Day 3 -XML

July 20th 2012

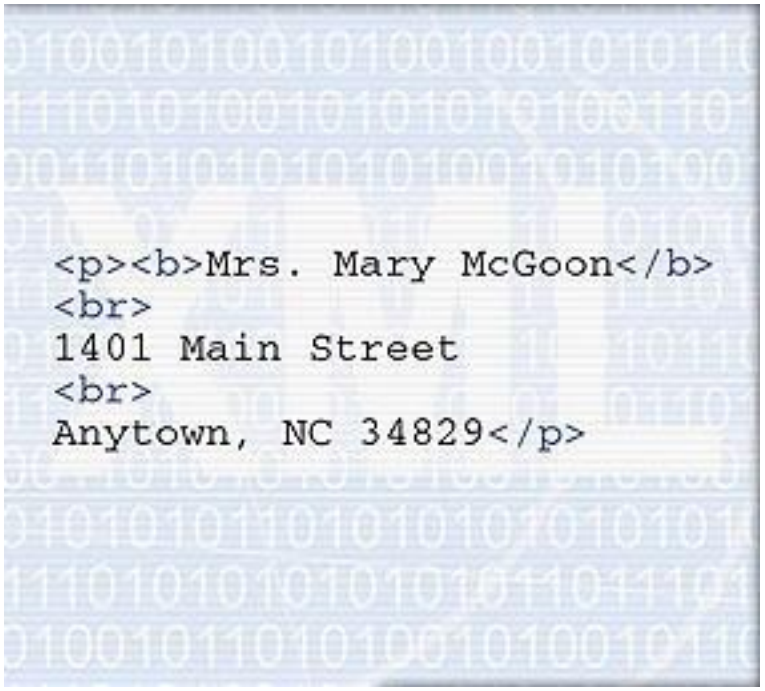
XML

Introduction

- XML stands for e**X**tensible **M**arkup **L**anguage.
- XML is
  - used to hold and store the data.
  - a World Wide Web (w3) consortium recommendation.
  - license-free, platform-independent and well supported
  - universally adopted, thereby assuring long-term support and a continually growing pool of technical expertise
- It offers an opportunity to create sophisticated solutions that are *data driven*.

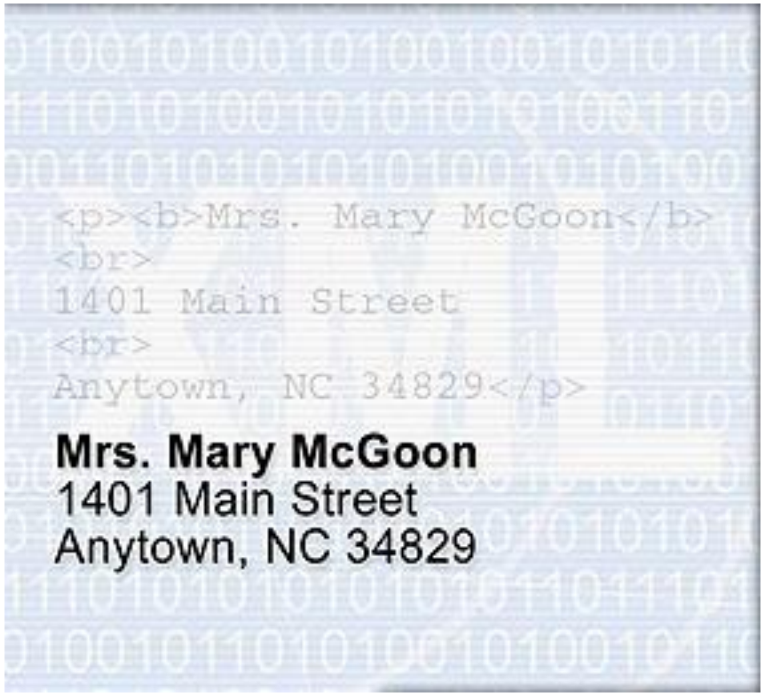
Why do we need XML?

.. when every one’s browser supports HTML?



```
<p><b>Mrs. Mary McGoon</b>
<br>
1401 Main Street
<br>
Anytown, NC 34829</p>
```

Data in HTML



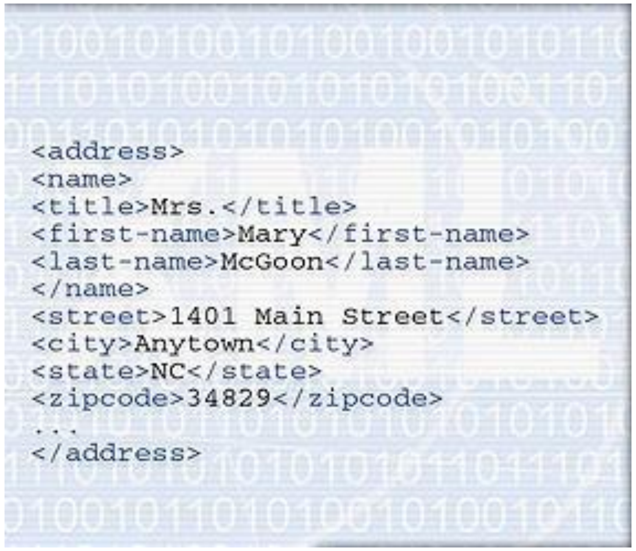
```
<p><b>Mrs. Mary McGoon</b>
<br>
1401 Main Street
<br>
Anytown, NC 34829</p>
```

