UNIT-III

3.1 INTRODUCTION TO XML:

XML is a software- and hardware-independent tool for storing and transporting data.

What is XML?

- XML stands for EXtensible Markup Language
- XML is a markup language much like HTML
- XML was designed to store and transport data
- XML was designed to be self-descriptive
- XML is a W3C Recommendation

The Difference Between XML and HTML

XML and HTML were designed with different goals:

- XML was designed to carry data - with focus on what data is
- HTML was designed to display data - with focus on how data looks
- XML tags are not predefined like HTML tags are

XML Does Not Use Predefined Tags

- The XML language has no predefined tags.
- The tags in the example above (like <to> and <from>) are not defined in any XML standard. These tags are "invented" by the author of the XML document.
- HTML works with predefined tags like <p>, <h1>, <table>, etc.
- With XML, the author must define both the tags and the document structure.

XML is Extensible

- Most XML applications will work as expected even if new data is added (or removed).
- Imagine an application designed to display the original version of note.xml (<to> <from> <heading> <data>).
- Then imagine a newer version of note.xml with added <date> and <hour> elements, and a removed <heading>.

The way XML is constructed; older version of the application can still work:

```
<note>
      <date> 2015-09-01 </date>
      <hour> 08:30 </hour>
      <to> Tove </to>
      <from> Jani </from>
      <body> Don't forget me this weekend! </body>
</note>
```

Note
To: Tove
From: Jani
Date: 2015-09-01 08:30
Head: (none)
Don't forget me this weekend!

3.2 XML DOCUMENT

XML Document Types:

1. An XML document with correct syntax is called "Well Formed".
2. A "Valid" XML document must also conform to a document type definition.

(i) Well Formed XML Documents:

- XML with correct syntax is Well Formed XML.

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.

Example

```
<?xml version="1.0" encoding="ISO-8859-1"?>
        <note>
                <to> Tove </to>
                <from> Jani </from>
                <heading> Reminder </heading>
                <body> Don't forget me this weekend! </body>
        </note>
```

(ii) Valid XML Documents

- A "valid" XML document is not the same as a "well formed" XML document.
- A "valid" XML document must be well formed. In addition it must conform to a document type definition.
- Rules that define the legal elements and attributes for XML documents are called Document Type Definitions (DTD) or XML Schemas.

There are two different document type definitions that can be used with XML:

1. DTD - The original Document Type Definition
2. XML Schema - An XML-based alternative to DTD

(ii) XML DTD

- An XML document with correct syntax is called "Well Formed".
- An XML document validated against a DTD is both "Well Formed" and "Valid".

Valid XML Documents

A "Valid" XML document is a "Well Formed" XML document, which also conforms to the rules of a DTD:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE note SYSTEM "Note.dtd">
<note>
        <to> Tove </to>
        <from> Jani </from>
        <heading> Reminder </heading>
        <body> Don't forget me this weekend! </body>
```

```
</note>
```

The DOCTYPE declaration, in the example above, is a reference to an external DTD file.

3.3 DTD

- A DTD is a Document Type Definition.
- A DTD defines the structure and the legal elements and attributes of an XML document.

Why Use a DTD?

- 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.

(i) An Internal DTD Declaration

If the DTD is declared inside the XML file, it must be wrapped inside the <!DOCTYPE> definition:

XML document with an internal 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>
```

(ii) An External DTD Declaration

If the DTD is declared in an external file, the <!DOCTYPE> definition must contain a reference to the DTD file:

XML document with a reference to an external DTD

```
<?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>
```

And here is the file "note.dtd", which contains the DTD:

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

3.4 XML SCHEMA

- An XML Schema describes the structure of an XML document.
- The XML Schema language is also referred to as XML Schema Definition (XSD).

XSD Example

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="note">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name="to" type="xs:string"/>
                        <xs:element name="from" type="xs:string"/>
                        <xs:element name="heading" type="xs:string"/>
                        <xs:element name="body" type="xs:string"/>
                    </xs:sequence>
            </xs:complexType>
    </xs:element>
</xs:schema>
```

The purpose of an XML Schema is to define the legal building blocks of an XML document:

- the elements and attributes that can appear in a document
- the number of (and order of) child elements
- data types for elements and attributes
- default and fixed values for elements and attributes

XML Schemas Support Data Types

XML Schemas is the support for data types.

- It is easier to describe allowable document content
- It is easier to validate the correctness of data
- It is easier to define data facets (restrictions on data)
- It is easier to define data patterns (data formats)
- It is easier to convert data between different data types

XML Schemas use XML Syntax

XML Schemas is that they are written in XML.

- You don't have to learn a new language

- You can use your XML editor to edit your Schema files
- You can use your XML parser to parse your Schema files
- You can manipulate your Schema with the XML DOM
- You can transform your Schema with XSLT

XML Schemas are extensible, because they are written in XML.
With an extensible Schema definition you can:

- Reuse your Schema in other Schemas
- Create your own data types derived from the standard types
- Reference multiple schemas in the same document

XML Schemas Secure Data Communication

- When sending data from a sender to a receiver, it is essential that both parts have the same "expectations" about the content.
- With XML Schemas, the sender can describe the data in a way that the receiver will understand.
- A date like: "03-11-2004" will, in some countries, be interpreted as 3.November and in other countries as 11.March.

However, an XML element with a data type like this:

