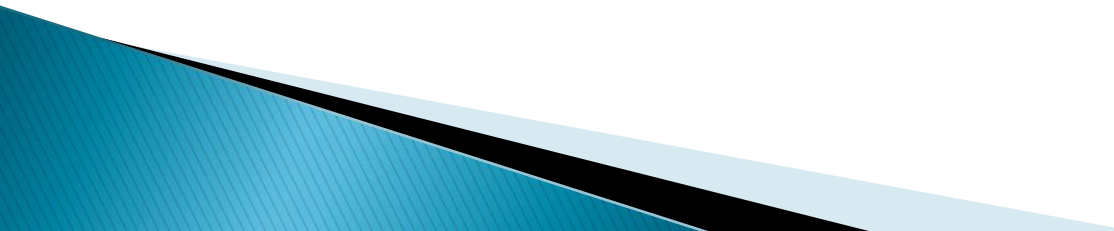


XML


XML

- XML – eXtensible Markup Language
- Not actually a markup language but a way of defining Markup languages
- Tags are not for layout but meaning (semantics)

XML

- ▶ XML – Metalanguage → a language that can be used to describe other languages.
 - ▶ XML documents contain text and can be written using any text editor.
 - ▶ Normally saved with the .xml extension.
- 

HTML and XML

- ▶ XML is not a replacement for HTML.
 - ▶ **HTML** : Used to format information, isn't very useful when it comes to describing information.
 - ▶ HTML tags are predefined.
 - ▶ HTML was designed to display data, with focus on how data looks
 - ▶ **XML** : Used to define the structure of data rather than its format.
 - ▶ XML doesn't rely on predefined tags and attributes.
 - ▶ XML was designed to transport and store data, with focus on what data is
- 

XML Document

- ▶ Two supporting files

- 1) One that specifies structural syntactic rules

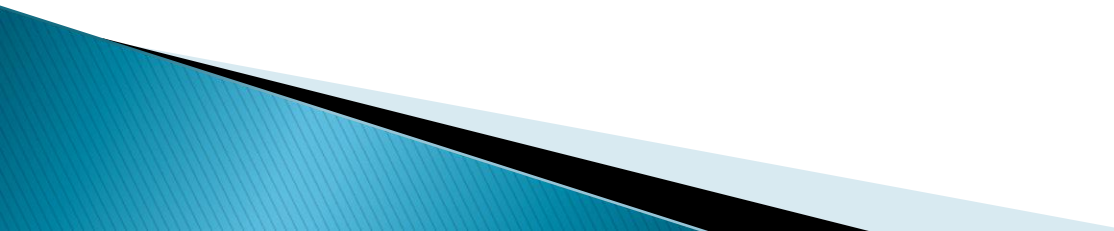
- DTD or a schema

- 2) One that contains a style sheet to describe how the content of document is to be printed or displayed.

XML Document

- ▶ Example – Database values
- ▶ Document describing items in an inventory
- ▶ Tags could be
 - <item>
 - <item_number>
 - <item_name>
 - <item_description>

XML Document

- ▶ In addition to tags, XML document can contain
 - ▶ Other types of markup
 - ▶ Attributes
 - ▶ Processing instructions
 - ▶ Entity references
 - ▶ Comments
 - ▶ Character data
- 

XML Naming rules

- ▶ A specific naming convention is required.
- ▶ Alphanumeric characters (a–z, A–Z and 0–9)
- ▶ Three punctuation characters:
 - ✓ Underscore (_)
 - ✓ Hyphen (–)
 - ✓ Period(.)
- ▶ Names may not contain **white space** and they may not begin with a hyphen, a period or a number.
- ▶ May begin with the letters a–z, A–Z and the underscore character.

