UNIT-3

Lecture-19

XML

We have a requirement to save the data with some additional content which can describe the data so that we can further understand and use it and to meet this requirement we used to design our own encoding format and write logic of encoding and decoding the content as a part of our Application Development.

- This makes us to concentrate on low level logic and increases the development time and cost.
- To Solve this problem first IBM has introduced GML(Generalized Markup Language) where GML was used only for the IBM internal purpose i.e,IBM Projects
- A small advancement for GML was given by IBM in the form of SGML(Standard general markup language) but later on SGML is taken by w3C(world wide web consortium) where W3C is an open community
- At this point SGML was be more standardized and was declared standards for developing markup languages

Example:

- HTML is a markup language designed following SGML Standards
- But using SGML for developing a markup language was time taking and complex because of the number of features in it and flexibility
- To solve the above problems and make advantage of markup languages available to even small requirements w3c wants to simplify the standards.
- And as a part of this requirement XML was introduced.

XML:(eXtensible markup language)

- is a meta markup language i.e is a language used to develop some other markup languages
- is a subset of SGML added with some additional services to simplify the language development
- we can say that XML is a restricted form SGML.

Markup language:

are used to describe structured data,it is tag based language which can describe the content which it is enclosing XML-standards for developing markup language

XML Markup language:

A markup language developed according to the standards of XML

i.e following XML standards Example: XHTML,MathML,CML,VML,WML etc

XML Dcoument

is a document which is designed following XML and one of the XML markup language standards

To Develop a markup language we require to define the following things.

1. Declare all the elements of the language
    a. i.e tags(elements),attributes,entities.....
2. Define the grammar rules for elements declared
3. An application which can put the document in to action

To perform the first 2 operations we can use DTD or XML Schema which are part of XML Specification.
And to develop an XML Application we can use XML Parsers
which are even standardized under XML specification by W3c ...i.e parser specifications
where XML Application is an application using XML Document and can be developed using any programing language like JAVA,JAVASCRIPT,C,C++,C#.....

UNIT-3

Lecture-20

DTD:(Document Type Defination):

is used to declare the elements and give the type definition,where XML document can be designed based on the type defination given by DTD

Using DTD we can declare and define:

I. Elements
II. Attributes
III. Entities
IV. Notations

i)Element

Definition:

Elements are used to describe the content which it encloses

Types of Elements:

i)child only
ii)Text only
iii)Empty
iv)Mixed
v)ANY(is a special type)

i)Child only:

these type of elements consists of one or more elements as a contents

Syntax:

```
<!ELEMENT elemnet_name(list of child element names)>
```

Example:

```
<account>
<name>      </name>
<bal>       </bal>
</account>
<!ELEMENT account(name,bal)>
```

Example2:

```
<bank>
<account>  </account>
<account>  </account>
</bank>
<!ELEMENT bank(account*)>
```

occurence Specifiers

*	indicate 0 or More
+	indiactes 1 or More
?	indicates 0 or 1
No Symbol	----only for one time

Example:

```
<emps>
<emp>
<name>   </name>
<sal>   </sal>
</emp>

<emp>
<name>                    </name>
<wages>                   </wages>
</emp>
</emps>
<!ELEMENT emps(emp+)>
<!ELEMENT emp(name,(sal|wages))>
```

ii)Text only:

These type of elements can take only text as a content where char,string,int,float,double,boolean...are considered as a text. and are refered with a type PCDATA
PCDATA:Parsed character DATA

Syntax:
<!ELEMENT element_name(#PCDATA)>
Example:
<name>cmrcet</name>
<!ELEMENT name(#PCDATA)>
<sal>1000</sal>
<!ELEMENT sal(#PCDATA)>
PCDATA allows all the characters of our encoding format except markup char like <..

iii)Empty:
These type of elements does not takes any content
Syntax:
<!ELEMENT element_name EMPTY>
Example:
<br> </br>
or
<br/>
<!ELEMENT br EMPTY>
iv)Mixed:
These type of elements can contain child elements or text or child elemnets and text or even it can be empty
Syntax:
<!ELEMENT element_name(#PCDATA|list of child elements with | as a separator)*>
Example:
<p>Welcome,<b>to CMRCET</b> and <i>B.Tech(CSE)</i><br/>Hello
</p>
<!ELEMENT p(#PCDATA|b|i|br)*>
v)ANY
These type of elements can take any type of content i.e:text or can be empty or any element declared in the document
Syntax:
<!ELEMENT element_name ANY>
Example:
<!ELEMENT MyElement ANY>

