              message.innerHTML=outputMessage;
          }
       </SCRIPT>
   </HEAD>

   <BODY>
      <CENTER>
         <H1>
             Retrieving data from an XML document
         </H1>

         <DIV ID="message"></DIV>
         <P>
         <INPUT TYPE="BUTTON" VALUE="Read the heading" ONCLICK="getData()">
      </CENTER>
   </BODY>
</HTML>
```

Extracting data from an XML document in Internet Explorer


Retrieving data from an XML document - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address
Go Links
Done
My Computer

Retrieving data from an XML document

Heading: Hello From XML

Read the heading

**Figure 3.2. Extracting data from an XML document in Internet Explorer**

## 3.8 XML DTD

**How does an XML processor check your document?**

There are two main checks that XML processors make:

1. Checking that your document is well-formed
2. Checking that it's valid

**Why need XML Validator**

- Use our XML validator to syntax-check your XML.
- Errors in XML documents will stop your XML applications.
- The W3C XML specification states that a program should stop processing an XML document if it finds an error.
- HTML browsers will display HTML documents with errors (like missing end tags). **With XML, errors are not allowed.**

**Creating Valid XML Documents**

An XML processor will usually check whether your XML document is well-formed, but only some are also capable of checking whether it's valid that is syntax in either a Document Type Definition (DTD) or an XML schema.

**XML DTD:**

- An XML document with correct syntax is called "Well Formed".
- An XML document validated against a DTD is "Well Formed" and "Valid".
- The purpose of a DTD is to **define the structure of an XML document**. It defines the **structure with a list of legal elements:**

As an example, you can see how you add a DTD to our XML document. DTDs can be separate documents, or they can be built into an XML document as we've done here using a **special element named <!DOCTYPE>.**

***An XML Document with a DTD (example4.xml)***

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/css" href="css1.css"?>
<!DOCTYPE document
 [
   <!ELEMENT document (heading, message)>
   <!ELEMENT heading (#PCDATA)>
   <!ELEMENT message (#PCDATA)>
]>
<document>
   <heading>
      Hello From XML
   </heading>
   <message>
      This is an XML document!
   </message>
</document>
```

***Valid XML Document with DTD (example.5.xml)***

A "Valid" XML document is a "Well Formed" XML document, which also conforms to the rules of a DTD:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE note SYSTEM "note.dtd">
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>
```

The DOCTYPE declaration, in the example above, is a reference to an external DTD file **"Note.dtd"**.

The content of the file is shown in below:

**Note.dtd**

```
<!DOCTYPE note
[
<!ELEMENT note (to,from,heading,body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
]>
```

The DTD above is interpreted like this:

- !DOCTYPE note defines that the root element of the document is note
- !ELEMENT note defines that the note element contains four elements: "to, from, heading, body"
- !ELEMENT to defines the to element to be of type "#PCDATA"
- !ELEMENT from defines the from element to be of type "#PCDATA"
- !ELEMENT heading defines the heading element to be of type "#PCDATA"
- !ELEMENT body defines the body element to be of type "#PCDATA"

**Note**

#PCDATA means **parse-able text data.**

**When NOT to Use a Document Definition?**

When you are working with small XML files, creating document definitions may be a waste of time.

## 3.9 XML Schema

- Another way of validating XML documents: using XML schemas.
- The XML Schema language is also referred to as **XML Schema Definition (XSD),** describes the structure of an XML document.
- defines the legal building blocks (elements and attributes) of an XML document like DTD.
- defines which elements are child elements
- defines the number and order of child elements
- defines whether an element is empty or can include text
- defines data types for elements and attributes
- defines default and fixed values for elements and attributes

It is believed that XML Schemas will be used in most Web applications as a replacement for DTDs.

Here are some reasons:

- XML Schemas are extensible to future additions
- XML Schemas are richer and more powerful than DTDs
- XML Schemas are written in XML
- XML Schemas support data types and namespaces

**Creating XML Schemas by Using XML Schema-Creation Tools**

Software tools that can generate XML schemas for you. A growing number of XML schema-creation tools are:

- HiT Software- online automatic XML schema generator and DTD to XML schema converter. You just let it upload a document, and it creates an XML schema for free.
- xmlArchitect- XML editor for creating schemas.
- XMLspy- XMLspy is a product family of tools that aid in the creation of XML schemas.
- XML Ray-This tool provides support for XML schemas and has an integrated online XML tutorial system.
- Microsoft Visual Studio .NET-Visual Studio .NET can also generate XML schemas for you automatically.

**XSD Simple Element**

A simple element is an XML element that can contain only text. It cannot contain any other elements or attributes. The text can be of many different types. It can be one of the types included in the XML Schema definition (boolean, string, date, etc.), or it can be a custom type that you can define yourself. You can also add restrictions (facets) to a data type in order to limit its content, or you can require the data to match a specific pattern.

***The syntax for defining a simple element is:***

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

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

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

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

**Example**

Here are some XML elements:

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

And here are the corresponding simple element definitions:

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

***Default and Fixed Values for Simple Elements***

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

1. A default value is automatically assigned to the element when no other value is specified. In the following example the default value is "red":

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

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

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

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

**XSD Attributes**

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

***The syntax for defining an attribute is:***

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

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

**Example**

Here is an XML element with an attribute:

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

And here is the corresponding attribute definition in XML schema:

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

***Default and Fixed Values for Attributes***

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

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

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

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

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

***Optional and Required Attributes***

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

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

**XSD Complex Elements**

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

There are four kinds of complex elements:

1. empty elements
2. elements that contain only other elements
3. elements that contain only text
4. elements that contain both other elements and text

**Note:** Each of these elements may contain attributes as well!

**Examples of Complex Elements**

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

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

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

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

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

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

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

