XML,WebServices & AJAX

Rs=36/=

Course Objective:

1. Developing an xml document that acts as textual databases and developing java applications that communicate with xml documents, Use XML to transport the data across different applications in language independent manner(**XML**)
2. Exposing java based business components (java Bean/EJB / Spring beans...etc) as services over the web. And writing an application which consumes those services(**Webservices**)
3. Developing rich internet applications, which provides asynchronous/fast response without page refresh(**AJAX**)

XML

Q.) What is XML?

- XML stands for EXtensible Markup Language
- XML is a markup language much like HTML
- It is also a specification. [From W3C {World Wide Web Consortium }]
- XML was designed to describe data.
- XML is self describing.

Q.) What is markup? (Markup = Tagging)

Enclosing textual content with in textual codes(tags) is nothing but markup.

Q.) What are the similarities between HTML and XML?

- Both are languages of web.
- Both are markup languages.
- Both are originated form SGML. [Standardized General Markup Language(complex) <- GML (platform dependent)]
- Tags are basic building blocks of both HTML and XML documents.

Q.) Is XML is Replacement for HTML?

No. Their goals are different.

Q.) What are the differences between HTML and XML?

HTML	XML

1. Pre defined tags	1. User defined tags
2. Limited no. of tags	2. Tags are extensible
3. Tags are case insensitive	3. Tags are case sensitive
4. Tags are meant for displaying the data but not for describing the data.	4.Tags are meant for describing the data
5. HTML focuses on how data looks	5.) XML focuses on what data is

Q.) What is the purpose of XML?

- XML is used to exchange the information between applications in language (java, .net, PHP) independent, vendor independent and platform independent manner over web.
- XML documents are used as textual databases.
- XML documents are used as deployment descriptors
- XML is Used to Create new Internet Languages(**XHTML, WSDL, WAP, SMIL...etc**)

Q.) What XML Does?

- Nothing!!!
- Just XML was created to structure, store and transport the data.

The following example is a email to SardarG, from Bantha, stored as XML:

```
<email>
    <to>SardarG</to>
    <from>Bantha</from>
    <subject>Reminder(Party)</subject>
    <body>Don't forget me this weekend! </body>
</ email>
```

The email above is quite self descriptive. It has sender and receiver information, it also has a subject and a message body.

But still, this XML document does not DO anything. It is just information wrapped in tags. Someone must write a piece of software to send, receive or display it.

Q.) Who invented xml tags?

With XML You Invent Your Own 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.

That is because the XML language has no predefined tags.

The tags used in HTML are predefined. HTML documents can only use tags defined in the HTML standard (like <p>, <h1>, etc.).

XML allows the author to define his/her own tags and his/her own document structure.

Q.) What is an XML document?

- Any text file that is developed with **.xml** extension is an xml document.

Q.) What is xml application?

- Any computer application that works with an xml document is nothing but an xml application.
- Working with XML document means, perform CRUD operations on the XML.

XML Syntax

An example XML document:

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

The first line in the document: *The XML declaration* should always be included. It defines the XML version of the document. In this case the document conforms to the 1.0 specification of XML:

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

The next line defines the first element of the document (the root element):

```
<note>
```

The next lines defines 4 child elements of the root (to, from, heading, and body):

```
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
```

The last line defines the end of the root element:

```
</note>
```

All XML elements must have a closing tag

In HTML some elements do not have to have a closing tag. The following code is legal in HTML:

```
<p>This is a paragraph
```

```
<p>This is another paragraph
```

In XML all elements must have a closing tag like this:

```
<p>This is a paragraph</p>
<p>This is another paragraph</p>
```

XML tags are case sensitive

XML tags are case sensitive. The tag <Letter> is different from the tag <letter>.

Opening and closing tags must therefore be written with the same case:

```
<Message>This is incorrect</message>
```

```
<message>This is correct</message>
```

All XML elements must be properly nested

In HTML some elements can be improperly nested within each other like this:

```
<b><i>This text is bold and italic</b></i>
```

In XML all elements must be properly nested within each other like this

```
<b><i>This text is bold and italic</i></b>
```

All XML documents must have a root tag

All XML documents must contain a single tag pair to define the root element. All other elements must be nested within the root element. All elements can have sub (children) elements. Sub elements must be in pairs and correctly nested within their parent element:

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

Attribute values must always be quoted

XML elements can have attributes in name/value pairs just like in HTML. In XML the attribute value must always be quoted. Study the two XML documents below. The first one is incorrect, the second is correct:

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

```
<note date=12/11/99>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

```
<?xml version="1.0"?>
<note date="12/11/99">
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

Entity References

Some characters have a special meaning in XML.

If you place a character like "<" inside an XML element, it will generate an error because the parser interprets it as the start of a new element.

This will generate an XML error:

```
<message>if salary < 1000 then</message>
```

To avoid this error, replace the "<" character with an **entity reference**:

```
<message>if salary &lt; 1000 then</message>
```

There are 5 predefined entity references in XML:

<	<	less than
>	>	greater than
&	&	ampersand
'	'	apostrophe
"	"	quotation mark

Note: Only the characters "<" and "&" are strictly illegal in XML. The greater than character is legal, but it is a good habit to replace it.

Comments in XML

The syntax for writing comments in XML is similar to that of HTML.

<!-- This is a comment -->

White-space is Preserved in XML

HTML truncates multiple white-space characters to one single white-space:

HTML:	Hello Tove
Output:	Hello Tove

With XML, the white-space in a document is not truncated.

Q.) What is a well-formed XML document?

- If an XML document confirms the syntactical rules (above specified) of XML specification is said to be well-formed.
- An XML document is said to be well-formed if it observes the following rules.

1. It must begin with the XML declaration
2. It must have one unique root element
3. Start-tags must have matching end-tags
4. Elements are case sensitive
5. All elements must be closed
6. All elements must be properly nested
7. All attribute values must be quoted
8. Entities must be used for special characters

Q.)What is an XML Element?

An XML element is everything from (including) the element's start tag to (including) the element's end tag.

An element can contain:

1. other elements
2. text
3. attributes
4. or a mix of all of the above...

Q.) What are the XML Naming Rules?

XML elements must follow these naming rules:

1. Names can contain letters, numbers, and other characters
2. Names cannot start with a number or punctuation character
3. Names cannot start with the letters xml (or XML, or Xml, etc)

4. Names cannot contain spaces

Any name can be used, no words are reserved.

Q.)What are the Best Naming Practices?

1. Make names descriptive. Names with an underscore separator are nice: <first_name>, <last_name>.
2. Names should be short and simple, like this: <book_title> not like this: <the_title_of_the_book>.
3. **Avoid "-" characters**. If you name something "first-name," some software may think you want to subtract name from first.
4. **Avoid "." characters.** If you name something "first.name," some software may think that "name" is a property of the object "first."
5. **Avoid ":" characters.** Colons are reserved to be used for something called **namespaces**.

XML Attributes

- XML elements can have attributes, just like HTML.
- Attributes provide additional information about an element.
- Attributes can occur in any order in an XML element.
- Attribute is unique in an XML element. i.e. It should not be repeated.

For example,

<x a="10" a="20" /> (wrong)

XML Attributes Must be Quoted

Attribute values must always be quoted. Either single or double quotes can be used. For a person's gender, the person element can be written like this:

```
<person gender="female">
```

or like this:

```
<person gender='female'>
```

If the attribute value itself contains double quotes you can use single quotes, like in this example:

```
<gangster name='George "Shotgun" Ziegler'>
```

or you can use character entities:

```
<gangster name="George &quot;Shotgun&quot; Ziegler">
```

Use of Elements vs. Attributes

Take a look at these examples:

```
<person gender="female">
  <firstname>Anna</firstname>
  <lastname>Smith</lastname>
</person>
```

```
<person>
  <gender>female</gender>
  <firstname>Anna</firstname>
  <lastname>Smith</lastname>
</person>
```

In the first example `gender` is an attribute. In the last, `gender` is an element. Both examples provide the same information.

There are no rules about when to use attributes or when to use elements. Attributes are handy in HTML. In XML my advice is to avoid them. Use elements instead.

Avoid using attributes? (I say yes!)

These are some of the problems using attributes:

- attributes cannot contain multiple values (elements can)
- attributes are not expandable (for future changes)
- attributes cannot describe structures (like child elements can)
- attributes are more difficult to manipulate by program code
- attribute values are not easy to test against a DTD/XSD

My Favorite Way

The following three XML documents contain exactly the same information:

A date attribute is used in the first example:

```
<note date="10/01/2008">
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

A date element is used in the second example:

```
<note>
  <date>10/01/2008</date>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

An expanded date element is used in the third: (THIS IS MY FAVORITE):

```
<note>
  <date>
    <day>10</day>
    <month>01</month>
    <year>2008</year>
  </date>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

XML Attributes for Metadata

Sometimes ID references are assigned to elements. These IDs can be used to identify XML elements in much the same way as the id attribute in HTML. This example demonstrates this:

```
<messages>
  <note id="501">
    <to>Tove</to>
    <from>Jani</from>
    <heading>Reminder</heading>
    <body>Don't forget me this weekend!</body>
  </note>
  <note id="502">
    <to>Jani</to>
    <from>Tove</from>
    <heading>Re: Reminder</heading>
    <body>I will not</body>
  </note>
</messages>
```

The id attributes above are for identifying the different notes. It is not a part of the note itself.

What I'm trying to say here is that metadata (data about data) should be stored as attributes, and the data itself should be stored as elements.

XML Documents Form a Tree Structure

The elements in an XML document form a document tree. All elements can have sub elements (child elements):

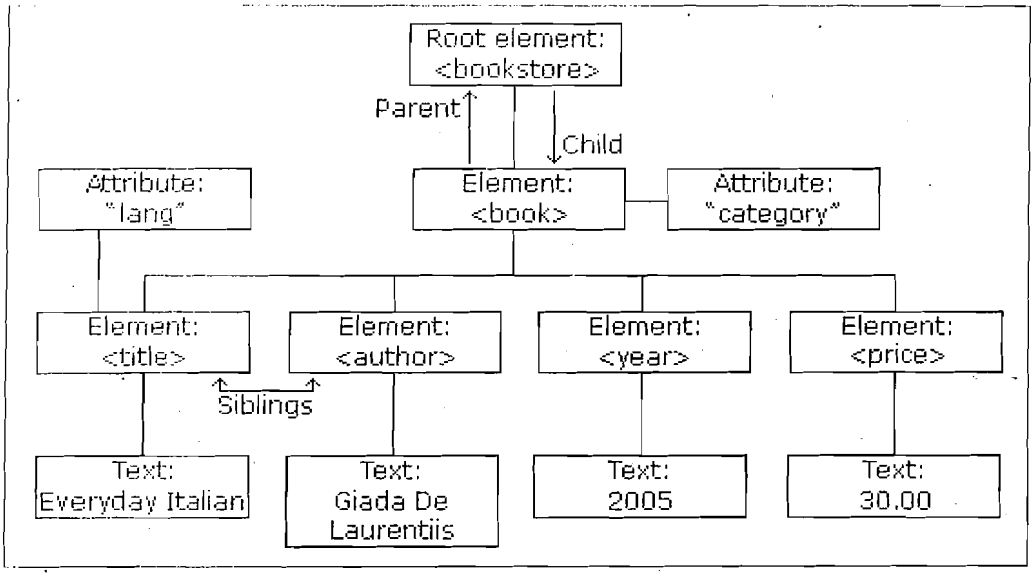
```
<root>
  <child>
```

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

The terms **parent, child, and sibling** are used to describe the relationships between elements. Parent elements have children. Children on the same level are called siblings (brothers or sisters).

All elements can have text content and attributes (just like in HTML).

Example:



The image above represents one book in the XML below:

```
<bookstore>
  <book category="COOKING">
    <title lang="en">Everyday Italian</title>
    <author>Giada De Laurentiis</author>
    <year>2005</year>
    <price>30.00</price>
  </book>
  <book category="CHILDREN">
    <title lang="en">Harry Potter</title>
    <author>J K. Rowling</author>
    <year>2005</year>
    <price>29.99</price>
  </book>
  <book category="WEB">
    <title lang="en">Learning XML</title>
    <author>Erik T. Ray</author>
    <year>2003</year>
    <price>39.95</price>
  </book>
</bookstore>
```

Q.) How to view XML files?

- By using Browsers, text editors, XML editors we can view XML files.

Q.) How to develop an XML document?

- In order to develop a useful xml document, we need to develop one more file. I.e. DTD or XSD file.
- In DTD file or XSD file we specify the XML vocabulary.

Q.) What is a valid XML?

- If an XML document is developed according to rules specified in DTD / XSD, it is known as valid XML document.

NOTE: Every valid XML is a well-formed, but vice versa need not be true.

[First well-formedness is checked then it checks for validness will be checked]

Q.) How to check the well-formedness of XML document?

- Using XML parsers, XML Editors, Browsers....etc.

Q.) How to validate XML document?

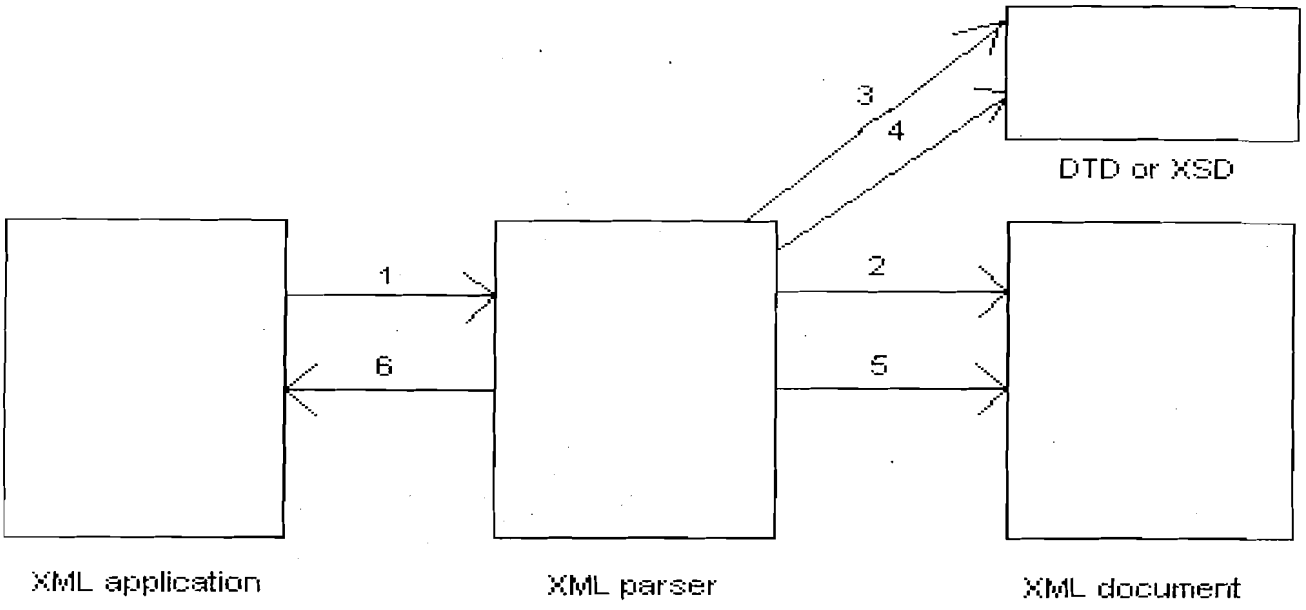
- Using XML parsers, XML editors...etc.

Every XML application needs two things to work with XML

1. Parser
2. API

Q.) What is an XML parser? What are its functions?

- XML parser is an API that enables XML applications to work with XML document.
- An XML parser performs the following things.
  1. Reading the XML document.
  2. Checking well-formedness
  3. Verify its validity.
  4. Making XML data available to XML application.



1. XML application instantiating the parser and specifying the XML file to the parser. (i.e. Passing the name of the XML file name to the Parser)
2. Parser reads the specified XML document and verifies its well-formedness.
3. In the XML document parser gets the information about the DTD or XSD. Parser verifies the correctness of the DTD or XSD.
4. Parser reads the metadata specified in the DTD or XSD file into the memory.
5. Basing on the metadata read into memory; XML Parser verifies the validity of the XML document.
6. Parser makes XML content (data) available to the XML application either in the form of a tree structure or in the form of events.

{The way DOM makes available data to the XML application is differ from JAXB }

Valid XML document

- XML parser can make data stored in XML document available to XML application if and only iff the XML documents are valid.
- There are two approaches for developing valid XML.
  - Using DTD
  - Using XSD

Q.)What is DTD?

- DTD stands for Document Type Definition.
- A DTD is a text file with .dtd extension.
- If XML file holds data, its corresponding DTD holds Meta data.
- In a DTD legal building blocks of an XML documents are specified. I.e. XML vocabulary is specified in a DTD.

Q.) What are the constituents of DTD file? (Contains)

From a DTD point of view, all XML documents are made up by the following building blocks:

- Elements
- Attributes
- Entities
- PCDATA
- CDATA

Elements

Elements are the **main building blocks** of both XML and HTML documents.

Examples of HTML elements are "body" and "table". Examples of XML elements could be "note" and "message". Elements can contain text, other elements, or be empty. Examples of empty HTML elements are "hr", "br" and "img".

Examples:

```
<body>some text</body>

<message>some text</message>
```

Attributes

Attributes provide **extra information about elements**.

Attributes are always placed inside the opening tag of an element. Attributes always come in name/value pairs. The following "img" element has additional information about a source file:

```
<img src="computer.gif" />
```

The name of the element is "img". The name of the attribute is "src". The value of the attribute is "computer.gif". Since the element itself is empty it is closed by a " /".

Entities

Some characters have a special meaning in XML, like the less than sign (<) that defines the start of an XML tag.

Most of you know the HTML entity: " ". This "no-breaking-space" entity is used in HTML to insert an extra space in a document. Entities are expanded when a document is parsed by an XML parser.

The following entities are predefined in XML:

Entity References	Character
<	<
>	>
&	&

" "

' '

PCDATA - Parsed Character Data

XML parsers normally parse all the text in an XML document.

When an XML element is parsed, the text between the XML tags is also parsed:

```
<message>This text is also parsed</message>
```

The parser does this because XML elements can contain other elements, as in this example, where the <name> element contains two other elements (first and last):

```
<name><first>Bill</first><last>Gates</last></name>
```

and the parser will break it up into sub-elements like this:

```
<name>
  <first>Bill</first>
  <last>Gates</last>
</name>
```

Parsed Character Data (PCDATA) is a term used about text data that will be parsed by the XML parser.

CDATA - (Unparsed) Character Data

The term CDATA is used about text data that should not be parsed by the XML parser.

Characters like "<" and "&" are illegal in XML elements.

"<" will generate an error because the parser interprets it as the start of a new element.

"&" will generate an error because the parser interprets it as the start of an character entity.

Some text, like JavaScript code, contains a lot of "<" or "&" characters. To avoid errors script code can be defined as CDATA.

Everything inside a CDATA section is ignored by the parser.

A CDATA section starts with "**<![CDATA[**" and ends with "**]]>**":

```
<script>
<![CDATA[
```

```
function matchwo(a,b)
{
if (a < b && a < 0) then
  {
  return 1;
  }
else
  {
  return 0;
  }
}
]]>
</script>
```

In the example above, everything inside the CDATA section is ignored by the parser.

Notes on CDATA sections:

A CDATA section cannot contain the string "]]>". Nested CDATA sections are not allowed.

The "]]>" that marks the end of the CDATA section cannot contain spaces or line breaks.

DTD - Elements

Declaring an Element

In the DTD, XML elements are declared with an element declaration. An element declaration has the following syntax:

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

Empty elements

Empty elements are declared with the keyword EMPTY inside the parentheses:

```
<!ELEMENT element-name (EMPTY)>

example:
<!ELEMENT br (EMPTY)>
```

Elements with data

Elements with data are declared with the data type inside parentheses:

```
<!ELEMENT element-name (#CDATA)>
or
<!ELEMENT element-name (#PCDATA)>
or
<!ELEMENT element-name (ANY)>
example:
<!ELEMENT note (#PCDATA)>
```

#CDATA means the element contains character data that is not supposed to be parsed by a parser.
#PCDATA means that the element contains data that IS going to be parsed by a parser.
The keyword ANY declares an element with any content.

If a #PCDATA section contains elements, these elements must also be declared.

Elements with children (sequences)

Elements with one or more children are defined with the name of the children elements inside the parentheses:

```
<!ELEMENT element-name (child-element-name)>
or
<!ELEMENT element-name (child-element-name,child-element-name,.....)>
example:
<!ELEMENT note (to,from,heading,body)>
```

When children are declared in a sequence separated by commas, the children must appear in the same sequence in the document. In a full declaration, the children must also be declared, and the children can also have children. The full declaration of the note document will be:

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

Q.) What is cardinality operator in DTD?

- An XML element can occur zero or more times in the XML document.
- Cardinality operator specifies the no. of times an XML element can occur in XML document.
- In a DTD we have three cardinality operators.
  - ? (0 or 1)
  - + (1 or n)
  - * (0 or n)

Note: If a cardinality operator is not associated with the XML element, that element should occur exactly once in the XML document.

Declaring only one occurrence of the same element

```
<!ELEMENT element-name (child-name)>
```

```
example
<!ELEMENT note (message)>
```

The example declaration above declares that the child element message can only occur one time inside the note element.

Declaring minimum one occurrence of the same element

```
<!ELEMENT element-name (child-name+)>
example
<!ELEMENT note (message+)>
```

The + sign in the example above declares that the child element message must occur one or more times inside the note element.

Declaring zero or more occurrences of the same element

```
<!ELEMENT element-name (child-name*)>
example
<!ELEMENT note (message*)>
```

The * sign in the example above declares that the child element message can occur zero or more times inside the note element.

Declaring zero or one occurrences of the same element

```
<!ELEMENT element-name (child-name?)>
example
<!ELEMENT note (message?)>
```

The ? sign in the example above declares that the child element message can occur zero or one times inside the note element.

Declaring mixed content

```
Example
<!ELEMENT note (to+,from,header,message*,#PCDATA)>
```

The example above declares that the element note must contain at least one **to** child element, exactly one **from** child element, exactly one **header**, zero or more **message**, and some other parsed **character data**as well.

DTD - Attributes

Declaring Attributes

In the DTD, XML element attributes are declared with an ATTLIST declaration. An attribute declaration has the following syntax:

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

As you can see from the syntax above, the ATTLIST declaration defines the element which can have the attribute, the name of the attribute, the type of the attribute, and the default attribute value.

The **attribute-type** can have the following values:

Value	Explanation
CDATA	The value is character data
(eval|eval|..)	The value must be an enumerated value
ID	The value is an 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 predefined

The **attribute-default-value** can have the following values:

Value	Explanation
#DEFAULT value	The attribute has a default value
#REQUIRED	Required attribute
#IMPLIED	optional attribute
#FIXED value	The attribute value is fixed

Attribute declaration example

