XML - Extensible Markup Language

XML

* XML is designed to transport and store data.

XML stands for EXtensible Markup Language
XML is a markup language much like HTML
XML was designed to carry 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

The Difference Between XML and HTML

- XML is not a replacement for HTML.
- XML and HTML were designed with different goals:
- XML was designed to transport and store data, with focus on what data is
- HTML was designed to display data, with focus on how data looks
- HTML is about displaying information, while XML is about carrying information.

XML was introduced due to a number of reasons

1. HTML has become a complex language with the latest version having more than 100 different tags
2. More tags are needed to handle the growing technologies such as E-commerce.
3. Mobile machines such as PDA'S don't have enough processing power to handle such as a complex language. Therefore better structuring required.
4. XML is text based, platform and language independent,

How Can XML be Used?

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

- XML Separates Data from HTML
- XML Simplifies Data Sharing
- XML Simplifies Data Transport
- XML Simplifies Platform Changes
- XML Makes Your Data More Available

XML Tree

XML documents form a tree structure that starts at "***the root***" and branches to "***the leaves***".

An Example XML Document
XML documents use a self-describing and simple syntax:

```
<?xml version="1.0" encoding="UTF-8"?>
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Coming home this weekend!</body>
</note>
```

XML Tree

The first line is the XML declaration. It defines the XML version (1.0) and the encoding used.
The next line describes the **root element** of the document (like saying: "this document is a note"):

<note>

The next 4 lines describe 4 **child elements** of the root (to, from, heading, and body):

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

And finally the last line defines the end of the root element: ***</note>***

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).

```
<bookstore>
  <book category="CHILDREN">
    <title lang="en">Harry Potter</title>
    <author>J K. Rowling</author>
    <year>2005</year>
    <price>29.99</price>
  </book>
  <book category="WEB">
    <title lang="en">Learning XML</title>
    <author>Erik T. Ray</author>
    <year>2003</year>
    <price>39.95</price>
  </book>
</bookstore>
```

XML Syntax Rules

- **Tag names are case sensitive.**

With XML, 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>

- **Every opening tag must have a corresponding closing tag (unless it is as an empty tag).**

In XML, it is illegal to omit the closing tag. All elements **must** have a closing tag:
*** This is a paragraph. ***
***
***

XML Syntax Rules

- **All XML elements must be properly nested:**

This text is bold and italic

In XML, all elements **must** be properly nested within each other:

This text is bold and italic

In the example above, "Properly nested" simply means that since the element is opened inside the element, it must be closed inside the element.

XML Syntax Rules

- **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>
```

XML Syntax Rules

- **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.
Study the two XML documents below. The first one is incorrect, the second is correct:

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

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

Entity References

Some characters have a special meaning in XML

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

Entity References

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 < 25000 then</message>

To avoid this error, replace the "<" character with an **entity**

reference:

<message>if salary < 25000 then</message>

Comments in XML

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

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.

Well Formed XML

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

XML has 3 fundamental building blocks: **elements**, **attributes**, and **values**.

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...

An element is used to describe or contain a piece of Information

Elements consist of two tags: an opening tag and a closing tag

```
<bookstore>
  <book category="CHILDREN">
    <title lang="en">Harry Potter</title>
    <author>J K. Rowling</author>
    <year>2005</year>
    <price>29.99</price>
  </book>
  <book category="WEB">
    <title lang="en">Learning XML</title>
    <author>Erik T. Ray</author>
    <year>2003</year>
    <price>39.95</price>
  </book>
</bookstore>
```

In the example above, <bookstore> and <book> have **element contents**, because they contain other elements. <book> also has an **attribute** (category="CHILDREN"). <title>, <author>, <year>, and <price> have **text content** because they contain text.

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.

Best Naming Practices

•Make names descriptive: ***<first_name>, <last_name>.***

•Make names short and simple, like this: ***<book_title>*** not like this: ***<the_title_of_the_book>.***

•Avoid "-". If you name something "***first-name***," some software may think you want to subtract name from first.

•Avoid ".". If you name something "***first.name***," some software may think that "name" is a property of the object "first.“

•Avoid ":". Colons are reserved to be used for something called namespaces (more later).

•Non-English letters like éòá are perfectly legal in XML, but watch out for problems if your software doesn't support them.

Naming Styles

There are no naming styles defined for XML elements. But here are some commonly used:

Style	Example	Description
Lower case	<firstname>	All letters lower case
Upper case	<FIRSTNAME>	All letters upper case
Underscore	<first_name>	Underscore separates words
Pascal case	<FirstName>	Uppercase first letter in each word
Camel case	<firstName>	Uppercase first letter in each words exept the first

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>

XML Attributes Must be Quoted

- Attribute values must always be quoted. Either single or double quotes can be used.
- For a person's sex, the person element can be written like this:

<person sex="female"> or <person sex='female'>

- If the attribute value itself contains double quotes you can use single quotes, like in this example:

<gangster name='George "Shotgun" Ziegler'>

or you can use character entities:

<gangster name="George "Shotgun" Ziegler">

XML Elements vs. Attributes