The above declaration describes that element MyElement can hold text and even any element declared in the document and it can be empty also

2. Attributes:

Are used to give a extra meaning for the content described by element

- Attribute resides in the opening tag of the element
- One element can be declared with any number of attributes,where element name and each of these attributes are separated with space character.
- Each of the attribute consist of one name and value where these are separated with '=' character and value should be in quotes ' or "(Single quotes or double quotes)
- Attribute name cannot have a space character

Example:

<emp empno="e101">

Syntax to declare an attribute:

<!ATTLIST element_name attribute_name type specifier[defaultvalue]>

Types:

1. CDATA(character data):
   This type allows all the characters including numbers and space character
2. NMTOKEN:
   is same as CDATA but does not accept space character
3. NMTOKENS:
   it accepts one or more tokens(where one token is a sequence of characters without space character) and in this case space is taken as separator between tokens
4. ID:The value of ID type attribute should be unique
   it should not start with number but it contain number
5. IDREF:it allows one of the ID type attribute value
6. IDREFS:it can take one or more ID type attribute values where space is the separator
7. enum:in this case while declaring attribute we will specify the list of values and it allows to use any one of the specified value.
8. ENTITY:it allows one entity name where this entity should be umparsed entity
9. ENTITIES:allows one or more entity names where space is the separator

Example:

<!ATTLIST empno working(yes|no) 'yes'>

Specifiers:

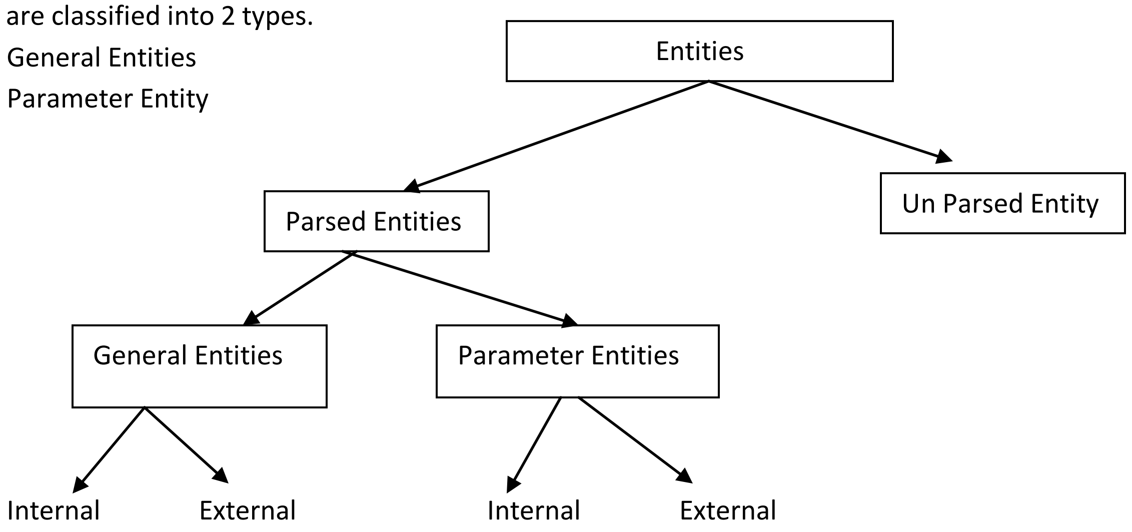
#REQUIRED --------------- Mandatory

#IMPLIED ---------------Optional

#FIXED ------------- -is Optional and even if it is used it has to be given with the value which is specified while declaring the attribute(i.e its value will be fixed)same as final in java

3)Entity:
is reference to some content.i.e is used to represent some reusable content.we have a requirement where some content is required to be used for more number of times within the XML documents and even in some cases we have content being repeated in DTD document also based on this requirement Entities are classified into 2 types.

