XML

Gaining Access to Diverse Data

Focus: Data integration in the relational model

Real-world data is often not in relational form

e.g., Excel spreadsheets, Web tables, Java objects, etc

One approach: convert using custom wrappers

– tools would adopt a standard export (and import) mechanism

➢ This is the role of **XML**, the eXtensible Markup Language

Overview of XML

- Open W3C standard
- Represents data hierarchically (in a tree)
- Encodes documents and structured data
- Provides context to data (Self-describing data)
- Representation of data across heterogeneous environments
  - Cross platform
  - Allows high degree of interoperability
- Strict rules
  - Syntax
  - Structure
  - Case sensitive

XML Usage

- XML can be used to exchange the information between organizations and systems
- XML can be used for offloading and reloading of databases.
- XML can be used to store and arrange the data, which can customize your data handling needs.
- XML can easily be merged with style sheets to create almost any desired output.
- Any type of data can be expressed as a XML document.

HTML and XML

HTML is used to mark up text so it can be **displayed** to users

HTML describes both **structure** (e.g. <p>, <h2>) and **appearance** (e.g.
, <font>, <i>)

HTML uses a fixed, unchangeable set of tags

XML is used to mark up data so it can be **processed** by computers

XML describes only content, or "meaning"

In XML**, it allows you to make up your own tags**

The XML Technologies

XML	**Extensible Markup Language**	**Defines XML documents**
Infoset	Information Set	Abstract model of XML data; definition of terms
DTD	Document Type Definition	Non-XML schema
XSD	XML Schema	XML-based schema language
XDR	XML Data Reduced	An earlier XML schema
CSS	Cascading Style Sheets	Allows you to specify styles
XSL	Extensible Stylesheet Language	Language for expressing stylesheets; consists of XSLT and XSL-FO
XSLT	XSL Transformations	Language for transforming XML documents
XSL-FO	XSL Formatting Objects	Language to describe precise layout of text on a page

The XML Technologies

XPath	XML Path Language	A language for addressing parts of an XML document, designed to be used by both XSLT and XPointer
XPointer	XML Pointer Language	Supports addressing into the internal structures of XML documents
XLink	XML Linking Language	Describes links between XML documents
XQuery	XML Query Language (draft)	Flexible mechanism for querying XML data as if it were a database
DOM	Document Object Model	API to read, create and edit XML documents; creates in-memory object model
SAX	Simple API for XML	API to parse XML documents; event-driven
Data Island	XML data embedded in a HTML page	
Data Binding	Automatic population of HTML elements from XML data	

The XML Technologies Contd.

Core of broader system

- Data – XML
- Schema – DTD and XML Schema
- Programmatic access – DOM
- Query – XPath, XSLT, XQuery
- Distributed programs – Web services

XML and Structured Data

- Pre-XML representation of data:

```
“PO-1234”,”CUST001”,”X9876”,”5”,”14.98”
```

- XML representation of the same data:

```
<PURCHASE_ORDER>
        <PO_NUM> PO-1234 </PO_NUM>
        <CUST_ID> CUST001 </CUST_ID>
        <ITEM_NUM> X9876 </ITEM_NUM>
        <QUANTITY> 5 </QUANTITY>
        <PRICE> 14.98 </PRICE>
</PURCHASE_ORDER>
```

XML Comments

- XML comment has following syntax:

  <!-------Your comment----->

A comment starts with <!-- and ends with -->.

```
<?xml version="1.0" encoding="UTF-8" ?>
<!---Students grades are uploaded after exams---->
<class_list>
    <student>
        <name>Bharat</name>
        <grade>A+</grade>
    </student>
</class_list>
```

XML Data

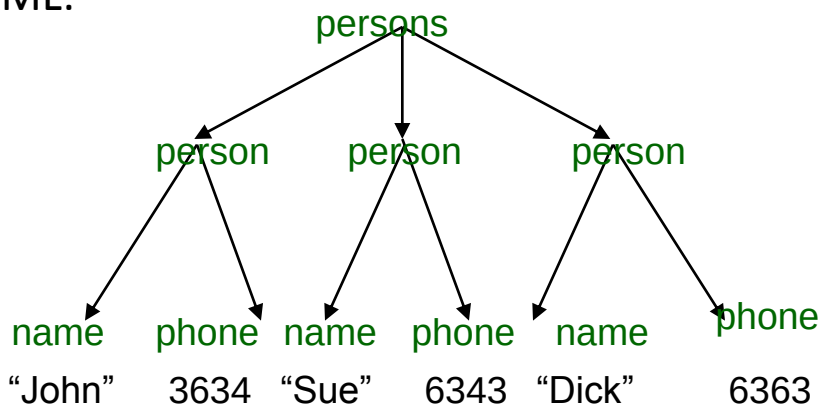
- XML is self-describing
- Schema elements become part of the data
  - Reational schema: persons(name,phone)
  - In XML <persons>, <name>, <phone> are part of the data, and are repeated many times
- Consequence: XML is much more flexible
- XML = semistructured data