```
<person sex="female">
  <firstname>Anna</firstname>
  <lastname>Smith</lastname>
</person>

<person>
  <sex>female</sex>
  <firstname>Anna</firstname>
  <lastname>Smith</lastname>
</person>
```

•In the first example sex is an attribute. In the last, sex is an element. Both examples provide the same information.

```
<note date="10/01/2008">
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Nice weekend!</body>
</note>
<note>
  <date>10/01/2008</date>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Nice weekend!</body>
</note>
```

```
<note>
  <date>
    <day>10</day>
    <month>01</month>
    <year>2008</year>
  </date>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Nice weekend!</body>
</note>
```

Avoid XML Attributes

Some of the problems with using attributes are:

- attributes cannot contain multiple values (elements can)
- attributes cannot contain tree structures (elements can)
- attributes are not easily expandable (for future changes)

Attributes are difficult to read and maintain. Use elements for data. Use attributes for information that is not relevant to the data.

XML Attributes for Metadata

Sometimes ID references are assigned to elements. These IDs can be used to identify XML elements in much the same way as the id attribute in HTML

```
<messages>
  <note id="501">
    <to>Tove</to>
    <from>Jani</from>
    <heading>Reminder</heading>
    <body>Don't forget me this weekend!</body>
  </note>
  <note id="502">
    <to>Jani</to>
    <from>Tove</from>
    <heading>Re: Reminder</heading>
    <body>I will not</body>
  </note>
</messages>
```

The id attributes above are for identifying the different notes. It is not a part of the note itself. Here is that **metadata** (data about data) should be stored as attributes, and the **data** itself should be stored as elements

Well-formedness & Validity

•An XML document with correct syntax is called "**Well Formed**".
•A "**Valid**" XML document must also conform to a specified document type.

Well Formed XML Documents
An XML document with correct syntax is "Well Formed".

•XML documents must have a root element
•XML elements must have a closing tag
•XML tags are case sensitive
•XML elements must be properly nested
•XML attribute values must be quoted

Valid XML Documents

A valid XML document is not the same as a well formed XML document.

•The first rule, for a valid XML document, is that it must be well formed
•The second rule is that a valid XML document must conform to a document type.

Rules that defines legal elements and attributes for XML documents are often called **document definitions**, or **document schemas**.

Document Definitions

There are different types of document definitions that can be used with XML:
•The original - Document Type Definition (DTD)
•The newer, and XML based - XML Schema – (XSD- XML Schema Type Definition)

XML – CDATA

- Character Data
- All text in an XML document will be parsed by the parser.
- But text inside a CDATA section will be ignored by the parser.

Example of a CDATA section is:
This is a face:

```
<![CDATA[
        *****
    * @ @ *
       * ) *
    * ~~~ *
       *****
]]>
```

PCDATA

- PCDATA means parsed character data.
- Think of character data as the text found between the start tag and the end tag of an XML element.
- PCDATA is text that WILL be parsed by a parser. The text will be examined by the parser for entities and markup.

Document Type Definition (DTD)

A Document Type Definition (DTD) defines the legal building blocks of an XML document. It defines the document structure with a list of legal elements and attributes.

A DTD can be declared inline inside an XML document, or as an external reference.

Internal DTD – It can directly include markup declarations in the document that form the internal DTD.

External DTD – It can reference external markup declarations that form the external DTD.

Standlone

Use 'yes' if the XML document has an internal DTD. Use 'no' if the XML document is linked to an external DTD, or any external entity

Internal DTD Declaration

If the DTD is declared inside the XML file, it should be wrapped in a DOCTYPE definition with the following syntax:

<!DOCTYPE root-element [element-declarations]>