```
DTD example:
<!ELEMENT square EMPTY>
  <!ATTLIST square width CDATA "0">

XML example:
<square width="100"></square>
```

In the above example the element square is defined to be an empty element with the attributes width of type CDATA. The width attribute has a default value of 0.

Default attribute value

```
Syntax:
<!ATTLIST element-name attribute-name CDATA "default-value">

DTD example:
<!ATTLIST payment type CDATA "check">

XML example:
<payment type="check">
```

Specifying a default value for an attribute, assures that the attribute will get a value even if the author of the XML document didn't include it.

Implied attribute

```
Syntax:
<!ATTLIST element-name attribute-name attribute-type #IMPLIED>
DTD example:
<!ATTLIST contact fax CDATA #IMPLIED>

Valid XML:
<contact fax="555-667788" />

Valid XML:
<contact />
```

Use an implied attribute if you don't want to force the author to include an attribute and you don't have an option for a default value either.

Required attribute

```
Syntax:
<!ATTLIST element-name attribute_name attribute-type #REQUIRED>
DTD example:
<!ATTLIST person number CDATA #REQUIRED>

XML example:
<person number="5677">

Invalid XML:
<person />
```

Use a required attribute if you don't have an option for a default value, but still want to force the attribute to be present.

Fixed attribute value

```
Syntax:
<!ATTLIST element-name attribute-name attribute-type #FIXED "value">


DTD example:
<!ATTLIST sender company CDATA #FIXED "Microsoft">


XML example:
<sender company="Microsoft">
Invalid XML:
<sender company="ysreddy" />
```

Use a fixed attribute value when you want an attribute to have a fixed value without allowing the author to change it. If an author includes another value, the XML parser will return an error.

Enumerated attribute values

```
Syntax:
<!ATTLIST element-name attribute-name (eval|eval|..) default-value>

DTD example:
<!ATTLIST payment type (check|cash) "cash">

XML example:
<payment type="check">
or
<payment type="cash">
```

Use enumerated attribute values when you want the attribute values to be one of a fixed set of legal values.

Element with multiple attributes

```
<!ATTLIST book
      publication CDATA "sekahr"
      pages CDATA #FIXED "150"
      color CDATA #IMPLIED
      isbn ID #REQUIRED  >
```

DTD - Entities

Entities

- Entities as variables used to define shortcuts to common text.
- Entity references are references to entities.
- Entities can be declared internal.
- Entities can be declared external

Internal Entity Declaration

```
Syntax:
<!ENTITY entity-name "entity-value">

DTD Example:
<!ENTITY writer "Jan Egil Refsnes.">
<!ENTITY copyright "Copyright XML101.">

XML example:
<author>&writer;&copyright;</author>
```

External Entity Declaration

```
Syntax:
<!ENTITY entity-name SYSTEM "URI/URL">
```

```
DTD Example:
<!ENTITY writer    SYSTEM "http://www.xml101.com/entities/entities.xml">
<!ENTITY copyright SYSTEM "http://www.xml101.com/entities/entities.dtd">

XML example:
<author>&writer;&copyright;</author>
```

Q.) How to associate a DTD with xml?

- We need to specify DOCTYPE declaration after XML declaration, in order to associate a DTD with the XML document.
- A DTD can be declared
  - Inside an XML document
  - External reference.

Internal DTD Declaration

If the DTD is declared inside the XML file, it should be wrapped in a DOCTYPE definition with the following syntax:

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

Example 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>
```

The DTD above is interpreted like this:

- **!DOCTYPE note** defines that the root element of this document is note
- **!ELEMENT note** defines that the note element contains four 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"

External DTD Declaration

If the DTD is declared in an external file, it should be wrapped in a DOCTYPE definition with the following syntax:

```
<!DOCTYPE root-element SYSTEM "filename and location">
```

This is the same XML document as above, but with 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 this 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)>
```

Note: Internal DTD's are seldom used. Reusability of DTD's across XML documents is not possible with internal DTD.

Q.) Develop a DTD file for the following XML file?

Message.xml

```
<message>
    <to>ram</to>
    <from>sam</from>
    <body>hi, how are you</body>
</message>
```

message.dtd

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

Q.) Develop an external DTD for the following XML document?

Student.xml

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

```
<!DOCTYPE students SYSTEM "student.dtd" >
<students>
        <student regno="nit001" >
                <name>david</name>
                <course>XML</course>
                <fees-status>NotYet</fees-status>
                <working>yes</working>
        </student>
        <student regno="nit002">
                <name>clair</name>
                <course>JAVA</course>
                <fees-status>paid</fees-status>
        </student>
</students>
```

student.dtd

```
<!ELEMENT students(student+) >
<!ELEMENT student(name, course, fees-status, working?) >
<!ELEMENT name (#PCDATA) >
<!ELEMENT course (PCDATA) >
<!ELEMENT  fees-status (PCDATA) >
<!ELEMENT working (#PCDATA) >
<!ATTLIST student ID #REQUIRED >
```

items.xml

```
<?xml version="1.0" ?>
<!DOCTYPE items SYSTEM "items.dtd">

<items>
        <item code="1001" >
                <price>2500.0</price>
        </item>

        <item code="1002">
                <price>150</price>
        </item>
</items>
```

items.dtd

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT items (item+) >
<!ELEMENT item (price) >
<!ELEMENT price (#PCDATA) >
<!ATTLIST item code ID #REQUIRED >
```

bookstore.dtd

```
<!ELEMENT bookstore(name, topic+)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT topic(topic-name, book+)>
```

```
<!ELEMENT topic-name (#PCDATA)>
<!ELEMENT book(title, author) >
<!ELEMENT title (#PCDATA)>
<!ELEMENT author (#PCDATA)>
<!ATTLIST  book isbn ID #REQUIRED >
```

bookstore.xml

```
<?xml version="1.0" ?>
<!DOCTYPE bookstore SYSTEM "bookstore.dtd">
< bookstore >
        <name>HydBookStore</name>
        <topic>
                <topic-name>XML</topic-name>
                <book isbn="b12345" color="red"
                        pages="150"  publication="sekhar"  >
                        <title>Mike's Guide to DTD's and XML Schemas</title>
                        <author>Mike Jervis</author>
                        <price>68.0</price>
                </book>
                <book isbn="b12346">
                        <title>XSD in 24 hours</title>
                        <author>David</author>
                </book>
        </topic>
</ bookstore >
```

Q.) What is a prolog in an XML file or in an XML document?

- An XML document is divided into two parts.
  - Prolog
  - Body

- Root element is nothing but the body of the XML document.
- Prolog is that part of the XML document which comes before the root element.
- Generally prolog contains XML declaration and DOCTYPE declaration.

Q.) What do you know about SYSTEM keyword in DOCTYPE declaration?

- SYSTEM keyword indicates that the specified DTD is a private one. Almost always we use this (SYSTEM) keyword only in DOCTYPE declaration.
- Another keyword i.e. an alternative for SYSTEM is PUBLIC.

Example:

```
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN"
  "http://java.sun.com/j2ee/dtds/web-app_2_2.dtd">
```

+ = Standardized body

- = Not Standardized body

W3C = Organization Name

Hibernate = Name of the DTD

Version 1.0 = Version name

EN = Language

Q.) What are the limitations of DTD?

1. Weaker data types.
2. Constraints can't be effectively specified.
3. Doesn't follow XML syntax, and hence not productive.

XML schemas

Q.) What is an XSD?

- XSD Stands **for XML Schema Definition**
- An XML schema is used to define the structure of an XML document.
- Schemas are successors to DTDs.
- Schemas are another approach used by the XML parser to validate an XML document.
- It is a text file with **.xsd** extension.

Q.) What XSD defines?

1. Defines **elements** that can appear in a document
2. Defines **attributes** that can appear in a document
3. Defines the number of **child elements** and their sequence
4. Defines whether an element is **empty** or can include text
5. Defines **data types** for elements and attributes
6. Defines **default and fixed values** for elements and attributes

Q.) Compare and contrast DTDs and XSDs?

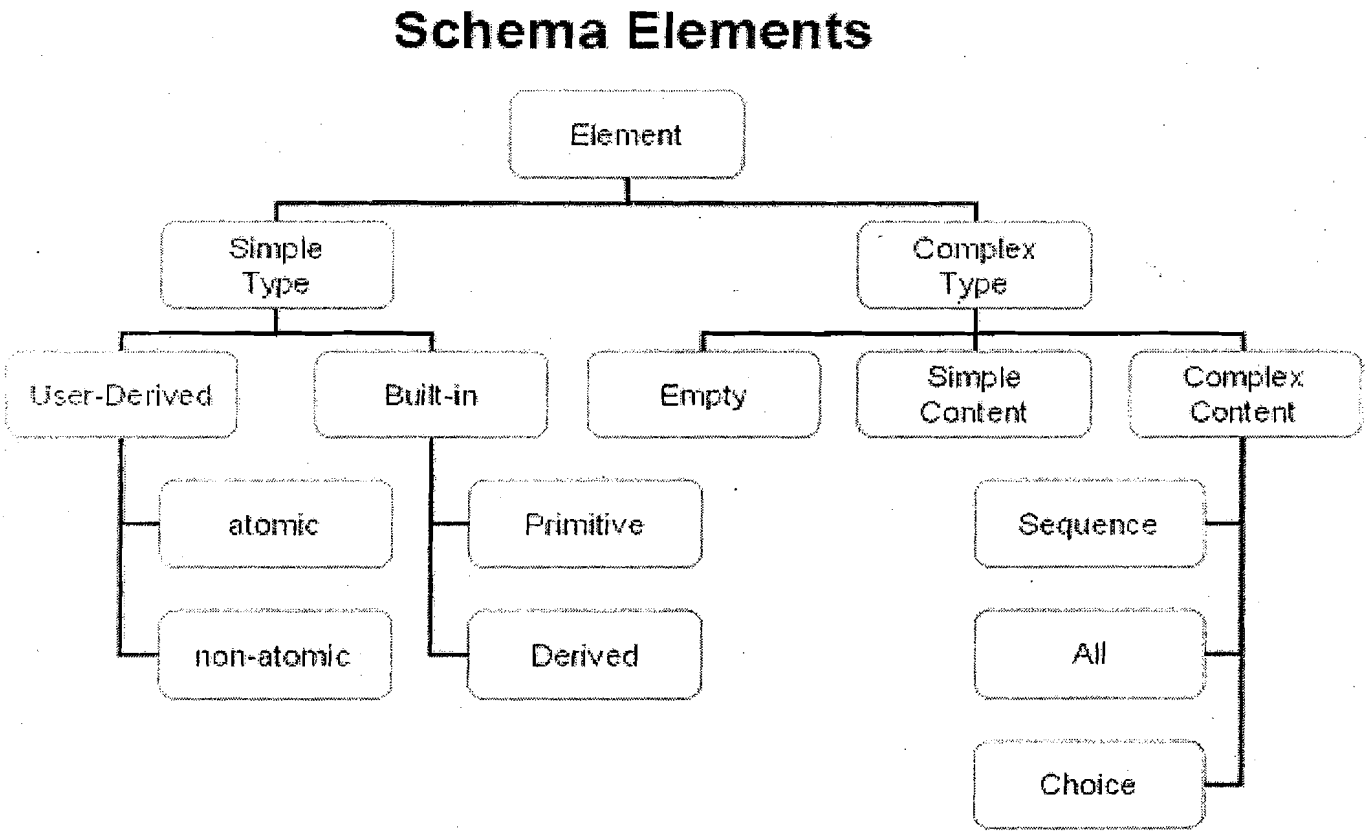
DTDs	XSDs

1. Used for legacy XML documents	1. Used for current XML documents
2. Follows EBNF(Extensible Backup Norm Form) syntax.	2. Follows XML syntax.
3. Limited data types. So not type safe.	3. Rich in data types, So it is type safe.
4. Cardinality control over the elements is limited	4.Provides much more cardinality constraints
5. Cannot define our own data types	5. Can define our won data types.
6. Doesn't support namespaces	6. Supports namespaces

Data Types

One of the **greatest strength** of XML Schemas is the **support for data types**. An XSD comprises of two kinds of data types.

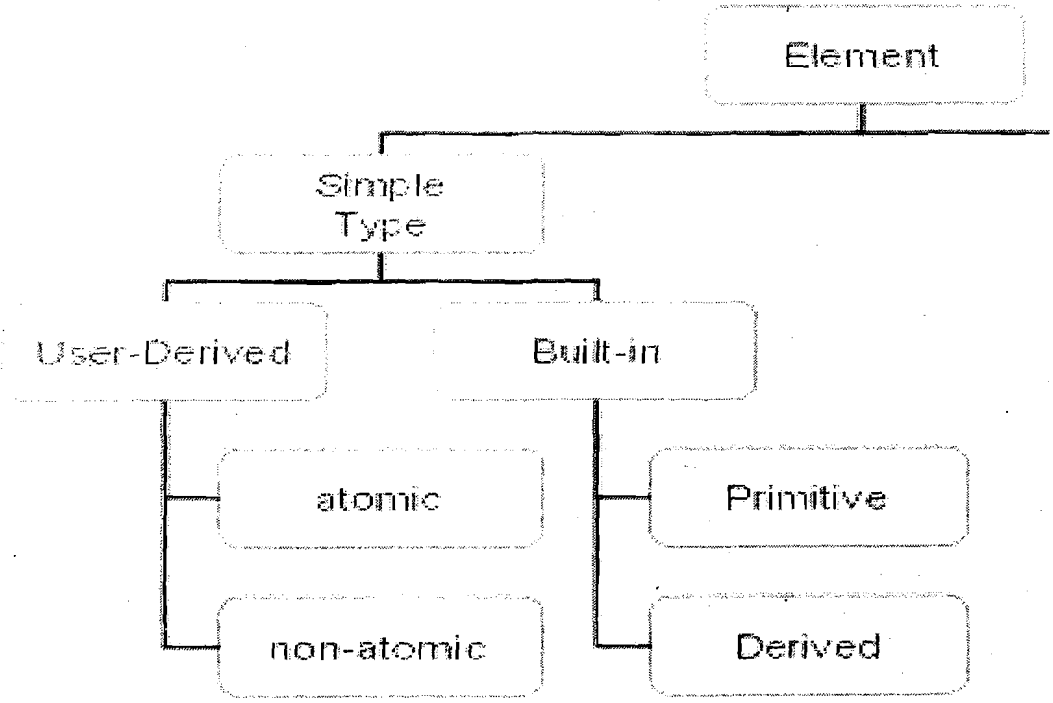
1. Simple Types
2. Complex Types



Simple Types

Q.) What is a Simple Element?

A simple element is an XML element that can contain only text. But is shouldn't contain any sub elements or attributes.



XML Schema specifies 44 built-in types,

19 of which are primitive.

String, Boolean, decimal, float, double, duration, dateTime, time, date, gYearMonth, gYear, gMonthDay, gDay, gMonth, hexBinary, base64Binary, anyURI, QName, NOTATION

25 of which Built-in Derived Data Types

normalizedString, token, language, NMTOKEN, NMTOKENS, Name, NCName, ID, IDREF, IDREFS, ENTITY, ENTITIES, integer, nonPositiveInteger, negativeInteger, long, int, short, byte, nonNegativeInteger, unsignedLong, unsignedInt, unsignedShort, unsignedByte, positiveInteger.

The most common types are:

1. xs:string
2. xs:decimal
3. xs:integer
4. xs:boolean
5. xs:date
6. xs:time

Defining a Simple Element

Syntax:

```
<xs:element name="element-name" type="element-type"/>
```

where **element-name** is the name of the element and **element-type** is the data type of the element.

Q.) What is the XSD equivalent for the following XML element?

XML elements:

```
<lastname>Refsnes</lastname>
<age>36</age>
<dateborn>1970-03-27</dateborn>
```

XSD definition:

```
<xs:element name="lastname" type="xs:string"/>
<xs:element name="age" type="xs:integer"/>
<xs:element name="dateborn" type="xs:date"/>
```

Q.) How to give Default and Fixed Values to Simple Elements ?

"color" element default value is "red":

```
<xs:element name="color" type="xs:string" default="red"/>
```

"color" element fixed value is "yellow":

```
<xs:element name="color" type="xs:string" fixed="yellow"/>
```

Q.) How to Define an Attribute?

Syntax:

```
<xs:attribute name="attribute-name" type="attribute-type"/>
```

Where **attribute-name** is the name of the attribute and **attribute-type** specifies the data type of the attribute.

Q.) What is the XSD equivalent for the following XML element?

ML element:

```
<lastname lang="EN">Smith</lastname>
```

XSD definition:

```
<xs:attribute name="lang" type="xs:string"/>
<xs:element name="lastname" type="xs:string" />
```

Q.) How to define fixed and default values to attributes?

Fixed Values for Attributes

```
<xs:attribute name="lang" type="xs:string" fixed="EN"/>
```

Default Values for Attributes

```
<xs:attribute name="lang" type="xs:string" default="EN"/>
```

Q.)How to specify an attribute as optional/ required?

Optional Attributes

```
<xs:attribute name="lang" type="xs:string" />
```

NOTE: By default an attribute is optional attribute

Required Attributes

```
<xs:attribute name="lang" type="xs:string" use="required"/>
```

XSD Restrictions/Facets

Restrictions are used to define acceptable values for XML elements or attributes. Restrictions on XML elements are called facets.

Restrictions on Values

The following example defines an element called "age" with a restriction. The value of age cannot be lower than 18 or greater than 30:

```
<xs:element name="age">
  <xs:simpleType>
    <xs:restriction base="xs:integer">
      <xs:minInclusive value="18"/>
      <xs:maxInclusive value="30"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

Restrictions on a Set of Values

To limit the content of an XML element to a set of acceptable values, we would use the enumeration constraint.

The example below defines an element called "car" with a restriction. The only acceptable values are: Audi, Golf, BMW:

```
<xs:element name="car">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:enumeration value="Audi"/>
      <xs:enumeration value="Golf"/>
      <xs:enumeration value="BMW"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

The example above could also have been written like this:

```
<xs:element name="car" type="carType"/>

<xs:simpleType name="carType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="Audi"/>
    <xs:enumeration value="Golf"/>
    <xs:enumeration value="BMW"/>
  </xs:restriction>
</xs:simpleType>
```

Note: In this case the type "carType" can be used by other elements because it is not a part of the "car" element.

Restrictions on a Series of Values

pattern constraint can be used to limit the content of an XML element to define a series of numbers or letters

Example#1 Acceptable value is ONE of the LOWERCASE letters from a to z:

```
<xs:element name="letter">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="[a-z]"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

Example#2 Acceptable value is THREE of the UPPERCASE letters from a to z:

```
<xs:element name="initials">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="[A-Z][A-Z][A-Z]"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

Example#3 Acceptable value is THREE of the LOWERCASE OR UPPERCASE letters from a to z:

```
<xs:element name="initials">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="[a-zA-Z][a-zA-Z][a-zA-Z]"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

Example#4 Acceptable value is ONE of the following letters: x, y, OR z:

```
<xs:element name="choice">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="[xyz]"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

Example#5 Acceptable value is FIVE digits in a sequence, and each digit must be in a range from 0 to 9:

```
<xs:element name="prodid">
  <xs:simpleType>
    <xs:restriction base="xs:integer">
```

```
      <xs:pattern value="[0-9][0-9][0-9][0-9][0-9]"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

Example#6 Acceptable value is zero or more occurrences of lowercase letters from a to z:

```
<xs:element name="letter">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="([a-z])*"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

Example#7 Acceptable value is one or more pairs of letters, each pair consisting of a lower case letter followed by an upper case letter. For example, "sToP" will be validated by this pattern, but not "Stop" or "STOP" or "stop":

```
<xs:element name="letter">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="([a-z][A-Z])+"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

Example#8 Acceptable value is male OR female:

```
<xs:element name="gender">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="male|female"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

Example#9 There must be exactly eight characters in a row and those characters must be lowercase or uppercase letters from a to z, or a number from 0 to 9:

```
<xs:element name="password">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="[a-zA-Z0-9]{8}"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

Example# 10 : Element to restrict the Password simple type to consist of between six and twelve characters, which can only be lowercase and uppercase letters and underscores

```
<xs:simpleType name="Password">
  <xs:restriction base="xs:string">
    <xs:pattern value="[A-Za-z_]{6,12}"/>
  </xs:restriction>
</xs:simpleType>
```

Number of Digits

Using `totalDigits` and `fractionDigits`, we can further specify that the `Salary` type should consist of seven digits, two of which come after the decimal point. Both `totalDigits` and `fractionDigits` are maximums. That is, if `totalDigits` is specified as 5 and `fractionDigits` is specified as 2, a valid number could have no more than five digits total and no more than two digits after the decimal point.

```
<xs:simpleType name="Salary">
 <xs:restriction base="xs:decimal">
    <xs:minInclusive value="10000"/>
    <xs:maxInclusive value="90000"/>
    <xs:fractionDigits value="2"/>
    <xs:totalDigits value="7"/>
 </xs:restriction>
</xs:simpleType>
```

Restrictions on Whitespace Characters

To specify how whitespace characters should be handled, we would use the **whiteSpace** constraint.

This example defines an element called "address" with a restriction. The **whiteSpace** constraint is set to "**preserve**", which means that the XML processor WILL NOT remove any white space characters:

```
<xs:element name="address">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:whiteSpace value="preserve"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

- `replace` - all tabs, line feeds, and carriage returns are replaced by single spaces.
- `collapse` - all tabs, line feeds, and carriage returns are replaced by single spaces and then all groups of single spaces are replaced with one single space. All leading and trailing spaces are then removed (i.e, trimmed).

Restrictions on Length

To limit the length of a value in an element, we would use the **length, maxLength, and minLength** constraints.

This example defines an element called "password" with a restriction. The value must be exactly eight characters:

```
<xs:element name="password">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:length value="8"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

This example defines another element called "password" with a restriction. The value must be minimum five characters and maximum eight characters:

```
<xs:element name="password">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:minLength value="5"/>
      <xs:maxLength value="8"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

Restrictions for Datatypes

Constraint	Description
Enumeration	Defines a list of acceptable values
fractionDigits	Specifies the maximum number of decimal places allowed. Must be equal to or greater than zero
Length	Specifies the exact number of characters or list items allowed. Must be equal to or greater than zero
maxExclusive	Specifies the upper bounds for numeric values (the value must be less than this value)
maxInclusive	Specifies the upper bounds for numeric values (the value must be less than or equal to this value)
maxLength	Specifies the maximum number of characters or list items allowed. Must be equal to or greater than zero
minExclusive	Specifies the lower bounds for numeric values (the value must be greater than this value)
minInclusive	Specifies the lower bounds for numeric values (the value must be greater than or equal to this value)

minLength	Specifies the minimum number of characters or list items allowed. Must be equal to or greater than zero
pattern	Defines the exact sequence of characters that are acceptable
totalDigits	Specifies the exact number of digits allowed. Must be greater than zero
whiteSpace	Specifies how white space (line feeds, tabs, spaces, and carriage returns) is handled

Nonatomic Types

All of XML Schema's built-in types are atomic, meaning that they cannot be broken down into meaningful bits. XML Schema provides for two nonatomic types: lists and unions.

