# XML

TLS

Rewards and Recognition

2012 ASTD BEST AWARD WINNER

training TOP 125 Winner - 2013

Tata Institute of Social Sciences CLO Awards India Best Virtual Learning Award

## Objectives

At the end of this session, you will be able to,

- Write XML document
- Write XSD and confirm XML document to a XSD

## Agenda

Following topics are covered during the session,

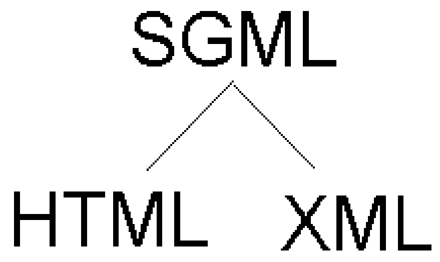
- Introduction to XML
- XML Syntax (well formed XML)
- XML Namespace
- XSD

## What is XML?

- XML stands for EXtensible Markup Language
- XML is a markup language much like HTML
- XML was designed to describe data
- XML tags are not predefined. You must define your own tags

## What is XML? (Contd…)

- XML is a W3C Recommendation
- Based on Standard Generalized Markup Language (SGML)



- XML uses a Document Type Definition (DTD) or an XML Schema (XSD) to provide grammar for the data

## XML Versions

- XML 1.0 (First Edition)
  - XML 1.0 was released as a W3C Recommendation 10, February 1998

- XML 1.0 (Second Edition)
  - XML 1.0 (SE) was released as a W3C Recommendation 6, October 2000
    Second Edition is only a correction to XML 1.0 that incorporates the first-edition errata (bug fixes)

- XML 1.0 (Third Edition)
  - Third Edition is only a correction to XML 1.0 that incorporates the first- and second-edition errata (bug fixes)

- XML 1.1
  - XML 1.1 was released as a Working Draft 13, December 2001, and as a Candidate Recommendation 15, October 2002.
    XML 1.1 allows almost any Unicode characters to be used in names.

## How XML Differs from HTML ??

- HTML tags are predefined while as XML tags are user defined.

- XML and HTML were designed with different goals:
  - XML was designed to describe data and to focus on what data is
  - HTML was designed to display data and to focus on how data looks

- XML when used in conjuction with HTML, vastly extends the capability of web pages to:
  - Deliver virtually any type of document
  - Sort, filter, rearrange, find and manipulate the information
  - Present highly structured information

## XML Document

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

## How can XML be used?

- XML can separate data from HTML
  - With XML, your data is stored outside your HTML
- XML is used to exchange data
  - With XML, data can be exchanged between incompatible systems
- XML and B2B
  - With XML, information can be exchanged over the Internet
- XML can be used to create new languages
  - XML is the mother of WAP and WML
- XML is a cross-platform, software and hardware independent tool for transmitting information

## Anatomy of an XML document

- Three Parts
  - Prolog
  - Document Element [Root Element]
  - Epilog

## Prolog (Optional)

- **XML Declaration**
  - States that this is an XML Document. Must be the first line in the XML document.

    <?xml version="1.0" encoding="ISO-8859-1"?>

- **Document Type Declaration**
  - Defines the type and the structure of the document. If used, it must come after the XML declaration.

    <!DOCTYPE catalog SYSTEM “catalog.dtd”>  e.g. of external subset

- **Processing Instructions**
  - Provides information that the XML processor passes on to the application. There can be one or more processing instructions that can appear in prolog.

    <?xml-stylesheet type="text/css" href="cd_catalog.css"?>

- **Comments**

    <!-- Edited with XML Spy v4.2 -- >

## Document Element

- Also known as Root element.
- It contains the document’s information content.
- The element content can be character data, other (nested) elements, or combination of both.

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

- Document element is <note>. It contains 4 nested elements.

## Epilog (Optional)

- The area after the root element is known as epilog
- Some markup constructs like comments, white spaces can come after the root element
- This is not an official term used in XML standard

## Example

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE CATALOG SYSTEM "cd_catalog.dtd">
<?xml-stylesheet type="text/css" href="cd_catalog.css"?>

