INTRODUCTION TO XML

INTRODUCTION

- XML stands for eXtensible Markup Language.
- It is a markup language much like HTML, but there are several differences between them.
- XML is a cross-platform, software and hardware independent tool for transmitting information.
- XML was developed in 1996 by the World Wide Web Consortium's (W3C's) XML Working Group.
- Web applications use XML extensively, and web browsers provide many XML-related capabilities.

XML ELEMENTS

- XML documents contain text that represents content (data) and elements.
- XML elements define the framework of an XML document and the structure of data items.

XML ELEMENT STRUCTURE

- XML documents delimit elements with start tags and end tags.
- A start tag consists of the element name (defined by user like variables in a programming language) in angle brackets.
- e.g. <firstName>, here firstName is element name.
- An end tag consists of the element name preceded by a forward slash (/) in angle brackets
- e.g. </firstName>

XML ELEMENT STRUCTURE

- An element start and end tags enclose text that represents a piece of data.
- e.g.

```
<firstName>
                    Ram
</firstName>
```

XML ELEMENT STRUCTURE

- An XML elements have different content types that is enclosed as data.
- An XML element is everything from (including) the element's start tag to (including) the element's end tag.
- An element can have
  - Element content : contains other elements
  - Simple content : contains text only
  - Mixed content : contains both text and other elements
  - Empty content : contains no information

XML ELEMENT STRUCTURE

- An element can also have attributes like HTML elements have attributes.
- Every XML document must have exactly one root element that contains all the other elements.

XML ELEMENT NAMING

- XML elements must follow these naming rules:
  - Names can contain letters, numbers, and other characters.
  - Names must not start with a number or punctuation character.
  - Names must not start with the letters xml (or XML or Xml etc.)
  - Names cannot contain spaces.
  - Any name can be used, no words are reserved, but the idea is to make names descriptive.

XML ELEMENT NAMING

- XML elements must follow these naming rules:
  - Names with an underscore separator are nice, for example <first_name> and <last_name>.
  - The “:” should not be used in element names because it is reserved to be used for something called namespaces.
  - Name are case sensitive because XML is a case sensitive language.

XML ATTRIBUTES

- XML elements can have attributes in the start tag, just like HTML.
- Attributes are used to provide additional information about elements.
- Attributes often provide information that is not a part of the data of an element.
- Attribute values in XML must always be enclosed in quotes, but either single or double quotes can be used.

XML ATTRIBUTES

- If the attribute value itself contains double quotes it is necessary to use single quotes and vice versa.
- e. g.

```
<prod id="33-657" media="paper">
    XML TEXT BOOK
</prod>
```

- prod element has two attributes id and media.

XML STRUCTURING RULES

- There are basic rules for structuring XML:
  - All XML must have a root element.
  - All tags must be closed.
  - All tags must be properly nested.
  - Tag names are case sensitive.
  - Tag names cannot contain spaces.
  - Attribute values must appear within quotes (“ ").

CREATING AN XML FILE (XML DOCUMENT)

- XML document can be written using any text editor with file extension .xml.
- XML document must follow XML structuring rules.
- The Document Type Declaration
  - The first line that can be processed in an XML file must specify the version of XML.
  - <?xml version="version of XML" ?>

CREATING AN XML FILE (XML DOCUMENT)

- Encoding Declaration
  - It can be specified by encoding attribute in XML declaration.
  - <?xml version="1.0" encoding = "utf-8" ?>
- XML Comment
  - XML comments begin with <!-- and end with -->
  - Comments can be placed almost anywhere in an XML document and can span multiple lines.

CREATING AN XML FILE (XML DOCUMENT)

- XML Prolog
  - Everything before root element in a XML is considered XML prolog.
  - In an XML prolog, the XML declaration must appear before the comments and any other markup.

XML NAMESPACES

- XML allows document authors to create custom elements. This extensibility can result in naming collisions among elements in an XML document that have the same name.
- For example, we may use the element book to mark up data about a text book. A library book reservation system may use the element book to mark up data about a booking of text book.
- XML Namespace is a mechanism to avoid name conflicts by differentiating elements or attributes within an XML document that may have identical names, but different definitions.

XML NAMESPACES

- An XML namespace is a collection of element and attribute names.
- XML namespaces provide a means for document authors to unambiguously refer to elements with the same name.
- Name conflicts in XML can easily be avoided using a name prefix.
- A document author places a namespace prefix and colon (:) before an element name to specify the namespace to which that element belongs.

