XML – Extensible Markup Language

Buddhika Gayashani Jayaneththi

Dept. of ICT

Faculty of Technology

University of Ruhuna

Introduction

- XML is a markup language much like HTML, but designed to describe data not to display data.
- The tags are not pre-defined.
- XML was designed to describe data, with focus on what data is.
- HTML was designed to display data, with focus on how data looks.

XML-Example

```
<?xml version="1.0" encoding="UTF-8"?>
<note>
<to>Tover</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don’t get forget me this weekend</body>
</note>
```

HTML vs XML

HTML - Display data

XML – Describe data

- Name – Jane
- Age – 16 years
- Gender – female
- Hair color – brown

Exercise

- Write down the xml code for the given below scenario

Note
To: Mike
From: Jill
Reminder:
Come soon!

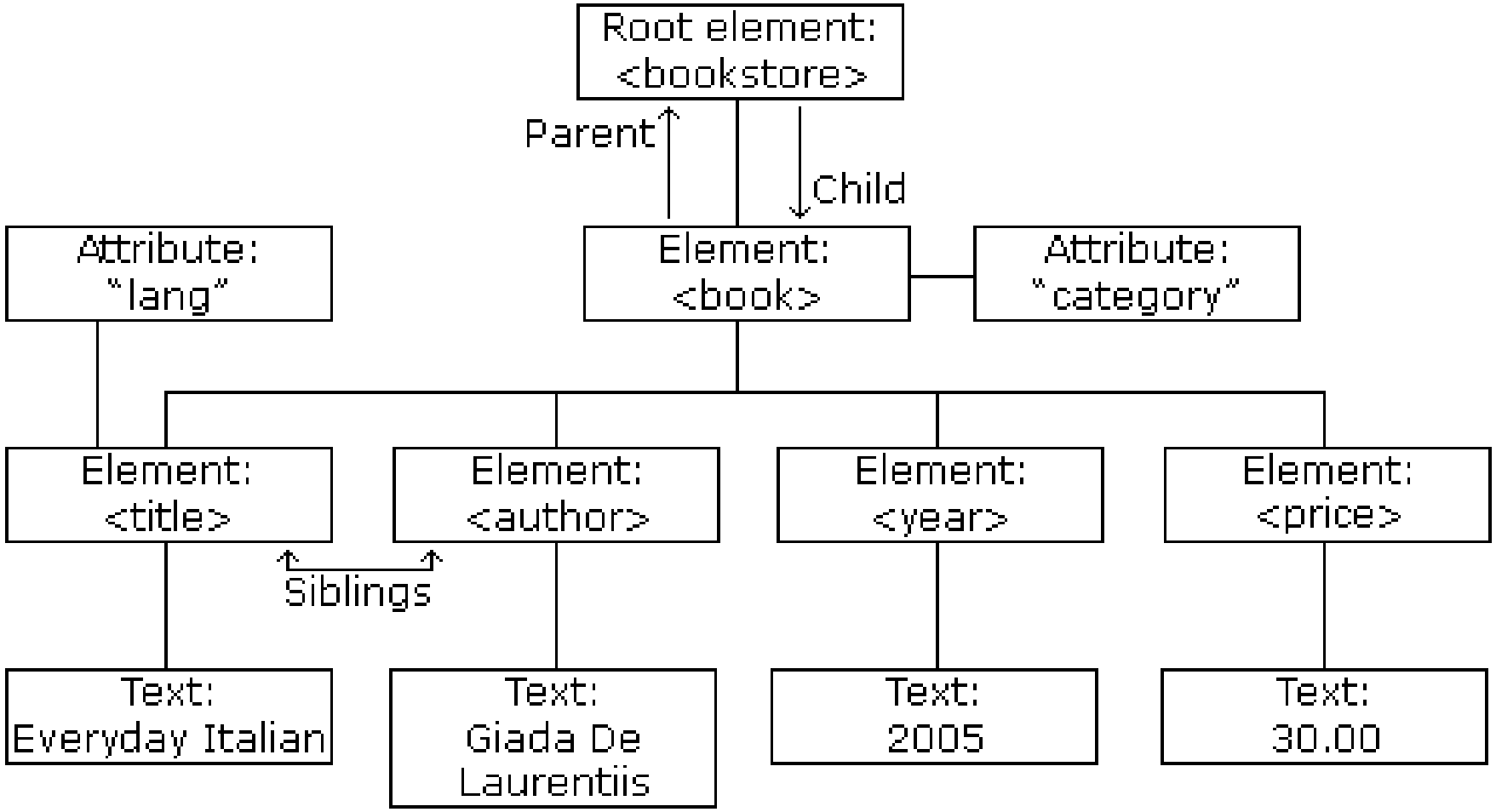
```
<?xml version="1.0" encoding="UTF-8"?>
<note>
        <to>Mike</to>
        <from>Jill</from>
        <reminder>Come soon!</reminder>
</note>
```

XML Tree Structure

- XML documents are formed as **element trees.**
- An XML tree starts at a **root element** and branches from the root to **child elements.**
- All elements can have sub elements (child elements).