Mrs. Mary McGoon
1401 Main Street
Anytown, NC 34829

HTML Rendering

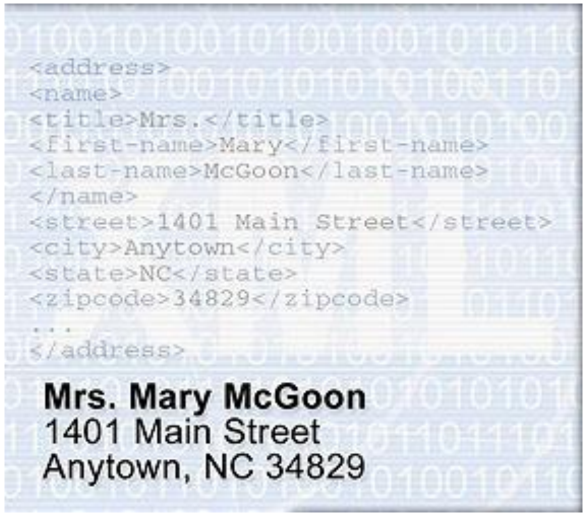
Why do we need XML?

.. when every one's browser supports HTML?



```
<address>
<name>
<title>Mrs.</title>
<first-name>Mary</first-name>
<last-name>McGoon</last-name>
</name>
<street>1401 Main Street</street>
<city>Anytown</city>
<state>NC</state>
<zipcode>34829</zipcode>
...
</address>
```

Sample XML code



```
<address>
<name>
<title>Mrs.</title>
<first-name>Mary</first-name>
<last-name>McGoon</last-name>
</name>
<street>1401 Main Street</street>
<city>Anytown</city>
<state>NC</state>
<zipcode>34829</zipcode>
...
</address>
```

Mrs. Mary McGoon
1401 Main Street
Anytown, NC 34829

XML Rendering

Mrs. Mary McGoon
1401 Main Street, Anytown, NC 34829
Mrs. Mary McGoon
1401 Main Street
Anytown, NC 34829

Second XML Rendering

Difference Between XML and HTML

- XML is not a replacement for HTML.
- XML and HTML were designed with different goals:

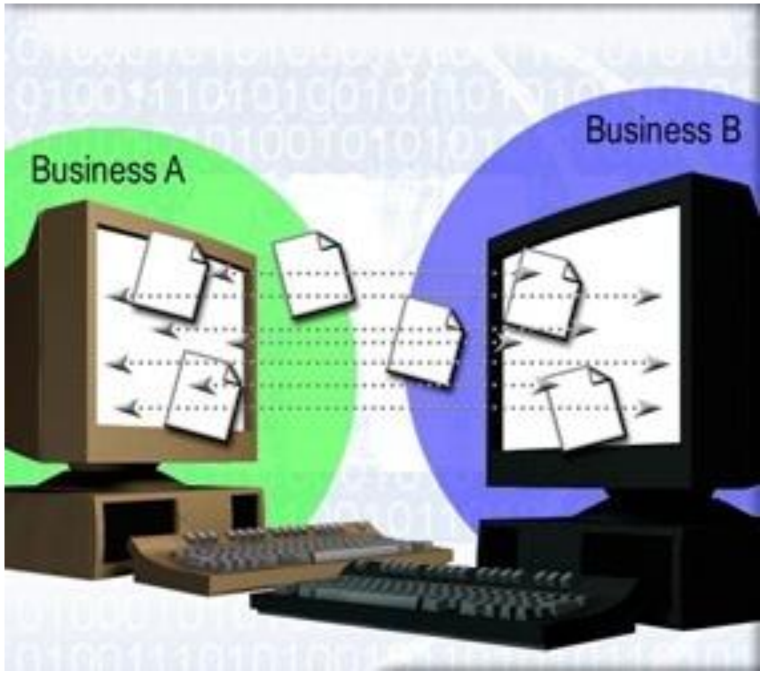
  XML was designed to transport and store data, with focus on what data is.
  HTML was designed to display data, with focus on how data looks.
- HTML is about displaying information, while XML is about carrying information.
- It allows creation of user defined tags. This feature allows to give structure and meaning to data.