Mapping Relational Data to XML Data

Persons

Name	Phone
John	3634
Sue	6343
Ram	6363

XML:




```
<persons>
    <person> <name>John</name>
        <phone> 3634</phone></person>
    <person> <name>Sue</name>
        <phone> 6343</phone> </person>
    <person> <name>Ram</name>
        <phone> 6363</phone></person>
</persons>
```

Mapping Relational Data to XML Data

Application specific mapping

Persons

Name	Phone
John	3634
Sue	6343

Orders

PersonName	Date	Product
John	2002	Gizmo
John	2004	Gadget
Sue	2002	Gadget

XML

```
<persons>
 <person>
    <name> John </name>
    <phone> 3634 </phone>
    <order>  <date> 2002 </date>
             <product> Gizmo </product>
    </order>
    <order> <date> 2004 </date>
             <product> Gadget </product>
    </order>
 </person>
 <person>
    <name> Sue </name>
    <phone> 6343 </phone>
    <order>  <date> 2004 </date>
             <product> Gadget </product>
    </order>
 </person>
</persons>
```

XML is Semi-structured Data

- Missing attributes:

```
<person>   <name> John</name>
           <phone>1234</phone>
 </person>

<person>  <name>Joe</name>
</person>
```

- Could represent in a table with nulls

name	phone
John	1234
Joe	-

XML is Semi-structured Data

- Repeated attributes

```
<person> <name> Mary</name>
          <phone>2345</phone>
          <phone>3456</phone>
</person>
```

Two phones !

- Not possible in tables:

name	phone	
Mary	2345	3456

???

```
<?xml version="1.0"?>
<contact-info>
    <address category="residence">
    <name>Tanmay Patil</name>
    <company>TutorialsPoint</company>
    <phone>(011) 12345678</phone>
    </address>
</contact-info>
```

Document Prolog Section

- Comes at the top of the document, contains:
  - XML declaration
  - Document Type Definition

XML Declaration

- <?xml version="1.0" encoding="UTF-8"?>
- The XML declaration is **case sensitive** and must begin with "<?xml>" where "xml" is written in lower-case.
- Needs to be the first statement of the XML document.

XML Declaration

```
<?xml
version="version_number"
encoding="encoding_declaration"
standalone="standalone_status"
?>
```

Parameter	Parameter_value	Parameter_description
Version	1.0	Specifies the **version** of the XML standard used.
Encoding	UTF-8, UTF-16, ISO-10646-UCS-2, ISO-10646-UCS-4, ISO-8859-1 to ISO-8859-9, ISO-2022-JP, Shift JIS, EUC-JP	**UTF-8** is the **default character encoding** used.
Standalone	*yes or no.*	Default value is set to ***no***. information from an external source (DTD), Yes: tells the processor there are no external declarations required for parsing the document

Rules

- If the XML declaration is present in the XML, it must be placed as the first line in the XML document.
- If the XML declaration is included, it must contain version number attribute.
- The Parameter names and values are case-sensitive.
- The names are always in lower case.
- The order of placing the parameters is important. The correct order is: ***version, encoding and standalone***.
- Either single or double quotes may be used.
- The XML declaration has no closing tag i.e. </?xml>