XML NAMESPACES

- An Document authors can create their own namespace prefixes using virtually any name except the reserved namespace prefix xml.

- For example,

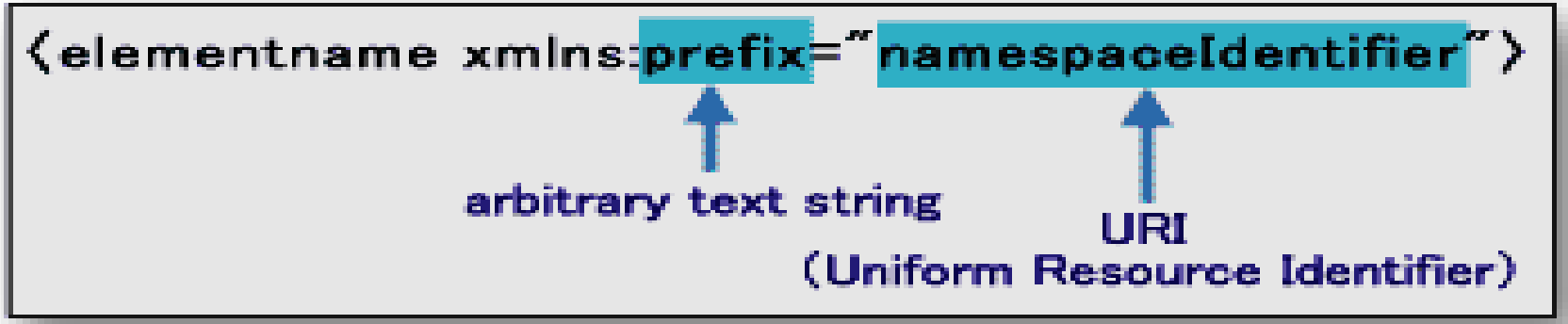
```
<subject>Geometry</subject>
<subject>Cardiology</subject>
```

- Namespaces can differentiate these two subject elements

```
<highschool:subject>Geometry</highschool:subject>
<medicalschool:subject>Cardiology</medicalschool:subject>
```

XML NAMESPACES DECLARATION

- A namespace requires declaration in the element's start tag



- Each namespace prefix is bound to a series of characters called a Uniform Resource Identifier (URI) that uniquely identifies the namespace.

- Document authors create their own namespace prefixes and URIs.

XML NAMESPACES DECLARATION

- A Unique URIs
  - To ensure that namespaces are unique, document authors must provide unique URIs.
  - Another common practice is to use URLs, which specify the location of a file or a resource on the Internet.
  - Using URLs guarantees that the namespaces are unique because the domain names are guaranteed to be unique.
  - The parser does not visit these URLs, nor do these URLs need to refer to actual web pages.

XML NAMESPACES DECLARATION

- Default Namespace
  - To eliminate the need to place namespace prefixes in each element, document authors may specify a default namespace for an element and its children.
  - Syntax:

```
<elementName xmlns="namespaceURI" >
```

  - When a default namespace is declared, the namespace is applied only to the element, and not to any attributes.

XML VOCABULARIES

- XML allows authors (developers) to create their own tags (describe own markup language) to describe data precisely.
- XML-based markup languages called XML vocabularies.
- Some of these markup languages are:
  - MathML (Mathematical Markup Language),
  - SVG (Scalable Vector Graphics),
  - WML (Wireless Markup Language),
  - XBRL (Extensible Business Reporting Language),
  - XUL (Extensible User Interface Language)

XML VOCABULARIES

- XML allows authors (developers) to create their own tags (describe own markup language) to describe data precisely.
- XML-based markup languages called XML vocabularies.
- Some of these markup languages are:
  - W3C XML Schema,
  - XSL (Extensible Stylesheet Language),
  - XHTML (Extensible HyperText Markup Language)

ACCESSING XML DOCUMENTS

- Viewing and Modifying XML Documents
  - XML documents are highly portable. Viewing or modifying an XML document—which is a text file that usually ends with the .xml filename extension does not require special software.
  - Any text editor that supports ASCII/Unicode characters can open XML documents for viewing and editing.
  - Most web browsers can display XML documents in a formatted manner that shows the XML's structure.

ACCESSING XML DOCUMENTS

- Processing XML Documents
  - Processing an XML document requires software called an XML parser (or XML processor).
  - A parser makes the document's data available to applications.
  - While reading an XML document's contents, a parser checks that the document follows the syntax rules.
  - A document that conforms to this syntax is a well-formed XML document and is syntactically correct.