Benefits of using XML

- It is web language for data interchange

Benefits of using XML

- Enable Business to Business Communication



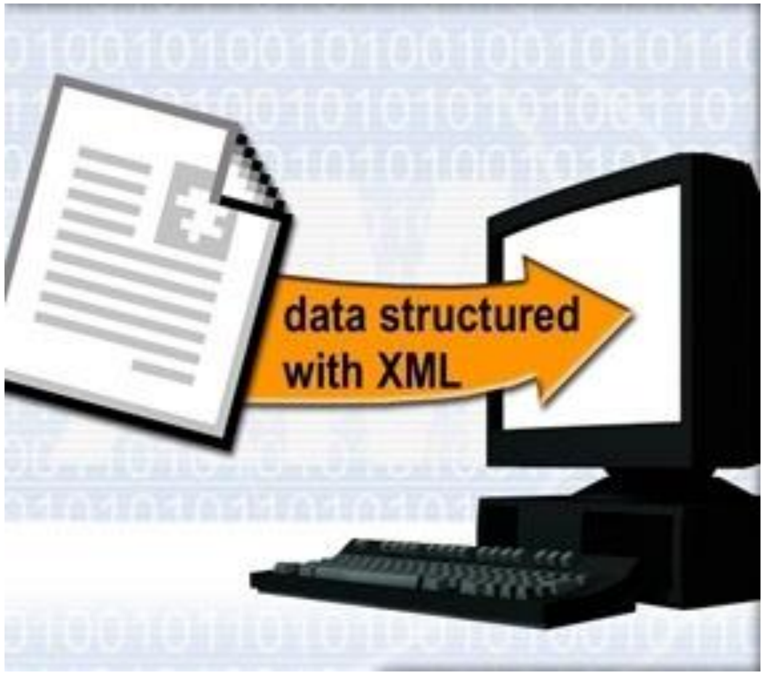
Benefits of using XML

- Enable Smart Searches

<first-name>Chip</first-name>

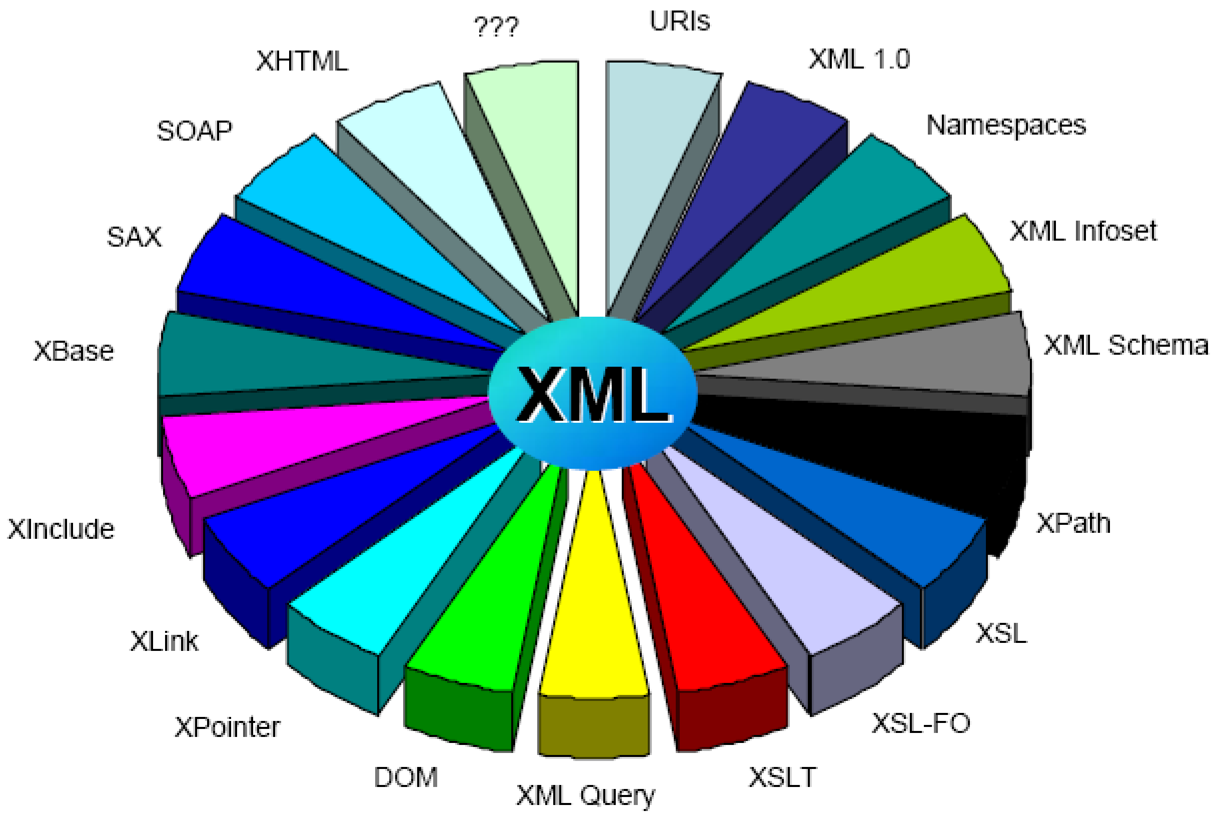
Benefits of using XML

- **Enable Smart Agents**



L&T Infotech

XML = eXtensible Meta Language



L&T Infotech

See it real

```
<?xml version="1.0"?>
<novel>
  <foreword>
         <paragraph> This is a great novel </paragraph>
  </foreword>
  <chapter number="1">
     <paragraph>It was a dark and stormy night. </paragraph>
          <paragraph>Suddenly, a shot rang out! </paragraph>
</chapter>
</novel>
```

XML SYNTAX RULES

L&T Infotech

Syntax

1. All XML documents must have closing tags.

```
<p> We are learning XML.
```

HTML - OK
XML - Not OK

```
<p> We are learning XML.</p>
```

✓

2. Tags are case-sensitive

```
<p> We are learning XML.</P>
```

HTML - OK
XML - Not OK

```
<p> We are learning XML.</p>
```

✓

L&T Infotech

Syntax

3. Elements must be properly nested.

```
<p><i> We are learning XML. </p></i>
```

HTML – OK
XML - Not OK

```
<p><i> We are learning XML.</i></p>
```

✓

4. Attribute values must be quoted.

```
<span id=first> We are learning XML.</span>
```

HTML - OK
XML – Not OK

```
<span id="first"> We are learning XML.</span>
```

✓

L&T Infotech

Syntax

5. Entity references

```
<condition> if Salary < 25000 </condition>
```

HTML – OK
XML – Not

```
<condition> If Salary &lt; 25000 </condition>
```

✓

There are 5 predefined entity references in XML:

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

L&T Infotech

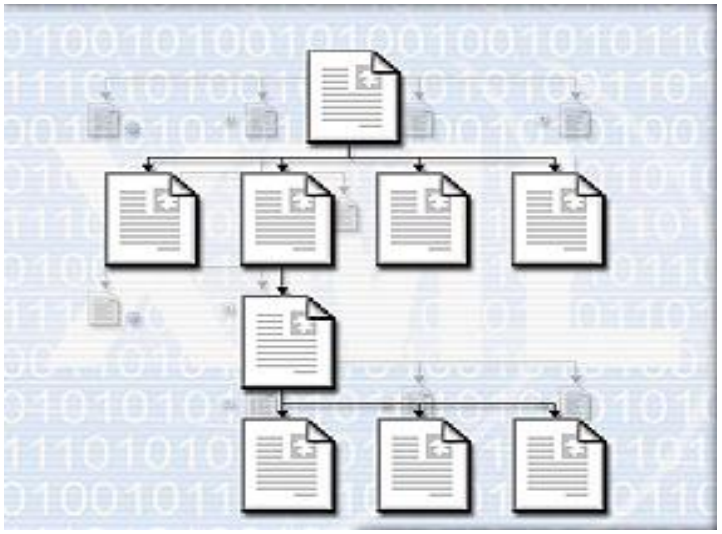
Syntax

6. White spaces are preserved.

HTML	XML tutorial.
Output	XML tutorial.