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

Example XML Tree



XML Code

```
<?xml version="1.0" encoding="UTF-8"?>
<bookstore>
  <book category="cooking">
        <title lang="en">Everyday Italian</title>
        <author>Giada De Laurentiis</author>
        <year>2005</year>
        <price>30.00</price>
  </book>
</bookstore>
```

XML File Display

- Raw XML files can be viewed in all major browsers.
- XML files cannot be displayed as HTML pages.

```
  <?xml version="1.0" encoding="UTF-8"?>
- <note>
      <to>Mike</to>
      <from>Jill</from>
      <reminder>Come soon!</reminder>
  </note>
```

Displaying XML Files with CSS

- CSS can be used to format an XML file.
- To add the css file into the xml file, include
  - **<?xml-stylesheet type="text/css" href=“*myStyle*.css"?>**

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/css" href="catalog.css"?>
<CATALOG>
<CD>
<TITLE>Empire burlesque</TITLE>
<ARTIST>Bob Dylan</ARTIST>
<COUNTRY>USA</COUNTRY>
<COMPANY>Columbia</COMPANY>
<PRICE>10.90</PRICE>
<YEAR>1985</YEAR>
</CD>
</CATALOG>
```

Displaying XML Files with CSS

- CSS file must contain the XML tags which are needed to be styled.
- Example
  - Catalog.css

```
CATALOG{
background-color: pink;
}

CD {
   display: block;
   margin-bottom: 30pt;
   margin-left: 0;
}
TITLE {
   display: block;
   color: green;
   font-size: 20pt;
}
```

Empire burlesque
Bob Dylan USA Columbia 10.90 1985

Name Conflicts

- Both given below codes contain a <table> element, but the elements have different content and meaning.

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

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

- If these XML fragments were added together, there would be a name conflict.

Namespaces

- In XML, element names are defined by the developer. This often results in a conflict when trying to mix XML documents from different XML applications.
- Using namespaces is a method to avoid element name conflicts.
- Part of XML's extensibility.
- Allow authors to differentiate between tags of the same name (using a prefix)
  - Allows multiple XML documents from multiple authors to be merged.
- Identified by a URI (Uniform Resource Identifier).
- The purpose of using an URI is to give the namespace a unique name.

Solving the naming conflicts using prefix.

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

```
<root>
<h:table xmlns:h="http://www.w3.org/TR/html4/">
  <h:tr>
    <h:td>Apples</h:td>
    <h:td>Bananas</h:td>
  </h:tr>
</h:table>
<f:table xmlns:f="https://www.w3schools.com/furniture">
  <f:name>African Coffee Table</f:name>
  <f:width>80</f:width>
  <f:length>120</f:length>
</f:table>
</root>
```

Solving the naming conflicts using prefix.

Namespaces can also be declared in the XML root element:

```
<root xmlns:h="http://www.w3.org/TR/html4/"
xmlns:f="https://www.w3schools.com/furniture">

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

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

</root>
```

XML Validation

- An XML document with correct syntax is called "Well Formed".
- The syntax rules:

  - XML documents must have a root element
  - XML elements must have a closing tag
  - XML tags are case sensitive
  - XML elements must be properly nested
  - XML attribute values must be quoted

XML Validation

- A "well formed" XML document is not the same as a "valid" XML document
- A "valid" XML document must be,
  - well formed
  - it must conform to a document type definition
- 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
- A document type definition defines the **rules and the legal elements and attributes** for an XML document
- http://www.xmlvalidation.com/

DTD

XML DTD

- A DTD is a Document Type Definition.
- A DTD defines the structure and the **legal elements and attributes** of an XML document.
- An XML document validated against a DTD is both "Well Formed" and "Valid“.
- With a DTD, independent groups of people can agree on a standard DTD for interchanging data.
- An application can use a DTD to verify that XML data is valid.

Option 1: Internal DTD Declaration