ACCESSING XML DOCUMENTS

- Processing XML Documents
  - If an XML parser can process an XML document successfully, that XML document is well-formed.
  - Parsers can provide access to XML-encoded data in well-formed documents only.
  - XML parsers are often built into browsers and other software.

ACCESSING XML DOCUMENTS

- Validating XML Documents
  - An XML document can reference a Document Type Definition (DTD) or a XML schema that defines the document's proper structure (sequence of elements, tag name etc.).
  - When an XML document references a DTD or a XML schema, some parsers can read it and check that the XML document follows the structure as defined known a validating parsers.
  - If the XML document conforms to the DTD/schema (i.e., has the appropriate structure), the document is valid.

ACCESSING XML DOCUMENTS

- Validating XML Documents
  - By definition, a valid XML document is well-formed.
  - Parsers that cannot check for document conformity against DTDs/schemas are non-validating parsers.
  - Not all well formed XML documents are valid documents.

DOCUMENT TYPE DEFINITION (DTD)

INTRODUCTION

- Document Type Definitions (DTDs) are one of two main types of documents you can use to specify XML document structure.
- It is a document on which structure of XML documents depends and validating parser use this document to conform the validation of XML document.
- Structure of an XML documents defines elements and tag names, there sequence, data type, mandatory elements etc.

INTRODUCTION

- DTDs allow users to check document structure and to exchange data in a standardized format.
- A DTD expresses the set of rules for document structure using an EBNF (Extended Backus-Naur Form) grammar.
- DTDs are not themselves XML documents.

CREATE DOCUMENT TYPE DEFINITION

- DTD can be created in two ways:
  - Internal DTD
    - Internal DTD is defined inside a XML document, but scope will be limited to that XML document.
    - Syntax:

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

<!DOCTYPE root_element [

                    DTD declarations    ]>
```

CREATE DOCUMENT TYPE DEFINITION

- DTD can be created in two ways:
  - External DTD
    - External DTD is a separate document with extension .dtd and referenced to XML document. Singe DTD can be referenced to multiple documents.
    - It is the collection of DTD declarations.

CREATE DOCUMENT TYPE DEFINITION

- DTD can be created in two ways:
  - External DTD
    - DTD is referenced in XML document by syntax:

      <?xml version="1.0"?>

      <!DOCTYPE root_element SYSTEM “filename.dtd”>
    - Content of “filename.dtd” file that contains the DTD for XML document.

DTD DECLARATIONS : ELEMENT TYPE

- Element type declarations set the rules for the type and number of elements that may appear in an XML document, what elements may appear inside each other, and what order they must appear in.
- Syntax:

  <!ELEMENT name allowable_contents>

- The keyword ELEMENT must be in upper case.
- name is the element tag name in XML document.
- allowable_contents of an element type is EMPTY, ANY, Mixed, or children element types.

DTD DECLARATIONS : ELEMENT TYPE

- EMPTY : refers to empty content type of elements.
- ANY : Refers to anything at all, as long as XML rules are followed.
- Mixed content: Refers to a combination of (#PCDATA) and children elements. Therefore, an element that has the allowable content (#PCDATA) may not contain any children.

```
<!ELEMENT parent_name (#PCDATA|child1_name)*>
```

DTD DECLARATIONS : ELEMENT TYPE

- Children elements: Any number of element types within another element type. These are called children elements, and the elements they are placed in are called parent elements.

- Children element types are declared using parentheses in the parent element type's declaration.

```
<!ELEMENT parent_name (child_name)>

<!ELEMENT child_name allowable_contents>
```

- The child element must be declared in a separate element type declaration.

DTD DECLARATIONS : ELEMENT TYPE

- Declaring Multiple Children (Sequence):
  - Multiple children are declared using commas (,).
  - Commas fix the sequence in which the children are allowed to appear in the XML document.

```
<!ELEMENT parent_name
(child1_name,child2_name,child3_name)>

<!ELEMENT child1_name allowable_contents>

<!ELEMENT child2_name allowable_contents>

<!ELEMENT child3_name allowable_contents>
```

DTD DECLARATIONS : ELEMENT TYPE

- Declaring Optional Children:
  - Optional children are declared using the (?) operator.
  - Optional means zero or one times appear in XML document.

```
<!ELEMENT parent_name (child1_name?)>

<!ELEMENT child1_name allowable_contents>
```

  - If the child element is used in the XML document it must be declared in a separate element type declaration.

DTD DECLARATIONS : ELEMENT TYPE

- Declaring Zero or More Children:
  - Zero or more children are declared using the (*) operator.
  - Zero or more means element can be appear zero or more times as children in XML document.