```
<date type="date">2004-03-11</date>
```

ensures a mutual understanding of the content, because the XML data type "date" requires the format "YYYY-MM-DD".

3.5 XML NAMESPACE

XML Namespaces provide a method to avoid element name conflicts.

Name Conflicts

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.

This XML carries HTML table information:

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

This XML carries information about a table (a piece of furniture):

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

Solving the Name Conflict Using a Prefix

Name conflicts in XML can easily be avoided using a name prefix.

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>
```

In the example above, there will be no conflict because the two <table> elements have different names.

XML Namespaces - The xmlns Attribute

- 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="http://www.w3schools.com/furniture">
                 <f:name> African Coffee Table </f:name>
                <f:width> 80 </f:width>
                 <f:length> 120 </f:length>
    </f:table>
</root>
```

In the example above:

- The xmlns attribute in the first <table> element gives the h: prefix a qualified namespace.
- The xmlns attribute in the second <table> element gives the f: prefix a qualified namespace.
- When a namespace is defined for an element, all child elements with the same prefix are associated with the same namespace.

Namespaces can also be declared the XML root element:

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

- The namespace URI is not used by the parser to look up information.
- The purpose of using an URI is to give the namespace a unique name.
- However, companies often use the namespace as a pointer to a web page containing namespace information.

Default Namespaces:

Defining a default namespace for an element saves us from using prefixes in all the child elements.

It has the following syntax:

xmlns="*namespaceURI*"

This XML carries HTML table information:

```
<table xmlns="http://www.w3.org/TR/html4/">
    <tr>
        <td>Apples</td>
        <td>Bananas</td>
    </tr>
</table>
```

XFILES:

3.6 XLINKS

XLink and its attributes

- XLink is used to create hyperlinks within XML documents.
- Any element in an XML document can behave as a link.
- With XLink, the links can be defined outside the linked files.

The anchor element, <a>, within HTML indicates a link to another resource on an HTML page. This could be a location within the same document or a document located elsewhere. In HTML terms, the anchor element creates a hyperlink to another location.

The hyperlink can either appear as straight text, a clickable image, or a combination of both. Although HTML anchor elements contain a lot of functionality, they are still limiting—they require the use of the anchor element (<a>) itself, and they basically sit there waiting for someone to click them before navigating to the specified location.

The **XML Linking Language**, XLink, addresses and overcomes these limitations by allowing a link to another document to be specified on any element within an XML document. The links to other documents can be much more complex than the simple links supported by the HTML specification.

Example for XLink