```
<?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>Come on this weekend</body>
</note>
```

Option 2: External DTD Declaration

Xml file

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

note.dtd file

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

The DTD above is interpreted like this:

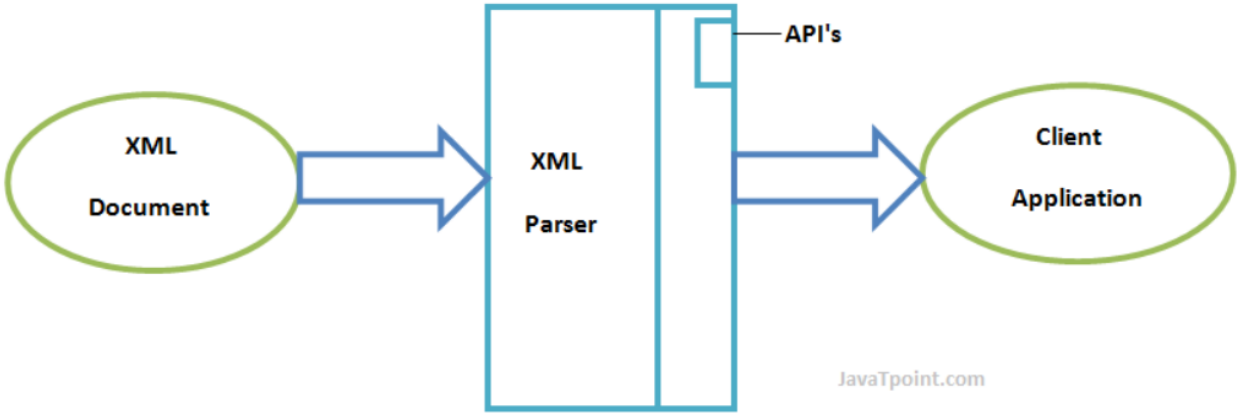
- !DOCTYPE note - Defines that the root element of the document is note
- !ELEMENT note (to,from,heading,body) - Defines that the note element must contain the elements: "to, from, heading, body"
- !ELEMENT to - Defines the to element to be of type "#PCDATA"
- !ELEMENT from - Defines the from element 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"

DTD Components

- In DTD point of view, all XML documents are made up by the following components
  - Elements
  - Attributes
  - Entities
  - PCDATA
  - CDATA

XML Parser

- A software library or package that provides interfaces for client applications to work with an XML document.
- The XML Parser is designed to read the XML and create a way for programs to use XML.
- XML parser validates the document and check that the document is well formatted.



Entities

- Entities are placeholders in XML.

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

<!DOCTYPE address [
    <!ELEMENT address (#PCDATA)>
    <!ENTITY name "Tanmay patil">
    <!ENTITY company "TutorialsPoint">
    <!ENTITY phone_no "(011) 123-4567">
]>

<address>
    &name;
    &company;
    &phone_no;
</address>
```

In the above example, the respective entity names *name*, *company* and *phone_no* are replaced by their values in the XML document.

PCDATA

- PCDATA - Parsed Character Data.
- **PCDATA is text that WILL be parsed by a parser.**
- Think of character data as the text found between the start tag and the end tag of an XML element.
- The text will be examined by the parser for entities and markup.
- Parsed character data should not contain any &, <, or > characters.

CDATA

- CDATA - 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
- Ex:

```
<!ELEMENT element-name category>
or
<!ELEMENT element-name (element-content)>
```

DTD - Elements

Description	Syntax
Empty elements	<!ELEMENT element-name EMPTY>
Elements with parsed character data	<!ELEMENT element-name (#PCDATA)>
Elements with any content	<!ELEMENT element-name ANY>
Elements with children	<!ELEMENT element-name (child1)> or <!ELEMENT element-name (child1,child2,...)>
Only one occurrence of an element	<!ELEMENT element-name (child-name)>
Declaring minimum one occurrence of an element	<!ELEMENT element-name (child-name+)>

DTD - Elements

Description	Syntax
Declaring zero or more occurrences of an element	<!ELEMENT element-name (child-name*)>
Declaring zero or one occurrence of an element	<!ELEMENT element-name (child-name?)>
Declaring either/or content	<!ELEMENT note (to,from,header,(message|body))>
Declaring mixed content	<!ELEMENT note (#PCDATA|to|from|header|message)*>

DTD-Attributes

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

Syntax

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

DTD example:

```
<!ATTLIST payment type CDATA "check">
```

XML example:

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

Attribute Types

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

Attribute Values

Value	**Explanation**
value	The default value of the attribute
#REQUIRED	The attribute is required
#IMPLIED	The attribute is optional
#FIXED *value*	The attribute value is fixed

Questions…