```
<?xml version="1.0"?>
<!DOCTYPE address [
<!ELEMENT address (name,no,street,city,country)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT no (#PCDATA)>
<!ELEMENT street (#PCDATA)>
<!ELEMENT city (#PCDATA)>
<!ELEMENT country (#PCDATA)>
]>
<address>
    <name>nimal</name>
    <no>24</no>
    <street>1st lane</street>
    <city>Colombo</city>
    <country>sri lanka</country>
</address >
```

The DTD above is interpreted like this:

– **!DOCTYPE address** defines that the root element of this document is address
– **!ELEMENT address** defines that the address element contains five elements: "name,no,street,city,country”
– **!ELEMENT name** defines the name element to be of type "#PCDATA
– **!ELEMENT no** defines the no element to be of type "#PCDATA“
– **!ELEMENT street** defines the street element to be of type "#PCDATA“
– **!ELEMENT city** defines the city element to be of type "#PCDATA“
– **!ELEMENT country** defines the country element to be of type "#PCDATA“

External DTD Declaration

If the DTD is declared in an external file, it should be wrapped in a DOCTYPE definition with the following syntax:

<!DOCTYPE root-element SYSTEM "filename">

This is the same XML document as above, but with an external DTD

```
<?xml version="1.0"?>
<!DOCTYPE address SYSTEM "address.dtd">
<address>
    <name>nimal</name>
    <no>24</no>
    <street>1st lane</street>
    <city>Colombo</city>
    <country>sri lanka</country>
</address >
```

External DTD Declaration

- And this is the file “address.dtd" which contains the DTD:

```
<!ELEMENT address (name,no,street,city,country)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT no (#PCDATA)>
<!ELEMENT street (#PCDATA)>
<!ELEMENT city (#PCDATA)>
<!ELEMENT country (#PCDATA)>
```

Declaring Elements

• In a DTD, XML elements are declared with an element declaration with the following syntax:

<!ELEMENT element-name category>

Or

<!ELEMENT element-name (element-content)>

Empty Elements

• Empty elements are declared with the category keyword EMPTY:

<!ELEMENT element-name EMPTY>

Example:

<!ELEMENT br EMPTY>

XML example:

*
*

Elements with Parsed Character Data

• Elements with only parsed character data are declared with #PCDATA inside parentheses:

<!ELEMENT element-name (#PCDATA)>

Example:

<!ELEMENT street (#PCDATA)>

Elements with any Contents

• Elements declared with the category keyword ANY, can contain any combination of parsable data:

<!ELEMENT element-name ANY>

Example:

<!ELEMENT address ANY>

Elements with Children (sequences)

• Elements with one or more children are declared with the name of the children elements inside parentheses:

<!ELEMENT element-name (child1)>

or

<!ELEMENT element-name (child1,child2,...)>

Example:

<!ELEMENT address (name,no,street,city,country)>

•When children are declared in a sequence separated by commas, the children must appear in the same sequence in the document.

• In a full declaration, the children must also be declared, and the children can also have children.

Elements with Children (sequences)

The full declaration of the “address" element is:

<!ELEMENT address (name,no,street,city,country)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT no (#PCDATA)>
<!ELEMENT street (#PCDATA)>
<!ELEMENT city (#PCDATA)>
<!ELEMENT country (#PCDATA)>

Declaring Only One Occurrence of an Element

<!ELEMENT element-name (child-name)>

Example:

<!ELEMENT address (street)>

• The example above declares that the child element "street" must occur once, and only once inside the "address" element.

Declaring Minimum One Occurrence of an Element

<!ELEMENT element-name (child-name+)>

Example:

<!ELEMENT address (street+)>

• The + sign in the example above declares that the child element "street" must occur one or more times inside the "address" element.

Declaring Zero or More Occurrences of an Element

<!ELEMENT element-name (child-name)>*

Example:

<!ELEMENT address (street)>*

• The * sign in the example above declares that the child element "street" can occur zero or more times inside the "address" element.

Declaring Zero or One Occurrences of an Element

<!ELEMENT element-name (child-name?)>

Example:

<!ELEMENT address (street?)>

• The ? sign in the example above declares that the child element "street" can occur zero or one time inside the "address" element.

Declaring either/or Content

Example:

<!ELEMENT note (to,from,header,(message|body))>

• The example above declares that the "note“ element must contain a "to" element, a "from“ element, a "header" element, and either a "message" or a "body" element.

Declaring Mixed Content

Example:

<!ELEMENT note (#PCDATA|to|from|header|message)>*

• The example above declares that the "note“ element can contain zero or more occurrences of parsed character data, "to", "from", "header", or "message" elements.

DTD - Attributes

Declaring Attributes

- An attribute declaration has the following syntax:

<!ATTLIST element-name attribute-name attribute-type default-value>

DTD example:

<!ATTLIST payment type CDATA "check">

XML example:

<payment type="check" />

Some of the **attribute-types** which can be used are following:

Type	Description
CDATA	The value is character data
(*en1*|*en2*|..)	The value must be one from an enumerated list
ID	The value is a unique id
IDREF	The value is the id of another element
IDREFS	The value is a list of other ids
NMTOKEN	The value is a valid XML name
NMTOKENS	The value is a list of valid XML names
ENTITY	The value is an entity
ENTITIES	The value is a list of entities
NOTATION	The value is a name of a notation
xml:	The value is a predefined xml value

The default-value can be one of the following:

Value	Explanation
value	The default value of the attribute
#REQUIRED	The attribute is required
#IMPLIED	The attribute is optional
#FIXED *value*	The attribute value is fixed

A Default Attribute Value

DTD:

<!ELEMENT square EMPTY>
<!ATTLIST square width CDATA "0">

Valid XML:

<square width="100" />

• In the example above, the "square" element is defined to be an empty element with a "width“ attribute of type CDATA. If no width is specified, it has a default value of 0. :

#REQUIRED

Use the #REQUIRED keyword if you don't have an option for a default value, but still want to force the attribute to be present.

Syntax

<!ATTLIST element-name attribute-name attribute-type #REQUIRED>

Example

DTD:

<!ATTLIST person number CDATA #REQUIRED>

Valid XML:

<person number="5677" />

Invalid XML:

<person />

#IMPLIED

Syntax

<!ATTLIST element-name attribute-name attribute-type #IMPLIED>

Example

DTD:

<!ATTLIST contact fax CDATA #IMPLIED>

Valid XML:

<contact fax="555-667788" />

Valid XML:

<contact />

• Use the #IMPLIED keyword if you don't want to force the author to include an attribute, and you don't have an option for a default value.

FIXED

Syntax

<!ATTLIST element-name attribute-name attribute-type #FIXED "value">

Example

DTD:

<!ATTLIST sender company CDATA #FIXED "Microsoft">

Valid XML:

<sender company="Microsoft" />

Invalid XML:

<sender company="W3Schools" />

• Use the #FIXED keyword when you want an attribute to have a fixed value without allowing the author to change it. If an author includes another value, the XML parser will return an error.

Enumerated Attribute Values

Syntax

<!ATTLIST element-name attribute-name (en1|en2|..) default-value>

Example

DTD:

<!ATTLIST payment type (check|cash) "cash">

XML example:

<payment type="check" />

or

<payment type="cash" />

• Use enumerated attribute values when you want the attribute value to be one of a fixed set of legal values.

XML Schema

* XML Schema is an XML-based alternative to DTD.
* An XML schema describes the structure of an XML document.
* The XML Schema language is also referred to as XML Schema Definition (XSD).

The purpose of an XML Schema is to define the legal building blocks of an XML document, just like a DTD.

An XML Schema:
defines elements that can appear in a document
defines attributes that can appear in a document
defines which elements are child elements
defines the order of child elements
defines the number 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

XML Schema

We think that very soon 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
XML Schemas support namespaces

Namespaces
XML Namespaces provide a method to avoid element name conflicts.

XML Schema

- XML Schemas Support Data Types

* One of the greatest strength of XML Schemas is the support for data types.

With support for data types:

It is easier to describe allowable document content
It is easier to validate the correctness of data
It is easier to work with data from a database
It is easier to define data facets (restrictions on data)
It is easier to define data patterns (data formats)
It is easier to convert data between different data types

XML Schema

- XML Schemas use XML Syntax

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

Some benefits of that XML Schemas are written in XML:

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

XML Schema

- XML Schemas Secure Data Communication
- XML Schemas are Extensible

XML Schemas are extensible, because they are written in XML.
With an extensible Schema definition you can:
Reuse your Schema in other Schemas
Create your own data types derived from the standard types
Reference multiple schemas in the same document

XML Schema

Well-Formed is not Enough
A well-formed XML document is a document that conforms to the XML syntax rules, like:

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

schema.xsd

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.w3schools.com"
xmlns="http://www.w3schools.com"
elementFormDefault="qualified">

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

</xs:schema>
```

