XML: eXtensible Markup Language

- XML is a software- and hardware-independent tool for storing and transporting data.
- XML is a markup language much like HTML.
- XML was designed to store and transport data and to be self-descriptive.
- XML is a W3C Recommendation(a standard or guideline that the World Wide Web Consortium (W3C) endorses for web technologies).
- XML is often used for distributing data over the Internet.

Example: The XML below is quite self-descriptive:

  - It has sender & receiver information
  - It has a heading
  - It has a message body
- But still, the XML above does not DO anything. XML is just information wrapped in tags.
- Someone must write a piece of software to send, receive, store, or display it.

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

- **The Difference Between XML and HTML**
- XML and HTML were designed with different goals:
- XML was designed to carry data - with focus on what data is
- HTML was designed to display data - with focus on how data looks
- XML tags are not predefined like HTML tags are
- The tags in the example above (like <to> and <from>) are not defined in any XML standard. These tags are "invented" by the author of the XML document unlike HTML which works with predefined tags
- With XML, the author must define both the tags and the document structure.
- **XML is Extensible**
- Most XML applications will work as expected even if new data is added (or removed).
- Imagine an application designed to display the original version of note.xml (<to> <from> <heading> <body>).
- Then imagine a newer version of note.xml with added <date> and <hour> elements, and a removed <heading>.
- The way XML is constructed, older version of the application can still work:

XML Simplifies Things

- XML simplifies data sharing
- XML simplifies data transport
- XML simplifies platform changes
- XML simplifies data availability

- Many computer systems contain data in incompatible formats.
- Exchanging data between incompatible systems (or upgraded systems) is a time-consuming task for web developers.
- Large amounts of data must be converted, and incompatible data is often lost.
- XML stores data in plain text format.
- This provides a software- and hardware-independent way of storing, transporting, and sharing data.
- XML also makes it easier to expand or upgrade to new operating systems, new applications, or new browsers, without losing data.

XML Separates Data from Presentation

- XML does not carry any information about how to be displayed.
- The same XML data can be used in many different presentation scenarios.
- Because of this, with XML, there is a full separation between data and presentation.