XML	XML tutorial.
Output	XML tutorial.

7. XML documents must have one and only one root element.



Parts of XML document

- Elements
- Attributes
- Entities
- Parsed Character Data (PCData)
- Character data (CData)
- Comments
- DTD

Element

- Element with content

```
<name>
    <first> John </first>
    <last> Smith </last>
</name>
```

- Empty Element

```
<name first="John" last="Smith" />
```

- Begin with a letter or underscore
- Elements beginning with 'XML' are reserved.
- Names can not contain spaces.

Attributes

- Provides additional information about elements

```
<name first=“John” last=‘Smith’ />
```

- Double or single quoted

```
<name fullname=‘Smith “John” ’ />
```

- Always a string

```
<case id=“23” />
```

L&T Infotech

Attributes vs. Elements: Best Practices

- Elements are best for
  - Hierarchy (Parent / Child relationship)
  - Containers
  - Sequencing
  - Content without constraints

- Attributes are best for
  - Modifying information
  - Metadata (data about element)
  - Enumeration
  - Constrained values or types

- **Note**

  If elements are like nouns, then attributes are like adjectives.

Entities

- An entity reference is precluded by a & sign. It is used to refer to a previously defined entity, much like a variable may be used in a program.

The following entities are predefined in XML:

Entity References	Character
&lt;	&lt;
&gt;	&gt;
&amp;	&amp;
&quot;	"
&apos;	'

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.
- 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 `&amp;` `&lt;` and `&gt;` entities, respectively.

Character Data (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.
- Used to enter text that contains special characters not displayed normally by the browser such as less than or greater than sign. These signs are used to enclose tags and are special characters.

Comments

- Same way as HTML comments
- May be included anywhere except inside an element tag (markup)

```
<!-- This is a comment and is not displayed by the browser or renderer. -->
```

L&T Infotech

Well formed XML

- Document has one and only one root element
- Elements must have a closing tag
- Element names are case sensitive
- Hierarchical nesting is enforced
- Attributes
  - Have quoted values
  - Attributes are only included on the start tag
  - No name duplicate within the single element
  - No ‘<’ (or equivalent) or external entities in the value

Types of XML documents

- Well formed
  - XML with correct syntax is “Well Formed” XML.

- Valid
  - XML validated against a DTD is “Valid” XML.
    - Must be a “well formed” document
    - conform to a DTD or schema

XML DTD

What is DTD ??

- The purpose of a DTD (Document Type Definition) is to define the legal building blocks of an XML document.
- A DTD defines the document structure with a list of legal elements and attributes.

DTD - Example

```
<!DOCTYPE NEWSPAPER [

    <!ELEMENT NEWSPAPER (ARTICLE+)>
    <!ELEMENT ARTICLE (HEADLINE,BYLINE,LEAD,BODY,NOTES)>
    <!ELEMENT HEADLINE (#PCDATA)>
    <!ELEMENT BYLINE (#PCDATA)>
    <!ELEMENT LEAD (#PCDATA)>
    <!ELEMENT BODY (#PCDATA)>
    <!ELEMENT NOTES (#PCDATA)>

    <!ATTLIST ARTICLE AUTHOR CDATA #REQUIRED>
    <!ATTLIST ARTICLE EDITOR CDATA #IMPLIED>
    <!ATTLIST ARTICLE DATE CDATA #IMPLIED>
    <!ATTLIST ARTICLE EDITION CDATA #IMPLIED>
]>
```

Introduction to 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 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]>
```

L&T Infotech

Internal DTD Declaration

- Example XML document with an internal DTD:

```
<?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>
```

L&T Infotech

Internal DTD Declaration

The DTD in previous example is interpreted like this:

- !DOCTYPE note defines that the root element of this 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"

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">
```

L&T Infotech

Internal DTD Declaration

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

```
<?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 this 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)>
```

Why Use a DTD?

- With a DTD, each of your XML files can carry a description of its own format.
- With a DTD, independent groups of people can agree to use a standard DTD for interchanging data.
- Your application can use a standard DTD to verify that the data you receive from the outside world is valid.
- You can also use a DTD to verify your own data.

DTD - XML Building Blocks

The main building blocks of both XML and HTML documents are elements.

- **The Building Blocks of XML Documents**

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

  - Elements
  - Attributes
  - Entities
  - PCDATA
  - CDATA

Elements

Elements are the **main building blocks** of both XML and HTML documents.

Examples of HTML elements are "body" and "table". Examples of XML elements could be "note" and "message". Elements can contain text, other elements, or be empty. Examples of empty HTML elements are "hr", "br" and "img".

Examples:

```
<body>some text</body>

<message>some text</message>
```

Attributes

Attributes provide **extra information about elements.**

Attributes are always placed inside the opening tag of an element. Attributes always come in name/value pairs. The following "img" element has additional information about a source file:

```
<img src="computer.gif" />
```

The name of the element is "img". The name of the attribute is "src". The value of the attribute is "computer.gif". Since the element itself is empty it is closed by a " /".