```
<description>

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

</description>
```

**XSD Elements Only**

**How to Define a Complex Element using XML Scheme**

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

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

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

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

If you use the method described above, only the "employee" element can use the specified complex type. Note that the child elements, "firstname" and "lastname", are surrounded by the <sequence> indicator. This means that the child elements must appear in the same order as they are declared. The "employee" element can have a type attribute that refers to the name of the complex type to use:

**XSD Empty Elements**

An empty complex element cannot have contents, only attributes.

An empty XML element:

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

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

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

**XSD Indicators**

We can control HOW elements are to be used in documents with indicators.

**Order Indicators**

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

Order indicators are:

- All
- Choice
- Sequence

**All Indicator**

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

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

**Choice Indicator**

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

```
<xs:element name="person">
  <xs:complexType>
    <xs:choice>
      <xs:element name="employee" type="employee"/>
      <xs:element name="member" type="member"/>
    </xs:choice>
  </xs:complexType>
</xs:element>
```

**Sequence Indicator**

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

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

**An XML Document**

Let's have a look at this XML document called "shiporder.xml":

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<shiporder orderid="889923">
  <orderperson>John Smith</orderperson>
  <shipto>
    <name>Ola Nordmann</name>
    <address>Langgt 23</address>
    <city>4000 Stavanger</city>
    <country>Norway</country>
  </shipto>
 </shiporder>
```

The XML document above consists of a root element, "shiporder", that contains a required attribute called "orderid". The "shiporder" element contains child elements: "orderperson" and “shipto”.

**Create an XML Schema**

Now we want to create a schema for the XML document above. We start by opening a new file that we will call "shiporder.xsd". To create the schema we could simply follow the structure in the XML document and define each element as we find it. We will start with the standard **XML declaration followed by the xs:schema element** that defines a schema:

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

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

In the schema above we use the standard namespace (xs), and the URI associated with this namespace is the Schema language definition, which has the standard value of http://www.w3.org/2001/XMLSchema.