1. General Entities
2. Parameter Entity



General Entities:
Are declared in DTD and used in XML documents
Internal Entity:
In this casethe content which has to be replaced where ever the entity is refered,will be placed in the declaration of the entity directly i.e in DTD document itself.
Syntax:

```
<!ENTITY entity_name "content which had to replaced">
To use the entity
this can done in XML document
&entity_name
```

Example:
```
<!ENTITY copyrights "copyrights Myshop 2013-2014">
```
External Entity:
Here the content which has to be replaced will be placed in separate file and in the declaration of the entity insted of specifying the content we will provide the filename with its path.
Syntax:
```
<!ENTITY enitity_name SYSTEM "filename with path">
```

Example:
```
<!ENTITY mylogo SYSTEM "shoplogo.gif">
```
Parameter Entity:
These entities are declared and used in DTD itself

Internal entity:

Syntax:

<!ENTITY % entity_name "content">
to use (i.e in DTD it self)
%entity_name

External Entities:

Syntax:
<!ENTITY % entity_name SYSTEM "filename">
to use
%entity_name
Example:

<!ENTITY % text "#PCDATA">
<!ELEMENT name(%text;)>

Unparsed Entities:

To refer some content which is of different encoding format we have to go for unparsed entities

i.e like to refer gif,.bmp,.jpg......files

Syntax:

<!ENTITY entity_name SYSTEM "filenamewith path" NDATA notation_name>

Notations:

These are used to refer some content which provides some additional description like MIME/Contenttype ....

Syntax:

<!NOTATION notation_name "content">

Example:

<!NOTATION gif "image|gif">

<!ENTITY mylogo SYSTEM "shoplogo.gif" NDATA gif>

<!ATTLIST shop logo ENTITY #REQUIED>

Example:

```
<emps logo="mylogo">
```

To associate the definitions to the XML document i.e the definitions which are given following DTD Standards. We use DOCTYPE element

There are 2 Types of DTD

1. Internal DTD
2. External DTD

Internal DTD:

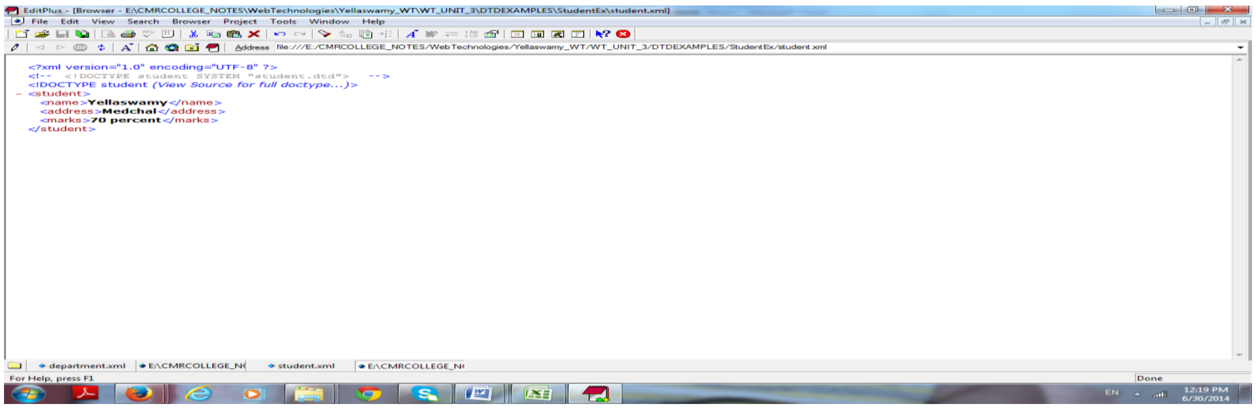
Here the DTD Content is placed inside the XML document.

```
<!DOCTYPE  root_element_name [DTD code]>
```

Example:Student.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<!-- <!DOCTYPE student SYSTEM "student.dtd"> -->
<!DOCTYPE student [<!ELEMENT student (name,address,marks)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT address (#PCDATA)>
<!ELEMENT marks (#PCDATA)>]>
<student>
<name>Yellaswamy</name>
<address>Medchal</address>
<marks>70 percent</marks>
</student>
```

Output:


EditPlus - [Browser - E:\CMRCOLLEGE_NOTES\WebTechnologies\Yellaswamy_WT\WT_UNIT_3\DTDEXAMPLES\StudentEx\student.xml]
File Edit View Search Browser Project Tools Window Help
Address file:///E:/CMRCOLLEGE_NOTES/WebTechnologies/Yellaswamy_WT/WT_UNIT_3/DTDEXAMPLES/StudentEx/student.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<!--  <!DOCTYPE student SYSTEM "student.dtd">   -->
<!DOCTYPE student (View Source for full doctype...)>
- <student>
    <name>Yellaswamy</name>
    <address>Medchal</address>
    <marks>70 percent</marks>
  </student>
```

department.xml
E:\CMRCOLLEGE_N
student.xml
E:\CMRCOLLEGE_N
For Help, press F1
Done
EN
12:19 PM
6/30/2014

External DTD:

Here the DTD code is written in to a separate file and referred by the XML document.