Entities

- Some characters have a special meaning in XML, like the less than sign (<) that defines the start of an XML tag.
- Most of you know the HTML entity: "&nbsp;". This "no-breaking-space" entity is used in HTML to insert an extra space in a document. Entities are expanded when a document is parsed by an XML parser.
- The following entities are predefined in XML:

Entity References	Character
&lt;	<
&gt;	>
&amp;	&
&quot;	"
&apos;	'

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.**
- 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 &amp; &lt; and &gt; entities, respectively.

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

In a DTD, elements are declared with an ELEMENT declaration.

- **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)>
```

DTD - Elements

- **Empty Elements**

  Empty elements are declared with the category keyword EMPTY:

```
<!ELEMENT element-name EMPTY>

Example:

<!ELEMENT br EMPTY>

XML example:

<br />
```

DTD - Elements

- **Elements with Parsed Character Data**

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

```
<!ELEMENT element-name (#PCDATA)>

Example:

<!ELEMENT from (#PCDATA)>
```

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>
```

DTD - Elements

- **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 note (to,from,heading,body)>
```

L&T Infotech

DTD - Elements

- **Elements with Children (sequences)**

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

L&T Infotech

DTD - Elements

- **Declaring Only One Occurrence of an Element**

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

Example:

<!ELEMENT note (message)>
```

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

DTD - Elements

- **Declaring Minimum One Occurrence of an Element**

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

Example:

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

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

L&T Infotech

DTD - Elements

- **Declaring Zero or More Occurrences of an Element**

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

Example:

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

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

L&T Infotech

DTD - Elements

- **Declaring Zero or One Occurrences of an Element**

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

Example:

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

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

DTD - Elements

- **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.

DTD - Elements

- **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

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

- **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" />
```

DTD - Attributes

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

DTD - Attributes

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 not required
#FIXED value	The attribute value is fixed

L&T Infotech

DTD - Attributes

- **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.

DTD - Attributes

#REQUIRED

Syntax

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

Example:

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

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

Invalid XML:
<person />
```

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 - Attributes

#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.

L&T Infotech

DTD - Attributes

#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.

DTD - Attributes

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 Elements vs. Attributes

In XML, there are no rules about when to use attributes, and when to use child elements.

L&T Infotech

XML Elements vs. Attributes

Use of Elements vs. Attributes

- Data can be stored in child elements or in attributes.

  Take a look at these examples:

```
<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 a child element. Both examples provide the same information.

There are no rules about when to use attributes, and when to use child elements. My experience is that attributes are handy in HTML, but in XML you should try to avoid them. Use child elements if the information feels like data.

XML Elements vs. Attributes

My Favorite Way(?)

- **I like to store data in child elements.**
- The following three XML documents contain exactly the same information:
- A date attribute is used in the first example:

```
<note date="12/11/2002">
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

XML Elements vs. Attributes

A date element is used in the second example:

```
<note>
  <date>12/11/2002</date>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

XML Elements vs. Attributes

An expanded date element is used in the third: (THIS IS MY FAVORITE):

```
<note>
  <date>
    <day>12</day>
    <month>11</month>
    <year>2002</year>
  </date>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

XML Elements vs. Attributes

Avoid using attributes?

- Should you avoid using attributes?

Some of the problems with attributes are:

- attributes cannot contain multiple values (child elements can)
- attributes are not easily expandable (for future changes)
- attributes cannot describe structures (child elements can)
- attributes are more difficult to manipulate by program code
- attribute values are not easy to test against a DTD

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.

XML Elements vs. Attributes

Don't end up like this (this is not how XML should be used):

```
<note day="12" month="11" year="2002“ to="Tove" from="Jani"
heading="Reminder“ body="Don't forget me this weekend!">
</note>
```

XML Elements vs. Attributes

An Exception to my Attribute Rule

- Rules always have exceptions.
- My rule about attributes has one exception:
- Sometimes I assign ID references to elements. These ID references can be used to access XML elements in much the same way as the NAME or ID attributes in HTML. This example demonstrates this:

```
<messages>
<note id="p501">
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>

<note id="p502">
  <to>Jani</to>
  <from>Tove</from>
  <heading>Re: Reminder</heading>
  <body>I will not!</body>
</note>
</messages>
```

XML Elements vs. Attributes

- The ID in these examples is just a counter, or a unique identifier, to identify the different notes in the XML file, and not a part of the note data.
- What I am trying to say here is that metadata (data about data) should be stored as attributes, and that data itself should be stored as elements.

DTD - 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

L&T Infotech

DTD - Entities

An Internal Entity Declaration

Syntax

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

Example

DTD Example:

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

XML example:

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

Note: An entity has three parts: an ampersand (&), an entity name, and a semicolon (;).

DTD - 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>
```

XML Schema

XML Schema

- An XML Schema describes the structure of an XML document.
- In this tutorial you will learn how to create XML Schemas, why XML Schemas are more powerful than DTDs, and how to use XML Schema in your application.

XML Schema

XML Schema Example

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

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

</xs:schema>
```

Introduction to 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).

What You Should Already Know

- Before you continue you should have a basic understanding of the following:
  - HTML / XHTML
  - XML and XML Namespaces
  - A basic understanding of DTD

Introduction to XML Schema

What is an XML Schema?

- 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

Introduction to XML Schema

XML Schemas are the Successors of DTDs

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

XML Schema is a W3C Recommendation

- XML Schema became a W3C Recommendation 02 May 2001.

L&T Infotech

Why Use XML Schemas?

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

L&T Infotech

Why Use XML Schemas?

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

Why Use XML Schemas?

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".

L&T Infotech

Why Use XML Schemas?

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

Why Use XML Schemas?