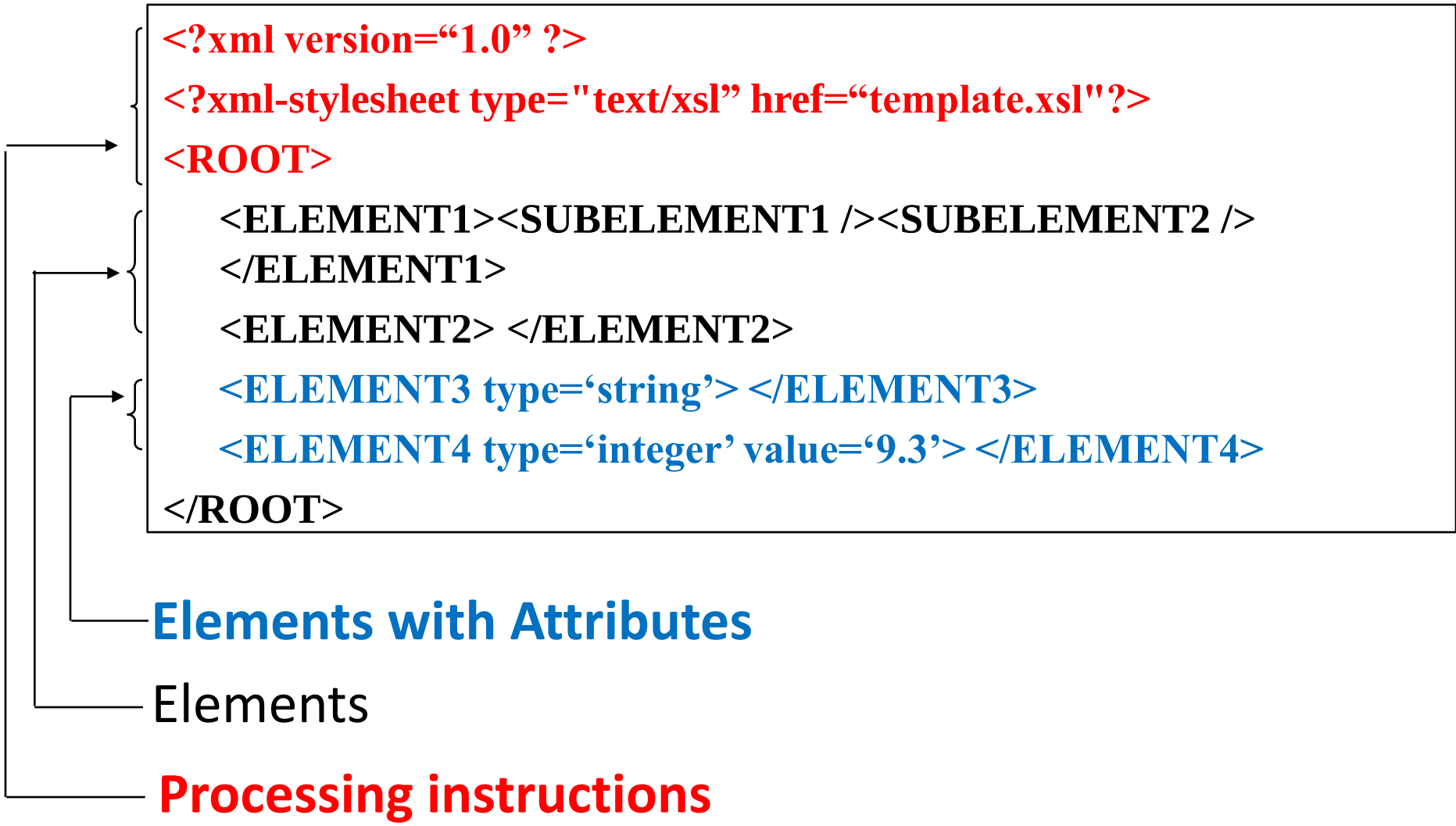
Example

- <?xml version='1.0' encoding='ISO-8859-1' standalone='no' ?>

Components of an XML Document

- **Elements**
  - Each element has a beginning and ending tag
    - <TAG_NAME>...</TAG_NAME>
  - Elements can be empty (<TAG_NAME />)
- **Attributes**
  - Describes an element; e.g. data type, data range, etc.
  - Can only appear in beginning tag
- **Processing instructions**
  - Encoding specification (Unicode by default)
  - Namespace declaration
  - Schema declaration

Components of a XML Document



Document Elements

- Building blocks of XML.
- These divide the document into a hierarchy of sections, each serving a specific purpose.

XML Tags

- Define the scope of an element in the XML
- XML tags are **case-sensitive**
- XML tags must be closed in an appropriate order
  - Start Tag : <address>
  - End Tag: </address>

XML Elements

- Several XML elements
- enclosed by **triangular brackets < >**
- <element>....</element>
- **Nesting of elements**

XML Elements

- One or more elements
- Elements behave as **containers** to hold text, elements, attributes, media objects or all of these.
  - <element-name attribute1 attribute2>
  - ....content
  - </element-name>
- attribute1, attribute2 are attributes of the element separated by white spaces.