XML Schema

```xml
<?xml version="1.0"?>
<note
xmlns="http://www.w3schools.com"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.w3schools.com schema.xsd">
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

The <schema> Element

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

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

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

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

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

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.w3schools.com"
xmlns="http://www.w3schools.com"
elementFormDefault="qualified">
...
...
</xs:schema>
```

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

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

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

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

xmlns="http://www.w3schools.com"
indicates that the default namespace is http://www.w3schools.com".

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

Once you have the XML Schema Instance namespace available:
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

you can use the schemaLocation attribute. This attribute has two values, separated by a space. The first value is the namespace to use. The second value is the location of the XML schema to use for that namespace:

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

XSD Simple Elements

XML Schemas define the elements of your XML files.
A simple element is an XML element that contains only text. It cannot contain any other elements or attributes.

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

XSD Simple Elements

Example
Here are some XML elements:

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

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"/>
```

XSD Simple Elements

Example
Here are some XML elements:

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

And here are the corresponding simple element definitions:

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

Default and Fixed Values for Simple Elements

Simple elements may have a default value OR a fixed value specified.
A default value is automatically assigned to the element when no other value is specified.
In the following example the default value is "red":

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

A fixed value is also automatically assigned to the element, and you cannot specify another value.
In the following example the fixed value is "red":

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

XSD Attributes

All attributes are declared as simple types.
What is an Attribute?
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"/>

Example
Here is an XML element with an attribute:

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

And here is the corresponding attribute definition:

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

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 Restrictions/Facets

Restrictions are used to define acceptable values for XML elements or attributes. Restrictions on XML elements are called facets.

Restrictions on Values

The following example defines an element called "age" with a restriction. The value of age cannot be lower than 0 or greater than 120:

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

XSD Restrictions/Facets

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

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

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

XSD Restrictions/Facets

Restrictions on a Series of Values

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

The example below defines an element called "letter" with a restriction.

The only acceptable value is ONE of the LOWERCASE letters from

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

XSD Restrictions/Facets

The only acceptable value is THREE of the UPPERCASE letters from a to z:

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

The only acceptable value is THREE of the LOWERCASE OR UPPERCASE letters from a to z:

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

XSD Restrictions/Facets

The next example defines an element called "prodid" with a restriction. The only acceptable value is FIVE digits in a sequence, and each digit must be in a range from 0 to 9:

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

Restrictions for Datatypes

Constraint	Description
enumeration	Defines a list of acceptable values
fractionDigits	Specifies the maximum number of decimal places allowed. Must be equal to or greater than zero
length	Specifies the exact number of characters or list items allowed. Must be equal to or greater than zero
maxExclusive	Specifies the upper bounds for numeric values (the value must be less than this value)
maxInclusive	Specifies the upper bounds for numeric values (the value must be less than or equal to this value)
maxLength	Specifies the maximum number of characters or list items allowed. Must be equal to or greater than zero
minExclusive	Specifies the lower bounds for numeric values (the value must be greater than this value)
minInclusive	Specifies the lower bounds for numeric values (the value must be greater than or equal to this value)
minLength	Specifies the minimum number of characters or list items allowed. Must be equal to or greater than zero
pattern	Defines the exact sequence of characters that are acceptable
totalDigits	Specifies the exact number of digits allowed. Must be greater than zero
whiteSpace	Specifies how white space (line feeds, tabs, spaces, and carriage returns) is handled

Complex Element

A complex element is an XML element that contains other elements and/or attributes.
There are four kinds of complex elements:

- empty elements
- elements that contain only other elements
- elements that contain only text
- elements that contain both other elements and text

Complex Element

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