XML element

- ▶ XML has no predefined elements.
- ▶ XML element is everything from (including) the element's start tag to (including) the element's end tag.
- ▶ Opening tag `<name>`
- ▶ Ending tag `</name>`
- ▶ Empty Elements → `<name />`
`<id empnum="123" />`

XML element & attribute

- ▶ XML elements can have attribute
 <title lang="en">Everyday Italian</title>
 <author>Giada De Laurentiis</author>
- ▶ Metadata (data about data) should be stored as **attributes**, and the data itself should be stored as **elements**.

XML attribute

- ▶ Attributes provide additional information about an element.
 - `<book category="COOKING">`
`<title lang="en">Everyday Italian</title>`

XML Elements vs. Attributes

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

```
<person>  
  <sex>female</sex>  
  <firstname>Anna</firstna  
me>  
  <lastname>Smith</lastna  
me>  
</person>
```

▶ **<!-- A tag with one attribute -->**
 <patient name = "Maggie Dee Mag"> ...
 </patient>

▶ **<!-- A tag with one nested tag -->**
 <patient>
 <name> Maggie Dee Mag</name>
 </patient>

• **<!-- A tag with one nested tag, which contains three nested tags -->**

 <patient>
 <name>
 <first> Maggie</first>
 <middle> Dee </middle>
 <last> Mag </last>
 </name>
 </patient>

XML Syntax Rules

- ▶ All XML Elements Must Have a Closing Tag
- ▶ XML Tags are Case Sensitive
 - `<Message>This is incorrect</message>`
 - `<message>This is correct</message>`
- ▶ XML Elements Must be Properly Nested
 - `<i>This text is bold and italic</i>`
 - `<i>This text is bold and italic</i>`
- ▶ XML Documents Must Have a Root Element
- ▶ Comments in XML
 - `<!-- This is a comment -->`

XML Syntax Rules

- ▶ XML Attribute Values Must be Quoted
 - `<note date=12/11/2007>`
 `<to>Tove</to>`
 `<from>Jani</from>`
 `</note>`
 - `<note date="12/11/2007">`
 `<to>Tove</to>`
 `<from>Jani</from>`
 `</note>`

XML Syntax Rules

- ▶ There are 5 predefined entity references in XML:

Entity References		
&lt;	<	Less than
&gt;	>	Greater than
&amp;	&	ampersand
&apos;	'	Apostrophe
&quot;	"	quotation mark

Character data sections

- ▶ Character data sections contain raw character data and aren't to processed by XML parsers.
- ▶ Beginning designator `<![CDATA[`
- ▶ Ending designator `]]>`
- ▶ `<book title = "XHTML">`
 `<page>`
 `<![CDATA[` `<p>` A text paragraph `</p>`
 `
` for line breaks
 `]]>`
 `</page>`
 `</book>`

Processing Instructions

- ▶ Used to pass information to applications.
- ▶ Processing instruction begin with `<?` and with `?>`.

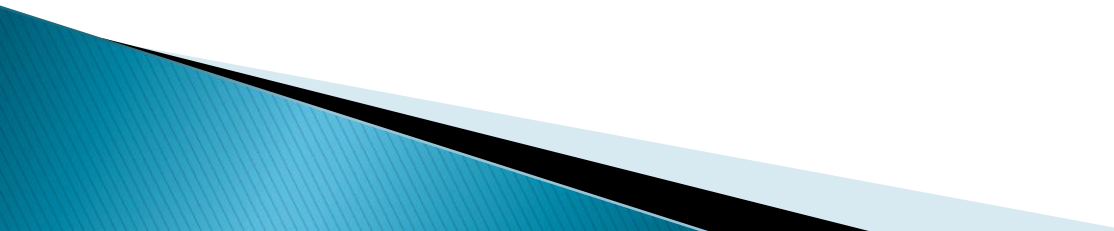
1) Specify a style sheet attached to a document.

e.g. `<?xml - stylesheet href="corp.css" type="text/css"?>`

2) Set the XML version.