<!-- Edited with XML Spy v4.2 -->
<CATALOG>
      <CD>
        <TITLE>Empire Burlesque</TITLE>
        <ARTIST>Bob Dylan</ARTIST>
        <COUNTRY>USA</COUNTRY>
        <COMPANY>Columbia</COMPANY>
        <PRICE>10.90</PRICE>
        <YEAR>1985</YEAR>
      </CD>
      <CD>
        <TITLE>Unchain my heart</TITLE>
        <ARTIST>Joe Cocker</ARTIST>
        <COUNTRY>USA</COUNTRY>
        <COMPANY>EMI</COMPANY>
        <PRICE>8.20</PRICE>
        <YEAR>1987</YEAR>
      </CD>
</CATALOG>
<!-- Edited with XML Spy v4.2 -->
```

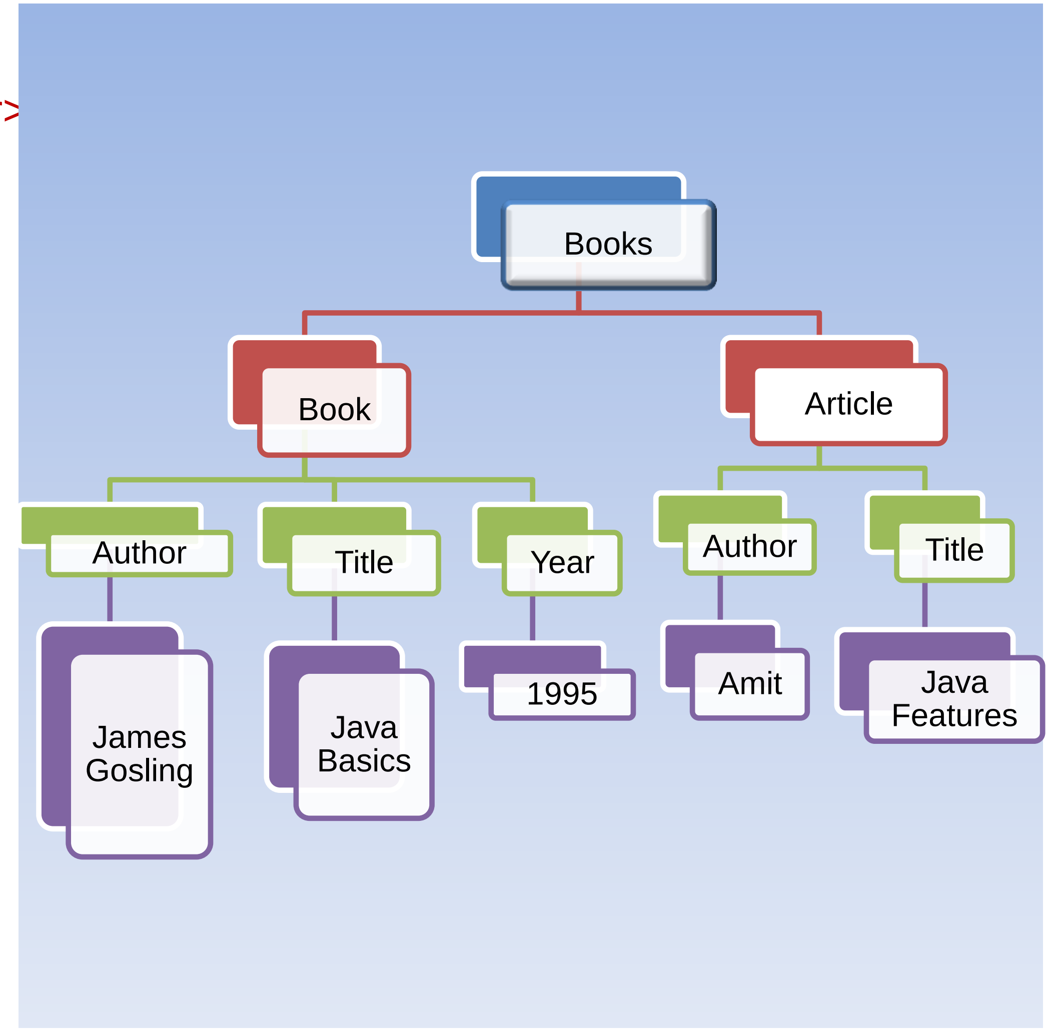
Prolog (Optional)

Document Element

Epilog (Optional)

## Example