Next, we have to define the **"shiporder" element**. This element has an attribute and it contains other elements, therefore we consider it as a **complex type**. The child elements of the "shiporder" element is surrounded by a **xs:sequence element** that defines an ordered sequence of sub elements:

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

Then we have to define the **"orderperson" element as a simple type** (because it does not contain any attributes or other elements). The type (xs:string) is prefixed with the namespace. The prefix associated with XML Schema that indicates a predefined schema data type:

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

Next, we have to define two elements that are of the complex type: "shipto". We start by defining the "shipto" element:

```
<xs:element name="shipto">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="name" type="xs:string"/>
   <xs:element name="address" type="xs:string"/>
   <xs:element name="city" type="xs:string"/>
   <xs:element name="country" type="xs:string"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

We can now declare the attribute of the "shiporder" element. Since this is a required attribute we specify use="required".

**Note:** The attribute declarations must always come last:

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

Here is the complete listing of the schema file called "shiporder.xsd":

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

<xs:element name="shiporder">
 <xs:complexType>
  <xs:sequence>
```

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

     <xs:element name="shipto">
      <xs:complexType>

        <xs:sequence>
          <xs:element name="name" type="xs:string"/>
          <xs:element name="address" type="xs:string"/>
          <xs:element name="city" type="xs:string"/>
          <xs:element name="country" type="xs:string"/>
        </xs:sequence>

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

</xs:sequence>
<xs:attribute name="orderid" type="xs:string" use="required"/>
</xs:complexType>
</xs:element>

</xs:schema>
```

**An XSD Example**

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


<xs:element name="note">
<xs:complexType>
  <xs:sequence>
    <xs:element name="to" type="xs:string"/>
    <xs:element name="from" type="xs:string"/>
    <xs:element name="heading" type="xs:string"/>
    <xs:element name="body" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
</xs:element>
```

The Schema above is interpreted like this:

- <xs:element name="note"> defines the element called "note"
- <xs:complexType> the "note" element is a complex type
- <xs:sequence> the complex type is a sequence of elements
- <xs:element name="to" type="xs:string"> the element "to" is of type string (text)
- <xs:element name="from" type="xs:string"> the element "from" is of type string
- <xs:element name="heading" type="xs:string"> the element "heading" is of type string
- <xs:element name="body" type="xs:string"> the element "body" is of type string

Everything is wrapped in "Well Formed" XML.

## 3.10 XML Parser (Parsing XML documents)

All modern browsers have a built-in XML parser. An XML parser converts an XML document into an XML DOM object - which can then be manipulated with JavaScript.

**Parse an XML Document**

The following code fragment parses an XML document into an XML DOM object:

```
if (window.XMLHttpRequest)
  {// code for IE7+, Firefox, Chrome, Opera, Safari
  xmlhttp=new XMLHttpRequest();
  }
else
  {// code for IE6, IE5
  xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
  }
xmlhttp.open("GET","books.xml",false);
xmlhttp.send();
xmlDoc=xmlhttp.responseXML;
```

**XML DOM**

The XML DOM defines a standard way for accessing and manipulating XML documents. The XML DOM views an XML document as a tree-structure. All elements can be accessed through the DOM tree. Their content (text and attributes) can be modified or deleted, and new elements can be created. The elements, their text, and their attributes are all known as nodes.

**The HTML DOM**

The HTML DOM defines a standard way for accessing and manipulating HTML documents.

All HTML elements can be accessed through the HTML DOM.

**Load an XML File - Cross-browser Example**

The following example parses an XML document ("note.xml") into an XML DOM object and then extracts some information from it with a JavaScript:

**Example**

```
<html>
<body>

 <span id="to"></span>
<span id="from"></span>
<span id="message"></span>

<script>
if (window.XMLHttpRequest)
  {// code for IE7+, Firefox, Chrome, Opera, Safari
  xmlhttp=new XMLHttpRequest();
  }
else
  {// code for IE6, IE5
  xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
 }
xmlhttp.open("GET","note.xml",false);
xmlhttp.send();
xmlDoc=xmlhttp.responseXML;

document.getElementById("to").innerHTML=
        xmlDoc.getElementsByTagName("to")[0].childNodes[0].nodeValue;
document.getElementById("from").innerHTML=
        xmlDoc.getElementsByTagName("from")[0].childNodes[0].nodeValue;
document.getElementById("message").innerHTML=
        xmlDoc.getElementsByTagName("message")[0].childNodes[0].nodeValue;
</script>

</body>
</html>
```

**Important Note!**

To extract the text "Tove" from the <to> element in the XML file above ("note.xml"), the syntax is: getElementsByTagName("to")[0].childNodes[0].nodeValue Notice that even if the XML file contains only ONE <to> element you still have to specify the array index [0]. This is because the getElementsByTagName() method returns an array.