e.g. `<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>`

XML Declaration

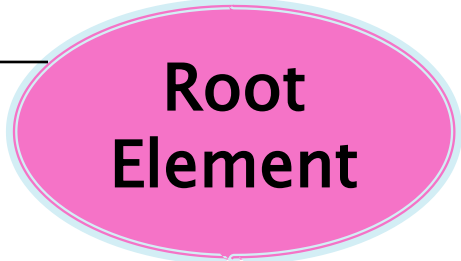

- ▶ Three attributes: version, encoding and standalone.
 - ▶ **Version** → sets the version of XML used in document.
 - ▶ **Encoding** → usually assumed that XML documents are encoded using either UTF-8 or UTF-16.
 - ▶ **Standalone** → if its value is no then the document may have to read an external DTD and if yes then internal DTD is specified.
- 

XML Tree

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

Working with Root, Parent and Child Elements

- ▶ The root element → first element in document.

- ▶ `<inventory>` ← 
 - `<item tracking_number="45154" manufacturer="Not listed">`
 - `<item_type> Floppy Disk Drive`
 - `</item_type>`
 - `<description> Standard Floppy Drive`
 - `</ description >`
 - `</item>`
- `</inventory>` ← 

Nesting Parent and child elements

▶ `<item>` ← **Parent Element**

`<item_type>` ←

`<description>` ←

Standard Floppy Drive

`</description>`

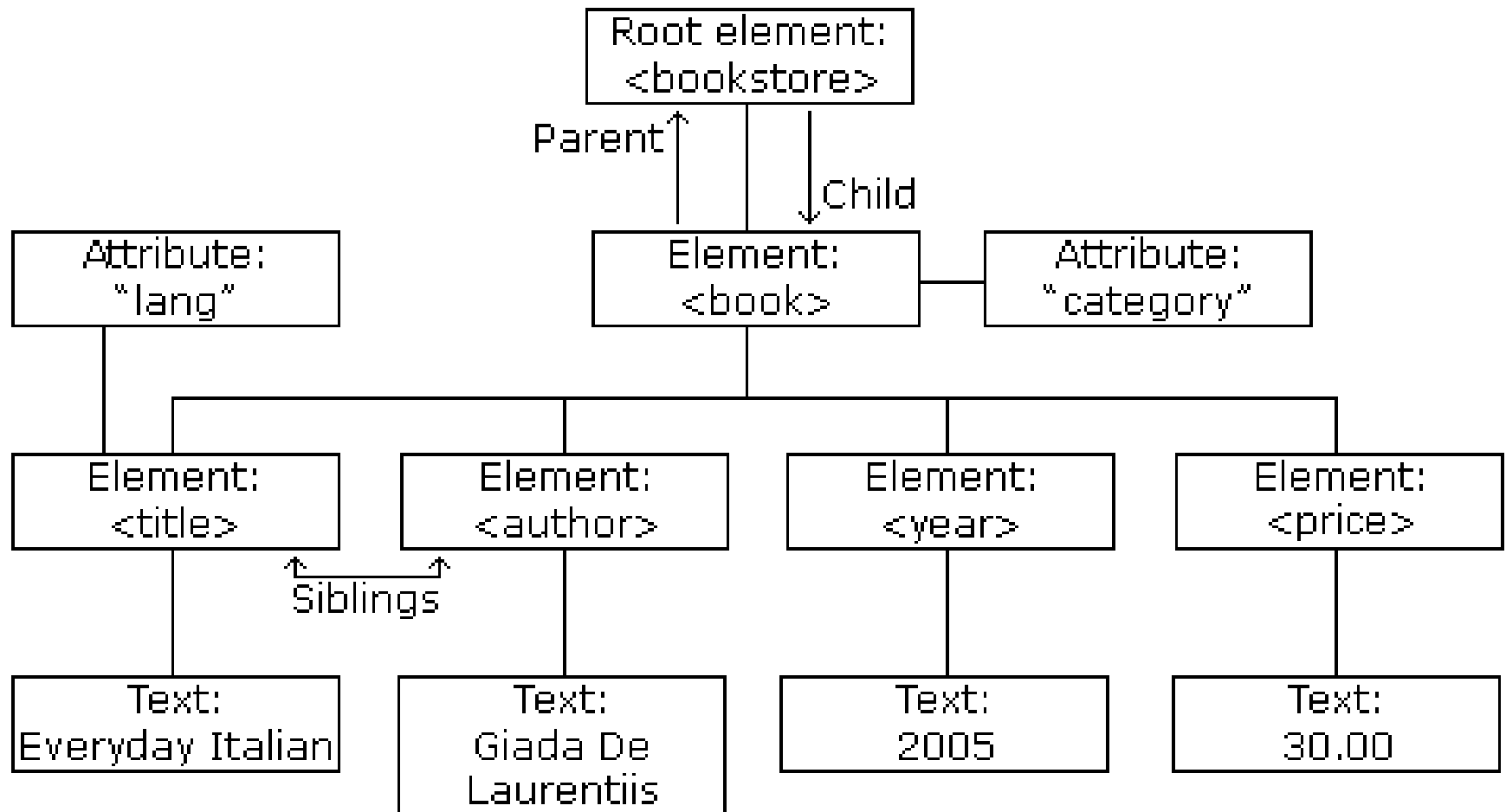
`</item_type>`

▶ `</item>`

**Parent
Element**

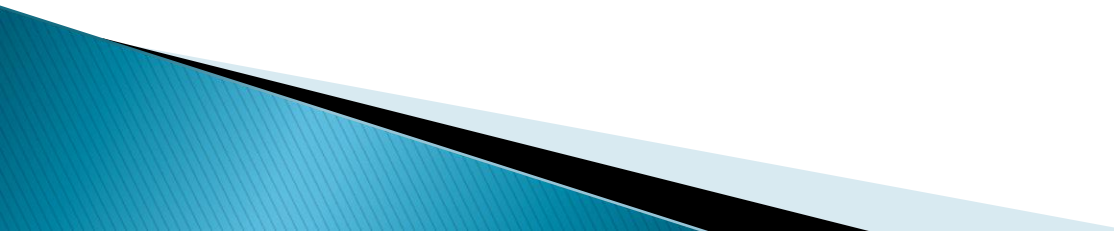
**Child
Elements**

Example



Example

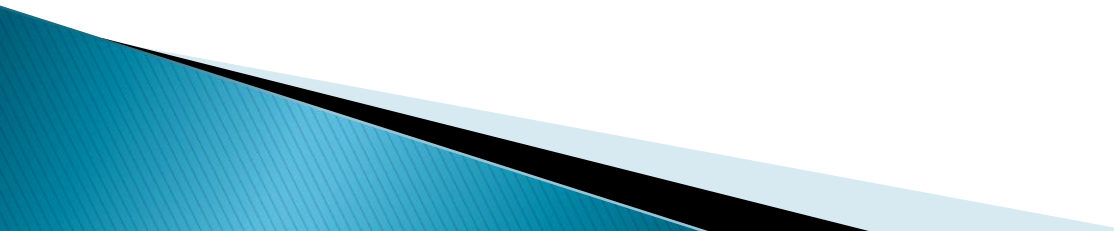
```
<bookstore>  
  <book category="COOKING">  
    <title lang="en">Everyday Italian</title>  
    <author>Giada De Laurentiis</author>  
    <year>2005</year>  
    <price>30.00</price>  
  </book>  
</bookstore>
```



Formatting XML

- ▶ DTD – Document type definition
- ▶ Extensible Stylesheet Language(XSL)