```
<!ELEMENT parent_name (child1_name*)>

<!ELEMENT child1_name allowable_contents>
```

  - If the child element is used in the XML document it must be declared in a separate element type declaration.

DTD DECLARATIONS : ELEMENT TYPE

- Combinations of Children (Choice):
  - A choice between children element types is declared using the (|) operator.

```
<!ELEMENT parent_name (child1_name|child2_name)>

        <!ELEMENT child1_name allowable_contents>

        <!ELEMENT child2_name allowable_contents>
```

DTD DECLARATIONS : ATTLIST TYPE

- Attributes are additional information associated with an element type glossary.
- The ATTLIST declarations identify which element types may have attributes, what type of attributes they may be, and what the default value of the attributes are.
- Syntax:

<!ATTLIST element_name attribute_name attribute_type default_value>

- attribute_type: specifies type of attribute. There are three main attribute types; a string type (CDATA), tokenized types, and enumerated types.

XSD: XML SCHEMA DEFINITION

INTRODUCTION

- An XML schema, commonly known as an XML Schema Definition (XSD).
- Similar to DTD, XML Schema is also used to check whether the given XML document is “well formed” and “valid”.
- XML schema is an alternative to DTD.
- An XML document is considered “well formed” and “valid” if it is successfully validated against XML Schema.

INTRODUCTION

- It describes what a given XML document can contain, in the same way that a database schema describes the data that can be contained in a database.
- The XML schema defines the shape, or structure, of an XML document, along with rules for data content and semantics.
- It can also describe the type and values that can be placed into each element or attribute.
- The extension of Schema file is .xsd.

ADVANTAGES OF USING XML SCHEMA OVER DTD

- Schema uses XML as language so you don't have to learn new syntax.
- DTD describes an XML document's structure, not the content of its elements, but XML schema supports data types of contents and namespaces.
- You can use XML parser to parse the XML schema as well, So no need of another parser for document definition documents like in DTD.
- Just like XML, the XML schema is extensible.
  - You can reuse the schema in other schema, as well as you can reference more than one schemas in a single XML document.

CREATING AN XML SCHEMA DOCUMENT

- To create the schema the standard XML declaration followed by the root element xs:schema element that defines a schema.
- The standard namespace (xs), and the URI associated with this namespace is the Schema language definition, which has the standard value of http://www.w3.org/2001/XMLSchema.
- targetNamespace attribute identifies the namespace of the XML vocabulary that this schema defines and value of attribute is used to reference this schema in an XML document.

CREATING AN XML SCHEMA DOCUMENT

- To create the schema the standard XML declaration followed by the root element xs:schema element that defines a schema.

```
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema  xmlns:xs="http://www.w3.org/2001/XMLSchema"  targetNamespace = "targetNamespaceURI">
.....
....
</xs:schema>
```

DEFINING AN ELEMENT IN XML SCHEMA

- The element tag defines an element to be included in an XML document that conforms to the schema.
- element specifies the actual elements (in XML document) that can be used to mark up data.
- Attributes name and type specify the element's name and type, respectively.
- An element's type indicates the data that the element may contain.

DEFINING AN ELEMENT IN XML SCHEMA

- The element tag defines an element to be included in an XML document that conforms to the schema.
- element specifies the actual elements (in XML document) that can be used to mark up data.
- Attributes name and type specify the element's name and type, respectively.
- An element's type indicates the data that the element may contain.

DEFINING AN ELEMENT (SIMPLE) EXAMPLE

Sample XSD	Sample XML

`<xs:element name="Customer_dob" type="xs:date" />`	`<Customer_dob>`
	`2000-01-12T12:13:14Z`
	`</Customer_dob>`

`<xs:element name="Customer_address" type="xs:string" />`	`<Customer_address>`
	`99 London Road`
	`</Customer_address>`

`<xs:element name="OrderID" type="xs:int" />`	`<OrderID>`
	`5756`
	`</OrderID>`

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

DEFINING AN ELEMENT (COMPLEX TYPES)

- A user-defined type that contains attributes or child elements must be defined as a complex type.
- Simple type element cannot contain attribute and child elements.
- A xs:complexType element provides the definition for an XML Element,
- It specifies which element and attributes are permitted and the rules regarding where they can appear and how many times.

DEFINING AN ELEMENT (COMPLEX TYPES)

- A user-defined type that contains attributes or child elements must be defined as a complex type.
- Simple type element cannot contain attribute and child elements.