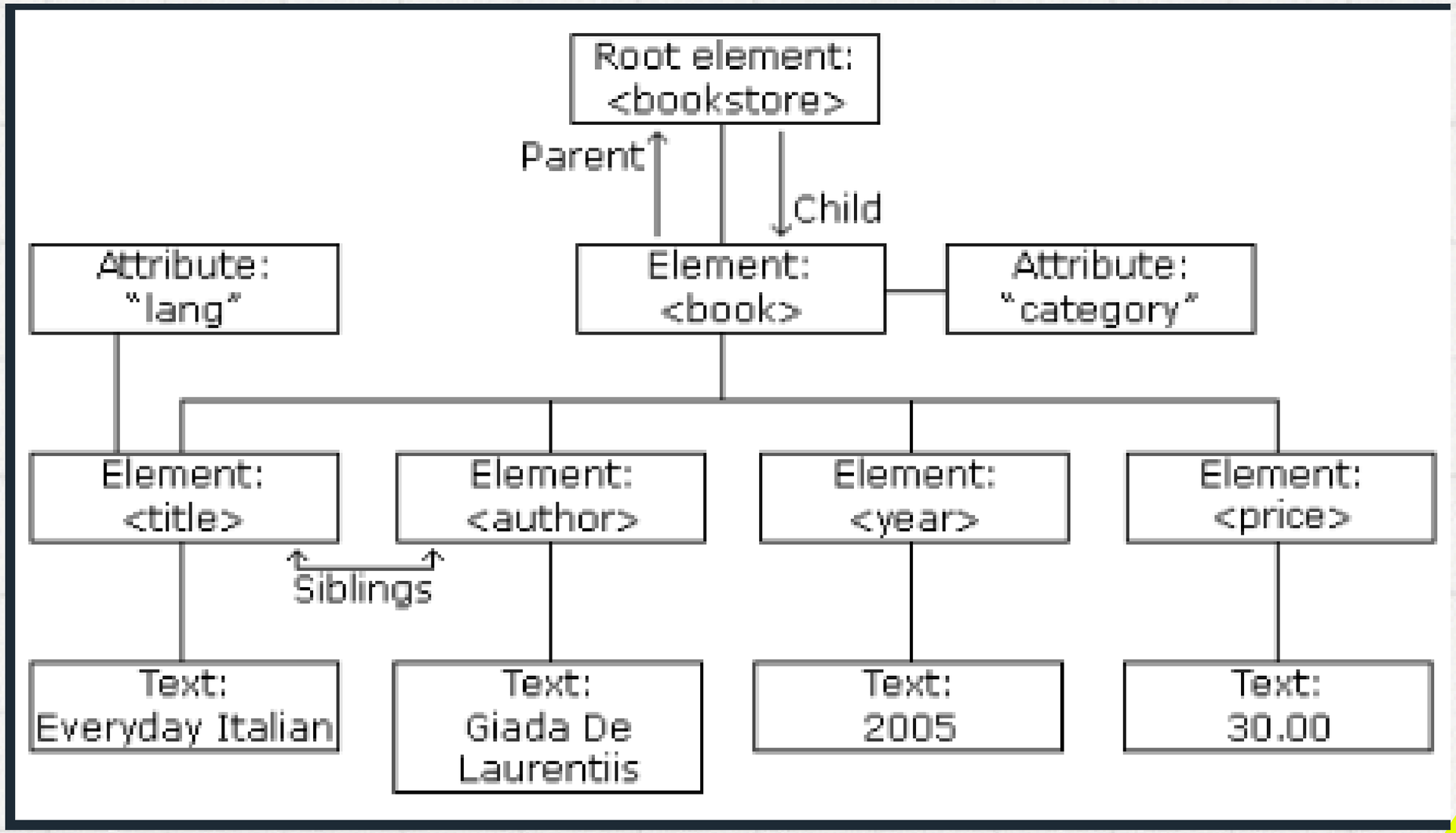
XML is Often a Complement to HTML

- In many HTML applications, XML is used to store or transport data, while HTML is used to format and display the same data.

XML Separates Data from HTML

- When displaying data in HTML, you should not have to edit the HTML file when the data changes.
- With XML, the data can be stored in separate XML files.
- With a few lines of JavaScript code, you can read an XML file and update the data content of any HTML page.

XML Tree Structure



- XML documents are formed as element trees.
- An XML tree starts at a root element and branches from the root to child elements.
- All elements can have sub-elements (child elements):
- The terms parent, child, and sibling are used to describe the relationships between elements.
- Parents have children. Siblings are children on the same level (brothers and sisters).

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

- XML Documents must contain one root element that is the parent of all other elements.
- All XML Elements Must Have a Closing Tag.
- XML Tags are Case Sensitive
- XML Elements Must be Properly Nested.

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

- XML elements can have attributes in name/value pairs just like in HTML which must always be quoted
- 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.

```
<b><i>This text is bold and italic</i></b>
```

```
<message>salary < 1000</message>
```

- To avoid this error, replace the "<" character with an entity reference:

```
<message>salary &lt; 1000</message>
```

Comments in XML

The syntax for writing comments in XML is similar to that of HTML:

```
<!-- This is a comment -->
```

<	<	less than
>	>	greater than
&	&	ampersand
'	'	apostrophe
"	"	quotation mark

White-space is Preserved in XML

XML does not truncate multiple white-spaces (HTML truncates multiple white-spaces to one single white-space):

XML:	Hello	Tove
HTML:	Hello Tove	

XML Element

An XML element is everything from (including) the element's start tag to (including) the element's end tag.

```
<price>29.99</price>
```

An element can contain:

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

In the example

- <title>, <author>, <year>, and <price> have text content because they contain text.
- <bookstore> and <book> have element contents, because they contain elements.
- <book> has an attribute (category="children").

Empty XML Elements

- An element with no content is said to be empty.
- You can also use a so called self-closing tag

```
<element></element>
```

```
<element />
```

XML Naming Rules

- Element names are case-sensitive
- Element names must start with a letter or underscore
- Element names cannot contain spaces.
- Element names cannot start with the letters xml (or XML, or Xml, etc).
- Element names can contain letters, digits, hyphens, underscores, and periods.

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

Best Naming Practices

- **Create descriptive name:** like this: <person>, <firstname>, <lastname>.
- **Create short and simple names:** 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 for namespaces.

XML Elements are Extensible

- XML elements can be extended to carry more information
- Consider that we created an application that extracted the <to>, <from>, and <body> elements from the XML document to produce this output.
- Consider that the author of the XML document added some extra information to it.
- The application should still be able to find the <to>, <from>, and <body> elements in the XML document and produce the same output (without breaking applications.)