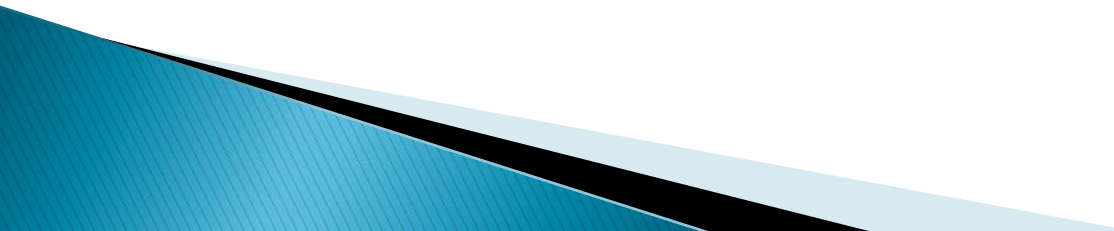
DTD – Document type definition

- ▶ DTD – the general set of rules for a document's tags and attributes is defined in a DTD file.
 - ▶ XML processors can use the DTD to determine if the document is constructed properly or not.
 - ▶ If do have a DTD, they should be structured to conform to the DTD.
- 

DTD

- ▶ A sequence of declarations enclosed in the block of a DOCTYPE markup declaration.
- ▶ `<!keyword >`
- ▶ Four possible keywords :
 - ▶ ELEMENT → used to define tags
 - ▶ ATTLIST → used to define tag attributes
 - ▶ ENTITY → used to define entities
 - ▶ NOTATION → used to define data type notations

DTD

- ▶ XML document → Form of a general tree.
 - ▶ Root node
 - ▶ Element is either a leaf node or an internal node.
 - ▶ Leaf node → syntactic description is character pattern.
 - ▶ Internal node → Syntax is a list of child elements, now each is either a leaf node or an internal node.
- 

DTD

Element Declaration

- ▶ `<!ELEMENT element_name (list of names of child elements) >`
- ▶ E.g. `<!ELEMENT note(to,from,heading,body)>`

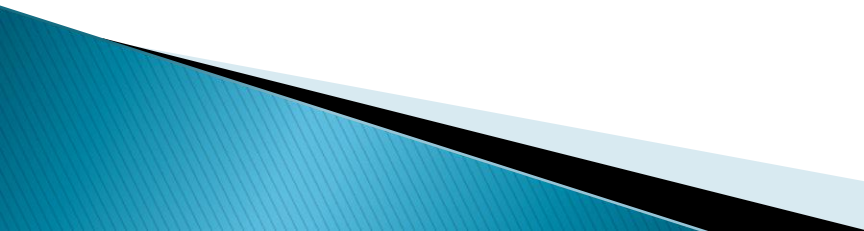
DTD

▶ Child element specification modifiers

Modifier	Meaning
(...)	Choice
(...,...,...)	Sequence
+	One or more occurrences
*	Zero or more occurrences
?	Zero or one occurrence

▶ `<!ELEMENT person(parent+, age, spouse?, sibling*)>`

DTD

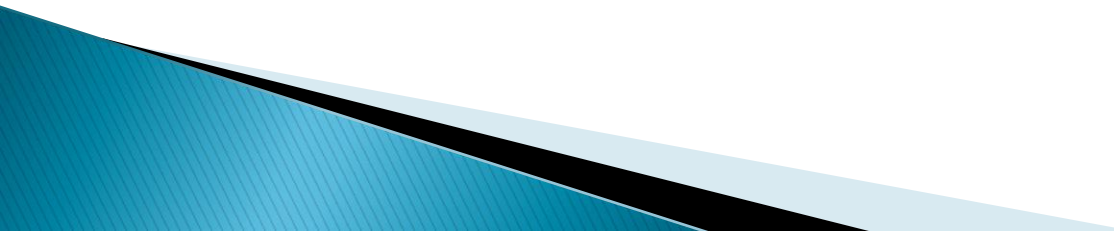
- ▶ Content of an element is type PCDATA, for parsable character data.
 - ▶ (#PCDATA): "mixed content", arbitrary sequence of character data and listed elements
 - ▶ Two other content type:
 - ▶ EMPTY → is used to specify that element has no content.
 - ▶ ANY → is used when element may have any content.
 - ▶ E.g. <!ELEMENT participant (#PCDATA)>
- 