```
<xs:element name="Customer">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="Dob" type="xs:date" />
            <xs:element name="Address" type="xs:string" />
            </xs:sequence>
    </xs:complexType>
</xs:element>
```

DEFINING COMPOSITORS

- Compositors provide rules that determine how and in what order children elements can appear within XML document.
- There are three types of compositors.

Compositor	Tag	Description
Sequence	<xs:sequence>	The child elements in the XML document MUST appear in the order they are declared in the XSD schema.
Choice	<xs:choice>	Only one of the child elements described in the XSD schema can appear in the XML document.
All	<xs:all>	The child elements described in the XSD schema can appear in the XML document in any order.

DEFINING XML ATTRIBUTES

- An attribute provides extra information within an element.
- Attributes have name and type properties and are defined within an XSD as follows:

```
<xs:attribute name="x" type="y" />
```

- An Attribute can appear 0 or 1 times within a given element in the XML document.
- The "use" property in the XSD definition is used to specify if the attribute is optional or mandatory.
- To specify that an attribute must be present, use="required“.

DEFINING XML ATTRIBUTES (EXAMPLES)

Sample XSD	Sample XML
<xs:element name="Order"> <xs:complexType> <xs:attribute name="OrderID" type="xs:int" /> </xs:complexType> </xs:element>	<Order OrderID="6" /> - *or no attribute –* <Order />
<xs:element name="Order"> <xs:complexType> <xs:attribute name="OrderID" type="xs:int" use="required" /> </xs:complexType> </xs:element>	<Order OrderID="6" />

REFERENCE XSD TO XML

- There are two ways to associate an XML schema (XSD) with an XML document depending on whether you use namespaces.
- If your documentRootElement does not have a namespace (targetNamespace attribute is not specified in XSD), you can use the noNamespaceSchemaLocation attribute.
- If you do use namespaces, you can use th targetNamespaceURI (specified as a value of targetNamespace in XSD.

REFERENCE XSD TO XML

- There are two ways to associate an XML schema (XSD) with an XML document depending on whether you use namespaces.

- If your documentRootElement does not have a namespace (targetNamespace attribute is not specified in XSD), you can use the noNamespaceSchemaLocation attribute.

```
<?xml version="1.0"?>
<documentRootElement
xsi:noNamespaceSchemaLocation="[pathToXMLSchema]"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"/>
```

REFERENCE XSD TO XML

- There are two ways to associate an XML schema (XSD) with an XML document depending on whether you use namespaces.
- If your documentRootElement does not have a namespace (targetNamespace attribute is not specified in XSD), you can use the noNamespaceSchemaLocation attribute.
- If you do use namespaces, you can use th targetNamespaceURI (specified as a value of

```
<?xml version="1.0"?>
<prefix:documentRootElement
xmlns:prefix="targetNamespaceURI"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"/>
```

PROGRAMMING INTERFACES FOR XML

- The two most popular APIs used to parse XML documents:
  - Document Object Model (DOM)
    - DOM is an official recommendation of the W3C.
    - DOM is designed to build a tree view of your document.
    - All XML documents must be contained in a single element. That single element then becomes the root of the tree.
    - The DOM specification defines several language-neutral interfaces.

PROGRAMMING INTERFACES FOR XML

- The two most popular APIs used to parse XML documents:
  - Simple API for XML (SAX)
    - SAX is a de facto standard.
    - It is a platform-independent language neutral standard interface for event-based XML parsing.
    - SAX defines events that can occur as a parser is reading through an XML document, such as the start or the end of an element.

APPLICATIONS OF XML

- XML has a variety of uses for Web, e-business, and portable applications. The following are some of the applications for which XML is useful:

- **Web publishing:** With XML, you store the data once and then render that content for different viewers or devices based on style sheet processing using an Extensible Style Language (XSL)/XSL Transformation (XSLT) processor.

- **Web searching and automating Web tasks:** Using XML restricts the search to the correct context and returns only the information that you want. Web agents and robots (programs that automate Web searches or other tasks) are more efficient and produce more useful results.

APPLICATIONS OF XML

- XML has a variety of uses for Web, e-business, and portable applications. The following are some of the applications for which XML is useful:

- **General applications:** XML provides a standard method to access information, making it easier for applications and devices of all kinds to use, store, transmit, and display data.

- **Metadata applications:** XML makes it easier to express metadata in a portable, reusable format. Like XML is used for web server configuration.