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

```xml
<note>
  <date>2008-01-10</date>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

- **XML Attributes**
- XML elements can have attributes, just like HTML.
- Attributes are designed to contain data related to a specific element.
- XML Attributes Must be Quoted: Either single or double quotes can be used.

Style	Example	Description
Lower case	<firstname>	All letters lower case
Upper case	<FIRSTNAME>	All letters upper case
Snake case	<first_name>	Underscore separates words (commonly used in SQL databases)
Pascal case	<FirstName>	Uppercase first letter in each word (commonly used by C programmers)
Camel case	<firstName>	Uppercase first letter in each word except the first (commonly used in JavaScript)

Avoid XML Attributes?

Some things to consider when 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)

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. This example demonstrates this:

- The id attributes above are for identifying the different notes. It is not a part of the note itself.
- Metadata (data about data) should be stored as attributes, and the data itself should be stored as elements.
- **XML Namespaces**
- XML Namespaces provide a method to avoid element 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:
- 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
<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>
```

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

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

Solving the Name Conflict Using a Prefix

- Name conflicts in XML can easily be avoided using a name prefix.
- This XML carries information about an HTML table, and a piece of furniture:
- In the example above, there will be no conflict because the two <table> elements have different names.

XML Namespaces - The xmlns Attribute

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

In the example above:

- The xmlns attribute in the first <table> element gives the h: prefix a qualified namespace.
- The xmlns attribute in the second <table> element gives the f: prefix a qualified namespace.

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

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

- When a namespace is defined for an element, all child elements with the same prefix are associated with the same namespace.
- Namespaces can also be declared in the XML root element:
- The namespace URI is not used by the parser to look up information.
- The purpose of using an URI is to give the namespace a unique name.
- However, companies often use the namespace as a pointer to a web page containing namespace information.

```
<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="https://www.w3schools.com/furniture">
  <f:name>African Coffee Table</f:name>
  <f:width>80</f:width>
  <f:length>120</f:length>
</f:table>

</root>
```

Namespaces in Real Use

- XSLT is a language that can be used to transform XML documents into other formats.
- The XML document below, is a document used to transform XML into HTML.
- The namespace "http://www.w3.org/1999/XSL/Transform" identifies XSLT elements inside an HTML document:

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

<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>
      <th style="text-align:left">Title</th>
      <th style="text-align:left">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>
```

This XML carries HTML table information

```xml
<table xmlns="http://www.w3.org/TR/html4/"
  <tr>
    <td>Apples</td>
    <td>Bananas</td>
  </tr>
</table>
```

DTD: Document Type Definition.

- A DTD defines the structure and the legal elements and attributes of an XML document.

Why Use a DTD?

- With a DTD, independent groups of people can agree on a standard DTD for interchanging data.
- An application can use a DTD to verify that XML data is valid.

An Internal DTD Declaration

- If the DTD is declared inside the XML file, it must be
- wrapped inside the <!DOCTYPE> definition.

```
<?xml version="1.0"?>
<!DOCTYPE note [
<!ELEMENT note (to,from,heading,body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
]>
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend</body>
</note>
```

The DTD above is interpreted like this:

- !DOCTYPE note defines that the root element of this document is note
- !ELEMENT note defines that the note element must contain four
- elements: "to,from,heading,body"
- !ELEMENT to, from, heading and body defines the to element to be of type "#PCDATA".
- !ELEMENT body defines the body element to be of type "#PCDATA”.

An External DTD Declaration

If the DTD is declared in an external file, the <!DOCTYPE> definition must contain a reference to the DTD file:

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

And here is the file "note.dtd", which contains the DTD:

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

The Building Blocks of XML Documents

Seen from a DTD point of view, all XML documents are made up by the following building blocks:

- **Elements:** are the main building blocks of both XML and HTML documents. Eg of HTML elements are "body" and "table". Eg of XML elements could be "note" and "message" could also contain empty tags such as "hr", "br".
- **Attributes:** Attributes provide extra information about elements and are always placed inside the opening tag of an element. Attributes always come in name/value pairs.

- **Entities:** Some characters have a special meaning in XML, like the less than sign (<) that defines the start of an XML tag.
- **PCDATA:** stands for 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.
- Tags inside the text will be treated as markup and entities will be expanded.
- However, parsed character data should not contain any &, <, or > characters; these need to be represented by the < and > entities, respectively.

Entity References	Character
<	<
>	>
&	&
"	"
'	'

- **CDATA:** CDATA means character data.
- CDATA is text that will NOT be parsed by a parser. Tags inside the text will NOT be treated as markup and entities will not be expanded.

DTD Elements

Elements with any Contents

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

```
<!ELEMENT element-name ANY>

Example:

<!ELEMENT note ANY>
```

Elements with Children (sequences)

- Elements with one or more children are declared with the name of the children elements inside parentheses:
- **Note:** 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. The full declaration of the "note" element is:

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

Example:

<!ELEMENT note (to,from,heading,body)>
```

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

Declaring Only One Occurrence of an Element

- The example above declares that the child element "message" must occur once, and only once inside the "note" element.