DTD

<!ATTLIST element-name attr-name attr-type attr-default ...>

- ▶ It declares which attributes are allowed or required in which elements attribute types
- ▶ CDATA: any value is allowed (the default)
- ▶ (*value*|...): list of allowed values
- ▶ ID, IDREF, IDREFS: ID attribute values must be unique (contain "element identity"), IDREF attribute values must match some ID (reference to an element)

DTD

- ▶ **attribute defaults:**
 - ▶ **#REQUIRED:** the attribute must be explicitly provided
 - ▶ **#IMPLIED:** attribute is optional, no default provided
 - ▶ **"value":** if not explicitly provided, this value inserted by default
 - ▶ **#FIXED "value":** as above, but only this value is allowed
- 

Number of occurrences	Syntax	Example
Only one	<!ELEMENT element-name (child-name)>	<!ELEMENT memo (message)>
One or more	<!ELEMENT element-name (child-name+)>	<!ELEMENT memo (message+)>
Zero or more	<!ELEMENT element-name (child-name*)>	<!ELEMENT memo (message*)>
Zero or one	<!ELEMENT element-name (child-name?)>	<!ELEMENT memo (message?)>
Either/or	<!ELEMENT element-name (child-name child-name)>	<!ELEMENT memo (note message)>

Value	Description
CDATA	Character data
(enum1 enum2 . . .)	Must be from enumerated list
ID	Unique ID
IDREF	ID of another element
IDREFS	List of other IDs
NMTOKEN	XML name
NMTOKENS	List of XML names
ENTITY	Entity
ENTITIES	List of entities
NOTATION	Notation name
xml :	Predefined XML value

Table 8-2 DTD attribute types

Value	Description
value	Default value of an attribute
#REQUIRED	Attribute must be included
#IMPLIED	Doesn't have to be included
#FIXED value	This value is fixed

Table 8-3 Attribute required or default

Internal DTD Declaration

- ▶ DTD is declared inside the XML file.

Syntax :

```
<!DOCTYPE root-element [element-  
declarations]>
```

- ▶ It is helpful when you want to apply constraints to an individual document and developing a complex 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>

External DTD Declaration

- ▶ With an external DTD, you place a reference to a DTD in a file.
- ▶ This makes it easy to apply the DTD to multiple documents.
- ▶ No need to edit the DTD definition in each and every XML document.
- ▶ Two types of external DTDs :
 - Public
 - Nonpublic

Syntax:

<!DOCTYPE root-element SYSTEM "filename">



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

DTD

- ▶ DTDs specify the purchase order details as defined below :
- ▶ Purchase Order
 - One and only one order number
 - One or more requested products
 - One order date
 - One customer identifier


```
<?xml version="1.0" ?>
<!DOCTYPE purchase_order [
  <!ELEMENT purchase_order (customer)>
  <!ELEMENT customer (account_id, name)>
  <!ELEMENT name (first, mi, last)>
  <!ELEMENT first (#PCDATA)>
  <!ELEMENT mi (#PCDATA)>
  <!ELEMENT last (#PCDATA)>
]>
<purchase_order>
  <customer>
    <account_id>10-456</account_id>
    <name>
      <first> William </first>
      <mi> R </mi>
      <last> Stanek </last>
    </name>
  </customer>
</purchase_order>
```

DTD file → purspec.dtd

```
<!ELEMENT
purchase_order
(customer)>
<!ELEMENT customer
(account_id, name)>
<!ELEMENT name (first,
mi, last)>
<!ELEMENT first
(#PCDATA)>
<!ELEMENT mi
(#PCDATA)>
<!ELEMENT last
(#PCDATA)>
```


purchase.xml

```
<?xml version="1.0"
standalone="no"?>
<!DOCTYPE purchase_order
SYSTEM "purspec.dtd">
<purchase_order>
<customer>
<account_id>10-
456</account_id>
<name>
<first> William </first>
<mi> R </mi>
<last> Stanek </last>
</name>
</customer>
</purchase_order>
```