Root element

- **Only one root element**

```
<root>
    <x>...</x>
    <y>...</y>
</root>
```

- **Case sensitivity:** The names of XML-elements are **case-sensitive**

Attributes

- An attribute specifies a single property for the element, using a **name/value pair.**
- Element: one or more attributes
- **Attribute names** are defined **without** quotation marks, whereas **attribute values** must always appear in **quotation marks.**
- Same attribute cannot have two values in a syntax.
- **....** **incorrect**

- EnumeratedType: predefined values in its declaration

Entities

- Entities provide a mechanism for textual substitution, e.g.

Entity	Substitution
&lt;	<
&amp;	&

- You can define your own entities
- Parsed entities can contain text and markup

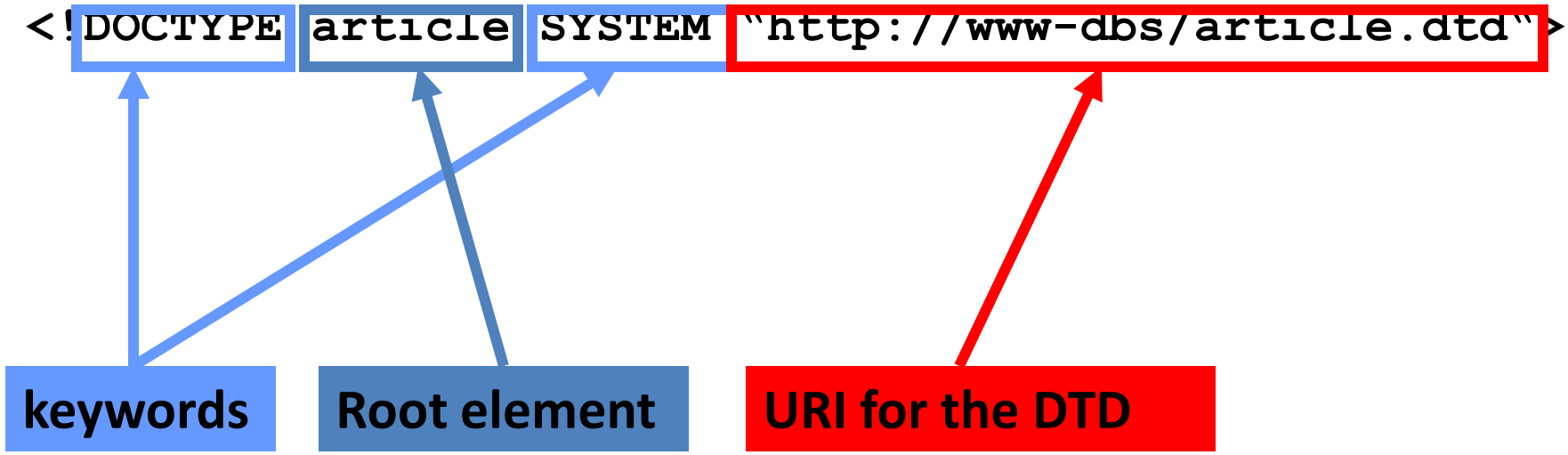
Document Type Definitions (DTD)

- Document Type Definition (DTD):
  - Supplies Metadata
  - Describe the structure of XML documents
  - Provide typing information of elements in those documents
- An XML document may have an optional DTD.
- DTD serves as **grammar** for the underlying XML document, and it is part of XML language.
- DTD has the form:

  <!DOCTYPE name [markupdeclaration]>

Linking DTD and XML Docs

- Document Type Declaration in the XML document:



DTD cont'd

- Consider an XML document:

<db><person><bg</name>
<age>42</age>
<email>bg@gmail.com </email>
</person>
<person>……</person>
……..
</db>

DTD cont'd

- DTD :

```
<!DOCTYPE db [
   <!ELEMENT db (person*)>
   <!ELEMENT person (name, age, email)>
   <!ELEMENT name (#PCDATA)>
   <!ELEMENT age (#PCDATA)>
   <!ELEMENT email (#PCDATA)>
 ]>
```

DTD cont'd

Indicator	Occurrence	
(no indicator)	Required	One and only one
?	Optional	None or one
*	Optional, repeatable	None, one, or more
+	Required, repeatable	One or more