```
<?xml version="1.0" encoding="UTF-8"?>
<homepages xmlns:xlink="http://www.w3.org/1999/xlink">
  <homepage xlink:type="simple" xlink:href="http://www.w3schools.com"> Welcome
</homepage>
  <homepage xlink:type="simple" xlink:href="http://www.w3.org"> to achariya
</homepage>
</homepages>
```

Output

Welcome to achariya

XLink Attributes

The XML Linking Language creates a link to another resource through the use of attributes specified on elements, not through the actual elements themselves. The XML Linking Language specification supports the attributes listed in the below table

Attribute	Description
xlink:type	This attribute must be specified and indicates what type of XLink is represented or defined.
xlink:href	This attribute contains the information necessary to locate the desired resource.
xlink:role	This attribute describes the function of the link between the current resource and another.
xlink:arcrole	This attributes describes the function of the link between the current resource and another.
xlink:title	This attribute describes the meaning of the link between the resources.
xlink:show	This attribute indicates how the resource linked to should be displayed
xlink:actuate	This attribute specifies when to load the linked resource.
xlink:label	This attribute is used to identify a name for a target resource.
xlink:from	This attribute identifies the starting resource.
xlink:to	This attribute identifies the ending resource.

TYPES OF XLINK

The XML Linking Language offers two major types of links:

1. Simple links
2. Extended links

Within XLink, a simple link is a convenient, by which to associate two resources. These resource - one local and one remote - are connected by an arc, always making

a simple link an outbound link. An extended link associates any number of resources together. Furthermore, those resources may be both local and remote.

3.7 XPATH

The **XML Path Language (XPath)** is a standard for creating expressions that can be used to find specific pieces of information within an XML document.

XPath expressions are used by both XSLT (for which XPath provides the core functionality) and XPointer to locate a set of nodes.

To understand how XPath works, it helps to imagine an XML document as a tree of nodes consisting of both elements and attributes. An XPath expression can then be considered a sort of roadmap that indicates the branches of the tree to follow and what limbs hold the information desired.

XPath expressions have the ability to locate nodes based on the nodes' type, name, or value or by the relationship of the nodes to other nodes within the XML document. In addition to being able to find nodes based on these criteria, an XPath expression can also return any of the following:

- ✓ A node set
- ✓ A Boolean value
- ✓ A string value
- ✓ A numeric value

XML documents are a hierarchical tree of nodes. There is a similarity between URLs and XPath expressions.

- URLs represent a navigation path of a hierarchical file system.
- XPath expressions represent a navigation path for a hierarchical tree of nodes.

The priority for evaluating XPath expressions is as follows:

1. Grouping
2. Filters
3. Path operations

XPath Syntax:

The XML Path Language provides a declarative notation, termed a *pattern*, used to select the desired set of nodes from XML documents. Each pattern describes a set of matching nodes to select from a hierarchical XML document. Each pattern describes a "navigation" path to the desired set of nodes similar to the Uniform Resource Identifier (URI) syntax. However, instead of navigating a file system, the XML Path Language navigates a hierarchical tree of nodes within an XML document.

Each "query" of an XML document occurs from a particular starting node that defines the context for the query. The context for the query has a very large impact on the results. For instance, the pattern that locates a node from the root of an XML document will most likely be a very different pattern when looking for the same node from somewhere else in the hierarchy.

One possible result from performing an XPath query is a node set, or a collection of nodes matching specified search criteria. To receive these results, a "location path" is needed to locate the result nodes. These location paths select the resulting node set relative to the current context. A location path is, itself, made up of one or more location steps. Each step is further comprised of three pieces:

- An axis
- A node test
- A Predicate

Therefore, the basic syntax for an XPath expression would be something like this:

axis::node test[predicate]

Using this basic syntax and the XML document in operators table, we could locate all the <c> nodes by using the following XPath expression:

/a/b/child::*

Alternatively, we could issue the following abbreviated version of the preceding expression:

/a/b/c

All XPath expressions are dependent on the current context. The context is the current location within the tree of nodes. Therefore, if we're currently on the second <b> element within the XML document, select all the <c> elements contained within that <b> element by using the following XPath expression:

./c

This is what's known as a "relative" XPath expression.

3.8 XPOINTER

- XPointer allows links to point to specific parts of an XML document.
- XPointer uses XPath expressions to navigate in the XML document.
- XPointer describes a location within an external document.
- XPointer can target a point within that XML document or a range within the target XML document.

The **XML Pointer Language (XPointer)** describes a location within an external document, an XPointer can target a point within that XML document or a range within the target XML document. XPointer builds on the XPath specification, the location steps within an XPointer are comprised of the same elements that make up XPath location steps.

XPointer provides two more important node tests:

- point()
- range()

The XPointer specification added the concept of a location within an XML document. Within XPointer, a location can be an XPath node, a point, or a range.

A **point** can represent the location immediately before or after a specified character or the location just before or just after a specified node.

A **range** consists of a start point and an endpoint and contains all XML information between those two points. In fact, the XPointer specification extends the node types to include points and ranges. XPointer expressions also allow predicates to be specified as part of a location step in much the same fashion XPath expressions allow for them. As with XPath expressions, XPointer expressions have specific functions to deal with each specific predicate type.

However, the XPointer specification also adds an additional function named unique().This new function indicates whether an XPointer expression selects a single location rather than multiple locations or no locations at all.

For an XPath expression, the result from a location step is known as a ***node set***; for an XPointer expression, the result is known as a ***location set***. To reduce the confusion, the XPointer specification uses a different term for the results of an expression:

Because an XPointer expression can yield a result consisting of points or ranges, the idea of the *node set* had to be extended to include these types. Therefore, to prevent confusion, the results of an XPointer expression are referred to *location sets*.

Four of the functions that return location sets, id(), root(), here(), and origin(),

Some XPointer Functions That Return Location Sets

Function	description
id()	Selects all nodes with the specified ID
root()	Selects the root element as the only location in a location set
here()	Selects the current element location in a location set
origin()	Selects the current element location for a node using an out-of-line link

3.9 XML WITH XSL

Creating the XSL Style Sheet

The next step is to create the XSL style sheet. XSL style sheets are XML documents; as a result, they must be well formed. An XSL style sheet has the following general structure:

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="URI" version="1.0">
    <!--XSL-T CONVERSION RULES-->
</xsl:stylesheet>
```

The <xsl:stylesheet> element defines how the XSLT processor should process the current XSL document. The xmlns attribute is the namespace definition. The XSL Transformation engine reads the xmlns attribute and determines whether it supports the given namespace. The xmlns attribute specifies the XSL prefix. All XSL elements and types in the document use the prefix.

The xmlns attribute value contains a Uniform Resource Identifier (URI), which serves as a generic method for identifying entities on the World Wide Web.