Name Conflicts

- ▶ `<table>`
 `<tr>`
 `<td>Apples</td>`
 `<td>Bananas</td>`
 `</tr>`
 `</table>`
- ▶ `<table>`
 `<name>African Coffee Table</name>`
 `<width>80</width>`
 `<length>120</length>`
 `</table>`

Namespaces

- ▶ XML Namespaces provide a method to avoid element name conflicts.
 - ▶ It is a collection of names used in XML documents as element types and attributes.
 - ▶ Name conflicts in XML can easily be avoided using a name prefix.
 - ▶ A **Uniform Resource Identifier (URI)** is a string of characters which identifies an Internet Resource.
 - ▶ The most common URI is the **Uniform Resource Locator (URL)** which identifies an Internet domain address.
- 

- ▶ This XML carries information about an HTML table, and a piece of 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>
```

XML Namespaces – The xmlns Attribute

- ▶ The namespace is defined by the **xmlns attribute** in the start tag of an element.
- ▶ The namespace declaration has the following syntax.

<element_name xmlns:prefix="URI">

- ▶ Example:

```
<birds xmlns:bd =  
  "http://www.aubdn.org/names/species">
```

- ▶ Within birds element, its children elements must be prefixed with bd.

```
<bd:lark>
```



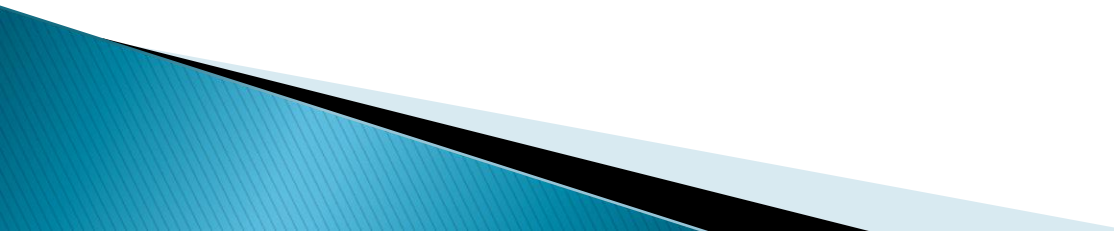
▶ **<root>**

```
<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>
```

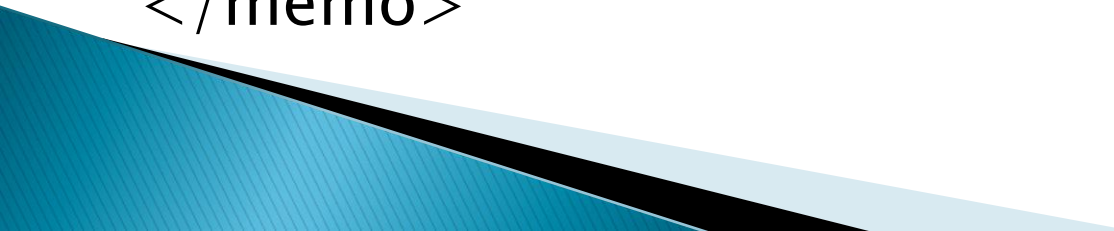
```
<f:table xmlns:f="http://www.w3schools.com/furnit  
ure">  
  <f:name>African Coffee Table</f:name>  
  <f:width>80</f:width>  
  <f:length>120</f:length>  
</f:table>
```

</root>

Schemas

- ▶ It performs same function as DTD.
 - ▶ Schema defines what is legal in an XML document.
 - ▶ It is saved with .xsd extension.
 - ▶ Root element is defined as complex as it contains a lot of the simple types.
- 