Parsing

- DTD: Document Type Definition

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<!DOCTYPE address [
    <!ELEMENT address (name,company,phone)>
    <!ELEMENT name (#PCDATA)>
    <!ELEMENT company (#PCDATA)>
    <!ELEMENT phone (#PCDATA)>
]>
<address>
    <name>ABC</name>
    <company>XYZ</company>
    <phone>9800000001</phone>
</address>
```

- The **DOCTYPE** declaration has **an exclamation mark (!) at the start of the element name.**
- **DTD Body-**

  <!ELEMENT address (name,company,phone)>

  <!ELEMENT name (#PCDATA)>

  <!ELEMENT company (#PCDATA)>

  <!ELEMENT phone_no (#PCDATA)>

#PCDATA means parseable text data.

- **End Declaration:** DTD is closed using a **closing bracket and a closing angle bracket** (**]>**).

External DTD

```
<?xml version="1.0" encoding="UTF-8" standalone="no"
   ?>
<!DOCTYPE address SYSTEM "address.dtd">
<address>
     <name>Bharat Gupta</name>
     <company>JIIT</company>
     <phone>(0120) 2594</phone>
</address>
```

address.dtd

<!ELEMENT address (name, company, phone)>

<!ELEMENT name (#PCDATA)>

<!ELEMENT company (#PCDATA)>

<!ELEMENT phone (#PCDATA)>

Attribute Declarations in DTDs

Attributes are declared per element:

<!ATTLIST section number CDATA #REQUIRED

title CDATA #REQUIRED>

declares two required attributes for element **section**.

Possible attribute defaults:

- **#REQUIRED** is required in each element instance
- **#IMPLIED** is optional
- **#FIXED default** always has this default value
- **default** **if the attribute is omitted from the element instance**

Attribute Types in DTDs

- **CDATA** string data
- **(A1|…|An)** enumeration of all possible values of the attribute (each is XML name)
- **ID** unique XML name to identify the element
- **IDREF** refers to **ID** attribute of some other element (“intra-document link“)
- **IDREFS** list of **IDREF**, separated by white space

XML CDATA Sections

- CDATA: Character Data.
- CDATA: Defined as blocks of text
- Commanding the parser that the particular section of the document **contains no markup** and **should be treated as regular text**.

Characters reserved

not allowed character	replacement-entity	character description
<	<	less than
>	>	greater than
&	&	ampersand
'	'	apostrophe
"	"	quotation mark

- **Syntax**:

  <![CDATA[

  characters with markup

  ]]>
- **CDATA Start section:**<![CDATA[
- **CDATA End section:]]> delimiter**
- **CData section -** Characters between these two enclosures are interpreted as characters, and not as markup. It may contain markup characters (<, >, and &), but they are **ignored by the XML processor**.

XML Schemas

XML Schemas

- "Schema" is a general term.
- A schema is "a structured framework"
- DTD's are a form of XML schemas
- When we say "XML Schemas," we usually mean the W3C XML Schema Language (XML Schema Definition or **XSD**)
- DTD's, XML Schema's are XML schema languages

Why XML Schemas?

- DTD's provide a weak specification language
  - Can't put any restrictions on text content
  - Less control over mixed content (text plus elements)
  - Less control over ordering of elements
- DTD's are written in a **non-XML format**
  - **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**

XML Schemas

- XML Schema is commonly known as XML Schema Definition **(XSD).**
- Describes and validate the structure and the content of XML data.
- XML schema **defines the elements, attributes and data types.**
- **Similar to a database schema** which describes the data in a database.

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
    xsi:noNamespaceSchemaLocation="**url.xsd**">
    (This is where your XML Schema definition can be found)
    ...
    </rootElement>

XML Namespaces

- Namespace is a mechanisms by which element and attribute name can be assigned to group.
- The Namespace is identified by Uniform Resource Identifier (URI).

Namespace Declaration: <element xmlns:name="URL">

- The Namespace starts with the keyword **xmlns.**
- The word **name** is the **Namespace prefix.**
- The **URL is the Namespace identifier.**