Lists

List types are sequences of atomic types separated by whitespace; you can have a list of integers or a list of dates. Lists should not be confused with enumerations. Enumerations provide optional values for an element. Lists represent a single value within an element.

```
<xs:simpleType name="DateList">
 <xs:list itemType="xs:date"/>
</xs:simpleType>

<xs:element name="Employee">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="Salary" type="xs:double"/>
   <xs:element name="Title" type="xs:string"/>
   <xs:element name="VacationDays" type="DateList"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

Example: XML

```
<Employee>
 <Salary>44000</Salary>
 <Title>Salesperson</Title>
 <VacationDays>2006-8-13 2006-08-14 2006-08-15</VacationDays>
</Employee>
```

Unions

Union types are groupings of types, essentially allowing for the value of an element to be of more than one type. In the example below, two atomic simple types are derived: RunningRace and Gymnastics. A third simple type, Event, is then derived as a union of the previous two. The Event element is of the Event type, which means that it can either be of the RunningRace or the Gymnastics type.

```
<xs:simpleType name="RunningRace">
        <xs:restriction base="xs:string">
                <xs:enumeration value="100 meters"/>
                <xs:enumeration value="10 kilometers"/>
                <xs:enumeration value="440 yards"/>
                <xs:enumeration value="10 miles"/>
                <xs:enumeration value="Marathon"/>
        </xs:restriction>
</xs:simpleType>
<xs:simpleType name="Gymnastics">
        <xs:restriction base="xs:string">
                <xs:enumeration value="Vault"/>
                <xs:enumeration value="Floor"/>
                <xs:enumeration value="Rings"/>
                <xs:enumeration value="Beam"/>
                <xs:enumeration value="Uneven Bars"/>
        </xs:restriction>
</xs:simpleType>
<xs:simpleType name="Event">
        <xs:union memberTypes="RunningRace Gymnastics"/>
</xs:simpleType>

<xs:element name="Program">
        <xs:complexType>
                <xs:sequence>
                        <xs:element name="Event" type="Event"/>
                </xs:sequence>
        </xs:complexType>
</xs:element>
```

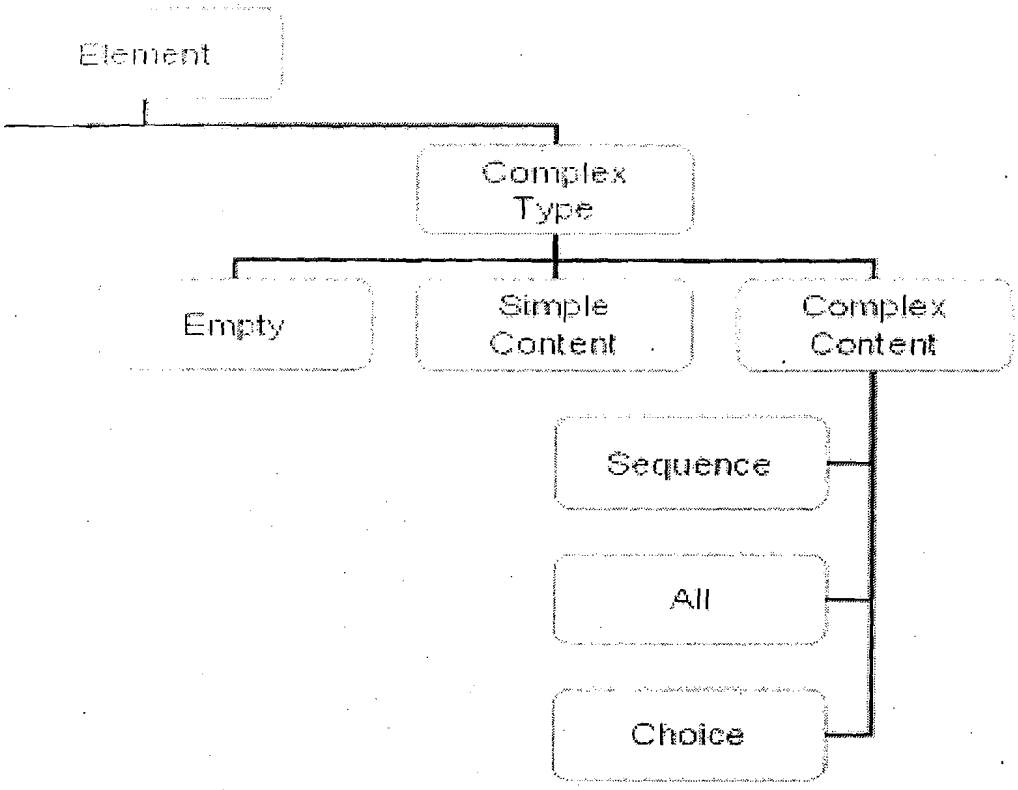
Example: XML

```
<Program >
 <Event>100-meters  Vault</Event>
</Program>
```

Complex Types

Q) What is a Complex Element?

A complex element is an XML element that contains other elements and/or attributes.



As the diagram shows, a complex-type element can be empty, contain simple content such as a string, or can contain complex content such as a sequence of elements.

Example: Define a full name of a person

```
<xs:complexType name="fullnameType" >
    <xs:sequence>
      <xs:element name="firstname" type="xs:string"/>
      <xs:element name="lastname" type="xs:string"/>
    </xs:sequence>
</xs:complexType>
```

Note: It is not necessary to explicitly declare that a simple-type element is a simple type, it is necessary to specify that a complex-type element is a complex type. This is done with the `xs:complexType` element

Q) How to Define a Complex Element

Complex types are defined using **<complextype>** element in two approaches

Approach #1: The **"person"** element can be declared directly by naming the element

```
<xs:element name="person">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="name" type="xs:string"/>
      <xs:element name="age" type="xs:integer"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

NOTE: <sequence> indicator. This means that the child elements must appear in the same order as they are declared.

Example:

```
<person>
  <name>Sekhar</name>
  <age>26</age>
</person>
```

Approach #2: The "**person**" element can have a **type** attribute that refers to the name of the complex type

```
<xs:element name="person" type="personType"/>

  <xs:complexType name="personType">
    <xs:sequence>
      <xs:element name="name" type="xs:string"/>
      <xs:element name="age" type="xs:integer"/>
    </xs:sequence>
  </xs:complexType>
```

NOTE: Advantage of this approach is several elements can refer the same complex type.

Example

```
<xs:element name="employee" type="personType"/>
<xs:element name="student" type="personType"/>
<xs:element name="member" type="personType"/>
```

There are four kinds of complex elements:

1. Elements with attributes
2. Elements with sub elements
3. Elements with attributes and text
4. Elements with sub elements and text

Note: Each of these elements may contain attributes as well!

1.Elements with attributes

An empty complex element cannot have contents, only attributes.

```
<xs:element name="product">
  <xs:complexType>
    <xs:attribute name="prodid" type="xs:positiveInteger"/>
  </xs:complexType>
</xs:element>
```

It is possible to declare the "product" element like below

```
<xs:element name="product">
  <xs:complexType>
    <xs:complexContent>
      <xs:restriction base="xs:integer">
        <xs:attribute name="prodid" type="xs:positiveInteger"/>
      </xs:restriction>
    </xs:complexContent>
  </xs:complexType>
</xs:element>
```

In the example above, The **complexContent** element signals that we intend to restrict or extend the content model of a complex type, and the restriction of integer declares one attribute but does not introduce any element content.

Example

```
<product prodid="1345" />
```

2.Elements with sub elements

You can define the "person" element in a schema, like this:

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

```
      <xs:element name="lastname" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Example

```
<person>
  <firstname>John</firstname>
  <lastname>Smith</lastname>
</person>
```

3.Elements with attributes and text

This type contains only simple content (text and attributes), therefore we add a **simpleContent** element around the content. When using **simple content**, you must define an **extension** OR a **restriction** within the **simpleContent** element

NOTE: Use the extension/restriction element to expand or to limit the base simple type for the element.

```
<xs:element name="somename">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="basetype">
        ....
        ....
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>

OR

<xs:element name="somename">
  <xs:complexType>
    <xs:simpleContent>
      <xs:restriction base="basetype">
        ....
        ....
      </xs:restriction>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
```

```
<xs:element name="shoesize">
  <xs:complexType>
    <xs:simpleContent>
```

```
      <xs:extension base="xs:integer">
        <xs:attribute name="country" type="xs:string" />
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
```

Example: <shoesize country="france">35</shoesize>

4.Elements with sub elements and text

XML element, "letter", that contains both text and other elements:

```
<xs:element name="letter">
  <xs:complexType mixed="true">
    <xs:sequence>
      <xs:element name="name" type="xs:string"/>
      <xs:element name="orderid" type="xs:positiveInteger"/>
      <xs:element name="shipdate" type="xs:date"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Example:

```
<letter>
  Dear Mr.<name>John Smith</name>.
  Your order <orderid>1032</orderid>
  will be shipped on <shipdate>2001-07-13</shipdate>.
</letter>
```

Empty Elements

An empty element is an element that contains no content, but it may have attributes. The HomePage element in the instance document below is an empty element.

```
<HomePage URL="http://www.marktwain.com"/>
```

Indicators

We can control HOW elements are to be used in documents with indicators.

There are seven indicators:

Order indicators:

1. All
2. Choice
3. Sequence

Occurrence indicators:

1. maxOccurs
2. minOccurs

Group indicators:

1. Group name
2. attributeGroup name

Order Indicators

Order indicators are used to define the order of the elements.

All Indicator

The <all> indicator specifies that the child elements can appear in any order, and that each child element must occur only once:

```
<xs:element name="person">
  <xs:complexType>
    <xs:all>
      <xs:element name="firstname" type="xs:string"/>
      <xs:element name="lastname" type="xs:string"/>
    </xs:all>
  </xs:complexType>
</xs:element>
```

Note: When using the <all> indicator you can set the <minOccurs> indicator to 0 or 1 and the <maxOccurs> indicator can only be set to 1 (the <minOccurs> and <maxOccurs> are described later).

Choice Indicator

The <choice> indicator specifies that either one child element or another can occur:

```
<xs:element name="servlet">
 <xs:complexType>
    <xs:choice>
      <xs:element name="servlet-class" type=" xs:string"/>
      <xs:element name="jsp-file" type="xs:string"/>
    </xs:choice>
  </xs:complexType>
</xs:element>
```

Sequence Indicator

The <sequence> indicator specifies that the child elements must appear in a specific order:

```
<xs:element name="servlet">
   <xs:complexType>
    <xs:sequence>
      <xs:element name="servlet-name" type="xs:string"/>
      <xs:element name="servlet-class" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Occurrence Indicators

Occurrence indicators are used to define how often an element can occur.

Note: For all "Order" and "Group" indicators (any, all, choice, sequence, group name, and group reference) the default value for maxOccurs and minOccurs is 1.

maxOccurs Indicator

The <maxOccurs> indicator specifies the maximum number of times an element can occur:

```
<xs:element name="person">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="full_name" type="xs:string"/>
      <xs:element name="child_name" type="xs;string" maxOccurs="10"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

The example above indicates that the "child_name" element can occur a minimum of one time (the default value for minOccurs is 1) and a maximum of ten times in the "person" element.

minOccurs Indicator

The <minOccurs> indicator specifies the minimum number of times an element can occur:

```
<xs:element name="person">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="full_name" type="xs:string"/>
      <xs:element name="child_name" type="xs:string"
      maxOccurs="10" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

The example above indicates that the "child_name" element can occur a minimum of zero times and a maximum of ten times in the "person" element.

NOTE: To allow an element to appear an unlimited number of times, use the **maxOccurs="unbounded"** statement.

NOTE: **minOccurs** and **maxOccurs** can also be applied to model groups (e.g, xs:sequence) to control the number of times a model group can be repeated.

Group Indicators

Group indicators are used to define related sets of elements.

Element Groups

Element groups are defined with the **group** declaration, like this:

```
<xs:group name="groupname">
...
</xs:group>
```

You must define an all, choice, or sequence element inside the group declaration. The following example defines a group named "**persongroup**", that defines a group of elements that must occur in an exact sequence:

```
<xs:group name="persongroup">
  <xs:sequence>
    <xs:element name="firstname" type="xs:string"/>
    <xs:element name="lastname" type="xs:string"/>
    <xs:element name="birthday" type="xs:date"/>
  </xs:sequence>
</xs:group>
```

After you have defined a group, you can reference it in another definition, like this:

```
<xs:group name="persongroup">
  <xs:sequence>
    <xs:element name="firstname" type="xs:string"/>
    <xs:element name="lastname" type="xs:string"/>
    <xs:element name="birthday" type="xs:date"/>
  </xs:sequence>
</xs:group>

<xs:element name="person" type="personinfo"/>

<xs:complexType name="personinfo">
  <xs:sequence>
    <xs:group ref="persongroup"/>
    <xs:element name="country" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
```

Attribute Groups

Attribute groups are defined with the attributeGroup declaration, like this:

```
<xs:attributeGroup name="groupname">
...
</xs:attributeGroup>
```

The following example defines an attribute group named "personattrgroup":

```
<xs:attributeGroup name="personattrgroup">
  <xs:attribute name="firstname" type="xs:string"/>
  <xs:attribute name="lastname" type="xs:string"/>
  <xs:attribute name="birthday" type="xs:date"/>
</xs:attributeGroup>
```

After you have defined an attribute group, you can reference it in another definition, like this:

```
<xs:attributeGroup name="personattrgroup">
  <xs:attribute name="firstname" type="xs:string"/>
  <xs:attribute name="lastname" type="xs:string"/>
  <xs:attribute name="birthday" type="xs:date"/>
</xs:attributeGroup>

<xs:element name="person">
  <xs:complexType>
    <xs:attributeGroup ref="personattrgroup"/>
  </xs:complexType>
</xs:element>
```

<any> Element

The <any> element enables us to extend the XML document with elements not specified by the schema.

person.xsd

```
<xs:schema>
<xs:element name="person">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="firstname" type="xs:string"/>
      <xs:element name="lastname" type="xs:string"/>
      <xs:any minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
</xs:schema>
```

children.xsd

```
<xs:schema >

<xs:element name="children">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="childname" type="xs:string"
      maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

</xs:schema>
```

family.xml

This XML file uses components from two different schemas; "perosn.xsd" and "children.xsd":

```
<persons >

<person>
  <firstname>Hege</firstname>
  <lastname>Refsnes</lastname>
  <children>
    <childname>Cecilie</childname>
  </children>
</person>

<person>
  <firstname>Stale</firstname>
  <lastname>Refsnes</lastname>
</person>

</persons>
```

NOTE: The <any> and <anyAttribute> elements are used to make EXTENSIBLE documents!

<anyAttribute> Element

The <anyAttribute> element enables us to extend the XML document with attributes not specified by the schema.

person.xsd

```
<xs:schema>
<xs:element name="person">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="firstname" type="xs:string"/>
      <xs:element name="lastname" type="xs:string"/>
    </xs:sequence>
    <xs:anyAttribute/>
```

```
  </xs:complexType>
</xs:element>
</xs:schema>
```

ttribute.xsd

```
<xs:schema >

<xs:attribute name="gender">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="male|female"/>
    </xs:restriction>
  </xs:simpleType>
</xs:attribute>

</xs:schema>
```

family.xml

This XML file uses components from two different schemas; "perosn.xsd" and "children.xsd":

```
<persons>

<person gender="female">
  <firstname>Hege</firstname>
  <lastname>Refsnes</lastname>
</person>

<person gender="male">
  <firstname>Stale</firstname>
  <lastname>Refsnes</lastname>
</person>

</persons>
```

Annotating XML Schemas

Overview

One of the nice features of XML Schema is that comments about the schema itself can be made within built-in XML elements. This makes it possible to run a transformation against a schema to builds documentation in HTML or some other human-readable format.

Annotating a Schema

The `xs:annotation` element is used to document a schema. It can take two elements: `xs:documentation` and `xs:appInfo`, which are used to provide human-readable and machine-readable notes, respectively.
The `xs:annotation` element can go at the beginning of most schema constructions, including `xs:schema`, `xs:element`, `xs:attribute`, `xs:simpleType`, `xs:complexType`, `xs:group`, and `xs:attributeGroup`.
Both the `xs:documentation` and `xs:appInfo` elements can contain any content, including undeclared elements and attributes. This allows the schema author to insert elements (e.g, HTML elements) to structure or format the documentation.

XML Namespaces

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>
```

If these XML fragments were added together, there would be a name conflict. Both contain a <table> element, but the elements have different content and meaning.

An XML parser will not know how to handle these differences.

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 so-called **namespace** for the prefix must be defined.

The namespace is defined by the **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.sekharit.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 <table> tag give the h: and f: prefixes 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 be declared in the elements where they are used or in the XML root element:

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

Note: The namespace URI is not used by the parser to look up information.

The purpose is to give the namespace a unique name. However, often companies 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>
```

This XML carries information about a piece of furniture:

```
<table xmlns="http://www.sekharit.com/furniture">
  <name>African Coffee Table</name>
  <width>80</width>
  <length>120</length>
</table>
```

) Develop a XSD file for the following xml file?

note.xml

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

note.xsd

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.sekharit.com"
xmlns="http://www.sekharit.com"
elementFormDefault="qualified">

<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 <schema> element is the root element of every XML Schema.

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

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

The <schema> element may contain some attributes. A schema declaration often looks something like this:

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

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://sekharit.com"
xmlns=" http://sekharit.com"
elementFormDefault="qualified">
...
...
</xs:schema>
```

Attribute name	Description	Exmaple
xmlns:xs	Elements and data types come from the "http://www.w3.org/2001/XMLSchema" namespace.	xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace	User defined elements and data types in our schema belongs to our namespace such as "http://sekharit.com"	targetNamespace="http://sekharit.com"
xmlns	Indicates default namespace	targetNamespace="http://sekharit.com"
elementFormDefault	Elements must be namespace qualified	elementFormDefault="qualified"

Q) How to Refer a Schema in an XML Document?

Using the **schemaLocation** attribute. This attribute has two values, separated by a space. The first value is the **namespace** to use. The second value is the **location of the XML** schema to use for that namespace:

schemaLocation="namespace location-of-XML"

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

<note xmlns="http://www.sekharit.com"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.sekharit.com note.xsd">

<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>
```

```
xmlns=http://www.sekharit.com
```

specifies the default namespace declaration. This declaration tells the schema-validator that all the elements used in this XML document are declared in the "http://www.sekharit.com" namespace.

Once you have the XML Schema Instance namespace available:

```
xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
```

you can use the schemaLocation attribute. This attribute has two values, separated by a space. The first value is the namespace to use. The second value is the location of the XML schema to use for that namespace:

```
xsi:schemaLocation=http://www.sekharit.com note.xsd
```

The XMLSchema-instance Namespace

The XMLSchema-instance namespace contains only four attributes. Here is a simplified version of the schema.

Code Sample: ***Namespaces/Demos/XMLSchema-instance.xsd***

```
<?xml version='1.0'?>
<xs:schema targetNamespace="http://www.w3.org/2001/XMLSchema-instance"
        xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:attribute name="nil"/>
 <xs:attribute name="type"/>
 <xs:attribute name="schemaLocation"/>
 <xs:attribute name="noNamespaceSchemaLocation"/>
</xs:schema>
```

An XSD Example

This will demonstrate how to write an XML Schema. You will also learn that a schema can be written in different ways.

An XML Document

Let's have a look at this XML document called "shiporder.xml":

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<shiporder orderid="889923"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="shiporder.xsd">
  <orderperson>John Smith</orderperson>
  <shipto>
    <name>Ola Nordmann</name>
    <address>Langgt 23</address>
    <city>4000 Stavanger</city>
    <country>Norway</country>
  </shipto>
  <item>
    <title>Empire Burlesque</title>
    <note>Special Edition</note>
    <quantity>1</quantity>
    <price>10.90</price>
  </item>
  <item>
    <title>Hide your heart</title>
    <quantity>1</quantity>
```

```
    <price>9.90</price>
  </item>
</shiporder>
```

The XML document above consists of a root element, "shiporder", that contains a required attribute called "orderid". The "shiporder" element contains three different child elements: "orderperson", "shipto" and "item". The "item" element appears twice, and it contains a "title", an optional "note" element, a "quantity", and a "price" element.

The line above: xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" tells the XML parser that this document should be validated against a schema. The line: xsi:noNamespaceSchemaLocation="shiporder.xsd" specifies WHERE the schema resides (here it is in the same folder as "shiporder.xml").

Create an XML Schema

Now we want to create a schema for the XML document above.

We start by opening a new file that we will call "shiporder.xsd". To create the schema we could simply follow the structure in the XML document and define each element as we find it. We will start with the standard XML declaration followed by the xs:schema element that defines a schema:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
...
</xs:schema>
```

In the schema above we use 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.

Next, we have to define the "shiporder" element. This element has an attribute and it contains other elements, therefore we consider it as a complex type. The child elements of the "shiporder" element is surrounded by a xs:sequence element that defines an ordered sequence of sub elements:

```
<xs:element name="shiporder">
  <xs:complexType>
    <xs:sequence>
      ...
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Then we have to define the "orderperson" element as a simple type (because it does not contain any attributes or other elements). The type (xs:string) is prefixed with the namespace prefix associated with XML Schema that indicates a predefined schema data type:

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

Next, we have to define two elements that are of the complex type: "shipto" and "item". We start by defining the "shipto" element:

```
<xs:element name="shipto">
```

```
  <xs:complexType>
    <xs:sequence>
      <xs:element name="name" type="xs:string"/>
      <xs:element name="address" type="xs:string"/>
      <xs:element name="city" type="xs:string"/>
      <xs:element name="country" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

With schemas we can define the number of possible occurrences for an element with the maxOccurs and minOccurs attributes. maxOccurs specifies the maximum number of occurrences for an element and minOccurs specifies the minimum number of occurrences for an element. The default value for both maxOccurs and minOccurs is 1!

Now we can define the "item" element. This element can appear multiple times inside a "shiporder" element. This is specified by setting the maxOccurs attribute of the "item" element to "unbounded" which means that there can be as many occurrences of the "item" element as the author wishes. Notice that the "note" element is optional. We have specified this by setting the minOccurs attribute to zero:

```
<xs:element name="item" maxOccurs="unbounded">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="title" type="xs:string"/>
      <xs:element name="note" type="xs:string" minOccurs="0"/>
      <xs:element name="quantity" type="xs:positiveInteger"/>
      <xs:element name="price" type="xs:decimal"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

We can now declare the attribute of the "shiporder" element. Since this is a required attribute we specify use="required".