```
<?xml version="1.0"?>
<!DOCTYPE memo [
  <!ELEMENT memo (to,from,title,message)>
  <!ELEMENT to (#PCDATA)>
  <!ELEMENT from (#PCDATA)>
  <!ELEMENT title (#PCDATA)>
  <!ELEMENT message (#PCDATA)>
]>
<memo>
  <to>Alan</to>
  <from>Sid</from>
  <title>Meeting</title>
  <message>We have meeting today</message>
</memo>
```

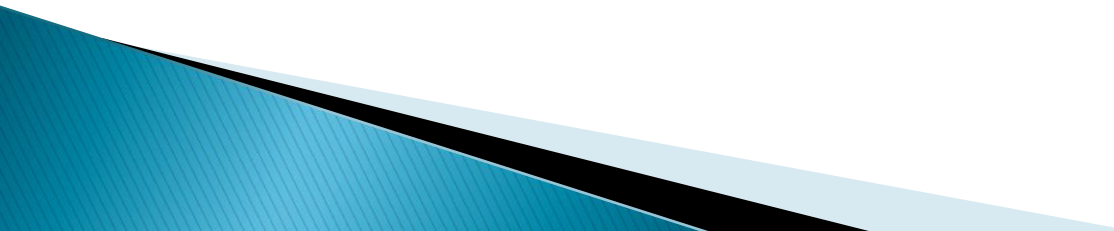


Schema example – memo.xsd

```
<?xml version="1.0" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  targetnamespace="http://www.myXMLnamespace.com"
  elementFormDefault="qualified">
  <xs:element name="memo">
  <xs:complexType>
  <xs:sequence>
  <xs:element name="to" type="xs:string" />
  <xs:element name="from" type="xs:string" />
  <xs:element name="title" type="xs:string" />
  <xs:element name="message" type="xs:string" />
  </xs:sequence>
  </xs:complexType>
  </xs:element>
  </xs:schema>
```



```
<?xml version="1.0"?>
<memo xmlns="http://www.w3schools.com"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.myXMLnamespace.com memo.xsd">
<to>Alan</to>
<from>Sid</from>
<title>Reminder</title>
<message> Meeting is today at 12.
</message>
</memo>
```



XML Schema

- ▶ The schema begins properly as its root element:

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

- ▶ **xmlns** → it declares namespace that elements and data types used in schema.
- ▶ All define with prefix `xs:.`
- ▶ The **targetnamespace** indicates where elements defined in this schema.

Elements

XML File syntax

```
<firstname>Greg</firstname>
```

```
<age>35</age>
```

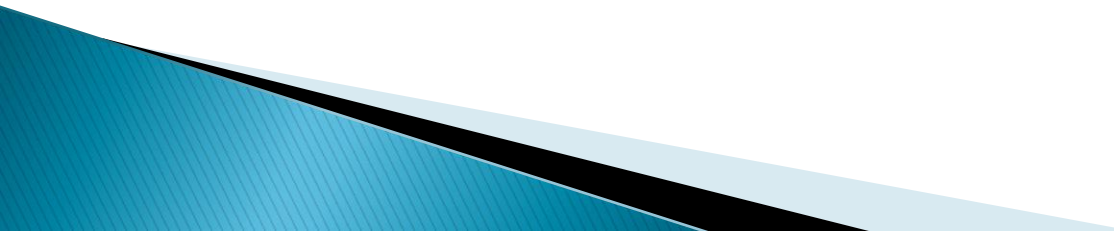
Scema syntax

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

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



Elements

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

Attribute

- ▶ `<firstname lang="English">Greg</firstname>`
- ▶ `<xs:attribute firstname="lang" type="xs:string">`
- ▶ `<xs:attribute firstname="lang" type="xs:string" default="English">`
- ▶ `<xs:attribute firstname="lang" type="xs:string" fixed="English">`

Attribute

- ▶ `<xs:attribute firstname="lang" type="xs:string" use="optional">`
- ▶ `<xs:attribute firstname="lang" type="xs:string" use="required">`

Well formed?

- ▶ DTDs or schema are defined which provides rules for well formed xml document.
- ▶ A parser can use the rules.
- ▶ **Parsers can be**
 - Any software
 - part of a browser
 - An object in a programming language
 - Javascript having ability to access parser → Microsoft's XMLDOM