```
<?xml version="1.0" encoding="UTF-8"?>
<cont:contact xmlns:cont="www.xyz.com/profile">
        <cont:name>ABC</cont:name>
        <cont:company>XYZ</cont:company>
        <cont:phone>(011) 123-4567</cont:phone>
</cont:contact>
```

- Namespace prefix is **cont,**
- the element names and attribute names with the **cont prefix (including the contact element), all belong to** the ***www.xyz.com/profile*** *namespace.*

The XSD document

- The file extension is .xsd
- The XSD starts like this:

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

- The root element is <schema>

“Simple” and “complex” elements

- Elements are the building blocks of XML document.
- A “simple” element is one that contains text and nothing else
  - It cannot be empty
  - It cannot contain other elements
  - It cannot have attributes
  - Text can be of many different types, and may have various restrictions applied to it
- **complex**
  - A complex element may have attributes
  - A complex element 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*

Restrictions

- 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 :tells what to do with whitespace
  - value="preserve" Keep all whitespace
  - value="replace" Change all whitespace characters to spaces
  - value="collapse" Remove leading and trailing whitespace, and replace 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>
```

XML Schema

- RDBMS Schema (id string, name string, age int,email string)
- XMLSchema
- XML Document and Schema

……..

```
<Students>
<Student id="p1">
<Name>Raj</Name>
<Age>19</Age>
<Email>raj@abc.com
</Email>
</Student>
</Students>
```

```
<xs:schema>
<xs:complexType name = "StudentType">
<xs:attribute name="id" type="xs:string" />
<xs:element name="Name" type="xs:string />
<xs:element name="Age" type="xs:integer" />
<xs:element name="Email" type="xs:string" />
</xs:complexType>
<xs:element name="Student" type="StudentType" />
</xs:schema>
```

Complex elements

- A complex type is a container for other element (child elements) definitions
- A complex element is defined as

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

Complex Type

```
<xs:element name="Address">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="name" type="xs:string" />
            <xs:element name="company" type="xs:string" />
            <xs:element name="phone" type="xs:int" />
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

- <xs:sequence> says that elements must occur in this order
- Remember that attributes are always simple types

Global and Local definitions

- Elements declared at the “top level” of a <schema> are available for use throughout the schema
- Elements declared within a xs:complexType are local to that type
- Thus, in

```
<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>
```

the elements firstName and lastName are only locally declared

Global Types