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

Example:

<!ELEMENT note (message)>
```

- **Declaring Minimum One Occurrence of an Element**
- The + sign declares that the child element "message" must occur one or more times inside the "note" element.
- **Declaring Zero or More Occurrences of an Element**
- The * sign declares that the child element "message" can occur zero or more times inside the "note" element.
- **Declaring Zero or One Occurrences of an Element**
- The ? sign in the example above declares that the child element "message" can occur zero or one time inside the "note" element.

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

Example:

<!ELEMENT note (message+)>
```

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

Example:

<!ELEMENT note (message*)>
```

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

Example:

<!ELEMENT note (message?)>
```

- **Declaring either/or Content**
- The example declares that the "note" element must contain a "to" element, a "from" element, a "header" element, and either a "message" or a "body" element.

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

- **Declaring Mixed Content**
- The example declares that the "note" element can contain zero or more occurrences of parsed character data, "to", "from", "header", or "message" elements.

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

DTD Attributes

In a DTD, attributes are declared with an ATTLIST declaration.

Syntax:

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

DTD example:

<!ATTLIST payment type CDATA "check">

XML example:

<payment type="check" />
```

The attribute-type can be one of the 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 attribute-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

Default

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.

```
DTD:
<!ELEMENT square EMPTY>
<!ATTLIST square width CDATA "0">

Valid XML:
<square width="100" />
```

#REQUIRED

Syntax:<!ATTLISTelement-name attribute-name attribute-type #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.

```
DTD:
<!ATTLIST person number CDATA #REQUIRED>

Valid XML:
<person number="5677" />

Invalid XML:
<person />
```

#IMPLIED

Syntax

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

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.

```
DTD:
<!ATTLIST contact fax CDATA #IMPLIED>

Valid XML:
<contact fax="555-667788" />

Valid XML:
<contact />
```

#FIXED

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

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.

```
DTD:
<!ATTLIST sender company CDATA #FIXED "Microsoft">

Valid XML:
<sender company="Microsoft" />

Invalid XML:
<sender company="W3Schools" />
```

Enumerated Attribute Values

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

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

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

XML example:
<payment type="check" />
or
<payment type="cash" />
```

Note: If you use attributes as containers for data, you end up with documents that are difficult to read and maintain. Try to use elements to describe data. Use attributes only to provide information that is not relevant to the data.

metadata (data about data) should be stored as attributes, and that data itself should be stored as elements.

XML Schema

- An XML Schema describes the structure of an XML document, just like a DTD.
- An XML document with correct syntax is called "Well Formed".
- An XML document validated against an XML Schema is both "Well Formed" and "Valid".
- 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:
  - the elements and attributes that can appear in a document
  - the number of (and order of) child elements
  - data types for elements and attributes
  - default and fixed values for elements and attributes

Why Learn XML Schema?

- In the XML world, hundreds of standardized XML formats are in daily use.
- Many of these XML standards are defined by XML Schemas.
- XML Schema is an XML-based (and more powerful) alternative to DTD.
- XML Schemas Support Data Types
- One of the greatest strength of XML Schemas is the support for data types.
- It is easier to describe, validate, define data facets (restrictions on data), define data patterns (data formats) allowable document content on data

XML Schemas use XML Syntax

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

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

XML Schemas Secure Data Communication

- When sending data from a sender to a receiver, it is essential that both parts have the same "expectations" about the content.
- With XML Schemas, the sender can describe the data in a way that the receiver will understand.
- A date like: "03-11-2004" will, in some countries, be interpreted as 3.November and in other countries as 11.March.
- However, an XML element with a data type like this:
- <date type="date">2004-03-11</date>
- ensures a mutual understanding of the content, because the XML data type "date" requires the format "YYYY-MM-DD".

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

- it must begin with the XML declaration
- it must have one unique root element
- start-tags must have matching end-tags
- elements are case sensitive
- all elements must be closed
- all elements must be properly nested
- all attribute values must be quoted
- entities must be used for special characters
- Even if documents are well-formed they can still contain errors, and those errors can have serious consequences.

The <schema> Element

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

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

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

- This 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="https://www.w3schools.com"

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

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

- This indicates that the default namespace is "https://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.