Note: The attribute declarations must always come last:

```
<xs:attribute name="orderid" type="xs:string" use="required"/>
```

Here is the complete listing of the schema file called "shiporder.xsd":

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

<xs:element name="shiporder">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="orderperson" type="xs:string"/>
      <xs:element name="shipto">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="name" type="xs:string"/>
            <xs:element name="address" type="xs:string"/>
            <xs:element name="city" type="xs:string"/>
```

```
            <xs:element name="country" type="xs:string"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="item" maxOccurs="unbounded">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="title" type="xs:string"/>
            <xs:element name="note" type="xs:string" minOccurs="0"/>
            <xs:element name="quantity" type="xs:positiveInteger"/>
            <xs:element name="price" type="xs:decimal"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
    <xs:attribute name="orderid" type="xs:string" use="required"/>
  </xs:complexType>
</xs:element>

</xs:schema>
```

Divide the Schema

The previous design method is very simple, but can be difficult to read and maintain when documents are complex.

The next design method is based on defining all elements and attributes first, and then referring to them using the ref attribute.

Here is the new design of the schema file ("shiporder.xsd"):

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

<!-- definition of simple elements -->
<xs:element name="orderperson" type="xs:string"/>
<xs:element name="name" type="xs:string"/>
<xs:element name="address" type="xs:string"/>
<xs:element name="city" type="xs:string"/>
<xs:element name="country" type="xs:string"/>
<xs:element name="title" type="xs:string"/>
<xs:element name="note" type="xs:string"/>
<xs:element name="quantity" type="xs:positiveInteger"/>
<xs:element name="price" type="xs:decimal"/>

<!-- definition of attributes -->
<xs:attribute name="orderid" type="xs:string"/>

<!-- definition of complex elements -->
<xs:element name="shipto">
  <xs:complexType>
    <xs:sequence>
```

```
        <xs:element ref="name"/>
        <xs:element ref="address"/>
        <xs:element ref="city"/>
        <xs:element ref="country"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:element name="item">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="title"/>
        <xs:element ref="note" minOccurs="0"/>
        <xs:element ref="quantity"/>
        <xs:element ref="price"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:element name="shiporder">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="orderperson"/>
        <xs:element ref="shipto"/>
        <xs:element ref="item" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute ref="orderid" use="required"/>
    </xs:complexType>
  </xs:element>

  </xs:schema>
```

Using Named Types

The third design method defines classes or types, that enables us to reuse element definitions. This is done by naming the simpleTypes and complexTypes elements, and then point to them through the type attribute of the element.

Here is the third design of the schema file ("shiporder.xsd"):

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

<xs:simpleType name="stringtype">
  <xs:restriction base="xs:string"/>
</xs:simpleType>

<xs:simpleType name="inttype">
  <xs:restriction base="xs:positiveInteger"/>
</xs:simpleType>

<xs:simpleType name="dectype">
```

```
  <xs:restriction base="xs:decimal"/>
</xs:simpleType>

<xs:simpleType name="orderidtype">
  <xs:restriction base="xs:string">
    <xs:pattern value="[0-9]{6}"/>
  </xs:restriction>
</xs:simpleType>

<xs:complexType name="shiptotype">
  <xs:sequence>
    <xs:element name="name" type="stringtype"/>
    <xs:element name="address" type="stringtype"/>
    <xs:element name="city" type="stringtype"/>
    <xs:element name="country" type="stringtype"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="itemtype">
  <xs:sequence>
    <xs:element name="title" type="stringtype"/>
    <xs:element name="note" type="stringtype" minOccurs="0"/>
    <xs:element name="quantity" type="inttype"/>
    <xs:element name="price" type="dectype"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="shipordertype">
  <xs:sequence>
    <xs:element name="orderperson" type="stringtype"/>
    <xs:element name="shipto" type="shiptotype"/>
    <xs:element name="item" maxOccurs="unbounded" type="itemtype"/>
  </xs:sequence>
  <xs:attribute name="orderid" type="orderidtype" use="required"/>
</xs:complexType>

<xs:element name="shiporder" type="shipordertype"/>

</xs:schema>
```

The restriction element indicates that the datatype is derived from a W3C XML Schema namespace datatype. So, the following fragment means that the value of the element or attribute must be a string value:

```
<xs:restriction base="xs:string">
```

The restriction element is more often used to apply restrictions to elements. Look at the following lines from the schema above:

```
<xs:simpleType name="orderidtype">
  <xs:restriction base="xs:string">
    <xs:pattern value="[0-9]{6}"/>
  </xs:restriction>
```

```
</xs:simpleType>
```

This indicates that the value of the element or attribute must be a string, it must be exactly six characters in a row, and those characters must be a number from 0 to 9.

XML application development

Rs: 14/2

Q.) What is an XML application?

- A program that works with XML document is known as XML application.
- To work with XML documents we need API

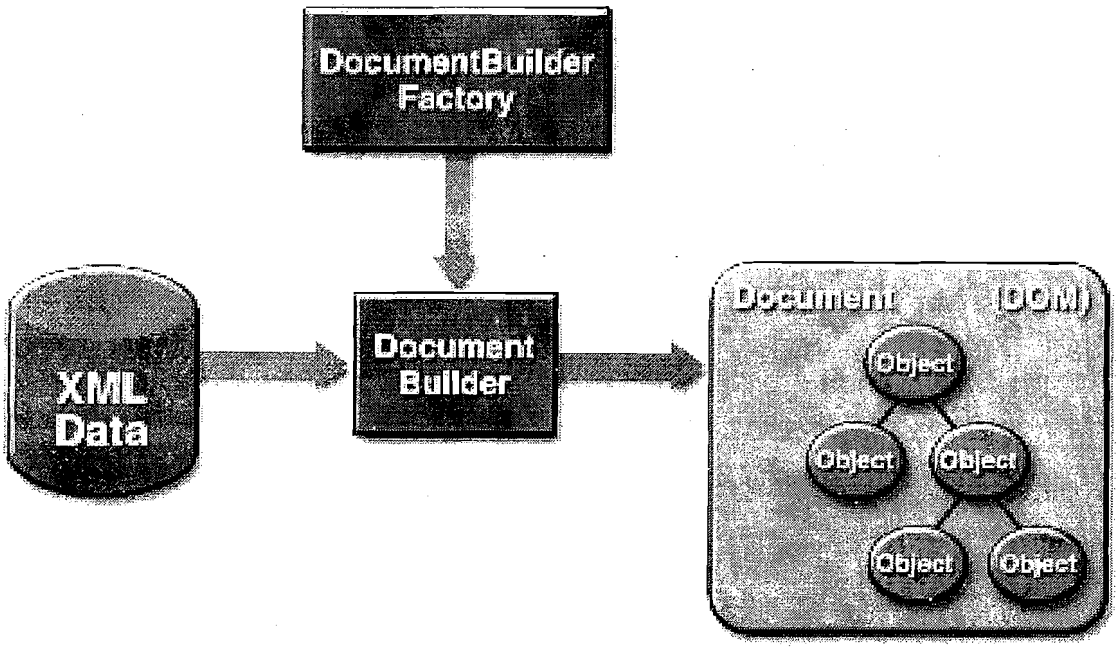
Q.) What is JAXP?

- JAXP stands for Java API for XML Processing
- JAXP is a specification from W3C.
- JAXP is an API from Sun
- Using JAXP API we can process XML documents in two methodologies
  - DOM
  - SAX
- JAXP API is implemented by multiple vendors(xcerses.jar, crimson.jar, oraclev2.jar ... etc)
- There are some parsing APIs which are proprietary (xml4j.jar, xmlbeans.jar ...etc)
- From Jdk 1.5 onwards, jdk itself has **xcerses.jar**(Apache) implementation.

DOM

- DOM stands for Document Object Model
- It is hierarchical model(tree based model)
- At a time total document will be loaded into the memory as a tree structure.
- In DOM each element is called Node
- DOM is R/W parser.
- Before jdk 1.5 there was only JAXP API as a part jdk, But there was no implementation.
- From jdk 1.5 onwards, jdk has JAXP API and xerces.jar(Apache) implementation.
- If we want different implementation, we need to set corresponding system properties and we need to add corresponding jar file in the class path.

DOM Architecture



Steps to work with DOM

Step 1: create DocumentBuilderFactory

```
DocumentBuilderFactory factory =
                    DocumentBuilderFactory.newInstance();
```

Step 2: create DocumentBuilder

```
DocumentBuilder builder = factory.newDocumentBuilder();
```

Step 3: get inputstream of xml file

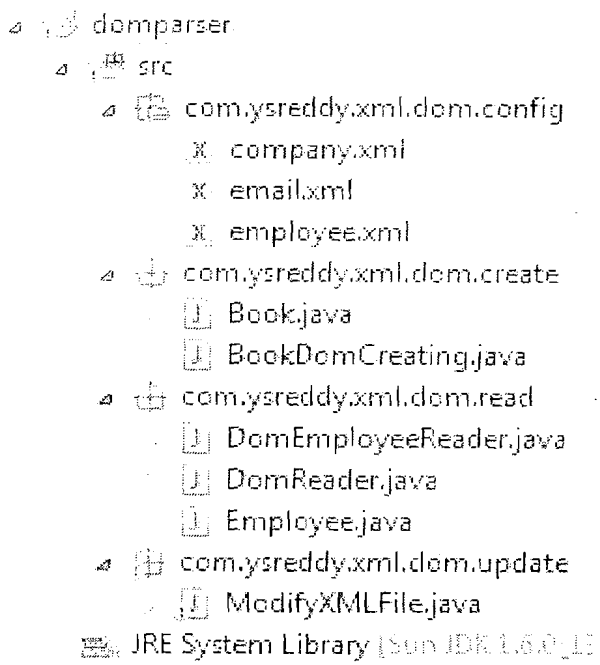
```
ClassLoader cl = DomReader.class.getClassLoader();
InputStream is = cl.getResourceAsStream("XML File");
```

Step 4: parse xml file and get Document object, by calling parse method on DocumentBuilder object.

```
Document document = builder.parse(is);
```

Step 5: Traverse DOM tree using document object and access XML data

Q.) Develop an XML application which can read, write and modify xml documents using DOM?



email.xml

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <mail>
3.     <subject>demo text</subject>
4.     <to>to@gmail.com</to>
5.     <from>from@gmail.com</from>
6. </mail>
```

DomReader.java

```
1. package com.ysreddy.xml.dom.read;
2.
3. import java.io.IOException;
4. import java.io.InputStream;
5.
6. import javax.xml.parsers.DocumentBuilder;
7. import javax.xml.parsers.DocumentBuilderFactory;
8. import javax.xml.parsers.ParserConfigurationException;
9.
10.    import org.w3c.dom.Document;
11.    import org.w3c.dom.Element;
12.    import org.w3c.dom.Node;
13.    import org.w3c.dom.NodeList;
14.    import org.xml.sax.SAXException;
15.
16.    public class DomReader {
17.          public static void main(String[] args) throws ParserConfigurationException,
18.                      SAXException, IOException {
19.                // create DocumentBuilderFactory
20.                DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
21.
22.                // create DocumentBuilder
23.                DocumentBuilder builder = factory.newDocumentBuilder();
24.
```

```
25.                 // get inputstream of email.xml file
26.                 ClassLoader cl = DomReader.class.getClassLoader();
27.                 InputStream is = cl.getResourceAsStream(
28.                                         "com/ysreddy/xml/dom/config/email.xml");
29.
30.                 // parse xml file and get Document object
31.                 Document document = builder.parse(is);
32.
33.                 // get root element of xml document
34.                 Element rootElement = document.getDocumentElement();
35.
36.                 // get <subject> tag value
37.                 Node first = rootElement.getFirstChild();
38.                 Node sibling = first.getNextSibling();
39.                 Node finalNode = sibling.getFirstChild();
40.                 String value = finalNode.getNodeValue();
41.                 System.out.println(value);
42.
43.                 // get <from> tab vlue
44.                 NodeList nodeList = rootElement.getElementsByTagName("from");
45.                 Node fromElement = nodeList.item(0);
46.                 Node fromChild = fromElement.getFirstChild();
47.                 String fromValue = fromChild.getNodeValue();
48.                 System.out.println(fromValue);
49.         }
50.   }
```

employee.xml

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <employees>
3.     <employee type="permanent">
4.           <name>Seagull</name>
5.           <id>3674</id>
6.           <age>34</age>
7.     </employee>
8.     <employee type="contract">
9.           <name>Robin</name>
10.                <id>3675</id>
11.                <age>25</age>
12.          </employee>
13.          <employee type="permanent">
14.                <name>Crow</name>
15.                <id>3676</id>
16.                <age>28</age>
17.          </employee>
18.   </employees>
```

Employee.java

```
1. package com.ysreddy.xml.dom.read;
2.
3. public class Employee {
4.     private String name;
5.     private int id;
6.     private int age;
7.     private String type;
```

```
8.
9.    public Employee() {
10.         }
11.
12.         public Employee(String name, int id, int age, String type) {
13.               this.name = name;
14.               this.id = id;
15.               this.age = age;
16.               this.type = type;
17.         }
18.
19.         public String getName() {
20.               return name;
21.         }
22.
23.         public void setName(String name) {
24.               this.name = name;
25.         }
26.
27.         public int getId() {
28.               return id;
29.         }
30.
31.         public void setId(int id) {
32.               this.id = id;
33.         }
34.
35.         public int getAge() {
36.               return age;
37.         }
38.
39.         public void setAge(int age) {
40.               this.age = age;
41.         }
42.
43.         public String getType() {
44.               return type;
45.         }
46.
47.         public void setType(String type) {
48.               this.type = type;
49.         }
50.
51.         @Override
52.         public String toString() {
53.               return "Employee Details  : name : " + name + "  age :" + age + " id:"
54.                           + id + " type:" + type;
55.         }
56.
57.   }
```

DomEmployeeReader.java

```
1. package com.ysreddy.xml.dom.read;
2.
```

```
3. import java.io.IOException;
4. import java.io.InputStream;
5. import java.util.ArrayList;
6. import java.util.List;
7.
8. import javax.xml.parsers.DocumentBuilder;
9. import javax.xml.parsers.DocumentBuilderFactory;
10.   import javax.xml.parsers.ParserConfigurationException;
11.
12.   import org.w3c.dom.Document;
13.   import org.w3c.dom.Element;
14.   import org.w3c.dom.Node;
15.   import org.w3c.dom.NodeList;
16.   import org.xml.sax.SAXException;
17.
18.   public class DomEmployeeReader {
19.         public static void main(String[] args) {
20.               // a) Getting a document object
21.               Document document = parseXmlFile();
22.
23.               // b) Get a list of employee elements
24.               List<Employee> employees = parseDocument(document);
25.
26.               // c) Iterating and printing.
27.               printData(employees);
28.
29.         }
30.
31.         private static List<Employee> parseDocument(Document document) {
32.
33.               List<Employee> employees = new ArrayList<Employee>();
34.
35.               Element rootElement = document.getDocumentElement();
36.               NodeList empNodelList = rootElement.getElementsByTagName("employee");
37.
38.               for (int i = 0; i < empNodelList.getLength(); i++) {
39.                     Element empElement = (Element) empNodelList.item(i);
40.                     Employee employee = getEmployee(empElement);
41.                     employees.add(employee);
42.               }
43.
44.               return employees;
45.         }
46.
47.         private static Employee getEmployee(Element empElement) {
48.
49.               String name = getTextValue(empElement, "name");
50.               int age = getIntValue(empElement, "age");
51.               int id = getIntValue(empElement, "id");
52.               String type = empElement.getAttribute("type");
53.               Employee employee = new Employee(name, id, age, type);
54.               return employee;
55.         }
56.
57.         private static int getIntValue(Element empElement, String tagName) {
```

```
58.               return Integer.parseInt(getTextValue(empElement, tagName));
59.         }
60.
61.         private static String getTextValue(Element empElement, String tagName) {
62.               NodeList nl = empElement.getElementsByTagName(tagName);
63.               Node nameNode = nl.item(0);
64.               Node textNode = nameNode.getFirstChild();
65.               String name = textNode.getNodeValue();
66.               return name;
67.         }
68.
69.         private static void printData(List<Employee> employees) {
70.
71.               for (Employee employee : employees) {
72.                     System.out.println(employee);
73.               }
74.
75.         }
76.
77.         private static Document parseXmlFile() {
78.               DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
79.               DocumentBuilder builder;
80.               Document document = null;
81.               try {
82.                     builder = factory.newDocumentBuilder();
83.                     InputStream is = DomEmployeeReader.class.getClassLoader()
84.                                                 .getResourceAsStream(
85.                                       "com/ysreddy/xml/dom/config/employee.xml");
86.                     document = builder.parse(is);
87.               } catch (ParserConfigurationException e) {
88.                     e.printStackTrace();
89.               } catch (SAXException e) {
90.                     e.printStackTrace();
91.               } catch (IOException e) {
92.                     e.printStackTrace();
93.               }
94.
95.               return document;
96.         }
97.   }
```

Book.java

```
1. package com.ysreddy.xml.dom.create;
2.
3. public class Book {
4.     private int id;
5.     private String author;
6.     private String title;
7.
8.     public Book() {
9.     }
10.
11.         public Book(int id, String author, String title) {
12.               this.id = id;
13.               this.author = author;
```

```
14.               this.title = title;
15.         }
16.
17.         public int getId() {
18.               return id;
19.         }
20.
21.         public void setId(int id) {
22.               this.id = id;
23.         }
24.
25.         public String getAuthor() {
26.               return author;
27.         }
28.
29.         public void setAuthor(String author) {
30.               this.author = author;
31.         }
32.
33.         public String getTitle() {
34.               return title;
35.         }
36.
37.         public void setTitle(String title) {
38.               this.title = title;
39.         }
40.
41.   }
```

BookDomCreating.java

```
1. package com.ysreddy.xml.dom.create;
2.
3. import java.io.IOException;
4. import java.util.ArrayList;
5. import java.util.Iterator;
6. import java.util.List;
7.
8. import javax.xml.parsers.DocumentBuilder;
9. import javax.xml.parsers.DocumentBuilderFactory;
10.   import javax.xml.parsers.ParserConfigurationException;
11.
12.   import org.w3c.dom.Document;
13.   import org.w3c.dom.Element;
14.   import org.w3c.dom.Text;
15.
16.   import com.sun.org.apache.xml.internal.serialize.OutputFormat;
17.   import com.sun.org.apache.xml.internal.serialize.XMLSerializer;
18.
19.   public class BookDomCreating {
20.         public static void main(String[] args) {
21.               // a) Load Data.
22.               List<Book> books = loadData();
23.
```

```
24.             // b) Getting an instance of DOM.
25.             Document dom = createDocument();
26.
27.             // c) Create the root element Books.
28.             createDOMTree(dom, books);
29.
30.             // d) Serialize DOM to FileOutputStream to generate the xml file
31.             // "book.xml".
32.             printToFile(dom);
33.
34.         }
35.
36.         private static List<Book> loadData() {
37.             List<Book> books = new ArrayList<Book>();
38.             books.add(new Book(1001, "Kathy Sierra", "SCJP"));
39.             books.add(new Book(1002, "James Gosling", "Java Architect"));
40.             return books;
41.         }
42.
43.         private static Document createDocument() {
44.
45.             Document dom = null;
46.
47.             // get an instance of factory
48.             DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
49.             try {
50.                 // get an instance of builder
51.                 DocumentBuilder db = dbf.newDocumentBuilder();
52.
53.                 // create an instance of DOM
54.                 dom = db.newDocument();
55.
56.             } catch (ParserConfigurationException pce) {
57.                 pce.printStackTrace();
58.             }
59.
60.             return dom;
61.         }
62.
63.         private static void createDOMTree(Document dom, List<Book> books) {
64.
65.             // create the root element
66.             Element rootEle = dom.createElement("books");
67.             dom.appendChild(rootEle);
68.
69.             Iterator<Book> it = books.iterator();
70.             while (it.hasNext()) {
71.                 Book b = it.next();
72.                 // For each Book object create element and attach it to root
73.                 Element bookEle = createBookElement(dom, b);
74.                 rootEle.appendChild(bookEle);
75.             }
76.
77.         }
78.
```

```
79.         private static Element createBookElement(Document dom, Book b) {
80.
81.               Element bookEle = dom.createElement("book");
82.               bookEle.setAttribute("id", String.valueOf(b.getId()));
83.
84.               // create author element and author text node and attach it to
85.               // bookElement
86.               Element authEle = dom.createElement("author");
87.               Text authText = dom.createTextNode(b.getAuthor());
88.               authEle.appendChild(authText);
89.               bookEle.appendChild(authEle);
90.
91.               //create title element and title text node and attach it to bookElement
92.
93.               Element titleEle = dom.createElement("title");
94.               Text titleText = dom.createTextNode(b.getTitle());
95.               titleEle.appendChild(titleText);
96.               bookEle.appendChild(titleEle);
97.
98.               return bookEle;
99.
100.        }
101.
102.        private static void printToFile(Document dom) {
103.
104.              try {
105.                    // print
106.                    OutputFormat format = new OutputFormat(dom);
107.                    format.setIndenting(true);
108.
109.                    // to generate output to console use this serializer
110.                    XMLSerializer serializer = new XMLSerializer(System.out, format);
111.
112.                    // to generate a file output use fileoutputstream instead of
113.                    // system.out
114.                    // XMLSerializer serializer = new XMLSerializer(new
115.                    // FileOutputStream(new File("D:\\book.xml")), format);
116.
117.                    serializer.serialize(dom);
118.
119.              } catch (IOException ie) {
120.                    ie.printStackTrace();
121.              }
122.        }
123.
124.  }
```

company.xml

```
1. <?xml version="1.0" encoding="UTF-8"  ?>
2. <company>
3.    <staff id="1">
4.    <firstname>yerragudi</firstname>
5.    <lastname>sreddy</lastname>
6.    <nickname>ysr</nickname>
7.    <salary>100000</salary>
```

```
8.     </staff>
9. </company>
```

ModifyXMLFile.java

```
1. package com.ysreddy.xml.dom.update;
2.
3. import java.io.File;
4. import java.io.IOException;
5. import java.io.InputStream;
6.
7. import javax.xml.parsers.DocumentBuilder;
8. import javax.xml.parsers.DocumentBuilderFactory;
9. import javax.xml.parsers.ParserConfigurationException;
10.    import javax.xml.transform.Transformer;
11.    import javax.xml.transform.TransformerException;
12.    import javax.xml.transform.TransformerFactory;
13.    import javax.xml.transform.dom.DOMSource;
14.    import javax.xml.transform.stream.StreamResult;
15.
16.    import org.w3c.dom.Document;
17.    import org.w3c.dom.Element;
18.    import org.w3c.dom.NamedNodeMap;
19.    import org.w3c.dom.Node;
20.    import org.w3c.dom.NodeList;
21.    import org.xml.sax.SAXException;
22.
23.    public class ModifyXMLFile {
24.
25.          public static void main(String argv[]) {
26.
27.                try {
28.                      InputStream compStream = ModifyXMLFile.class.getClassLoader()
29.                                  .getResourceAsStream(
30.                                        "com/ysreddy/xml/dom/config/company.xml");
31.
32.                      DocumentBuilderFactory docFactory = DocumentBuilderFactory
33.                                  .newInstance();
34.                      DocumentBuilder docBuilder = docFactory.newDocumentBuilder();
35.                      Document doc = docBuilder.parse(compStream);
36.
37.                      // Get the root element
38.                      Node company = doc.getFirstChild();
39.
40.                      // Get the staff element , it may not working if tag has spaces,
41.                      // or
42.                      // whatever weird characters in front...it's better to use
43.                      // getElementsByTagName() to get it directly.
44.                      // Node staff = company.getFirstChild();
45.
46.                      // Get the staff element by tag name directly
47.                      Node staff = doc.getElementsByTagName("staff").item(0);
48.
49.                      // update staff attribute
50.                      NamedNodeMap attr = staff.getAttributes();
51.                      Node nodeAttr = attr.getNamedItem("id");
```