```
<xs:element name="AddressType">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="name" type="xs:string" />
            <xs:element name="company" type="xs:string" />
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

```
<xs:element name="Address1">
    <xs:complexType>
        <xs:sequence>
        <xs:element name="address" type="AddressType" />
        <xs:element name="phone1" type="xs:int" />
        </xs:sequence>
    </xs:complexType>
</xs:element>
<xs:element name="Address2">
    <xs:complexType>
        <xs:sequence>
        <xs:element name="address" type="AddressType" />
        <xs:element name="phone1" type="xs:int" />
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

xs:all

- xs:all allows elements to appear in any order
-
```
<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>
```
- Despite the name, the members of an xs:all group can occur once or not at all

Referencing

- Once you have defined an element or attribute (with name="..."), you can refer to it with ref="..."
- Example:
- *<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>
  – <xs:element name="student" ref="person">

Predefined string types

- Recall that a simple element is defined as:
  <xs:element name="***name***" type="***type***" />
- Here are a few of the possible string types:
  - xs:string: a string
  - xs:normalizedString: a string that doesn't contain tabs, newlines, or carriage returns
  - xs:token: a string that doesn't contain any whitespace other than single spaces
- Allowable restrictions on strings:
  - enumeration, length, maxLength, minLength, pattern, whiteSpace

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-DDThh:mm:ss***
  - The T is part of the syntax
- Allowable restrictions on dates and times:
  - enumeration, minInclusive, minExclusive, maxInclusive, maxExclusive, pattern, whiteSpace

Predefined numeric types

- Here are some of the predefined numeric types:
- xs:decimal
- xs:byte
- xs:short
- xs:int
- xs:long
- xs:positiveInteger
- xs:negativeInteger
- xs:nonPositiveInteger
- xs:nonNegativeInteger
- Allowable restrictions on numeric types:
  - enumeration, minInclusive, minExclusive, maxInclusive, maxExclusive, fractionDigits, totalDigits, pattern, whiteSpace

XPath

- XPath is designed to allow the developer to select specific parts of an XML document.
- XPath is a simple language to identify parts of the XML document (for further processing)
- XPath operates on the tree representation of the document
- Result of an XPath expression is a set of elements or attributes.

Node

- A node is a logical part representation of an XML document.
- In XPath 1.0 there are seven types of nodes:
  - Root node
  - Element node
  - Attribute node
  - Text node
  - Namespace node
  - Comment node
  - Processing Instruction node

root node

- The **root node** represents the document itself, independent of any content.
- The **root element** is the first element in the document and is a child of the root node.
- The **element node** represents the document element, is a child of the root node
- The **XML declaration and the document type declaration** **are not children of the root node** (Not represented in the XPath data model)

Elements of XPath

- An XPath expression usually is a **location path** that consists of **location steps**, separated by **/**:

  /article/text/abstract: selects all **abstract** elements

- **root element :** A leading **/**
- Possible location steps:
  - child element **x**: select all child elements with name **x**
  - Attribute **@x**: select all attributes with name **x**
  - Wildcards ***** (any child), **@*** (any attribute)
  - Multiple matches, separated by **|** : **x|y|z**

XPath Operators

Operator	Usage Description
/	Child operator – selects only immediate children (when at the beginning of the pattern, context is root)
//	Recursive descendent– selects elements at any depth (when at the beginning of the pattern, context is root)
.	Indicates current context
..	Selects the parent of the current node
*	Wildcard
@	Prefix to attribute name (when alone, it is an attribute wildcard)
[ ]	Applies filter pattern

Predicates in Location Steps

- Added with **[ ]** to the location step
- Used to restricts elements that qualify as result of a location step to those that fulfil the predicate:
  - **a[b]** elements **a** that have a subelement **b**
  - **a[@d]** elements **a** that have an attribute **d**
  - Plus conditions on content/value:
    - **a[b="c"]**
    - **A[@d>7]**
    - **<, <=, >=, !=, …**
- position() : show the position of the context node .

/Book/Chapter[@number=2]

- Starting from the root node, take the child axis and look for element nodes called Book;
- then, for each of those Book element nodes, look for element nodes called Chapter,
- also using the child axis; then select only those Chapter elements that have a number attribute whose value is 2.

XPath Functions

- Numeric value functions:
  - abs, ceiling, floor, round, etc
- String functions:
  - compare, concat, substring, string-length, uppercase, lowercase, starts-with, ends-with, matches, replace, etc.

./author
(finds all author elements within current context)
/bookstore
(find the bookstore element at the root)

/*
(find the root element)

//author
(find all author elements anywhere in document)

/bookstore[@specialty = “textbooks”]
(find all bookstores where the specialty attribute = “textbooks”)

/book[@style = /bookstore/@specialty]
(find all books where the style attribute = the specialty attribute of the bookstore element at the root)

Xpath Examples

/literature/book/author

retrieves all book authors: starting with the root, traverses the tree, matches element names literature, book, author, and returns elements

<author>Suciu, Dan</author>,

<author>Abiteboul, Serge</author>, ...,

<author><firstname>Jeff</firstname>

<lastname>Ullman</lastname></author>

/literature/(book|article)/author

authors of books or articles

/literature/*/author

authors of books, articles, essays, etc.

/literature//author

authors that are descendants of literature

/literature//@year

value of the year attribute of descendants of literature

/literature//author[firstname]

authors that have a subelement firstname

/literature/book[price < “50“]

low priced books (price <50)

/literature/book[author//country = “Germany“]

books with German author

Path Expression	Result
/bookstore/book[1]	
/bookstore/book[last()]	
/bookstore/book[last()-1]	
/bookstore/book[position()<3]	
//title[@lang]	
//title[@lang='eng']	
/bookstore/book[price>35.00]	
/bookstore/book[price>35.00]/title	

Path Expression	Result
/bookstore/book[1]	Selects the **first book element** that is the child of the bookstore element
/bookstore/book[last()]	Selects the **last book element** that is the child of the bookstore element
/bookstore/book[last()-1]	Selects the **last but one book element** that is the child of the bookstore element
/bookstore/book[position()<3]	Selects the **first two book elements** that are children of the bookstore element
//title[@lang]	Selects **all the title elements** that have an **attribute named lang**
//title[@lang='eng']	Selects a**ll the title elements** that have an **attribute named lang** with a **value of 'eng'**
/bookstore/book[price>35.00]	Selects all the **book elements of the bookstore** element that have a **price element with a value greater than 35.00**
/bookstore/book[price>35.00]/title	Selects **all the title elements of the book** elements of the bookstore element that have a price element with a value greater than 35.00