Well-Formed is not Enough

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

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

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

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

L&T Infotech

XSD How To?

A Simple XML Document

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

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

L&T Infotech

XSD How To?

A DTD File

The following example is a DTD file called "note.dtd" that defines the elements of the XML document above ("note.xml"):

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

- The first line defines the note element to have four child elements: "to, from, heading, body".
- Line 2-5 defines the to, from, heading, body elements to be of type "#PCDATA".

L&T Infotech

XSD How To?

An XML Schema

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

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

<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>
```

XSD How To?

A Reference to an XML Schema

This XML document has a reference to an XML Schema:

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

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

XSD - The <schema> Element

The <schema> Element

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

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

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

XSD - The <schema> Element

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>
```

XSD - The <schema> Element

The following fragment:

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

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

XSD - The <schema> Element

This fragment:

```
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.

This fragment:

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

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

This fragment:

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

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

XSD - The <schema> Element

Referencing a Schema in an XML Document

This XML document has a reference to an XML Schema:

```
<?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 note.xsd">

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

XSD - The <schema> Element

The following fragment:

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

specifies the default namespace declaration. This declaration tells the schema-validator that all the elements used in this XML document are declared in the "http://www.w3schools.com" namespace.

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. 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.

XSD Simple Elements

What is a Simple Element?

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

XSD Simple Elements

What is a Simple Element?

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

XSD Simple Elements

Defining a Simple Element

The syntax for defining a simple element is:

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

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

- XML Schema has a lot of built-in data types. The most common types are:
  - xs:string
  - xs:decimal
  - xs:integer
  - xs:boolean
  - xs:date
  - xs:time

XSD String Data Types

String data types are used for values that contains character strings.

String Data Type

- The string data type can contain characters, line feeds, carriage returns, and tab characters.
- The following is an example of a string declaration in a schema:

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

An element in your document might look like this:

```
<customer>John Smith</customer>
```

Or it might look like this:

```
<customer>      John Smith     </customer>
```

Note: The XML processor will not modify the value if you use the string data type.

XSD String Data Types

NormalizedString Data Type

- The normalizedString data type is derived from the String data type.
- The normalizedString data type also contains characters, but the XML processor will remove line feeds, carriage returns, and tab characters.

The following is an example of a normalizedString declaration in a schema:

```
<xs:element name="customer" type="xs:normalizedString"/>
```

An element in your document might look like this:

```
<customer>John Smith</customer>
```

Or it might look like this:

```
<customer>	John Smith	</customer>
```

Note: In the example above the XML processor will replace the tabs with spaces.

L&T Infotech

XSD String Data Types

Token Data Type

- The token data type is also derived from the String data type.
- The token data type also contains characters, but the XML processor will remove line feeds, carriage returns, tabs, leading and trailing spaces, and multiple spaces.

The following is an example of a token declaration in a schema:

```
<xs:element name="customer" type="xs:token"/>
```

An element in your document might look like this:

```
<customer>John Smith</customer>
```

Or it might look like this:

```
<customer>     John Smith     </customer>
```

Note: In the example above the XML processor will remove the tabs.

XSD Date and Time Data Types

Date and time data types are used for values that contain date and time.

Date Data Type

The date data type is used to specify a date.

The date is specified in the following form "YYYY-MM-DD" where:

- YYYY indicates the year
- MM indicates the month
- DD indicates the day

Note: All components are required!

The following is an example of a date declaration in a schema:

```
<xs:element name="start" type="xs:date"/>
```

An element in your document might look like this:

```
<start>2002-09-24</start>
```

XSD Date and Time Data Types

Time Zones

- To specify a time zone, you can either enter a date in UTC time by adding a "Z" behind the date - like this:

```
<start>2002-09-24Z</start>
```

- or you can specify an offset from the UTC time by adding a positive or negative time behind the date - like this:

```
<start>2002-09-24-06:00</start>

or

<start>2002-09-24+06:00</start>
```

L&T Infotech

XSD Date and Time Data Types

Time Data Type

- The time data type is used to specify a time.
- The time is specified in the following form "hh:mm:ss" where:
  - hh indicates the hour
  - mm indicates the minute
  - ss indicates the second

Note: All components are required!

- The following is an example of a time declaration in a schema:

```
<xs:element name="start" type="xs:time"/>
```

- An element in your document might look like this:

```
<start>09:00:00</start>
```

- Or it might look like this:

```
<start>09:30:10.5</start>
```

XSD Date and Time Data Types

Time Zones

- To specify a time zone, you can either enter a time in UTC time by adding a "Z" behind the time - like this:

```
<start>09:30:10Z</start>
```

- or you can specify an offset from the UTC time by adding a positive or negative time behind the time - like this:

```
<start>09:30:10-06:00</start>

or

<start>09:30:10+06:00</start>
```

XSD Date and Time Data Types

DateTime Data Type

- The dateTime data type is used to specify a date and a time.
- The dateTime is specified in the following form "YYYY-MM-DDThh:mm:ss" where:
  - YYYY indicates the year
  - MM indicates the month
  - DD indicates the day
  - T indicates the start of the required time section
  - hh indicates the hour
  - mm indicates the minute
  - ss indicates the second
- **Note:** All components are required!