```
52.                     nodeAttr.setTextContent("2");
53.
54.                     // append a new node to staff
55.                     Element age = doc.createElement("age");
56.                     age.appendChild(doc.createTextNode("28"));
57.                     staff.appendChild(age);
58.
59.                     // loop the staff child node
60.                     NodeList list = staff.getChildNodes();
61.
62.                     for (int i = 0; i < list.getLength(); i++) {
63.
64.                           Node node = list.item(i);
65.
66.                           // get the salary element, and update the value
67.                           if ("salary".equals(node.getNodeName())) {
68.                                 node.setTextContent("2000000");
69.                           }
70.
71.                           // remove firstname
72.                           if ("firstname".equals(node.getNodeName())) {
73.                                 staff.removeChild(node);
74.                           }
75.
76.                     }
77.
78.                     // write the content into xml file
79.                     TransformerFactory transformerFactory = TransformerFactory
80.                                 .newInstance();
81.                     Transformer transformer = transformerFactory.newTransformer();
82.                     DOMSource source = new DOMSource(doc);
83.                     StreamResult result=new StreamResult(new File("D://modify.xml"));
84.                     transformer.transform(source, result);
85.
86.                     System.out.println("Done");
87.
88.               } catch (ParserConfigurationException pce) {
89.                     pce.printStackTrace();
90.               } catch (TransformerException tfe) {
91.                     tfe.printStackTrace();
92.               } catch (IOException ioe) {
93.                     ioe.printStackTrace();
94.               } catch (SAXException sae) {
95.                     sae.printStackTrace();
96.               }
97.         }
98.   }
```

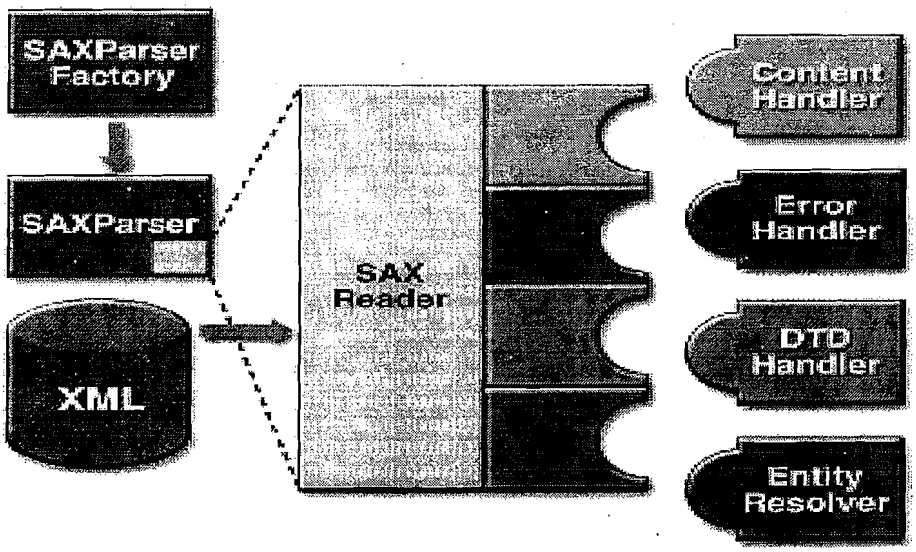
SAX

- ⇨ The **Simple API for XML** (**SAX**) is a serial access parser API for XML.
- ⇨ It is an event-driven model for processing XML, which implements the technique to register the handler to invoke the callback methods whenever an event is generated.
- ⇨ Event is generated when the parser encounters a new XML tag

Q.) What are the difference between DOM and SAX?

DOM PARSER	SAX PARSER
DOM is Document Object Model API for XML	SAX is a Simple API for XML
DOM is a tree model parser	SAX is an event based parser
Stores the entire XML document into memory before processing	Parses node by node
Occupies more memory	Occupies less memory
DOM is read and write parser.	SAX is read only parser
Random Access	Sequential Access
Preserves comments	Doesn't preserve comments
Slower than SAX	Faster than DOM

SAX Architecture



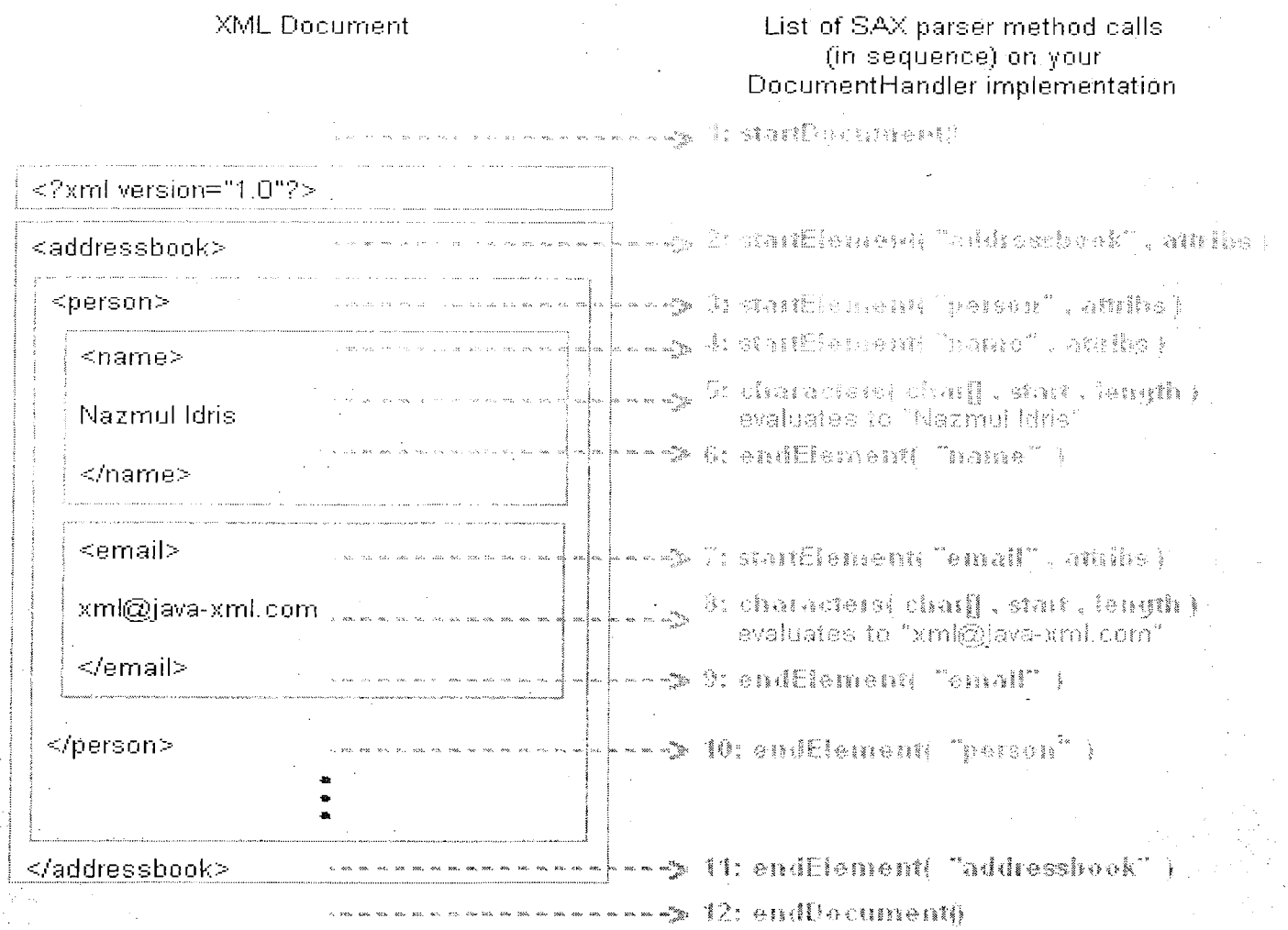


Figure 1 : SAX DocumentHandler interface methods and their sequence

Steps to work with SAX:

Step 1: Implementing the required handlers:

```
public class SAXXMLParserImpl extends DefaultHandler{  …. }
```

Step 2: New instance of SAXParserFactory

```
SAXParserFactory factory = SAXParserFactory.newInstance();
```

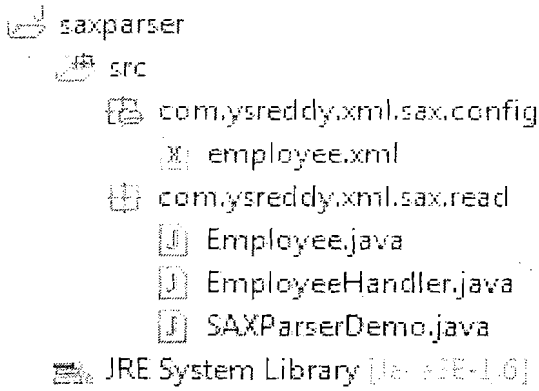
Step 3: New instance of SAX Parser:

```
SAXParser saxParser = factory.newSAXParser();
```

Step 4: Parsing the XML document:

```
saxParser.parse( new File(XML_FILE_TO_BE_PARSED),
                                    new SAXXMLParserImpl() );
```

Q.) Develop XML application where you can read employee.xml file using SAX parser?



employee.xml

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <employees>
3.     <employee type="permanent">
4.           <name>Seagull</name>
5.           <id>3674</id>
6.           <age>34</age>
7.     </employee>
8.     <employee type="contract">
9.           <name>Robin</name>
10.                <id>3675</id>
11.                <age>25</age>
12.          </employee>
13.          <employee type="permanent">
14.                <name>Crow</name>
15.                <id>3676</id>
16.                <age>28</age>
17.          </employee>
18.    </employees>
```

Employee.java

```
1. package com.ysreddy.xml.sax.read;
2.
3. public class Employee {
4.     private int id;
5.     private String name;
6.     private int age;
7.     private String type;
8.
9.     public Employee() {
10.          }
11.
```

```
12.         public Employee(int id, String name, int age, String type) {
13.               this.id = id;
14.               this.name = name;
15.               this.age = age;
16.               this.type = type;
17.         }
18.
19.         public int getId() {
20.               return id;
21.         }
22.
23.         public void setId(int id) {
24.               this.id = id;
25.         }
26.
27.         public String getName() {
28.               return name;
29.         }
30.
31.         public void setName(String name) {
32.               this.name = name;
33.         }
34.
35.         public int getAge() {
36.               return age;
37.         }
38.
39.         public void setAge(int age) {
40.               this.age = age;
41.         }
42.
43.         public String getType() {
44.               return type;
45.         }
46.
47.         public void setType(String type) {
48.               this.type = type;
49.         }
50.
51.         @Override
52.         public String toString() {
53.               return "Employee Details -- Eid :"+id +" Name : "+ name +" Age : "+age
54.                     +" Type : "+type ;
55.         }
56.
57.   }
```

EmplyeeHandler.java

```
1. package com.ysreddy.xml.sax.read;
2.
3. import java.util.ArrayList;
4. import java.util.List;
5.
6. import org.xml.sax.Attributes;
7. import org.xml.sax.SAXException;
```

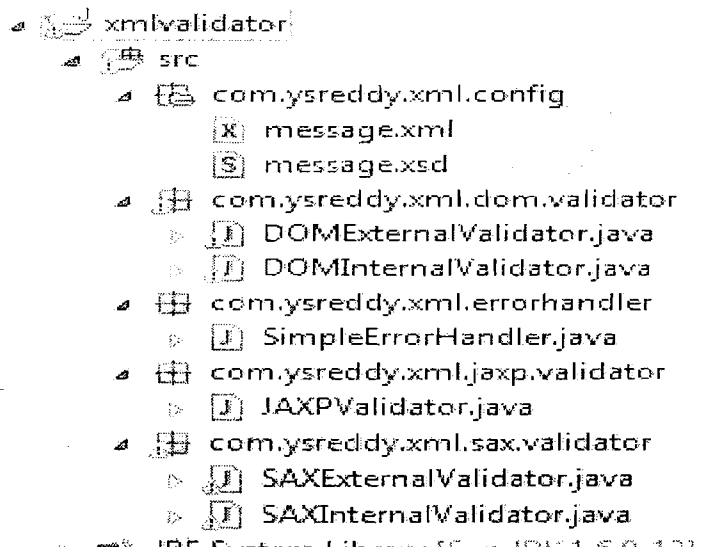
```
8. import org.xml.sax.helpers.DefaultHandler;
9.
10.   public class EmployeeHandler extends DefaultHandler {
11.         private Employee tempEmp;
12.         private String tempVal;
13.         private List<Employee> emps = new ArrayList<Employee>();
14.
15.         @Override
16.         public void startDocument() throws SAXException {
17.               System.out.println("....DOCUMENT STARTED...");
18.         }
19.
20.         @Override
21.         public void endDocument() throws SAXException {
22.               for(Employee employee : emps)
23.                     System.out.println(employee);
24.         }
25.
26.         @Override
27.         public void startElement(String uri, String localName, String qName,
28.                     Attributes attributes) throws SAXException {
29.               // System.out.println("\n.... starting....");
30.               // System.out.println("URI: " + uri);
31.               // System.out.println("localName: " + localName);
32.               // System.out.println("qName: " + qName);
33.
34.               tempVal = "";
35.               if (qName.equalsIgnoreCase("employee")) {
36.                     // create a new instance of employee
37.                     tempEmp = new Employee();
38.                     tempEmp.setType(attributes.getValue("type"));
39.               }
40.
41.         }
42.
43.         @Override
44.         public void characters(char[] ch, int start, int length)
45.                     throws SAXException {
46.               // System.out.println("\nch: "+ch);
47.               // System.out.println("start: "+start);
48.               // System.out.println("length: "+length);
49.               // System.out.println("value: "+new String(ch,start,length));
50.               tempVal = new String(ch, start, length);
51.
52.         }
53.
54.         @Override
55.         public void endElement(String uri, String localName, String qName)
56.                     throws SAXException {
57.               // System.out.println("\n.... ending....");
58.               // System.out.println("URI: " + uri);
59.               // System.out.println("localName: " + localName);
60.               // System.out.println("qName: " + qName);
61.               if (qName.equalsIgnoreCase("employee")) {
62.                     // add it to the list
```

```
63.                     emps.add(tempEmp);
64.
65.                 } else if (qName.equalsIgnoreCase("name")) {
66.                     tempEmp.setName(tempVal);
67.                 } else if (qName.equalsIgnoreCase("id")) {
68.                     tempEmp.setId(Integer.parseInt(tempVal));
69.                 } else if (qName.equalsIgnoreCase("age")) {
70.                     tempEmp.setAge(Integer.parseInt(tempVal));
71.                 }
72.
73.         }
74.
75.     }
```

SaxParserDemo.java

```
1. package com.ysreddy.xml.sax.read;
2.
3. import java.io.IOException;
4.
5. import javax.xml.parsers.ParserConfigurationException;
6. import javax.xml.parsers.SAXParser;
7. import javax.xml.parsers.SAXParserFactory;
8.
9. import org.xml.sax.SAXException;
10.
11.    public class SAXParserDemo {
12.         public static void main(String[] args) throws ParserConfigurationException,
13.                                                        SAXException, IOException {
14.                 SAXParserFactory factory = SAXParserFactory.newInstance();
15.                 SAXParser parser = factory.newSAXParser();
16.                 parser.parse(SAXParserDemo.class.getClassLoader().getResourceAsStream(
17.                    "com/ysreddy/xml/sax/config/employee.xml"), new EmployeeHandler());
18.
19.         }
20.    }
```

Q.) Develop java application to validate XML document against XSD?



message.xsd

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
3.     xmlns:msg="http://ysreddy.com/java/xml/types"
4.     targetNamespace="http://ysreddy.com/java/xml/types"
5.     elementFormDefault="qualified"
6.     attributeFormDefault="unqualified">
7.
8.     <xs:element name="message">
9.           <xs:complexType>
10.                 <xs:sequence>
11.                       <xs:element name="from" type="xs:string" />
12.                       <xs:element name="to" type="xs:string" minOccurs="1"
13.                             maxOccurs="unbounded" />
14.                       <xs:element name="subject">
15.                             <xs:simpleType>
16.                                   <xs:restriction base="xs:string">
17.                                         <xs:minLength value="10" />
18.                                         <xs:maxLength value="100" />
19.                                   </xs:restriction>
20.                             </xs:simpleType>
21.                       </xs:element>
22.                       <xs:element name="body" type="xs:string" minOccurs="0"
23.                             maxOccurs="1" />
24.                 </xs:sequence>
25.           </xs:complexType>
26.     </xs:element>
27. </xs:schema>
```

message.xml

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <m:message xmlns:m="http://ysreddy.com/java/xml/types"
3. xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4. xsi:schemaLocation="http://ysreddy.com/java/xml/types file:///E:/Y.S.Reddy/XML/xsd-
5. examples/xsd/message.xsd">
6.
7.     <m:from>ysr@gmail.com</m:from>
8.     <m:to>ys@gmail.com</m:to>
9.     <!--<m:subject>somthing something</m:subject> -->
10.    </m:message>
```

SimpleErrorHandler.java

```
1. package com.ysreddy.xml.errorhandler;
2.
3. import org.xml.sax.ErrorHandler;
4. import org.xml.sax.SAXException;
5. import org.xml.sax.SAXParseException;
6.
7. public class SimpleErrorHandler implements ErrorHandler {
8.     public void warning(SAXParseException e) throws SAXException {
9.         System.out.println(e.getMessage());
10.       }
11.
12.       public void error(SAXParseException e) throws SAXException {
```

```
13.             System.out.println(e.getMessage());
14.         }
15.
16.         public void fatalError(SAXParseException e) throws SAXException {
17.             System.out.println(e.getMessage());
18.         }
19.     }
```

DOMInternalValidator.java

```
1. package com.ysreddy.xml.dom.validator;
2.
3. import java.io.IOException;
4.
5. import javax.xml.parsers.DocumentBuilder;
6. import javax.xml.parsers.DocumentBuilderFactory;
7. import javax.xml.parsers.ParserConfigurationException;
8.
9. import org.w3c.dom.Document;
10.    import org.xml.sax.SAXException;
11.
12.    import com.ysreddy.xml.errorhandler.SimpleErrorHandler;
13.
14.    public class DOMInternalValidator {
15.          public static void main(String[] args) throws SAXException,
16.                       ParserConfigurationException, IOException {
17.                DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
18.                factory.setValidating(true);
19.                factory.setNamespaceAware(true);
20.                factory.setAttribute(
21.                            "http://java.sun.com/xml/jaxp/properties/schemaLanguage",
22.                            "http://www.w3.org/2001/XMLSchema");
23.
24.                DocumentBuilder builder = factory.newDocumentBuilder();
25.    //          builder.setErrorHandler(new SimpleErrorHandler());
26.                Document document= builder.parse(
27.                                  DOMInternalValidator.class.getClassLoader()
28.                                  .getResourceAsStream(
29.                                  "com/ysreddy/xml/config/message.xml"));
30.                System.out.println(".........");
31.          }
32.    }
```

DOMExternalValidator.java

```
1. package com.ysreddy.xml.dom.validator;
2.
3. import java.io.IOException;
4.
5. import javax.xml.parsers.DocumentBuilder;
6. import javax.xml.parsers.DocumentBuilderFactory;
7. import javax.xml.parsers.ParserConfigurationException;
8. import javax.xml.transform.Source;
9. import javax.xml.transform.stream.StreamSource;
10.    import javax.xml.validation.SchemaFactory;
11.
```

```
12.   import org.w3c.dom.Document;
13.   import org.xml.sax.InputSource;
14.   import org.xml.sax.SAXException;
15.
16.   import com.ysreddy.xml.errorhandler.SimpleErrorHandler;
17.   import com.ysreddy.xml.jaxp.validator.JAXPValidator;
18.
19.   public class DOMExternalValidator {
20.         public static void main(String[] args) throws SAXException,
21.                     ParserConfigurationException, IOException {
22.               DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
23.               factory.setValidating(false);
24.               factory.setNamespaceAware(true);
25.
26.               SchemaFactory schemaFactory =
27.                   SchemaFactory.newInstance("http://www.w3.org/2001/XMLSchema");
28.
29.               factory.setSchema(schemaFactory.newSchema(
30.                   new Source[] {new StreamSource(
31.                   DOMExternalValidator.class.getClassLoader()
32.                    .getResourceAsStream("com/ysreddy/xml/config/message.xsd"))}));
33.
34.               DocumentBuilder builder = factory.newDocumentBuilder();
35.
36.   //          builder.setErrorHandler(new SimpleErrorHandler());
37.
38.               Document document = builder.parse(new InputSource(
39.                         DOMExternalValidator.class.getClassLoader()
40.                         .getResourceAsStream("com/ysreddy/xml/config/message.xml")));
41.               System.out.println(".........");
42.         }
43.   }
```

SAXInternalValidator.java

```
1. package com.ysreddy.xml.sax.validator;
2.
3. import java.io.IOException;
4.
5. import javax.xml.parsers.ParserConfigurationException;
6. import javax.xml.parsers.SAXParser;
7. import javax.xml.parsers.SAXParserFactory;
8.
9. import org.xml.sax.InputSource;
10.   import org.xml.sax.SAXException;
11.   import org.xml.sax.XMLReader;
12.
13.   import com.ysreddy.xml.errorhandler.SimpleErrorHandler;
14.
15.   public class SAXInternalValidator {
16.         public static void main(String[] args) throws IOException, SAXException,
17.                                                   ParserConfigurationException {
18.
19.               SAXParserFactory factory = SAXParserFactory.newInstance();
20.               factory.setValidating(true);
21.               factory.setNamespaceAware(true);
```

```
22.
23.             SAXParser parser = factory.newSAXParser();
24.             parser.setProperty(
25.                         "http://java.sun.com/xml/jaxp/properties/schemaLanguage",
26.                         "http://www.w3.org/2001/XMLSchema");
27.
28.             XMLReader reader = parser.getXMLReader();
29.   //        reader.setErrorHandler(new SimpleErrorHandler());
30.             reader.parse(new InputSource(
31.                    SAXInternalValidator.class.getClassLoader()
32.                   .getResourceAsStream("com/ysreddy/xml/config/message.xml")));
33.             System.out.println(".........");
34.         }
35.   }
```