```
<books>
    <book id="123" loc="lib">
        <author>James Gosling</author>
        <title>Java Basics</title>
        <year>1995</year>
    </book>
    <article id="555" ref="123">
        <author>Amit</author>
        <title>Java Features </title>
    </article>
</ books >
```



## XML Syntax

- All XML documents must have a root element

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

- All XML elements must have a closing tag

  e.g. `<p>This is a paragraph </p>`

- XML tags are case sensitive

  e.g. `<Message>This is incorrect</message>`

  `<message>This is correct</message>`

- All XML elements must be properly nested

  e.g. `<b><i>Invalid Nesting</b></i>`

  `<b><i>Valid Nesting</i></b>`

## XML Syntax (Contd...)

- Attribute values must always be quoted

  e.g.  <note date="12/11/2002"></note>

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

    e.g.   <!-- This is a comment -->

- Naming Rules
  - The name must begin with a letter or underscore, followed by zero or more letters, digits, periods, hyphens or underscores.
  - Names beginning with xml (any combination of upper and lower case) are reserved for standardization. Don’t use.
  - Names cannot contain spaces

## XML Elements

- XML Elements have relationships
  - Elements are related as parents and children

```
<book>
   <title>My First XML</title>
   <prod id="33-657" media="paper"></prod>
   <chapter>Introduction to XML
      <para>What is HTML</para>
      <para>What is XML</para>
   </chapter>
   <chapter>XML Syntax
      <prod id=“12-234” />
      <para>Elements must have a closing tag</para>
      <para>Elements must be properly nested</para>
   </chapter>
</book>
```

## XML Elements

- Elements have Content
  - An element can have
    - element content
    - mixed content
    - simple content
    - empty content
    - An element can also have attributes

TLS

## XML Attributes

- Attributes are used to provide additional information about elements

- XML elements can have attributes in the start tag, just like HTML. e.g.  <note type=”reminder”>

- Attributes are enclosed in single or double quotes

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

## Use of Elements vs. Attributes

- Example 1

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

- Example 2

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

## Use of Elements vs. Attributes

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

## Avoid using Attributes

- Some of the problems with using 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 Document Type Definition (DTD) - which is used to define the legal elements of an XML document

- 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

## When to use  Attribute

- Assigning 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>
</messages>
```

## Entity Reference

- There are 5 predefined entity references in XML

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

## XML CDATA

- All the text in an XML document is parsed by the parser.
  - The parser does this because XML elements can contain other elements

- Only the text inside a CDATA section is ignored by the parser

```
<script>
  <![CDATA[
    function matchwo(a,b)
    {
      if (a < b && a < 0) then
      {
        return 1
      }
      else
      {
        return 0
      }
    }
  ]]>
</script>
```

- Nested CDATA sections are not allowed

## “Well-formed” XML Documents

- XML with correct syntax is well-formed XML
  - All elements must have an end tag
  - All elements must be cleanly nested
  - All attribute values must be enclosed in quotation marks
  - Each document must have a unique first element, the root node

## XML Namespaces

- Name conflicts
  - This XML document carries information in a table:

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

  - This XML document carries information about a table (a piece of furniture):

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

## XML Namespaces (Contd…)

Solving Name Conflicts Using a Prefix

- This XML document carries information in a table:

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

- This XML document carries information about a piece of furniture:

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

## XML Namespaces (Contd…)

- XML Namespaces provide a method to avoid element name conflicts
  - This XML document carries information in a table:

```
<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>
```

- This XML document carries information about a piece of furniture:

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

## XML Namespaces – xmlns Attribute

- The XML namespace attribute is placed in the start tag of an element and has the following syntax:

  xmlns:namespace-prefix="namespaceURI“

  - All child elements with the same prefix are associated with the same namespace.

  - Imp: the address used to identify the namespace is not used by the parser to look up information. The only purpose is to give the namespace a unique name. However, very often companies use the namespace as a pointer to a real web page containing information about the namespace.

## XML Namespaces – Default Namespace

- Defining a default namespace for an element saves us from using prefixes in all the child elements.

  xmlns="namespaceURI“

- This XML document carries information in a table:

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

- This XML document carries information about a piece of furniture:

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

## Valid XML

- A "well formed" XML document is not the same as a "valid" XML document.
- A "valid" XML document must be well formed. In addition, it must conform to a document type definition.
- There are two different document type definitions that can be used with XML:
- DTD - The original Document Type Definition
- XML Schema - An XML-based alternative to DTD

- Validation of XML is always done against DTD / Schema by Parser
- Parser is a program that parses XML document and occasionally modifies it.
- Various parsers are available in market along with XML editors Eg. Altova XMLSpy, Eclipse, MSXML Parser, Expat XML parser etc.

## What is XML Parser

XML Parser provides way how to access or modify data present in an XML document. Java provides multiple options to parse XML document. Following are various types of parsers which are commonly used to parse XML documents.