```
<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>
```

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

XML Namespaces

XML Namespaces provide a method to avoid element name conflicts.

Name Conflicts

In XML, element names are defined by the developer. This often results in a conflict when trying to mix XML documents from different XML applications.

This XML carries HTML table information

```
<table>
  <tr>
    <td>Apples</td>
    <td>Bananas</td>
  </tr>
</table>
```

This XML carries information about a table

```
<table>
  <name>African Coffee Table</name>
  <width>80</width>
  <length>120</length>
</table>
```

If these XML fragments were added together, there would be a name conflict. Both contain a <table> element, but the elements have different content and meaning.
A user or an XML application will not know how to handle these differences.

XML Namespaces - The xmlns Attribute

When using prefixes in XML, a so-called **namespace** for the prefix must be defined.
The namespace is defined by the **xmlns attribute** in the start tag of an element.
The namespace declaration has the following syntax. ***xmlns:prefix="URI".***

```
<root>
<h:table xmlns:h="http://www.w3.org/TR/html4/">
  <h:tr>
    <h:td>Apples</h:td>
    <h:td>Bananas</h:td>
  </h:tr>
</h:table>
<f:table xmlns:f="http://www.w3schools.com/furniture">
  <f:name>African Coffee Table</f:name>
  <f:width>80</f:width>
  <f:length>120</f:length>
</f:table>
</root>
```

In the example above, the xmlns attribute in the <table> tag give the h: and f: prefixes a qualified namespace.

When a namespace is defined for an element, all child elements with the same prefix are associated with the same namespace.

RDF

RDF stands for Resource Description Framework.
RDF is a standard for describing Web resources.
RDF can be used to describe title, author, content, and copyright information of web pages.

What is RDF?

- RDF stands for **R**esource **D**escription **F**ramework
- RDF is a framework for describing resources on the web
- RDF is designed to be read and understood by computers
- RDF is not designed for being displayed to people
- RDF is written in XML
- RDF is a part of the W3C's Semantic Web Activity
- RDF is a W3C Recommendation

RDF - Examples of Use

- Describing properties for shopping items, such as price and availability
- Describing time schedules for web events
- Describing information about web pages (content, author, created and modified date)
- Describing content and rating for web pictures
- Describing content for search engines
- Describing electronic libraries

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

<rdf:RDF
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:si="http://www.w3schools.com/rdf/">

<rdf:Description rdf:about="http://www.w3schools.com">
  <si:title>W3Schools</si:title>
  <si:author>Jan Egil Refsnes</si:author>
</rdf:Description>

</rdf:RDF>
```

XPath

XPath is used to navigate through elements and attributes in XML documents.

- XPath is a syntax for defining parts of an XML document
- XPath uses path expressions to navigate in XML documents
- XPath contains a library of standard functions
- XPath is a major element in XSLT
- XPath is a W3C recommendation
- XPath (XML Path Language) is a language used to point to (or select) parts of XML documents.
- XPath is not a complete language, but a syntax to be used by other languages.
- XPath was designed by W3C, as a major element in the W3C standards, XSLT, XQuery, XLink, and XPointer.
- Today XPath can also be used in JavaScript and Java, XML Schema, PHP, Python, C and C++, and lots of other languages.

XPath

```
<?xml version="1.0" encoding="UTF-8"?>
<bookstore>
<book>
  <title lang="eng">Harry Potter</title>
  <price>29.99</price>
</book>
<book>
  <title lang="eng">Learning XML</title>
  <price>39.95</price>
</book>
</bookstore>
```

- The following example selects all the title nodes

  /bookstore/book/title

- The following example selects the title of the first book node under the bookstore element:

  /bookstore/book[1]/title

Displaying XML with CSS

It is possible to use CSS to format an XML document.