SAXExternalValidator.java

```
1. package com.ysreddy.xml.sax.validator;
2.
3. import java.io.IOException;
4.
5. import javax.xml.parsers.ParserConfigurationException;
6. import javax.xml.parsers.SAXParser;
7. import javax.xml.parsers.SAXParserFactory;
8. import javax.xml.transform.Source;
9. import javax.xml.transform.stream.StreamSource;
10.   import javax.xml.validation.SchemaFactory;
11.
12.   import org.xml.sax.InputSource;
13.   import org.xml.sax.SAXException;
14.   import org.xml.sax.XMLReader;
15.
16.   import com.ysreddy.xml.dom.validator.DOMExternalValidator;
17.   import com.ysreddy.xml.errorhandler.SimpleErrorHandler;
18.
19.   public class SAXExternalValidator {
20.         public static void main(String[] args) throws SAXException, IOException,
21.                                                 ParserConfigurationException {
22.
23.             SAXParserFactory factory = SAXParserFactory.newInstance();
24.             factory.setValidating(false);
25.             factory.setNamespaceAware(true);
26.
27.             SchemaFactory schemaFactory =
28.                 SchemaFactory.newInstance("http://www.w3.org/2001/XMLSchema");
29.
30.             factory.setSchema(schemaFactory.newSchema(
31.                 new Source[] {new
32.                   StreamSource(SAXExternalValidator.class.getClassLoader()
33.                  .getResourceAsStream("com/ysreddy/xml/config/message.xsd"))}));
34.
35.             SAXParser parser = factory.newSAXParser();
36.
37.             XMLReader reader = parser.getXMLReader();
38.   //        reader.setErrorHandler(new SimpleErrorHandler());
39.             reader.parse(new InputSource(
```

```
40.                         SAXExternalValidator.class.getClassLoader()
41.                        .getResourceAsStream("com/ysreddy/xml/config/message.xml")));
42.       }
43.   }
```

JAXPValidator .java

```
1. package com.ysreddy.xml.jaxp.validator;
2.
3. import java.io.IOException;
4. import java.io.InputStream;
5.
6. import javax.xml.XMLConstants;
7. import javax.xml.transform.Source;
8. import javax.xml.transform.stream.StreamSource;
9. import javax.xml.validation.Schema;
10.   import javax.xml.validation.SchemaFactory;
11.   import javax.xml.validation.Validator;
12.
13.   import org.xml.sax.SAXException;
14.
15.   public class JAXPValidator {
16.         public static void main(String[] args) throws SAXException, IOException {
17.
18.           // 1. Lookup a factory for the W3C XML Schema language
19.           SchemaFactory factory =
20.               SchemaFactory.newInstance(XMLConstants.W3C_XML_SCHEMA_NS_URI);
21.
22.           // 2. Compile the schema.
23.           InputStream xsdInputStream = JAXPValidator.class.getClassLoader()
24.                       .getResourceAsStream("com/ysreddy/xml/config/message.xsd");
25.           Source xsdSource = new StreamSource(xsdInputStream);
26.           Schema schema = factory.newSchema(xsdSource);
27.
28.           // 3. Get a validator from the schema.
29.           Validator validator = schema.newValidator();
30.
31.           // 4. Parse the document you want to check.
32.           InputStream xmlInputStream = JAXPValidator.class.getClassLoader()
33.                       .getResourceAsStream("com/ysreddy/xml/config/message.xml");
34.           Source xmlSource = new StreamSource(xmlInputStream);
35.
36.           // 5. Check the document
37.           try {
38.               validator.validate(xmlSource);
39.               System.out.println(" Given is valid.");
40.           }
41.           catch (SAXException ex) {
42.               System.out.println("Given is not valid because ");
43.               System.out.println(ex.getMessage());
44.           }
45.
46.       }
47.   }
```

Q.) Which one is better, SAX or DOM ? In what cases, we prefer SAXParser and DOMParser?

Both SAX and DOM parser have their advantages and disadvantages. Which one is better should depends on the characteristics of your application.

StAX – Streaming API for XML

StAX solves SAX and DOM problems by providing more control of XML parsing to the programmer, in particular by exposing a simple iterator-based API and an underlying stream of events. Methods such as next() and hasNext() allow an application developer to ask for the next event, or pull the event, rather than handle the event in a callback. StAX also enables the programmer to stop processing the document at any time, skip ahead to sections of the document, and get subsections of the document.

Q.) What is JAXB?

- JAXB stands for **"Java Architecture for XML Binding"**
- JAXB is a Java standard that defines how Java objects are converted to/from XML

Example binding frameworks:

Caster, Glue, JAXB, XMLBeans, XStream, Liquid...etc.

Q.) Why JAXB?

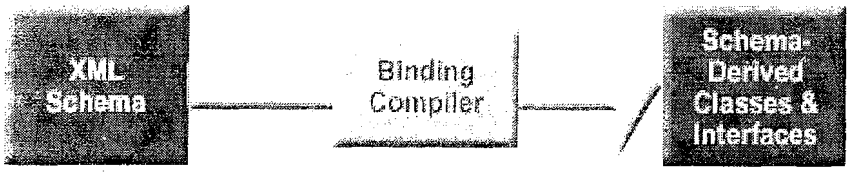
- Programmers don't have to deal with XML directly
- JAXB parser is faster than SAX parser
- JAXB also provides random accses like DOM
- It provides compiler that compiles XML schema to Java classes
- JAXB is high-level language while JAXP/SAX/DOM are assembly language for XML document Management.

Q.)How to use JAXB to access XML document?

Step 1: Generate the Java source files by submitting the XML Schema to the binding compiler(XJC)

C:/> xjc (-options) schema

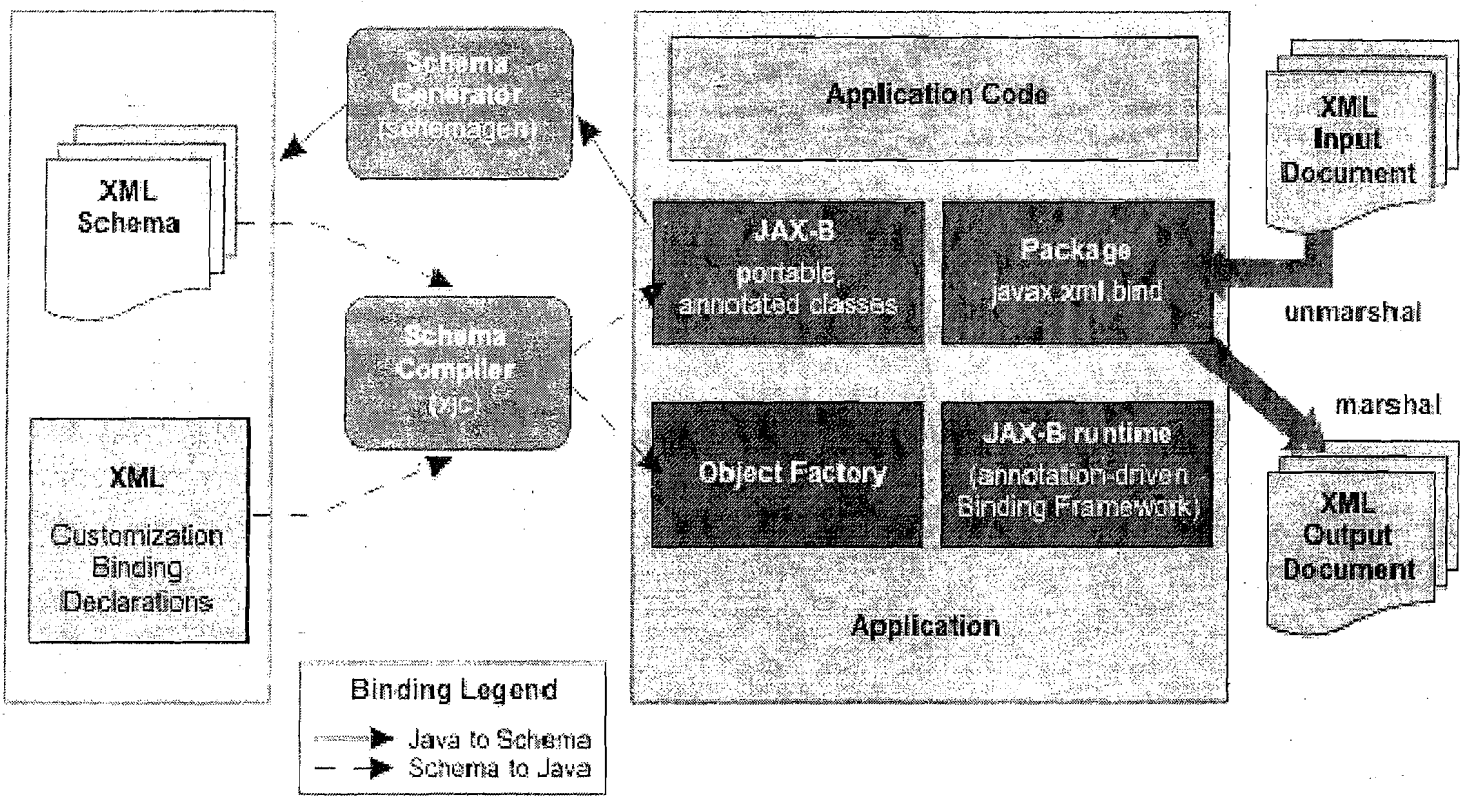
C:/> xjc -p <package-name> -d <destination> schema



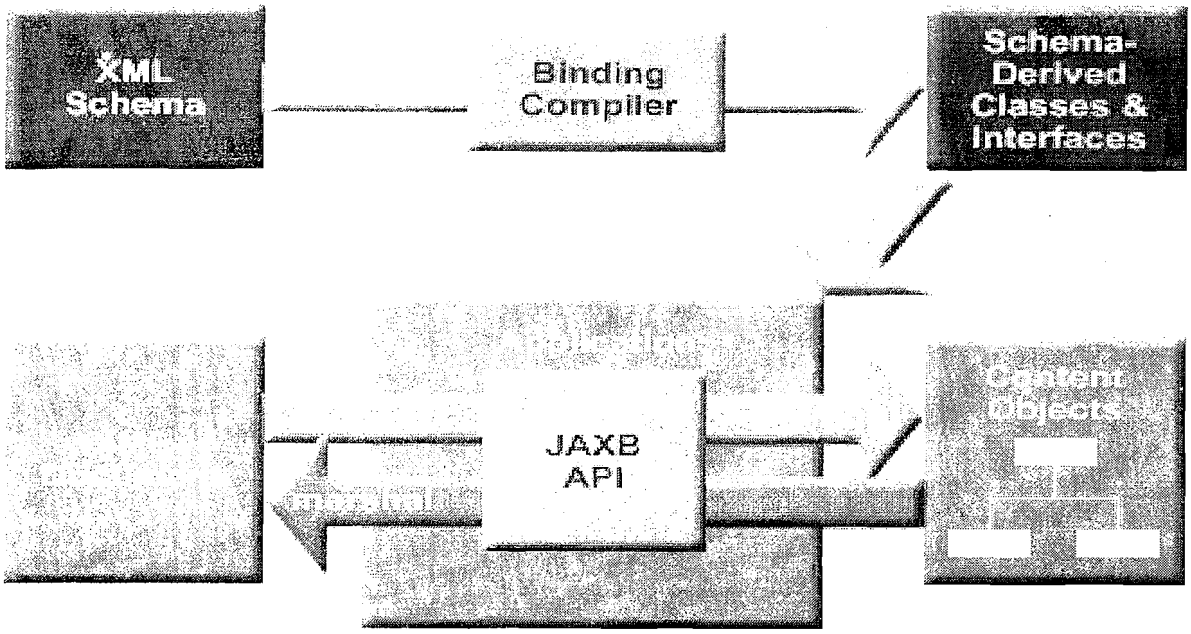
Step 2: Compile the Java source code.

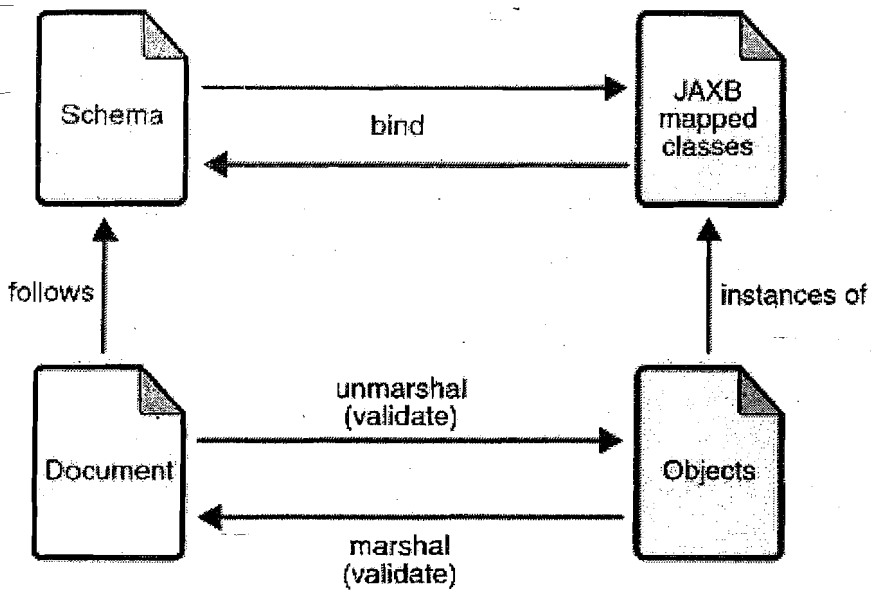
Step 3: Perform Marshall/Unmarshal operations

JAXB architecture



Binding Life Cycle





JAXB compile time steps(one time operation)

- Develop or obtain XML schema
- Generate the java source files by compiling the XML schema through the binding compiler
- Develop JAXB client application (with the javax:xml.bind JAXB interfaces)
- Compile the Java source codes

JAXB Runtime Steps

With the classes and the binding framework, write Java applications that

- Unmarshalling the data from an XML
- Access and modify the data.
- Optionally validate the against XML schema.
- Marshal in-memory data to a new XML document

Steps to work with JAXB(Using XJC of jwsdp-2.0)

Step 1: Create "java project"

Step 2: Create one source folder with name "resources"

Step 3: Place company.xsd file in "resources" folder

Step 4: Create binding classes by running XJC compiler as follows

(set the path to **java version "1.5.0_22"**, set path to "**C:\Sun\jwsdp-2.0\jaxb\bin**")

```
C:\Windows\system32\cmd.exe
E:\Y.S.Reddy\WEB-SERVICES\workspace\jaxb>xjc -d src -verbose resources/company.xsd
parsing a schema...
compiling a schema...
[INFO] generating code
unknown location

com\ysreddy\xml\jaxb\company\types\AddressType.java
com\ysreddy\xml\jaxb\company\types\CompanyType.java
com\ysreddy\xml\jaxb\company\types\EmployeeType.java
com\ysreddy\xml\jaxb\company\types\ObjectFactory.java
com\ysreddy\xml\jaxb\company\types\package-info.java

E:\Y.S.Reddy\WEB-SERVICES\workspace\jaxb>
```

Step 5: Place the required jar files in the classpath to work with JAXB

Step 6: Perform Marshall, UnMarsall operations using JAXB API.

NOTE: If we are using JDK 6 and more version, we no need to add any jar files. Because JDK itself has JAXB implementation.

Steps to work with JAXB(Using XJC of JDK 6.0)

Step 1: Create "java project"

Step 2: Create one source folder with name "resources"

Step 3: Place company.xsd file in "resources" folder

Step 4: Set the **path** to jdk 6.0

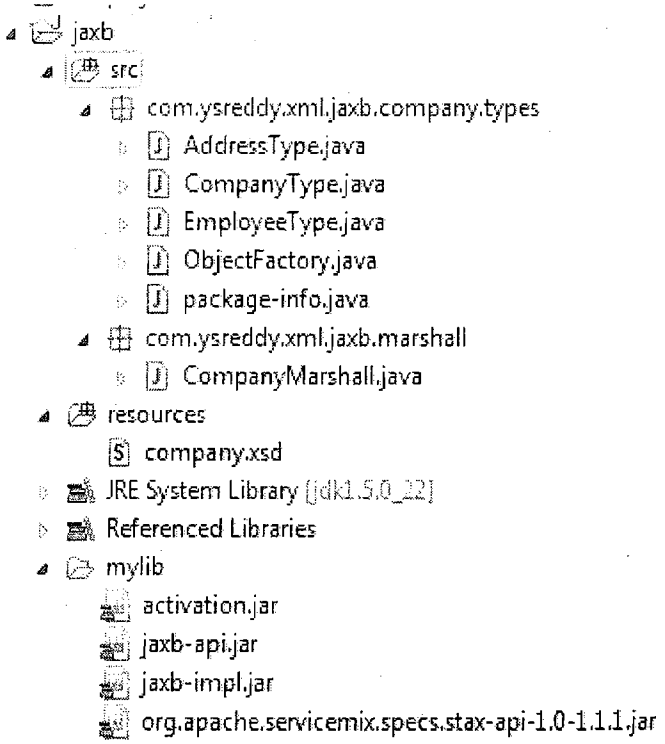
set path=C:\Program Files\Java\jdk1.6.0_18\bin;%path%

```
E:\Y.S.Reddy\WEB-SERVICES\workspace\jaxb>xjc -d src -verbose resources/company.xsd
parsing a schema...
compiling a schema...
[INFO] generating code
unknown location

com\ysreddy\xml\jaxb\company\types\AddressType.java
com\ysreddy\xml\jaxb\company\types\CompanyType.java
com\ysreddy\xml\jaxb\company\types\EmployeeType.java
com\ysreddy\xml\jaxb\company\types\ObjectFactory.java
com\ysreddy\xml\jaxb\company\types\package-info.java

E:\Y.S.Reddy\WEB-SERVICES\workspace\jaxb>
```

Step 5: Perform Marshall, UnMarsall operations using JAXB API.



company.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified" attributeFormDefault="unqualified"
          targetNamespace="http://ysreddy.com/xml/jaxb/company/types"
   xmlns:c="http://ysreddy.com/xml/jaxb/company/types">

          <xs:element name="company" type="c:companyType" />

          <xs:complexType name="companyType">
                <xs:sequence>
                      <xs:element name="employee" type="c:employeeType"
                            minOccurs="1" maxOccurs="unbounded" />
                </xs:sequence>
          </xs:complexType>

          <xs:complexType name="employeeType">
                <xs:sequence>
                      <xs:element name="eno" type="xs:int" />
                      <xs:element name="name" type="xs:string" />
                      <xs:element name="salary" type="xs:double" />
                      <xs:element name="address" type="c:addressType" />
                </xs:sequence>
          </xs:complexType>

         <xs:complexType name="addressType">
                <xs:sequence>
                      <xs:element name="hno" type="xs:string" />
                      <xs:element name="city" type="xs:string" />
                      <xs:element name="pincode" type="xs:int" />
                </xs:sequence>
          </xs:complexType>

   </xs:schema>
```

company.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<c:company xmlns:c="http://ysreddy.com/xml/jaxb/company/types"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://ysreddy.com/xml/jaxb/company/types
file:///E:/Y.S.Reddy/XML/xsd-examples/xsd/company.xsd">
    <c:employee>
          <c:eno>1001</c:eno>
          <c:name>ysreddy</c:name>
          <c:salary>2498.98</c:salary>
                <c:address>
                      <c:hno>1-2A/2</c:hno>
                      <c:city>Hyd</c:city>
                      <c:pincode>5000082</c:pincode>
                </c:address>
          </c:employee>

          <c:employee>
                <c:eno>1002</c:eno>
                <c:name>ys</c:name>
```

```
20.             <c:salary>6824.08</c:salary>
21.             <c:address>
22.                   <c:hno>45-2B/2</c:hno>
23.                   <c:city>Bang</c:city>
24.                   <c:pincode>5000080</c:pincode>
25.             </c:address>
26.       </c:employee>
27.
28.  </c:company>
```

CompanyMarshall.java

```
1. package com.ysreddy.xml.jaxb.marshall;
2.
3. import javax.xml.bind.JAXBContext;
4. import javax.xml.bind.JAXBException;
5. import javax.xml.bind.Marshaller;
6.
7. import com.ysreddy.xml.jaxb.company.types.AddressType;
8. import com.ysreddy.xml.jaxb.company.types.CompanyType;
9. import com.ysreddy.xml.jaxb.company.types.EmployeeType;
10.
11.  public class CompanyMarshall {
12.       public static void main(String[] args) throws JAXBException {
13.             JAXBContext context = JAXBContext.newInstance(
14.                                   "com.ysreddy.xml.jaxb.company.types");
15.             Marshaller marshaller = context.createMarshaller();
16.             marshaller.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, true);
17.
18.             CompanyType companyType = new CompanyType();
19.
20.             EmployeeType e1 = new EmployeeType();
21.             EmployeeType e2 = new EmployeeType();
22.
23.             AddressType a1 = new AddressType();
24.             a1.setHno("1/2A");
25.             a1.setCity("hyd");
26.             a1.setPincode(1234);
27.
28.             e1.setEno(10001);
29.             e1.setName("abc");
30.             e1.setSalary(4500.00);
31.             e1.setAddress(a1);
32.
33.             e2.setEno(10002);
34.             e2.setName("xyz");
35.             e2.setSalary(4569.00);
36.             e2.setAddress(a1);
37.
38.             companyType.getEmployee().add(e1);
39.             companyType.getEmployee().add(e2);
40.
41.             marshaller.marshal(companyType, System.out);
42.
43.
44.
```

```
45.         }
46.    }
```

CompanyUnmarshall.java

```
1. package com.ysreddy.xml.jaxb.marshall;
2.
3. import java.io.InputStream;
4.
5. import javax.xml.XMLConstants;
6. import javax.xml.bind.JAXBContext;
7. import javax.xml.bind.JAXBElement;
8. import javax.xml.bind.JAXBException;
9. import javax.xml.bind.Unmarshaller;
10.    import javax.xml.transform.Source;
11.    import javax.xml.transform.stream.StreamSource;
12.    import javax.xml.validation.Schema;
13.    import javax.xml.validation.SchemaFactory;
14.
15.    import org.xml.sax.SAXException;
16.
17.    import com.ysreddy.xml.jaxb.company.types.CompanyType;
18.    import com.ysreddy.xml.jaxb.company.types.EmployeeType;
19.
20.    public class CompanyUnMarshall {
21.          public static void main(String[] args) throws SAXException, JAXBException {
22.          SchemaFactory factory = SchemaFactory.newInstance(
23.                                                XMLConstants.W3C_XML_SCHEMA_NS_URI);
24.          InputStream inputStream = CompanyUnMarshall.class.getClassLoader()
25.                                                .getResourceAsStream("company.xsd");
26.          Source source = new StreamSource(inputStream);
27.          Schema schema = factory.newSchema(source);
28.          JAXBContext context = JAXBContext.newInstance(
29.                                               "com.ysreddy.xml.jaxb.company.types");
30.          Unmarshaller unmarshal= context.createUnmarshaller();
31.          unmarshal.setSchema(schema);
32.          JAXBElement<CompanyType> element=  (JAXBElement<CompanyType>)
33.             unmarshal.unmarshal(CompanyUnMarshall.class.getClassLoader()
34.                                                .getResourceAsStream("company.xml"));
35.          CompanyType companyType = element.getValue();
36.          System.out.println("Employees information....");
37.          for(EmployeeType employeeType: companyType.getEmployee()){
38.                System.out.println("\nENO : "+employeeType.getEno());
39.                System.out.println("NAME : "+employeeType.getName());
40.                System.out.println("SALARY : "+employeeType.getSalary());
41.                System.out.println("HNO : "+employeeType.getAddress().getHno());
42.                System.out.println("CITY : "+employeeType.getAddress().getCity());
43.                System.out.println("PINCODE : "
44.                                           +employeeType.getAddress().getPincode());
45.          }
46.    }
47.    }
```

JAXB uses annotations to indicate the central elements.