XSD Date and Time Data Types

- The following is an example of a dateTime declaration in a schema:

```
<xs:element name="startdate" type="xs:dateTime"/>
```

- An element in your document might look like this:

```
<startdate>2002-05-30T09:00:00</startdate>
```

- Or it might look like this:

```
<startdate>2002-05-30T09:30:10.5</startdate>
```

XSD Date and Time Data Types

Time Zones

- To specify a time zone, you can either enter a dateTime in UTC time by adding a "Z" behind the time - like this:

```
<startdate>2002-05-30T09:30:10Z</startdate>
```

- or you can specify an offset from the UTC time by adding a positive or negative time behind the time - like this:

```
<startdate>2002-05-30T09:30:10-06:00</startdate>

or

<startdate>2002-05-30T09:30:10+06:00</startdate>
```

XSD Date and Time Data Types

Duration Data Type

- The duration data type is used to specify a time interval.
- The time interval is specified in the following form "PnYnMnDTnHnMnS" where:
  - P indicates the period (required)
  - nY indicates the number of years
  - nM indicates the number of months
  - nD indicates the number of days
  - T indicates the start of a time section (required if you are going to specify hours, minutes, or seconds)
  - nH indicates the number of hours
  - nM indicates the number of minutes
  - nS indicates the number of seconds

L&T Infotech

XSD Date and Time Data Types

The following is an example of a duration declaration in a schema:

```
<xs:element name="period" type="xs:duration"/>
```

An element in your document might look like this:

```
<period>P5Y</period>
```

The example above indicates a period of five years.

Or it might look like this:

```
<period>P5Y2M10D</period>
```

The example above indicates a period of five years, two months, and 10 days.

Or it might look like this:

```
<period>P5Y2M10DT15H</period>
```

The example above indicates a period of five years, two months, 10 days, and 15 hours.

Or it might look like this:

```
<period>PT15H</period>
```

The example above indicates a period of 15 hours.

XSD Date and Time Data Types

Negative Duration

- To specify a negative duration, enter a minus sign before the P:

```
<period>-P10D</period>
```

- The example above indicates a period of minus 10 days.

XSD Date and Time Data Types

Date and Time Data Types

Name	Description
date	Defines a date value
dateTime	Defines a date and time value
duration	Defines a time interval
gDay	Defines a part of a date - the day (DD)
gMonth	Defines a part of a date - the month (MM)
gMonthDay	Defines a part of a date - the month and day (MM-DD)
gYear	Defines a part of a date - the year (YYYY)
gYearMonth	Defines a part of a date - the year and month (YYYY-MM)
time	Defines a time value

XSD Numeric Data Types

Decimal Data Type

- The decimal data type is used to specify a numeric value.
- The following is an example of a decimal declaration in a schema:

```
<xs:element name="prize" type="xs:decimal"/>
```

- An element in your document might look like this:

```
<prize>999.50</prize>
```

- Or it might look like this:

```
<prize>+999.5450</prize>
```

- Or it might look like this:

```
<prize>-999.5230</prize>
```

XSD Numeric Data Types

- Or it might look like this:

```
<prize>0</prize>
```

- Or it might look like this:

```
<prize>0</prize>
```

- Or it might look like this:

```
<prize>14</prize>
```

Note: The maximum number of decimal digits you can specify is 18.

XSD Numeric Data Types

Integer Data Type

- The integer data type is used to specify a numeric value without a fractional component.
- The following is an example of an integer declaration in a schema:

```
<xs:element name="prize" type="xs:integer"/>
```

- An element in your document might look like this:

```
<prize>999</prize>
```

- Or it might look like this:

```
<prize>+999</prize>
```

- Or it might look like this:

```
<prize>-999</prize>
```

- Or it might look like this:

```
<prize>0</prize>
```

L&T Infotech

XSD Numeric Data Types

Numeric Data Types

Note that all of the data types below derive from the Decimal data type (except for decimal itself)!

Name	Description
byte	A signed 8-bit integer
decimal	A decimal value
int	A signed 32-bit integer
integer	An integer value
long	A signed 64-bit integer
negativeInteger	An integer containing only negative values (..,-2,-1)
nonNegativeInteger	An integer containing only non-negative values (0,1,2,..)
nonPositiveInteger	An integer containing only non-positive values (..,-2,-1,0)
positiveInteger	An integer containing only positive values (1,2,..)
short	A signed 16-bit integer
unsignedLong	An unsigned 64-bit integer
unsignedInt	An unsigned 32-bit integer
unsignedShort	An unsigned 16-bit integer
unsignedByte	An unsigned 8-bit integer

XSD Miscellaneous Data Types

Other miscellaneous data types are boolean, base64Binary, hexBinary, float, double, anyURI, QName, and NOTATION.

Boolean Data Type

- The boolean data type is used to specify a true or false value.
- The following is an example of a boolean declaration in a schema:

```
<xs:attribute name="disabled" type="xs:boolean"/>
```

An element in your document might look like this:

```
<prize disabled="true">999</prize>
```

Note: Legal values for boolean are true, false, 1 (which indicates true), and 0 (which indicates false).

XSD Miscellaneous Data Types

Binary Data Types

- Binary data types are used to express binary-formatted data.
- We have two binary data types:
  - base64Binary (Base64-encoded binary data)
  - hexBinary (hexadecimal-encoded binary data)
- The following is an example of a hexBinary declaration in a schema:

```
<xs:element name="blobsrc" type="xs:hexBinary"/>
```

XSD Miscellaneous Data Types

AnyURI Data Type

- The anyURI data type is used to specify a URI.
- The following is an example of an anyURI declaration in a schema:

```
<xs:attribute name="src" type="xs:anyURI"/>
```

- An element in your document might look like this:

```
<pic src="http://www.w3schools.com/images/smiley.gif" />
```

Note: If a URI has spaces, replace them with %20.