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

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

Defining a Simple Element

Syntax: <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:

- 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"/>

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

Default and Fixed Values for Simple Elements

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

```
<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.

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

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.

How to Define an Attribute?

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

- The syntax for defining an attribute is:
- where xxx is the name of the attribute and yyy specifies the data type of the attribute
- Here is an XML element with an attribute: <lastname lang="EN">Smith</lastname>
- And here is the corresponding attribute definition: <xs:attribute name="lang" type="xs:string"/>

Default and Fixed Values for Attributes

- A default value is automatically assigned to the attribute when no other value is specified.
- A fixed value is also automatically assigned to the attribute, and you cannot specify another value.

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

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

Optional and Required Attributes

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

Restrictions on Content

When an XML element or attribute has a data type defined, it puts restrictions on the element's or attribute's content. Eg: if an element of type "xs:date" contains a string like "Hello World", the element will not validate.

What is a 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:** An empty complex element cannot have contents, only attributes.
- **elements that contain only other elements:** An "elements-only" complex type contains an element that contains only other elements.
- **elements that contain only text:** A complex text-only element can contain text and attributes.
- **elements that contain both other elements and text:** A mixed complex type element can contain attributes, elements, and text.

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

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

How to Define a Complex Element

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

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

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

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.

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

- If you use the method described above, several elements can refer to the same complex type, 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>
```

Eg of Empty Elements

- XML

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

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

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

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

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

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

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

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

Eg of Elements-only

- An XML element, "person", that contains only other elements:
- You can define the "person" element in a schema, like this:
- Notice the <xs:sequence> tag. It means that the elements defined ("firstname" and "lastname") must appear in that order inside a "person" element.

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

```
<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>
```

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

```
<xs:element name="person" type="persontype"/>

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

Eg of elements that contain only text

- This type contains only simple content (text and attributes), therefore we add a simpleContent element around the content. When using simple content, you must define an extension OR a restriction within the simpleContent element, like this:
- The following example declares a complexType, "shoesize". The content is defined as an integer value, and the "shoesize" element also contains an attribute named "country":

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

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

OR

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

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

```
<xs:element name="shoesize" type="shoetype"/>

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

- **Eg of mixed content**
- An XML element, "letter", that contains both text and other elements:
- The following schema declares the "letter" element:
- Note: To enable character data to appear between the child-elements of "letter", the mixed attribute must be set to "true". The <xs:sequence> tag means that the elements defined (name, orderid and shipdate) must appear in that order inside a "letter" element.
- We could also give the complexType element a name, and let the "letter" element have a type attribute that refers to the name of the complexType (if you use this method, several elements can refer to the same complex type):

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

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

```
<xs:element name="letter" type="lettertype"/>

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

XSD Indicators

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

There are seven indicators:

Order indicators:

- All
- Choice
- Sequence

Occurrence indicators:

- maxOccurs
- minOccurs

Group indicators:

- Group name
- attributeGroup name

```
<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>
```

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

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

- Note: When using the <all> indicator you can set the <minOccurs> indicator to 0 or 1 and the <maxOccurs> indicator can only be set to 1.
- **Choice Indicator**
- The <choice> indicator specifies that either one child element or another can occur:
- **Sequence Indicator**
- The <sequence> indicator specifies that the child elements must appear in a specific order:

```
<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>
```

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

Occurrence Indicators

- Occurrence indicators are used to define how often an element can occur.
- Note: For all "Order" and "Group" indicators (any, all, choice, sequence, group name, and group reference) the default value for maxOccurs and minOccurs is 1.
- **maxOccurs Indicator**
- The <maxOccurs> indicator specifies the maximum number of times an element can occur:
- **minOccurs Indicator**
- The <minOccurs> indicator specifies the minimum number of times an element can occur:
- The example above indicates that the "child_name" element can occur a minimum of zero times and a maximum of ten times in the "person" element.

```
<xs:element name="person">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="full_name" type="xs:string"/>
      <xs:element name="child_name" type="xs:string"
      maxOccurs="10" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Group Indicators

Group indicators are used to define related sets of elements.

- **Element Groups**
- Element groups are defined with the group declaration, like thi

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

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

```
<xs:group name="persongroup">
  <xs:sequence>
    <xs:element name="firstname" type="xs:string"/>
    <xs:element name="lastname" type="xs:string"/>
    <xs:element name="birthday" type="xs:date"/>
  </xs:sequence>
</xs:group>
```

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

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

The following example defines an attribute group named "personattrgroup":

```
<xs:attributeGroup name="personattrgroup">
  <xs:attribute name="firstname" type="xs:string"/>
  <xs:attribute name="lastname" type="xs:string"/>
  <xs:attribute name="birthday" type="xs:date"/>
</xs:attributeGroup>
```

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

```
<xs:attributeGroup name="personattrgroup">
  <xs:attribute name="firstname" type="xs:string"/>
  <xs:attribute name="lastname" type="xs:string"/>
  <xs:attribute name="birthday" type="xs:date"/>
</xs:attributeGroup>

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