Annotation	Description
@XmlRootElement(namespace = "namespace")	Define the root element for a XML tree
@XmlType(propOrder = { "field2", "field1",.. })	Allows to define the order in which the fields are written in the XML file
@XmlElement(name = "neuName")	Define the XML element which will be used. Only need to be used if the neuNeu is different then the JavaBeans Name

The general steps in the JAXB data binding process are:

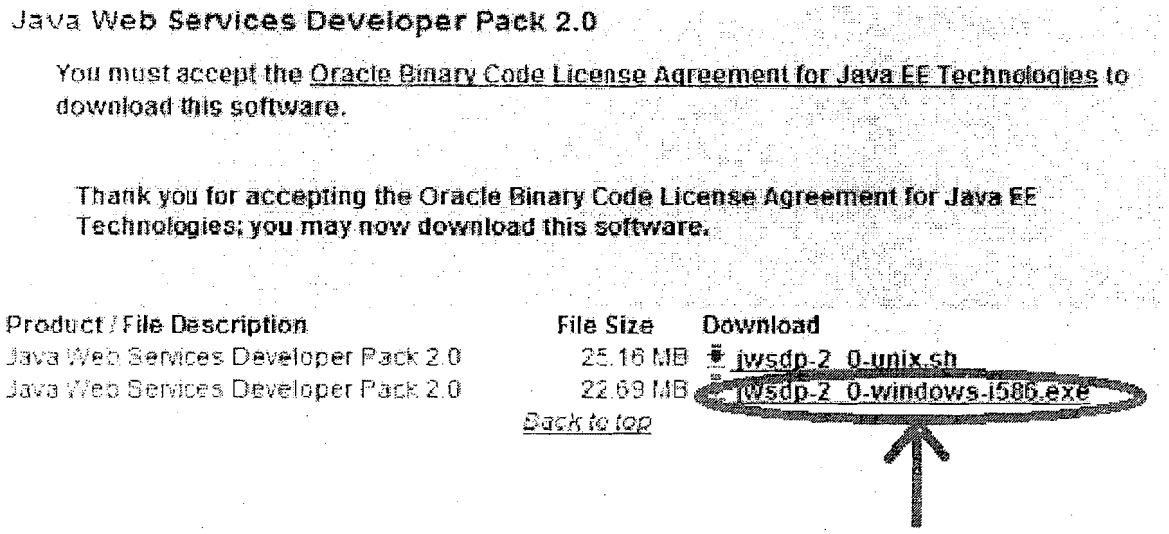
1. **Generate classes**. An XML schema is used as input to the JAXB binding compiler to generate JAXB classes based on that schema.
2. **Compile classes**. All of the generated classes, source files, and application code must be compiled.
3. **Unmarshal**. XML documents written according to the constraints in the source schema are unmarshalled by the JAXB binding framework.
4. **Generate content tree**. The unmarshalling process generates a content tree of data objects instantiated from the generated JAXB classes; this content tree represents the structure and content of the source XML documents.
5. **Validate** (optional). The unmarshalling process optionally involves validation of the source XML documents before generating the content tree.
6. **Process the content**. The client application can modify the XML data represented by the Java content tree by means of interfaces generated by the binding compiler.
7. **Marshal**. The processed content tree is marshalled out to one or more XML output documents. The content may be validated before marshalling.

jwsdp-2.0 Installation Steps

Step 1: Down load **jwsdp-2_0-windows-i586.exe** from the bellow URL

http://www.oracle.com/technetwork/java/javasebusiness/downloads/java-archive-downloads-jwsdp-419428.html#jwsdp-2.0-oth-JPR

- Java Web Services Developer Pack 2.0 Patch 1
- JavaServer Web Development Kit 1.0.1
- JavaServer Web Development Kit 1.0



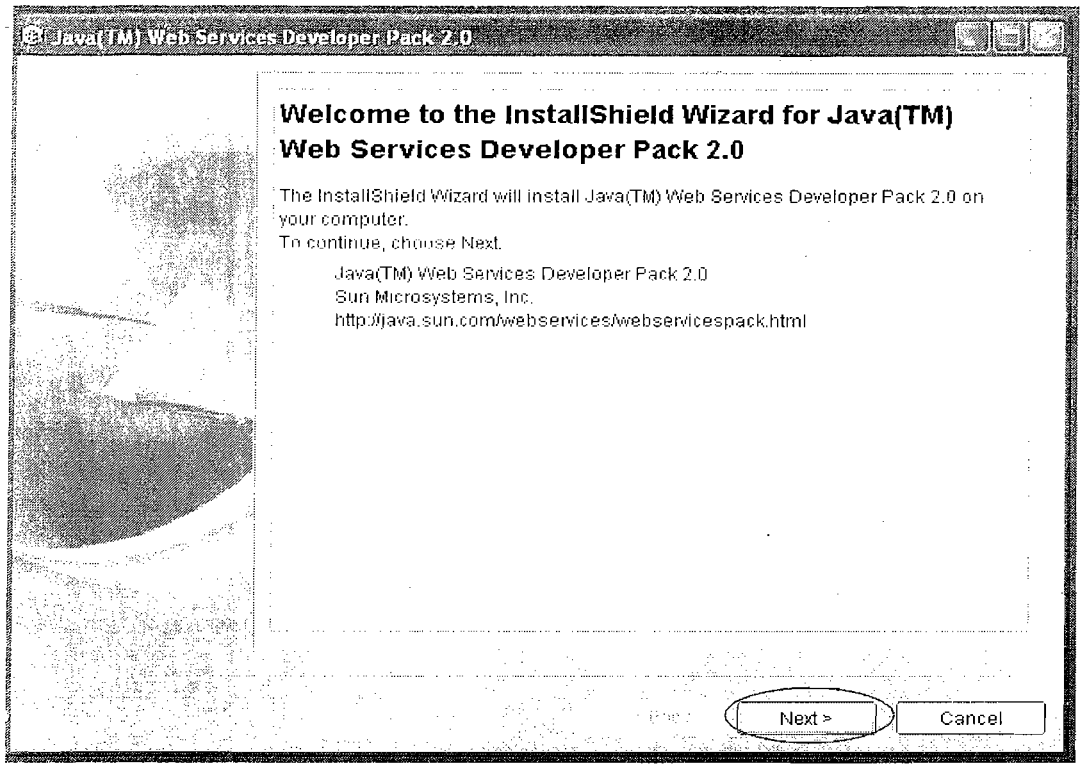
Step 2: The file `jwsdp-2_0-windows-i586.exe` is the Java WSDP installer. After downloading is finished double click on the installer icon and click on Run button to install.

Or

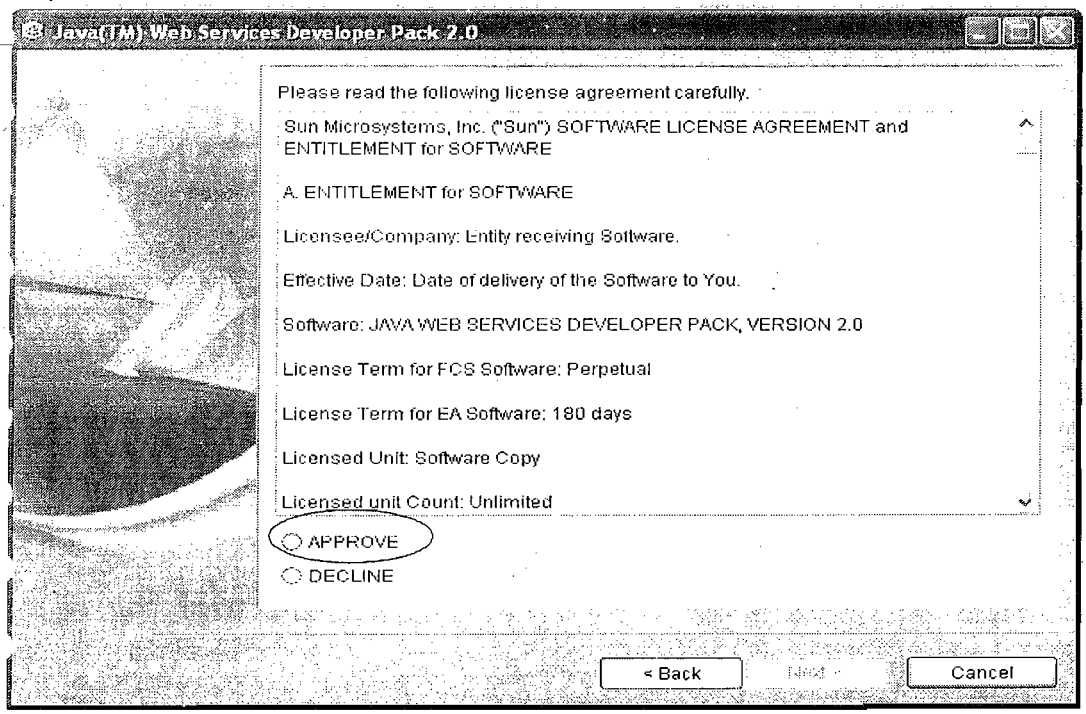
Install the Java WSDP from the command line using console option

```
C:\> jwsdp-2_0-windows-i586.exe -console
```

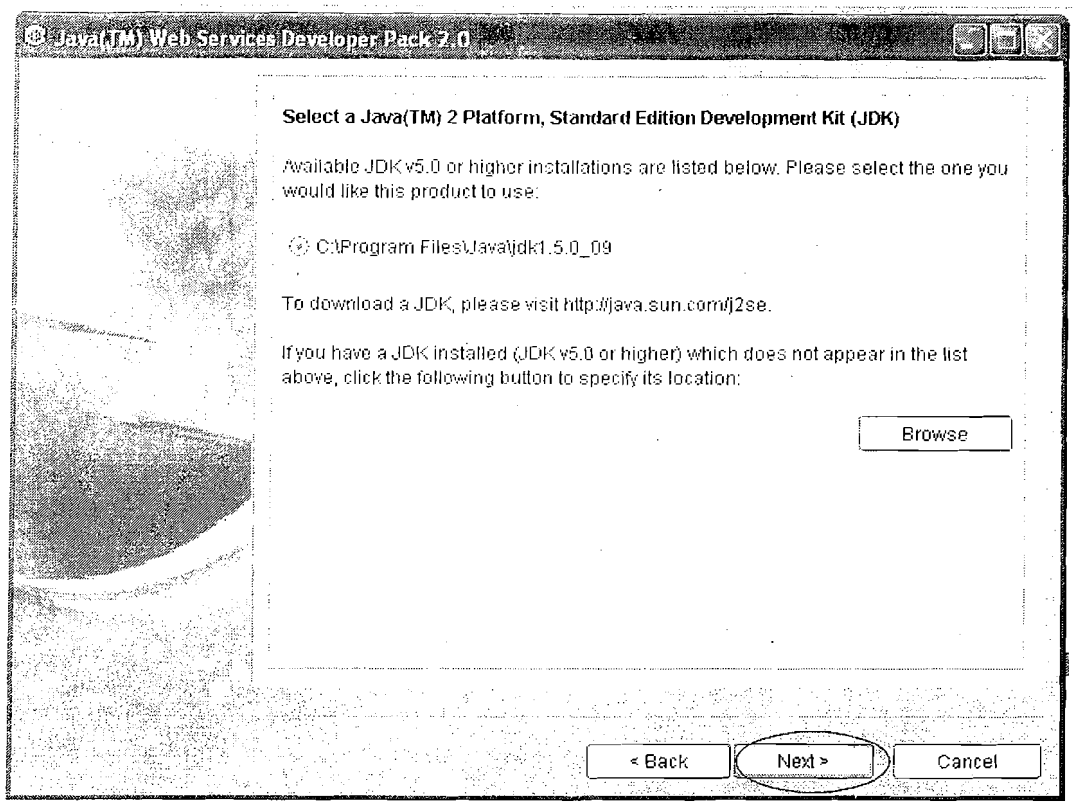
Click **Next** to continue. You should see



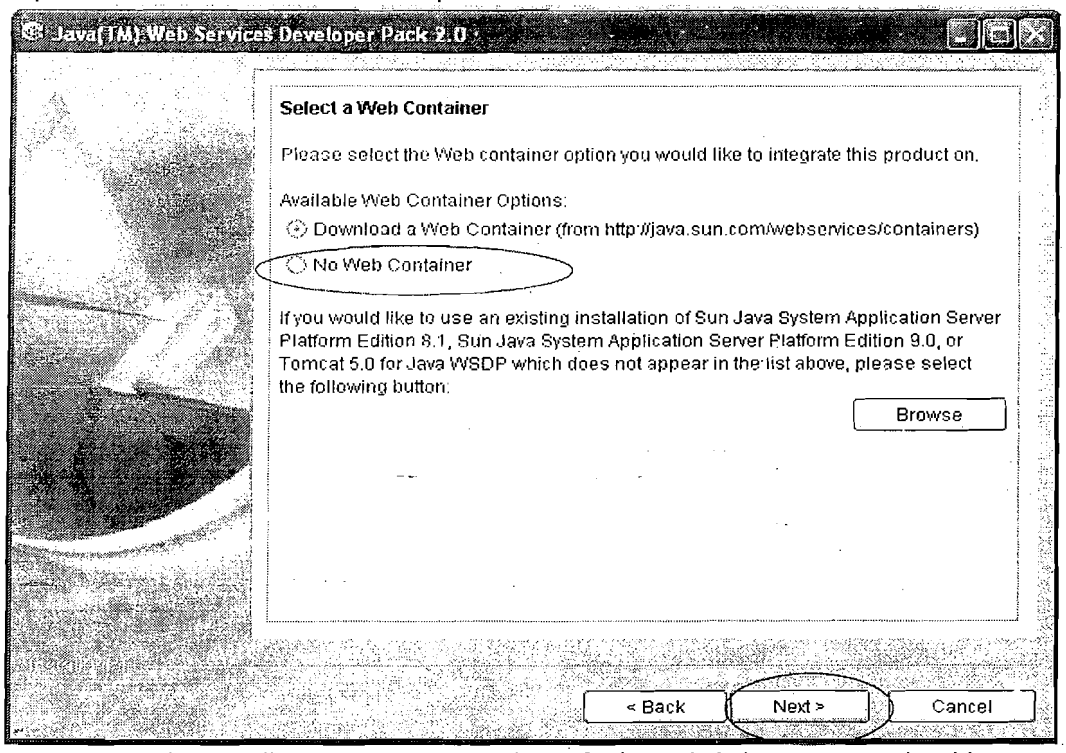
Step 3: Select APPROVE and click Next to see



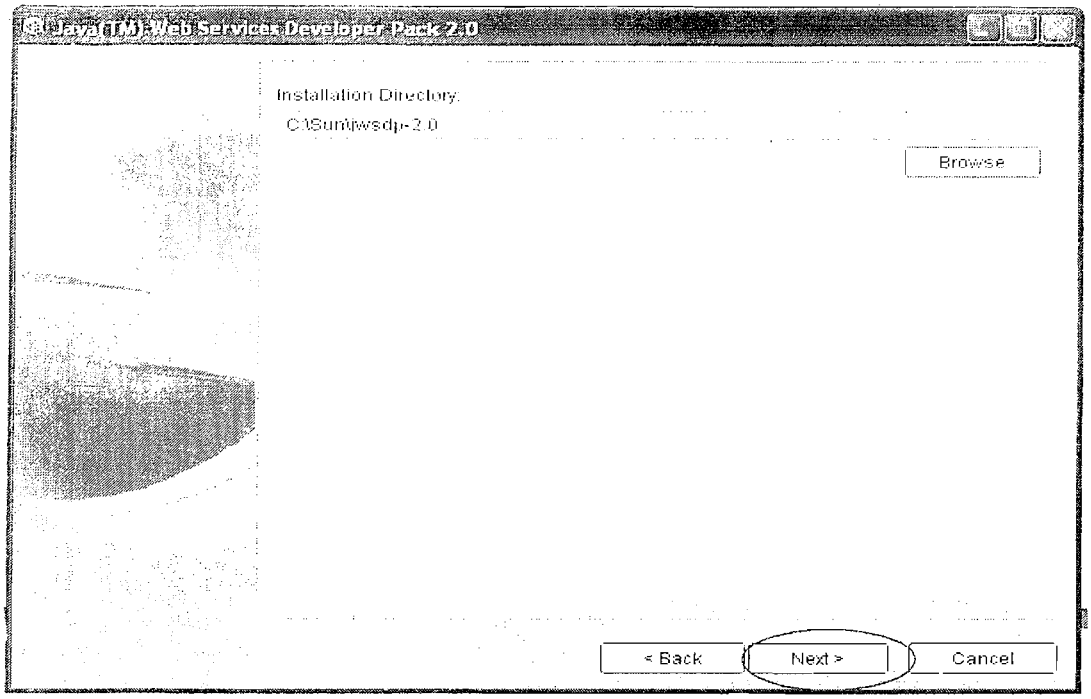
Step 4: Accept default values or Click Browser button to select JDK5.0 or higher version and click Next . you should see



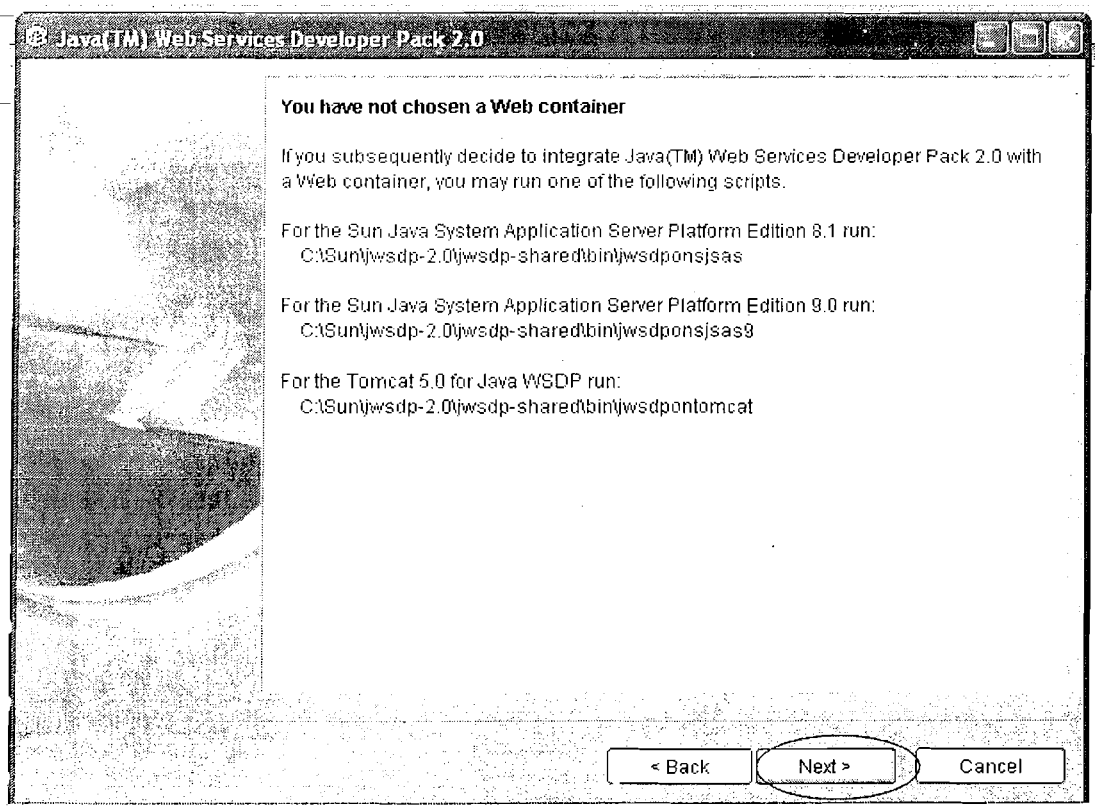
Step 5: Select "No Web Container" option and click Next button



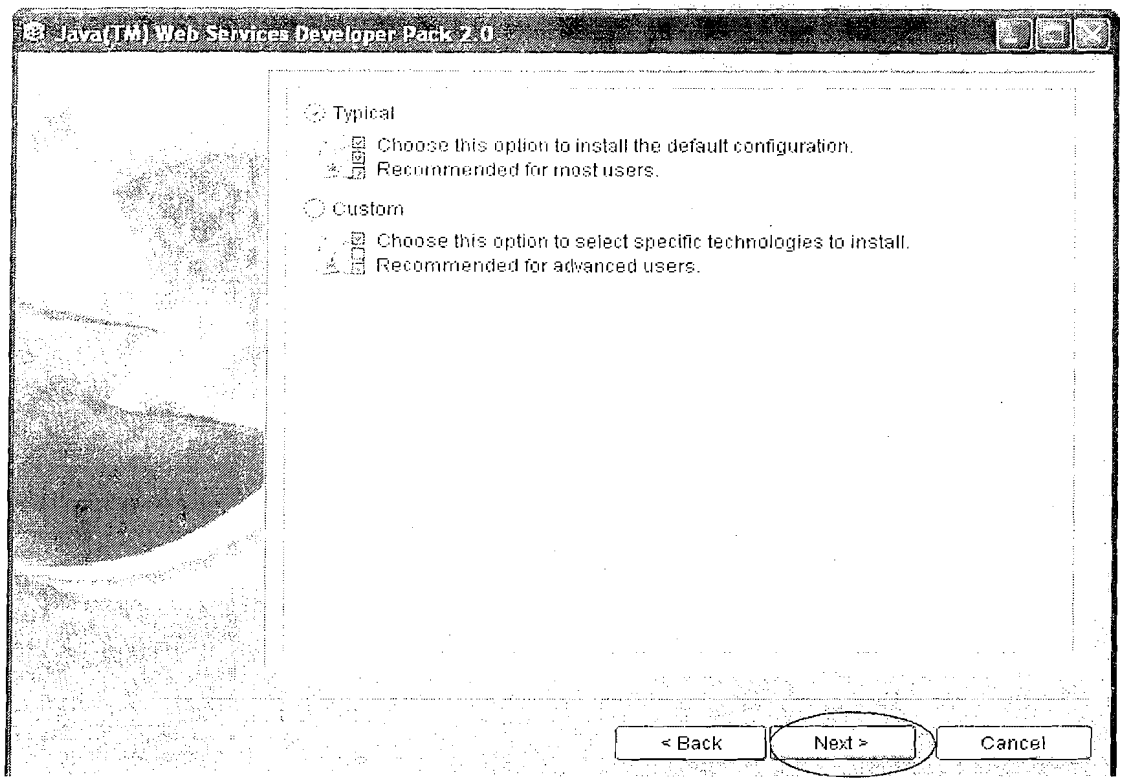
Step 6: Leave the Installation Directory on the default and click Next . You should see