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/css" href="cd_catalog.css"?>
<CATALOG>
 <CD>
  <TITLE>Empire Burlesque</TITLE>
  <ARTIST>Bob Dylan</ARTIST>
  <COUNTRY>USA</COUNTRY>
  <COMPANY>Columbia</COMPANY>
  <PRICE>10.90</PRICE>
  <YEAR>1985</YEAR>
 </CD>
 <CD>
  <TITLE>Hide your heart</TITLE>
  <ARTIST>Bonnie Tyler</ARTIST>
  <COUNTRY>UK</COUNTRY>
  <COMPANY>CBS Records</COMPANY>
  <PRICE>9.90</PRICE>
  <YEAR>1988</YEAR>
 </CD>
</CATALOG>
```

```
CATALOG{
background-color: #ffffff;
width: 100%; }
CD {
display: block;
margin-bottom: 30pt;
margin-left: 0;}
TITLE{
color: #FF0000;
font-size: 20pt;}
ARTIST{
color: #0000FF;
font-size: 20pt;}
COUNTRY,PRICE,YEAR,COMPANY {
display: block;
color: #000000;
margin-left: 20pt; }
```

XSLT

•XSL stands for Extensible Style sheet Language, and is a style sheet language for XML documents.

•XSLT is a language for transforming XML documents into XHTML documents or to other XML documents.

•XSLT is used to transform an XML document into another XML document, or another type of document that is recognized by a browser, like HTML and XHTML. Normally XSLT does this by transforming each XML element into an (X)HTML element.

•XSLT stands for XSL Transformations
•XSLT is the most important part of XSL
•XSLT transforms an XML document into another XML document
•XSLT uses XPath to navigate in XML documents
•XSLT is a W3C Recommendation

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
  <html>
  <body>
  <h2>My CD Collection</h2>
  <table border="1">
    <tr bgcolor="#9acd32">
      <th>Title</th>
      <th>Artist</th>
    </tr>
    <xsl:for-each select="catalog/cd">
    <tr>
      <td><xsl:value-of select="title"/></td>
      <td><xsl:value-of select="artist"/></td>
    </tr>
    </xsl:for-each>
  </table>
  </body>
  </html>
</xsl:template>
</xsl:stylesheet>
```

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet type="text/xsl" href="estyle.xsl"?>
<catalog>
  <cd>
    <title>Empire Burlesque</title>
    <artist>Bob Dylan</artist>
    <country>USA</country>
    <company>Columbia</company>
    <price>10.90</price>
    <year>1985</year>
  </cd>
.
.
</catalog>
```

There are two ways to access XML through a programming language

1. SAX- Simple API for XML
2. DOM – Document Object Model

Entities

Entities are variables used to define shortcuts to standard text or special characters.

- Entity references are references to entities
- Entities can be declared internal or external

An Internal Entity Declaration
Syntax

<!ENTITY entity-name "entity-value">

DTD Example:

<!ENTITY writer "Donald Duck.">
<!ENTITY copyright "Copyright W3Schools.">

XML example:

<author>&writer;&copyright;</author>

Entities

An External Entity Declaration
Syntax
<!ENTITY entity-name SYSTEM "URI/URL">

Example
DTD Example:

<!ENTITY writer SYSTEM
"http://www.w3schools.com/entities.dtd">
<!ENTITY copyright SYSTEM
"http://www.w3schools.com/entities.dtd">

XML example:

<author>&writer;&copyright;</author>

XSL-FO

XSL-FO stands for Extensible Stylesheet Language Formatting Objects.

XSL-FO is a language for formatting XML data for output to screen, paper or other media

XSL-FO Areas

The XSL formatting model defines a number of rectangular areas (boxes) to display output.

All output (text, pictures, etc.) will be formatted into these boxes and then displayed or printed to a target media.

We will take a closer look at the following areas:

- Pages
- Regions
- Block areas
- Line areas
- Inline areas

XSL-FO Block Areas

XSL-FO Block areas define small block elements (the ones that normally starts with a new line) like paragraphs, tables and lists. XSL-FO Block areas can contain other Block areas, but most often they contain Line areas.

XSL-FO List Blocks

There are four XSL-FO objects used to create lists:

fo:list-block (contains the whole list)

fo:list-item (contains each item in the list)

fo:list-item-label (contains the label for the list-item - typically an <fo:block> containing a number, character, etc.)

fo:list-item-body (contains the content/body of the list-item - typically one or more <fo:block> objects)