The XSL style sheet contains HTML text and XSL elements. The HTML text forms the basis of the desired output page. The XSL elements are template rules

for the XSLT processor. A template is associated with a given element in the XML document. In our example, a template is defined to match on the <book> element using the following code:

```
<xsl:template match="/book">
    <!--static text and xsl rules -->
</xsl:template>
```

XSLT defines the <xsl:value-of> element for retrieving data from a XML document. The <xsl:value-of> element contains a select attribute. This attribute value is the name of the actual XML element you want to retrieve. For example, the following code will retrieve the title of the book:

```
<xsl:value-of select="title" />
```

To create the file book_view.xsl. This style sheet will create an HTML page that contains information about the book, which is stored in the file book.xml. The style sheet contains the basic format of an HTML page and uses XSL elements to retrieve the data. Currently, the XSL elements are merely placeholders in the document. Once an XSL processor accesses the XSL style sheet, the processor executes the XSL elements and replaces them with the appropriate data from the XML document. Below code contains the complete code for book_view.xsl.

Example

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
<xsl:template match="/book">
  <html>
    <body>
      <b> Title: </b> <xsl:value-of select="title" />
      <p/>
      <b> By: </b> <xsl:value-of select="author" />
      <p/>
      <b> Cost: </b> <xsl:value-of select="price" />
      <p/>
      <b> Category: </b> <xsl:value-of select="category" />
      <p/>
      <b> Description </b>
      <p/>
      <i> <xsl:value-of select="summary" /> </i>
    </body>
  </html>
</xsl:template>
</xsl:stylesheet>
```

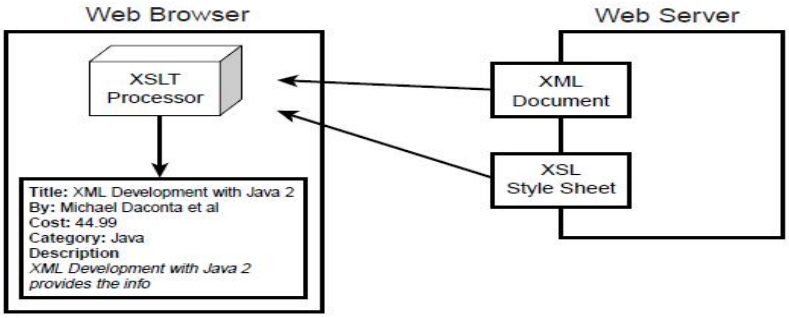
The XSLT Processor

Two techniques are available for performing the XSLT processing:

1. Client-side processing
2. Server-side processing

Client-side XSLT processing commonly occurs in a Web browser. The Web browser includes an XSLT processor and retrieves the XML document and XSL style sheet.

The client-side technique offloads the XSLT processing to the client machine. This minimizes the workload on the Web server. However, the disadvantage is that the Web browser must provide XSLT support. Netscape Communicator 6 and Microsoft Internet Explorer 6 support the XSLT 1.0 specification.



Processing XSLT in a Web browser

Server-side XSLT processing occurs on the Web server or application server. A server-side process such as an Active Server Page (ASP), Java Server Page (JSP), or Java servlet will retrieve the XML document and XSL style sheet and pass them to an XSLT processor. The output of the XSLT processor is sent to the client Web browser for presentation. The output is generally a markup language, such as HTML, that is understood by the client browser. The application interaction is shown below figure.

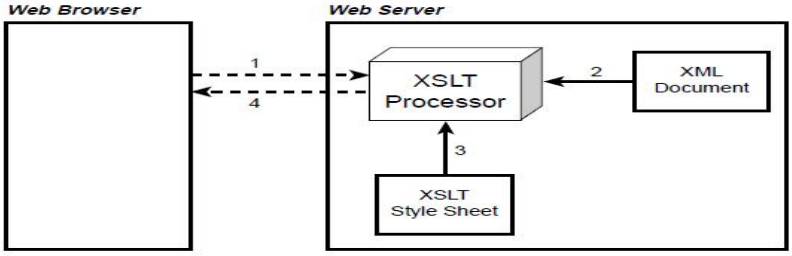


Fig 3.1: Processing XSLT on the server side

An advantage of the server-side technique is browser independence. The output document is simply an HTML file. This technique supports the older browser versions and makes the application more robust and versatile.

The server-side technique, the application can support a diverse collection of clients. The application can detect the user-agent, such as a WAP-enabled mobile phone, and send back a document containing a Wireless Markup Language (WML) tag. The WAP-enabled phone can render the content using the built-in WML mini-browser.

3.10 XSL FORMATTING OBJECTS

- The XSL technology is also composed of **XSL Formatting Objects (XSL-FO).**

- XSL-FO was designed to assist with the printing and displaying of XML data. The main emphasis is on the document layout and structure. This includes the dimensions of the output document, including page headers, footers, and margins.
- XSL-FO also allows the developer to define the formatting rules for the content, such as font, style, color, and positioning.XSL-FO is a sophisticated version of Cascading Style Sheets (CSS). XSL-FO borrows a lot of the terminology and elements from CSS.
- XSL-FO documents are well-formed XML documents. An XSL-FO formatting engine processes XSL-FO documents.
  The two techniques for creating XSL-FO documents.
  - The first is to simply develop the XSL-FO file with the included data.
  - The second technique is to dynamically create the XSL-FO file using an XSLT translation.

XSL-FO Formatting Engines

Many of the XSL-FO formatting engines implement a subset of the XSL-FO specification. Also, the browser support for XSL-FO is nonexistent. Engines are available that allow you to experiment with the basic features of XSL-FO. To use the Apache XSL-FOP to generate PDF documents from XML.

XSL-FO Engine	Web Site
Apache XSL-FOP	xml.apache.org
XEP	www.renderx.com
iText	www.lowagie.com/iText/
Unicorn	www.unicorn-enterprises.com

To create a simple XSL-FO document. Once the document is created, we will use the Apache XSL-FOP formatter to convert the document to a PDF file. The application interaction is illustrated below figure.

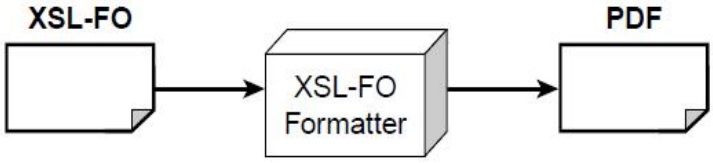


Fig 3.2: Apache XSL-FOP generating PDF documents

Basic Document Structure

An XML-FO document follows the syntax rules of XML; as a result, it is well formed. XSL-FO elements use the following namespace:

http://www.w3.org/1999/XSL/Format

The following code snippet shows the basic document setup for XSL-FO:

```
<?xml version=”1.0” encoding=”utf-8”?>
  <fo:root xmlns:fo=”http://www.w3.org/1999/XSL/Format”>
    <!-- layout master set -->
     <!-- page masters: size and layout -->
```

```
    <!-- page sequences and content -->
</fo:root>
```

The **element <fo:root>** is the root element for the XSL-FO document. An XSL-FO document can contain the following components:

1. Page master
2. Page master set
3. Page sequences

3.11 PARSING XML USING DOM

- The **Document Object Model (DOM)** provides a way of representing an XML document in memory so that it can be manipulated by your software.
- DOM is a standard application programming interface (API) that makes it easy for programmers to access elements and delete, add, or edit content and attributes.
- DOM was proposed by the World Wide Web Consortium (W3C) in August of 1997 in the User Interface Domain. The Activity was eventually moved to the Architecture Domain in November of 2000.
- DOM interfaces are defined independent of any particular programming language.
- DOM code in just about any programming language, such as Java, ECMAScript (a standardized version of JavaScript/JScript), or C++.

The XML DOM is:

- A standard object model for XML
- A standard programming interface for XML
- Platform- and language-independent
  - ✓ The XML DOM defines the **objects and properties** of all XML elements, and the **methods** (interface) to access them.
  - ✓ In other words: **The XML DOM is a standard for how to get, change, add, or delete XML elements.**

DOM Core

- DOM is a tree structure that represents elements, attributes, and content.

Consider a simple XML document