XSD Simple Elements

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"/>
```

L&T Infotech

XSD Simple Elements

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

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.

L&T Infotech

XSD Attributes

How to Define an Attribute?

The syntax for defining an attribute is:

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

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

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

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

XSD Attributes

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"/>
```

L&T Infotech

XSD Attributes

Default and Fixed Values for Attributes

- Attributes may have a default value OR a fixed value specified.
- A default value is automatically assigned to the attribute when no other value is specified.
- In the following example the default value is "EN":

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

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

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

L&T Infotech

XSD Attributes

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 Attributes

Restrictions on Content

- When an XML element or attribute has a data type defined, it puts restrictions on the element's or attribute's content.
- If an XML element is of type "xs:date" and contains a string like "Hello World", the element will not validate.
- With XML Schemas, you can also add your own restrictions to your XML elements and attributes. These restrictions are called facets. You can read more about facets in the next chapter.

XSD Restrictions/Facets

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

XSD Restrictions/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.

The example below defines an element called "car" with a restriction. The only acceptable values are: Audi, Golf, BMW:

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

XSD Restrictions/Facets

The last example could also have been written like this:

```
<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>
```

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

L&T Infotech

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 a to z:

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

XSD Restrictions/Facets

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

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

XSD Restrictions/Facets

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

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

XSD Restrictions/Facets

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

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

L&T Infotech

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>
```

XSD Restrictions/Facets

Other Restrictions on a Series of Values

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

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

XSD Restrictions/Facets

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

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

XSD Restrictions/Facets

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

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

XSD Restrictions/Facets

- The next example defines an element called "password" with a restriction. There must be exactly eight characters in a row and those characters must be lowercase or uppercase letters from a to z, or a number from 0 to 9:

```
<xs:element name="password">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="[a-zA-Z0-9]{8}"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

XSD Restrictions/Facets

Restrictions on Whitespace Characters

- To specify how whitespace characters should be handled, we would use the whiteSpace constraint.
- This example defines an element called "address" with a restriction. The whiteSpace constraint is set to "preserve", which means that the XML processor WILL NOT remove any white space characters:

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

XSD Restrictions/Facets

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

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

XSD Restrictions/Facets

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

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

XSD Restrictions/Facets

Restrictions on Length

- To limit the length of a value in an element, we would use the length, maxLength, and minLength constraints.
- This example defines an element called "password" with a restriction. The value must be exactly eight characters:

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

XSD Restrictions/Facets

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

```
<xs:element name="password">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:minLength value="5"/>
      <xs:maxLength value="8"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

XSD Restrictions/Facets

Restrictions for Datatypes

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

XSD Complex Elements

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

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

XSD Complex Elements

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>
```

XSD Complex Elements

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

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

XSD Complex Elements

How to Define a Complex Element

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

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

L&T Infotech

XSD Complex 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:

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

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

XSD Complex Elements

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

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

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

XSD Complex Elements

If you use the method described in previous slide, several elements can refer to the same complex type, 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>
```

XSD Complex Elements

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

```
<xs:element name="employee" type="fullpersoninfo"/>

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

<xs:complexType name="fullpersoninfo">
  <xs:complexContent>
    <xs:extension base="personinfo">
      <xs:sequence>
        <xs:element name="address" type="xs:string"/>
        <xs:element name="city" type="xs:string"/>
        <xs:element name="country" type="xs:string"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

L&T Infotech

XSD Empty Elements

An empty complex element cannot have contents, only attributes.

Complex Empty Elements

- An empty XML element:

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

XSD Empty Elements

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

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

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

XSD Empty Elements

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

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

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

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

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

XSD Elements Only

An "elements-only" complex type contains an element that contains only other elements.

Complex Types Containing Elements Only

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

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

XSD Elements Only

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

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

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

XSD Elements Only

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>
```

L&T Infotech

XSD Text-Only Elements

Complex Text-Only Elements

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

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

XSD Text-Only Elements

OR

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

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

XSD Text-Only Elements

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

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

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

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

L&T Infotech

XSD Text-Only Elements

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>
```

XSD Mixed Content

A mixed complex type element can contain attributes, elements, and text.

Complex Types with Mixed Content

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

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

XSD Mixed Content

The following schema declares the "letter" element:

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

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

L&T Infotech

XSD Mixed Content

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

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

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

XSD Indicators

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

Indicators

There are seven indicators:

- Order indicators:
  - All
  - Choice
  - Sequence
- Occurrence indicators:
  - maxOccurs
  - minOccurs
- Group indicators:
  - Group name
  - attributeGroup name

XSD Indicators

Order Indicators

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

All Indicator

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

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

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

L&T Infotech

XSD Indicators

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>
```

XSD Indicators

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>
```

XSD Indicators

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.

XSD Indicators

maxOccurs Indicator

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

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

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

L&T Infotech

XSD Indicators

minOccurs Indicator

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

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

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

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

XSD Indicators

A working example:

- An XML file called "Myfamily.xml":

```
<?xml version="1.0"?>
<persons xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="family.xsd">
<person>
  <full_name>Hege Refsnes</full_name>
  <child_name>Cecilie</child_name>
</person>
<person>
  <full_name>Tove Refsnes</full_name>
  <child_name>Hege</child_name>
  <child_name>Stale</child_name>
  <child_name>Jim</child_name>
  <child_name>Borge</child_name>
</person>
<person>
  <full_name>Stale Refsnes</full_name>
</person>
</persons>
```

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

L&T Infotech

XSD Indicators

- Here is the schema file "family.xsd":

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

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

XSD Indicators

Group Indicators

- Group indicators are used to define related sets of elements.

Element Groups

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

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

XSD Indicators

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>
```

L&T Infotech

XSD Indicators

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

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

<xs:element name="person" type="personinfo"/>

<xs:complexType name="personinfo">
  <xs:sequence>
    <xs:group ref="persongroup"/>
    <xs:element name="country" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
```

L&T Infotech

XSD Indicators

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>
```

XSD Indicators

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>
```

Thank You

Our Business Knowledge,
Your Winning Edge.