- **Dom Parser** - Parses the document by loading the complete contents of the document and creating its complete hierarchical tree in memory.
- **SAX Parser** - Parses the document on event based triggers. Does not load the complete document into the memory.
- **JDOM Parser** - Parses the document in similar fashion to DOM parser but in more easier way.
- **StAX Parser** - Parses the document in similar fashion to SAX parser but in more efficient way.
- **XPath Parser** - Parses the XML based on expression and is used extensively in conjuction with XSLT.
- **DOM4J Parser** - A java library to parse XML, XPath and XSLT using Java Collections Framework , provides support for DOM, SAX and JAXP.
- There are JAXB and XSLT APIs available to handle XML parsing in Object Oriented way.

## XML Schemas

- “XML Schema” is an XML-based alternative to DTDs
- An XML Schema describes the structure of an XML document
- The XML Schema language is also referred to as XML Schema Definition (XSD)

## Schema Definition

- An XML Schema defines:
  - Elements that can appear in a document
  - Attributes that can appear in a document
  - Which elements are child elements
  - Order of child elements
  - Number of child elements
  - Whether an element is empty or can include text
  - Data types for elements and attributes
  - Default and fixed values for elements and attributes

## Why XML Schemas?

- DTDs provide weak specification language
  - We can not put any restriction on text content
  - We have very little control over mixed content (text plus elements)
  - We have little control over ordering of elements
- DTDs are written in a strange (non-XML) format
  - We need separate parsers for DTDs and XML
- The XML Schema Definition language solves these problems
  - XSD gives you much more control over structure and content
  - XSD is written in XML

## Referring to a Schema

- To refer to a DTD in an XML document, the reference goes before the root element:

```
<?xml version="1.0"?>
<!DOCTYPE rootElement SYSTEM "url">
<rootElement> ... </rootElement>
```

- To refer to an XML Schema in an XML document, the reference goes in the root element:

```
<?xml version="1.0"?>
<rootElement
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         (The XML Schema Instance reference is required)
 xsi:noNamespaceSchemaLocation="url.xsd">
          (This is where the XML Schema definition can be found)
  ...
</rootElement>
```

TLS

## The XSD Document

- Since the XSD is written in XML, it can get confusing which we are talking about
- Except for the additions to the root element of our XML data document, we will discuss about the XSD schema document
- The file extension is .xsd
- The root element is <schema>
- The XSD starts like this:
  - <?xml version="1.0"?>
    <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

## <schema>

- The <schema> element may have attributes:
  - xmlns:xs="http://www.w3.org/2001/XMLSchema"
    - This is necessary to specify where all our XSD tags are defined

## “Simple” and “Complex” Elements

- A “simple” element is one that contains text and nothing else
  - It cannot have attributes
  - It cannot contain other elements
  - It cannot be empty
  - However, the text can be of many different types, and may have various restrictions applied to it
- If an element is not simple, it is “complex”
  - A complex element may have attributes
  - It may be empty, or it may contain text, other elements, or both text and other elements

## Defining a simple element

- A simple element is defined as
  `<xs:element   name="`*name*`"   type="`*type*`" />`
  where:
  - *name* is the name of the element
  - the most common values for *type* are

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

- Other attributes a simple element may have:
  - default="*default value*"  if no other value is specified
  - fixed="*value*"  no other value may be specified

## Defining an Attribute

- Attributes themselves are always declared as simple types
- An attribute is defined as

  &lt;xs:attribute name="*name*" type="*type*" /&gt;

  where:
  - *name* and *type* are the same as for xs:element
- Other attributes a simple element may have:
  - default="*default value*" if no other value is specified
  - fixed="*value*" no other value may be specified
  - use="optional" the attribute is not required (default)
  - use="required" the attribute must be present

## Restrictions, or “facets”

- The general form for putting a restriction on a text value is:

```
<xs:element  name="name">                (or xs:attribute)
    <xs:restriction base="type">
        ... the restrictions ...
    </xs:restriction>
</xs:element>
```

- For example:

```
<xs:element  name="age">
    <xs:restriction base="xs:integer">
        <xs:minInclusive value="0">
        <xs:maxInclusive value="140">
    </xs:restriction>
</xs:element>
```

## Restrictions on Numbers

- minInclusive: number must be ≥ the given *value*
- minExclusive: number must be > the given *value*
- maxInclusive: number must be ≤ the given *value*
- maxExclusive: number must be < the given *value*
- totalDigits: number must have exactly *value* digits
- fractionDigits: number must have no more than *value* digits after the decimal point

