XML Document Structure

UNIT - I

XML Document Structure

- It consist of following:

- The XML declaration
- The Document Type Declaration
- The element data
- The attribute data
- The character data or XML content

XML Declaration - Components

- <?xml   - Starts the beginning of the processing instruction

- Version=”xxx” - specific version of XML being used in the document

- standalone=”xxx” - defines whether documents are allowed to contain external markup declarations, can be set to “yes” or “no”

- encoding=”xxx” - indicates character encoding that the document uses, default is “US-ASCII”, the most common alternate setting is “UTF-8”

Eg. <?xml version=”1.0” encoding=”UTF-8” standalone=”no”?>

XML Document Type Declaration

- A Document Type Declaration names the document type and identifies the internal content by specifying the *root element*

- A DOCTYPE allows to validates the document by referencing internal / external DTD

- Eg. <!DOCTYPE shirt SYSTEM “shirt.dtd”>

Elements

- XML elements are either a matched pair of XML tags or single XML tags that are “self-closing”
- When elements do not come in pairs, the element name is suffixed by the forward slash.
- Eg. <on_sale/>
- In this case, there would be no other matching element, these “unmatched” elements are known as *empty elements*
- XML is a hierarchical tree
- Elements can be nested within other elements
- Element names has no size limitation

Attributes

- Attributes within elements communicate additional information to XML processors
- It modifies the nature of the encapsulated content

Eg. <price currency=”USD”>…</price>

<on_sale start_date=”10-15-2001”/>

- Attributes follow strict rules as to their value
- Attributes can be required, optional, or contain a fixedValue

Contd…

- Required or optional attributes can either contain text or contain one of set list of enumerated values
- Fixed attributes, if present, must contain a specific value
- Attributes can specify a default value that is applied if the attribute is optional but not present

Entity References

- Entities indicates to XML-processor that a text string is to follow that will be replaced with a different literal value
- Internal Entities

```
<?xml version="1.0"?>
<!DOCTYPE library [
<!ENTITY cpy "Copyright 2000">

<library>
<book>
<title>How to Win Friends</title>
<author>Joe Charisma</author>
<copyright>&cpy;</copyright>
</book>
```

Others

- **Comments**
- Eg. <!-- This is a comment -->
- <to>Tove</to>
- **Processing Instructions**
- Similar to comments but provide additional info' to application how the content should be processed
- *PI target*, the instruction name *is a special identifier that the processing* application is intended to understand
- Eg. <?instruction options?>

Rules of XML Structure

- All XML Elements Must Have a Closing Tag
- XML Tags Are Case Sensitive
- All XML Elements Must Have Proper Nesting
- All XML Documents Must Contain a Single Root Element
- Attribute Values Must Be Quoted
- Attributes May Only Appear Once in the Same Start Tag
- Attribute Values Cannot Contain References to External Entities
- All Entities Except amp, lt, gt, apos, and quot must Be declared before they are used

Namespaces in XML

- XML documents with multiple conflicting tag sets
- Namespace is a mechanism to associate correct elements from which tag set the elements come from
- It uses colon-delimited prefix to associate external semantics with elements identified URI
- Namespace-identified element acts as if element defined locally

Contd…

- XML developers may choose the same element and attribute names for their standards, but they could mean different things

```
Eg. <Customer>
<Name>John Smith</Name>
<Order>
<Product>
<Name>Hot Dog Buns</Name>
</Product>
</Order>
</Customer>
```

- Namespaces can easily tell the Xml parser, difference between the two <Name> elements

Contd…

```
Eg. <Customer>
<cust:Name xmlns:cust=”customer-namespace-URI”>John
   Smith</cust:Name>
<Order>
<Product>
<prod:Name xmlns:prod=”product-namespace-URI”>Hot
   Dog Buns</prod:Name>
</Product>
</Order>
</Customer>
```

Declaring Namespaces

- Can be declared in 2 ways: a) Default namespace b)Explicit declaration
- A default namespace declaration specifies a namespace to use for all child elements of the current element

Eg.

```
<Customer xmlns=”http://www.eps-software.com/po”>
<Name>Travis Vandersypen</Name>
<Order>
<Product>
<Name>Hot Dog Buns</Name>
</Product>
</Order>
</Customer>
```

Contd…

- Explicit Declaration: used for more readability with meaningful names

```
<cust:Customer xmlns:cust=”http://www.eps-software.com/customer”
xmlns:ord=”http://www.eps-software.com/order”>
<cust:Name>Travis Vandersypen</cust:Name>
<ord:Order>
<ord:Product>
<ord:Name xmlns:prod=”product-namespace-URI”>Hot Dog
    Buns</ord:Name>
</ord:Product>
</ord:Order> </cust:Customer>
```

- Two different namespaces are referenced: one for customers and one for orders

- Allows a different set of rules to be applied for customer names versus product names.

Identifying the Scope of Namespaces

- By default, “inherit” their parent element’s namespace
- Inherited namespace can be overwritten by specifying a new namespace on a particular child element

```
Eg. <Customer xmlns=”http://www.eps-
    software.com/customer”>
<Name>Travis Vandersypen</Name>
<Order xmlns=”http://www.eps-software.com/order”>
<Product>
<Name>Hot Dog Buns</Name>
</Product>
</Order>
</Customer>
```