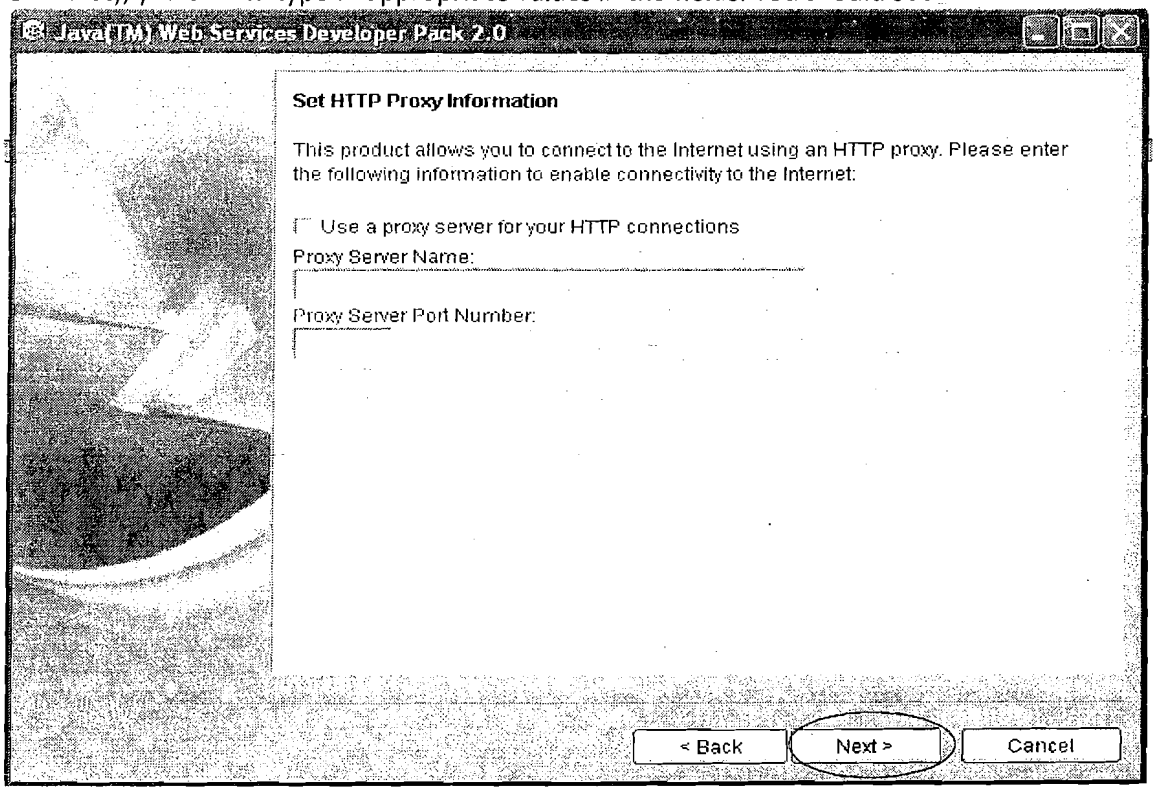
Step 7:Click **Next**. You should see



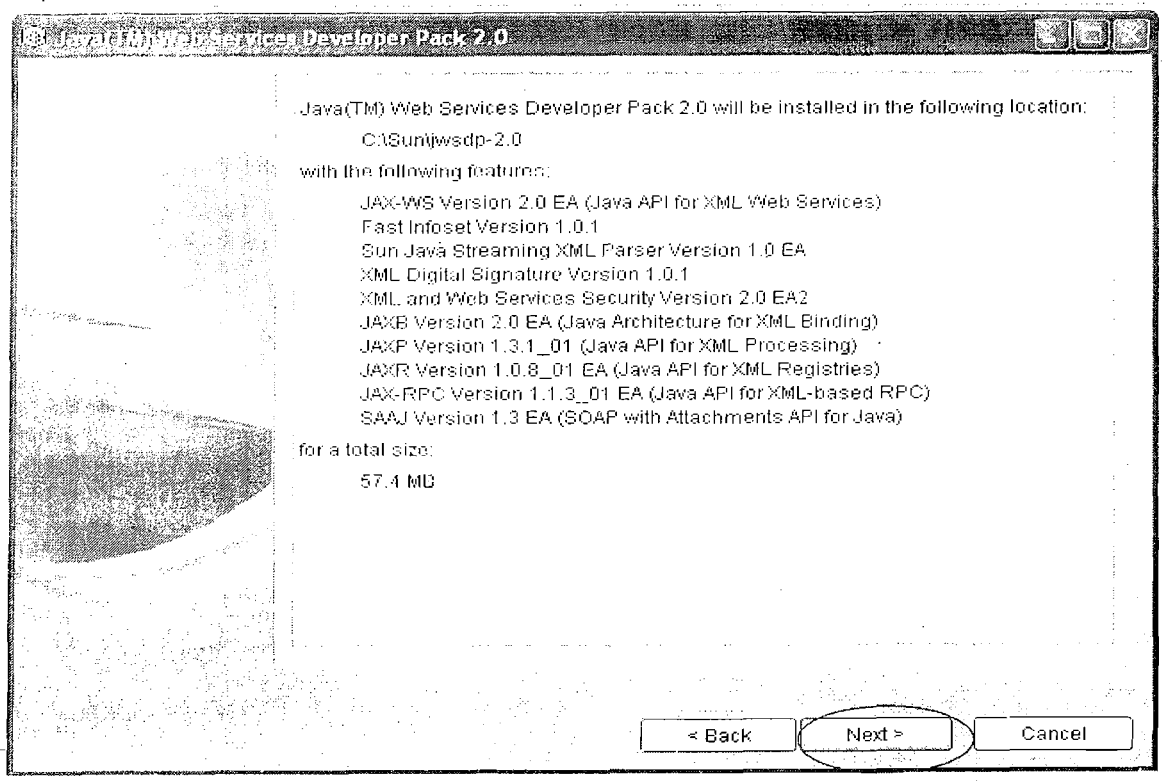
Step 8: Leave the Selection on Typical and click Next You should see



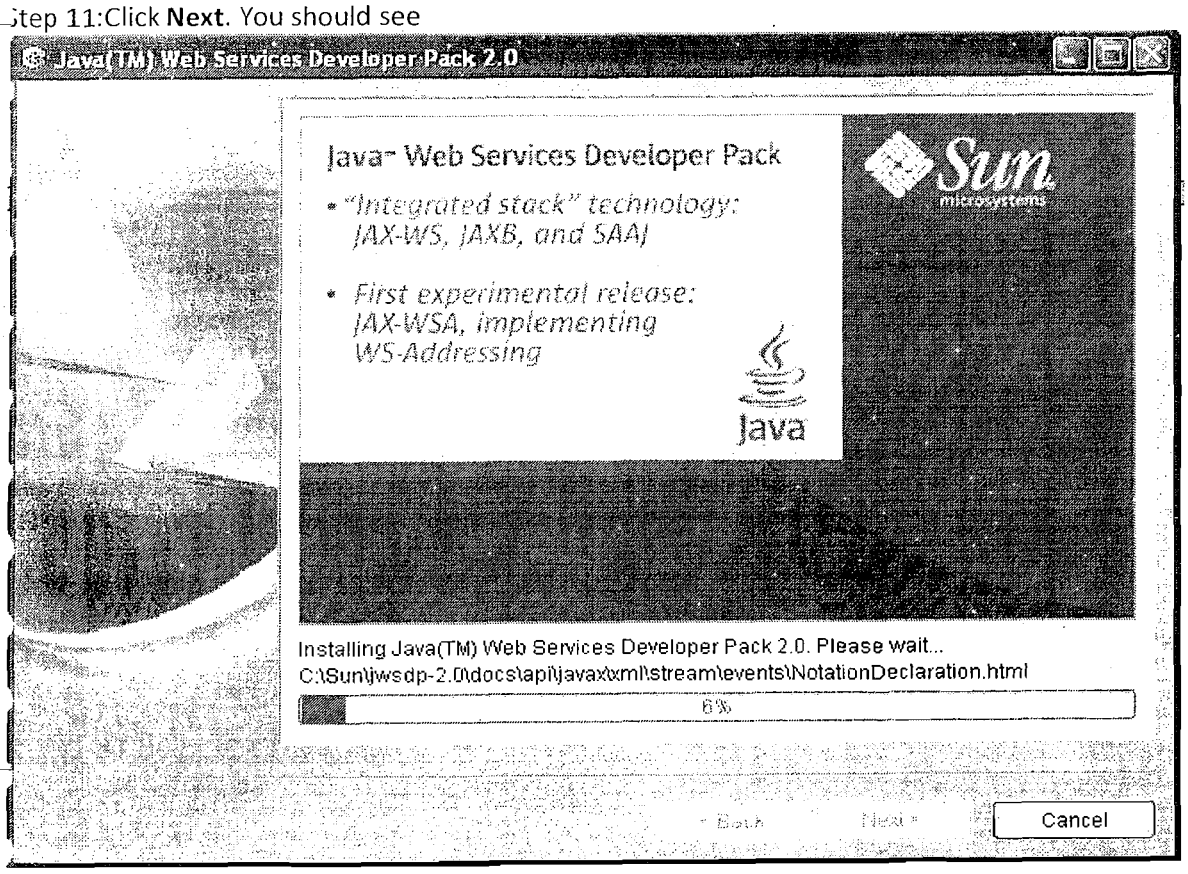
Step 9:Click **Next** to accept the defaults. If you use a proxy on your network to access this machine (not likely for localhost), you should type in appropriate values in the fields. You should see



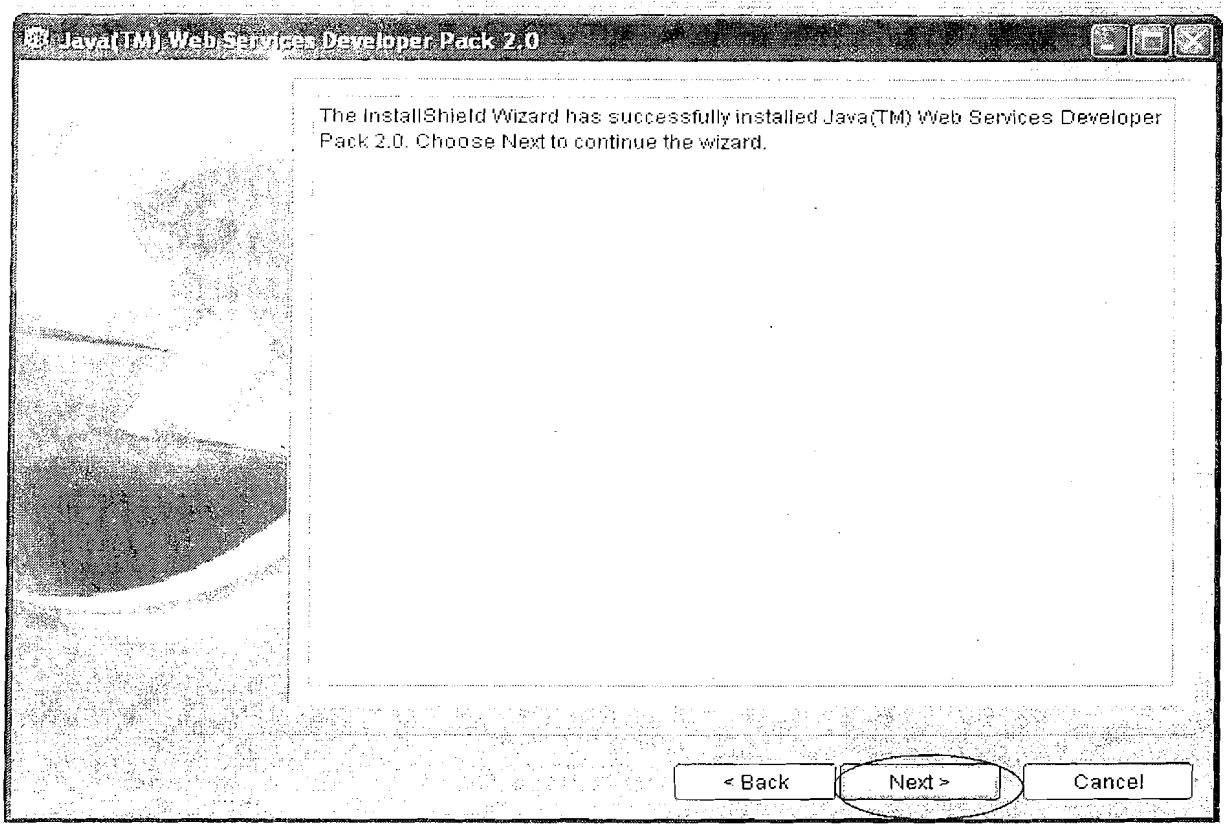
Step 10:Click **Next**. You should see



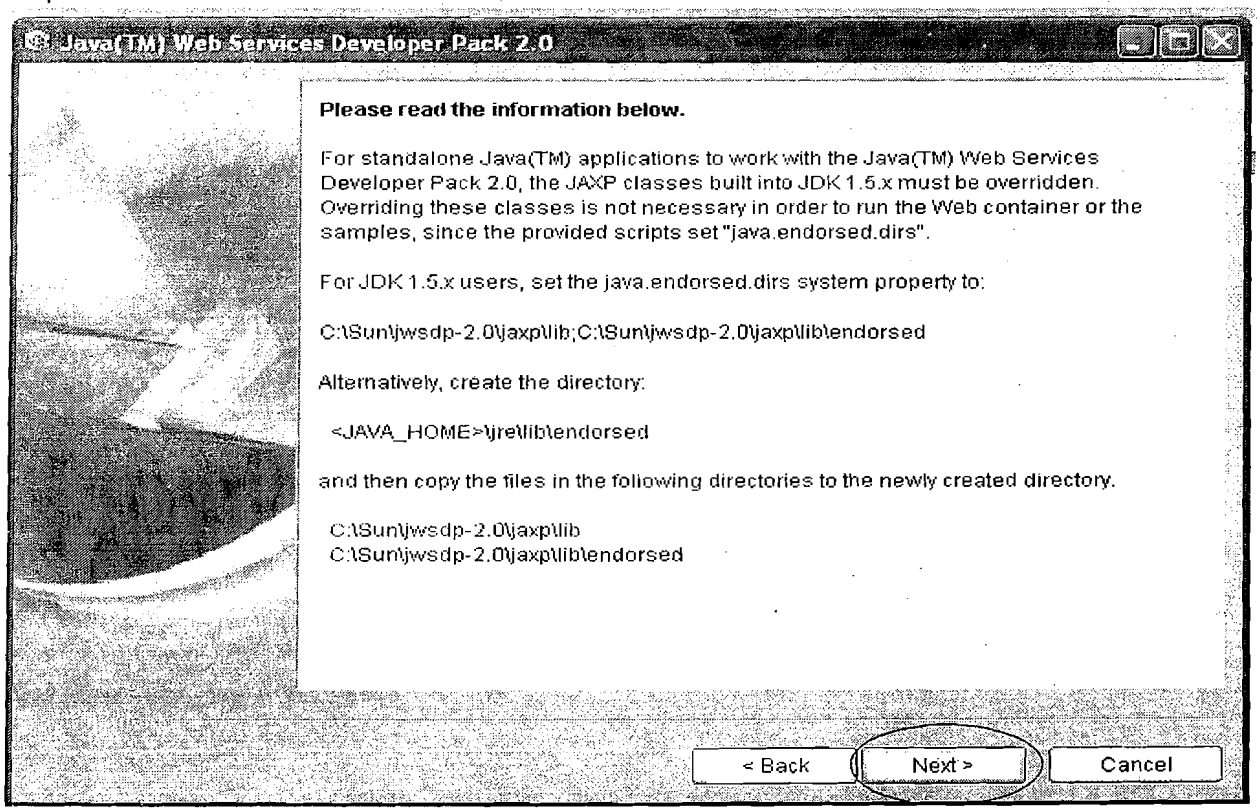
Step 11:Click **Next**. You should see



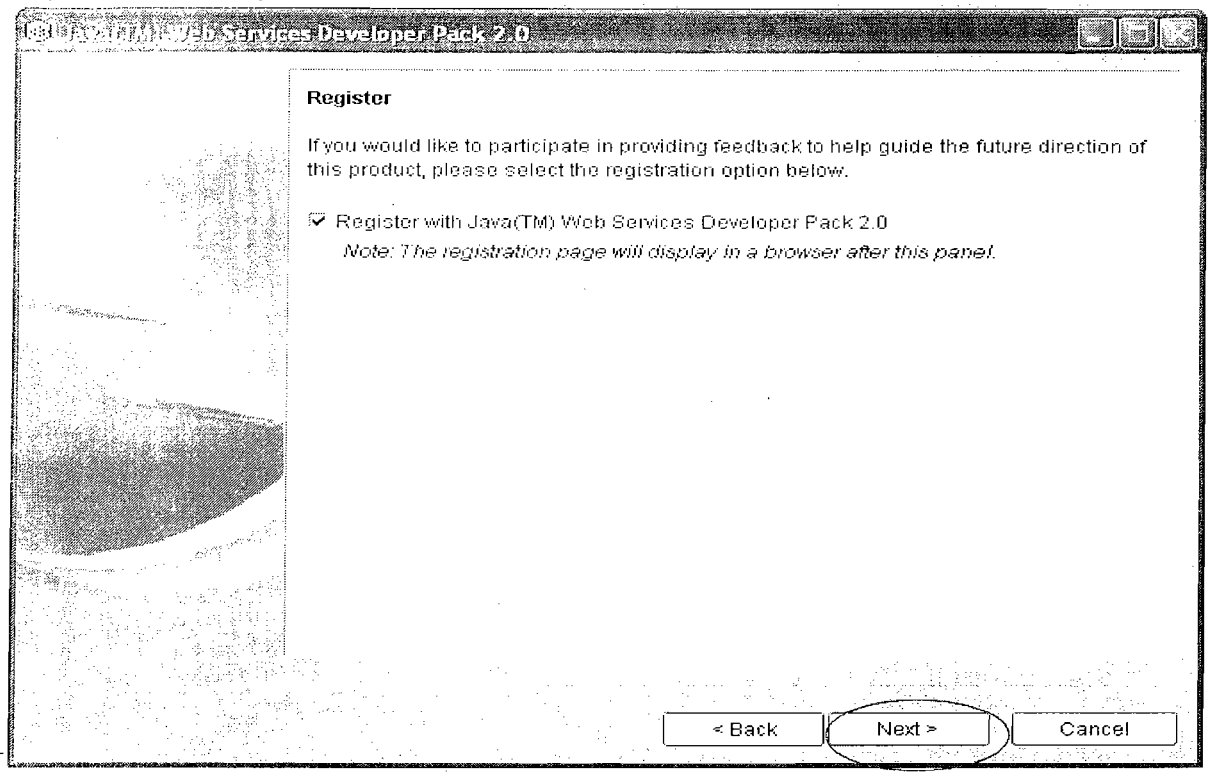
Step 12:Click **Next.** You should see



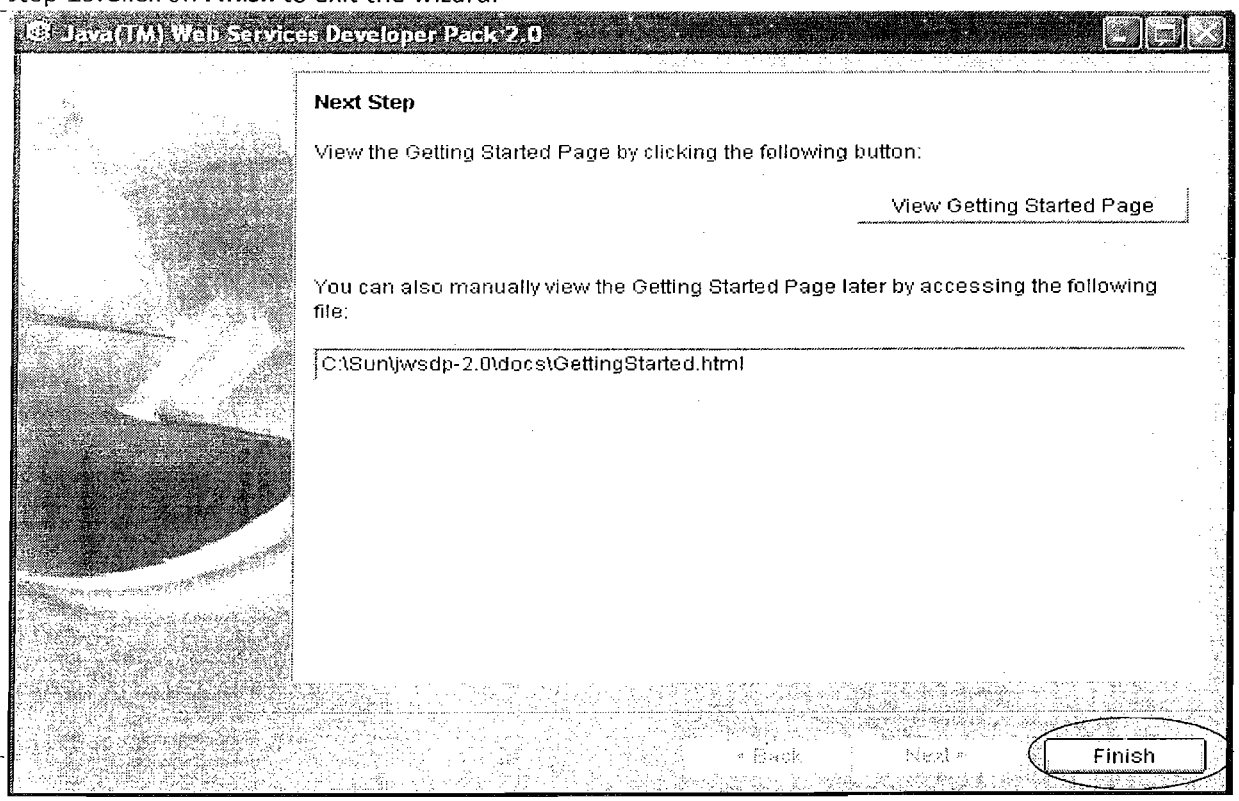
Step 13:Click **Next.** You should see



Step 14: Uncheck Register check box, click **Next**. You should see



Step 15:Click on **Finish** to exit the wizard.



You have now successfully installed JWSDP (despite the error message).

Step 16: Append the below path to "PATH" system variable. **"C:\Sun\jwsdp-2.0\jaxb\bin"**