## Restrictions on Strings

- Length: the string must contain exactly value characters
- minLength: the string must contain at least value characters
- maxLength: the string must contain no more than value characters
- Pattern: the value is a regular expression that the string must match
- whiteSpace: not really a “restriction”; tells what to do with whitespace
  - value="preserve“: Keeps all whitespace
  - value="replace“: Changes all whitespace characters to spaces
  - value="collapse“: Removes leading and trailing whitespace, and replaces all sequences of whitespace with a single space

## Enumeration

- An enumeration restricts the value to be one of a fixed set of values
- Example:

  -
    ```
    <xs:element name="season">
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:enumeration value="Spring"/>
          <xs:enumeration value="Summer"/>
          <xs:enumeration value="Autumn"/>
          <xs:enumeration value="Fall"/>
          <xs:enumeration value="Winter"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:element>
    ```

## Complex Elements

- A complex element is defined as

```
<xs:element  name="name">
    <xs:complexType>
        ... information about the complex type...
    </xs:complexType>
</xs:element>
```

- Example:

```
<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>
```

- Remember that attributes are always simple types

## Declaration and Use

- So far we've been talking about how to *declare* types, not how to *use* them
- To *use* a type we have declared, use it as the value of type="..."
  - Examples:
    - <xs:element name="student" type="person"/>
    - <xs:element name="professor" type="person"/>
  - Scope is important: you cannot use a type if is local to some other type

## xs:sequence

- We've already seen an example of a complex type whose elements must occur 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>
```

## Text element with attributes

- If a text element has attributes, it is no longer a simple type

```
<xs:element name="population">
        <xs:complexType>
             <xs:simpleContent>
                  <xs:extension  base="xs:integer">
                       <xs:attribute  name="year" type="xs:integer">
                  </xs:extension>
             </xs:simpleContent>
        </xs:complexType>
</xs:element>
```

## Predefined Date and Time Types

- xs:date: A date in the format *CCYY-MM-DD*, for example, 2002-11-05
- xs:time: A date in the format *hh:mm:ss* (hours, minutes, seconds)
- xs:dateTime: Format is *CCYY-MM-DD*T*hh:mm:ss*
- Allowable restrictions on dates and times:
  - enumeration, minInclusive, maxExclusive, maxInclusive, maxExclusive, pattern, whiteSpace

## Predefined Numeric Types

- Some of the predefined numeric types:

| | |
|---|---|
| xs:decimal | xs:positiveInteger |
| xs:byte | xs:negativeInteger |
| xs:short | xs:nonPositiveInteger |
| xs:int | xs:nonNegativeInteger |
| xs:long | |

- Allowable restrictions on numeric types:
  - enumeration, minInclusive, maxExclusive, maxInclusive, maxExclusive, fractionDigits, totalDigits, pattern, whiteSpace

## Sample XSD

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
        <xs:element name="customername" type="xs:string"/>
        <xs:element name="salesperson" type="xs:string"/>
    <xs:element name="shippingaddress" type="xs:string"/>
                <xs:element name="orderdate" type="xs:string"/>
    <xs:element name="order">
                <xs:complexType>
                        <xs:sequence>
                                <xs:element ref="orderid"/>
                                <xs:element ref="customername"/>
                                <xs:element ref="orderdate"/>
                                <xs:element ref="salesperson"/>
                                <xs:element ref="shippingaddress"/>
                        </xs:sequence>
                </xs:complexType>
    </xs:element>
                <xs:element name="orderid">
                <xs:simpleType>
                        <xs:restriction base="xs:byte">
                                <xs:enumeration value="1"/>
                                <xs:enumeration value="2"/>
                        </xs:restriction>
                </xs:simpleType>
    </xs:element>
    <xs:element name="orders">
                <xs:complexType>
                        <xs:sequence>
                                <xs:element ref="order" maxOccurs="unbounded"/>
                        </xs:sequence>
                </xs:complexType>
    </xs:element>
</xs:schema>
```

## Sample XML with XSD

```
<?xml version="1.0"?>
<orders
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation=“example2.xsd">
   <order>
              <orderid>1</orderid>
             <customername>Nova Systems</customername>
             <orderdate>10-april-2001</orderdate>
             <salesperson>Jack Nicholson</salesperson>
             <shippingaddress>12,green park</shippingaddress>
   </order>
   <order>
             <orderid>2</orderid>
             <customername>Iflex Systems</customername>
             <orderdate>20-april-2001</orderdate>
             <salesperson>Bill Gates</salesperson>
             <shippingaddress> M.G.Road </shippingaddress>
   </order>
</orders>
```

## Summary

- In this session, we have covered:
  - Introduction to XML
  - XML Syntax (well formed XML)
  - XML Namespace
  - XSD

## Thank You