```
<purchase-order>
        <customer> James Bond </customer>
        <merchant> Spies R Us </merchant>
        <items>
                <item> Night vision camera </item>
                <item> Vibrating massager </item>
        </items>
</purchase-order>
```

Tree structure representing the XML document as

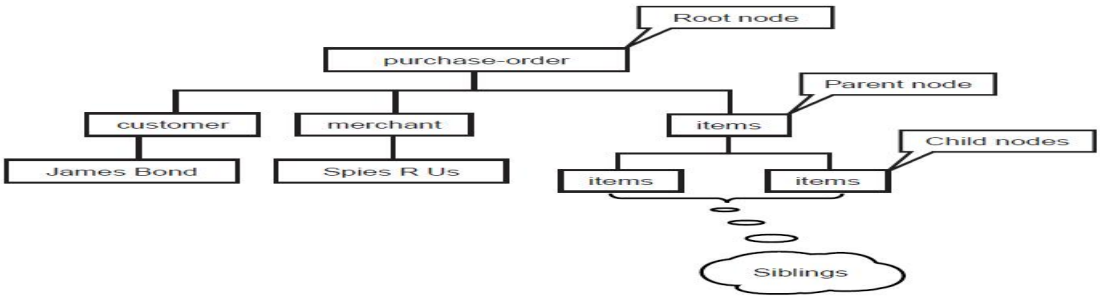


Fig 3.3 Tree Structure of XML Document

The DOM says:

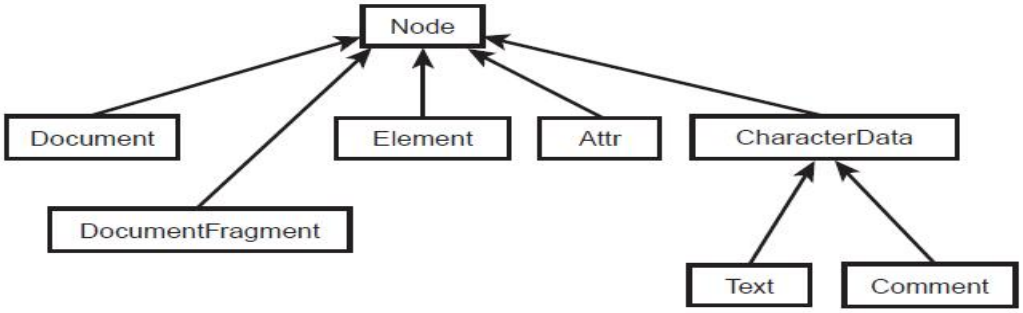
- The entire document is a document node
- Every XML element is an element node
- The text in the XML elements are text nodes
- Every attribute is an attribute node
- Comments are comment nodes

DOM Interfaces

DOM interfaces are defined in IDL so that they are language neutral.

Interface	Description
Node	The primary interface for the DOM. It can be an element, attribute, text, and so on, and contains methods for traversing a DOM tree.
NodeList	An ordered collection of Nodes.
NamedNodeMap	An unordered collection of Nodes that can be accessed by name and used with attributes.
Document	An Node representing an entire document. It contains the root Node.
DocumentFragment	A Node representing a piece of a document. It's useful for extracting or inserting a fragment into a document.
Element	A Node representing an XML element.
Attr	A Node representing an XML attribute.
CharacterData	A Node representing character data.
Text	A CharacterData node representing text.
Comment	A CharacterData node representing a comment.
DOMException	An exception raised upon failure of an operation.
DOMImplementation	Methods for creating documents and determining whether an implementation has certain features.

The relationships among the interfaces



3.12 SAX

- The **Simple API for XML (SAX)** can only be used for parsing existing documents.
- SAX (Simple API for XML) is an event-driven online algorithm for parsing XML documents, with an API interface developed by the XML-DEV mailing list.
- SAX parsers operate on each piece of the XML document sequentially.
- SAX is an API that can be used to parse XML documents. A *parser* is a program that reads data a character at a time and returns manageable pieces of data.
- For example, a parser for the English language might break up a document into paragraphs, words, and punctuation.
- SAX provides a framework for defining event listeners, or ***handlers*.**
- These handlers are written by developers interested in parsing documents with a known structure. The handlers are registered with the SAX framework in order to receive events.
- Events can include start of document, start of element, end of element, and so on. The handlers contain a number of methods that will be called in response to these events. Once the handlers are defined and registered, an input source can be specified and parsing can begin.

SAX Basics

To illustrate how SAX works, consider a simple document:

```
<?xml version=”1.0” encoding=”UTF-8”?>
<fiction>
        <book author=”Herman Melville”> Moby Dick </book>
</fiction>
```

If you want to parse this document using SAX, you would build a *content handler* by creating a Java class that implements the Content Handler interface in the org.xml.saxmpackage.

Once you have a content handler, you simply register it with a SAX XMLReader, set up the input source, and start the parser. Next, the methods in your content handler will be called when the parser encounters elements, text, and other data. Specifically, the events generated by the preceding example will look something like this:

```
start document
start element: fiction
start element: book (including attributes)
characters: Moby Dick
end element: book
```

end element: fiction
end document

The events reported follow the content of the document in a linear sequence. There are a number of other events that might be generated in response to processing instructions, errors, and comments.

XML Processing with SAX:

A parser that implements SAX (i.e., *a SAX Parser*) functions as a stream parser, with an event-driven API. The user defines a number of callback methods that will be called when events occur during parsing.

The SAX events include

- XML Text nodes
- XML Element Starts and Ends
- XML Processing Instructions
- XML Comments

SAX Packages

- The SAX 2.0 API is comprised of two standard packages and one extension package.
- The standard packages are
  - org.xml.sax
  - org.xml.helpers

org.xml.sax package

The org.xml.sax package contains the basic classes, interfaces, and exceptions needed for parsing documents.

Name	Description
Interfaces	
Attributes	Interface for a list of XML attributes.
ContentHandler	Receives notification of the logical content of a document.
DocumentHandler	ContentHandler interface, which includes namespace support.
DTDHandler	Receives notification of basic DTD-related events.
EntityResolver	Basic interface for resolving entities.
ErrorHandler	Basic interface for SAX error handlers.
Locator	Interface for associating a SAX event with a document location.
XMLFilter	Interface for an XML filter.
XMLReader	Interface for reading an XML document using callbacks.
Classes	
HandlerBase	DocumentHandler interface.
InputSource	A single input source for an XML entity.
Exceptions	
SAXException	Encapsulates a general SAX error or warning.

SAXNotRecognizedException	Exception class for an unrecognized identifier.
SAXNotSupportedException	Exception class for an unsupported operation.
SAXParseException	Encapsulates an XML parse error or warning.

(ii) org.xml.sax.helpers package

The org.xml.sax.helpers package contains additional classes that can simplify some of your coding and make it more portable

Class	Description
AttributesImpl	Default implementation of the Attributes interface.
DefaultHandler	Default base class for SAX2 event handlers.
LocatorImpl	Provides an optional convenience implementation of Locator.
NamespaceSupport	Encapsulate namespace logic for use by SAX drivers.
ParserAdapter	Adapts a SAX1 Parser as a SAX2 XMLReader.
XMLFilterImpl	Base class for deriving an XML filter.
XMLReaderAdapter	Adapts a SAX2 XMLReader as a SAX1 Parser.
XMLReaderFactory	Factory for creating an XML reader.

3.13 XML INTEGRATING WITH DATABASE

XML and database integration is important because XML provides a standard technique to describe data.

XML Database Solutions

A large number of XML database solutions are available, and they generally come in two flavors:

1. database mapping
2. native XML support

(i) XML Database Mapping

The XML database solution provides a mapping between the XML document and the database fields. The system dynamically converts SQL result sets to XML documents. Depending on the sophistication of the product, it may provide a graphical tool to map the database fields to the desired XML elements. Other tools support a configuration file that defines the mapping. These tools continue to store the information in relational database management system (RDBMS) format. The simply provide an XML conversion process that is normally implemented as a server-side Web application.

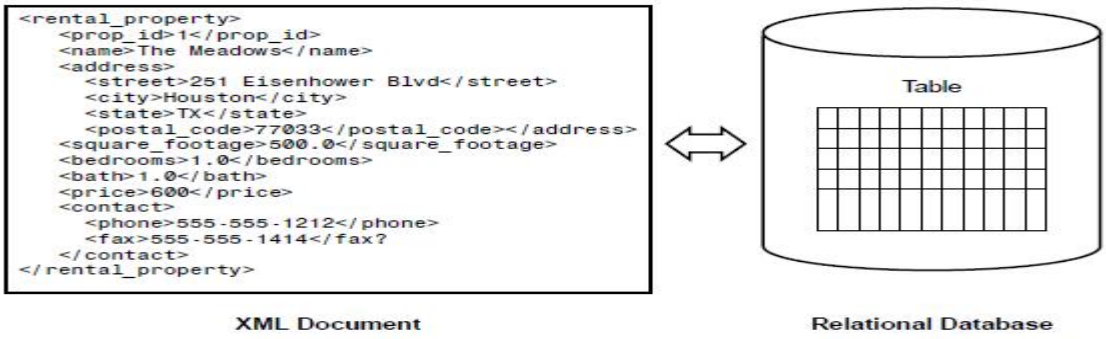


Fig 3.4: Mapping XML documents to database fields

(ii) Native XML Support:

The XML database solution actually stores the XML data in the document in its native format. Each product uses its own proprietary serialization technique to store the data. However, when the data is retrieved, it represents an XML document.

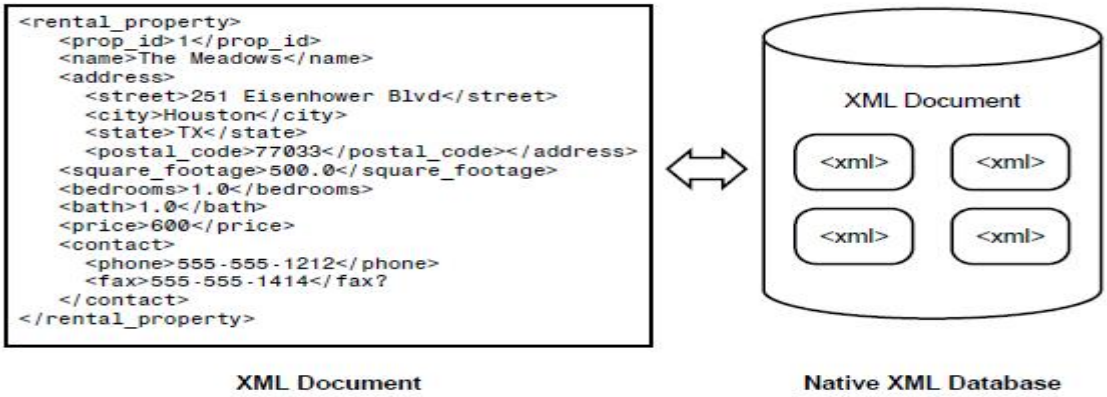


Fig 3.5: Native XML databases

Modeling Databases in XML:

Learn how to model a database in XML using Java. When we model a database, we provide an external representation of the database contents. For our sample program, we'll utilize a database that contains information on rental properties. We'll model the rental property database as an XML document.

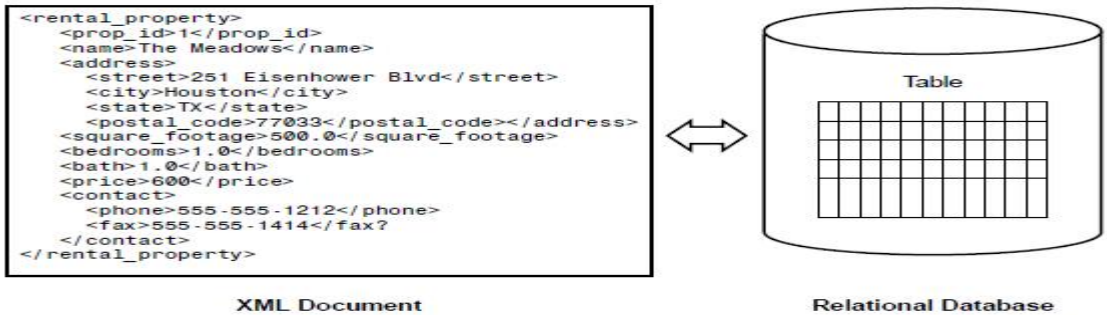


Fig 3.6: Modeling Databases in XML

One possible solution is to use Java servlets and JDBC. Java servlets are server-side components that reside in a Web server or application server. Java servlets are commonly used to handle requests from Web browsers using the HTTP protocol.

A key advantage to using servlets is the thin-client interface. The servlets handle the request on the server side and respond by generating an HTML page dynamically. This lowers the requirement on the client browser. The browser only has to provide support of HTML. As a result, there is zero client-side administration.

Java applets require the browser to support the correct version of the Java Virtual Machine (JVM). This has been a thorny issue with the Java community since the early days of applet development. If the browser doesn't support Java, the applet will not execute. Of course, there are a number of workarounds, such as the Java Plug-In and Java Web Start.

We can develop a servlet that uses JDBC. The servlet will make the appropriate query to the database and use Java database Connectivity (JDBC) API result set metadata to create the elements.

3.14 FORMATTING XML FOR THE WEB

(i) XML Presentation Using CSS:

```
<!-- This style sheet will be referenced as notestyle.css -->
Note
{
    display: block
}
From, To
{
   display:block;
   font-family:verdana;
   font-size:15px;
   margin-bottom:5px
}
Subject
{
    display:block;
    font-family:verdana;
    font-size:13px;
    font-weight:bold;
    margin-bottom:10px
}
Body
{
    display:block;
    font-family:verdana;
    font-size:12px
}
```

In this listing are five style selectors: Note, From, To, Subject, and Body. Each selector listed represents the name of an XML element. The styles associated with each selector will be applied to XML elements that have matching names A CSS style sheet may be attached to an XML document through the use of the special XML processing instruction <?xml-stylesheet?>. There are two attributes to the xml-stylesheet processing instruction: type and href. The type attribute sets the MIME type for the CSS style sheet. Its value should always be text/css. The href attribute gives the URL for the location of the CSS style sheet.

The href attribute of the xml-stylesheet processing instruction assumes that the CSS style sheet is located in the same directory. The href attribute value could also be a relative URL or an absolute URL.

Applying CSS to an XML Document

```
<?xml version=”1.0”?>
<!--
    This is referencing the style sheet from calling it notestyle.css here
-->
<?xml-stylesheet type=”text/css” href=”notestyle.css”?>
<Note>
    <From> From: Bob </From>
    <To> To: Jenny </To>
    <Subject> Subject: Hello Friend! </Subject>
    <Body> Just thought I would drop you a line. </Body>
</Note>
```

Output

From: Bob
To: Jenny
Subject: Hello Friend!
Just thought I would drop you a line.

(ii) XHTML

- XHTML stands for E**X**tensible **H**yper **T**ext **M**arkup **L**anguage
- XHTML is almost identical to HTML
- XHTML is stricter than HTML
- XHTML is HTML defined as an XML application
- XHTML is supported by all major browsers

Variants of XHTML

XHTML, a document must be validated against one of three DTDs

1. Strict
2. Transitional
3. Frameset

Strict DTD

A strictly conforming XHTML document that references the Strict DTD will have the following Document Type Declaration:

<!DOCTYPE html PUBLIC “-//W3C//DTD XHTML 1.0 Strict//EN”
“http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd”>

Strict DTD means that the XHTML document will have the following characteristics:

- There will be a strict separation of presentation from structure. Style sheets are used for formatting, and the XHTML markup is very clean and uncluttered. There are no optional vendor-specific HTML extensions.
- The Document Type Definition must be present and placed before the <html> element in the document.
- The root element of the document will be <html>.
- The <html> element will have the xmlns attribute in order to designate the XHTML namespace.
- The document is valid according to the rules defined in the Strict DTD.