<!DOCTYPE root_element_name SYSTEM “dtd file name with path”>

<!DOCTYPE root_element_name PUBLIC “fpi string” “dtd file url”>

FPI (Formal public identifier):

This String gives some information about the vendor and dtd which we are referring

This String is divided into 4 parts and these parts are separated with //

1. Part takes + or –
2. Part takes the company name or the person name who developed the DTD and responsible for the DTD
3. Part the purpose and version of DTD
4. Part the 2 letter language code(i.e the codes given under the ISO standards)
   Example:
   EN for English

Example:

-//CMRCET//Examples DTD 1.0//EN

Example for External DTD

Department.dtd

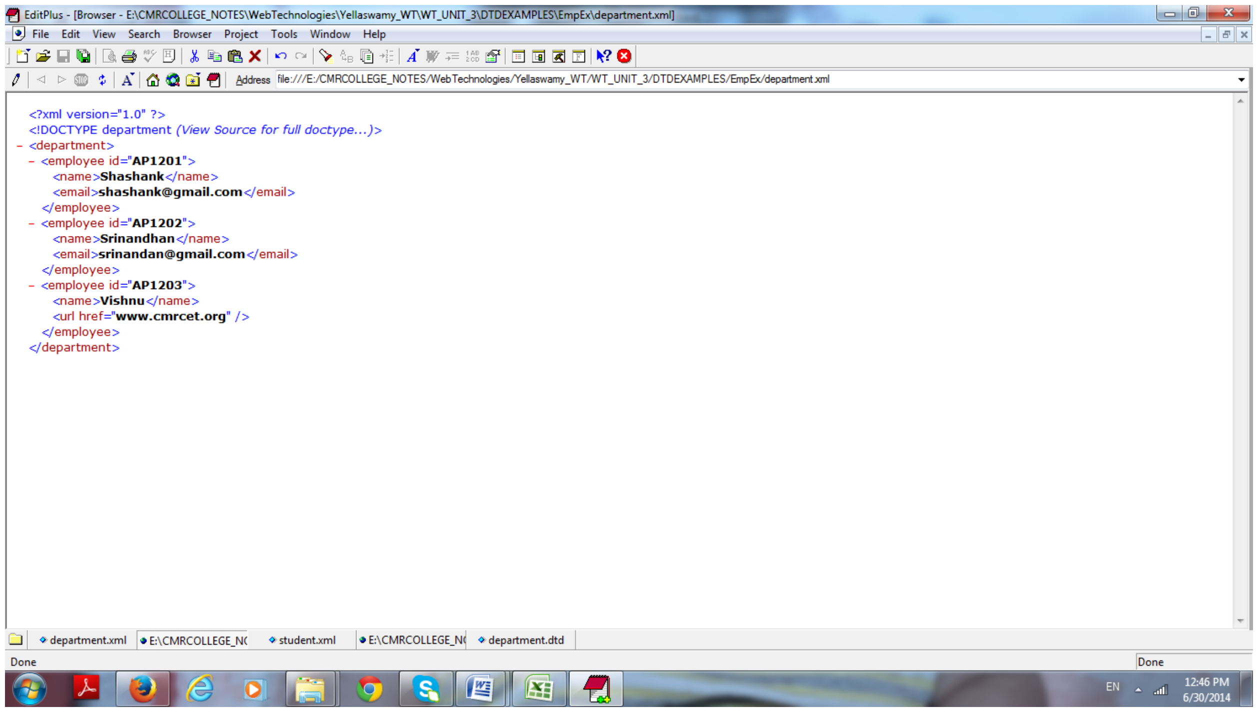
```
<!ELEMENT department (employee)*>
<!ELEMENT employee (name, (email | url))>
<!ATTLIST employee id CDATA #REQUIRED>
<!ELEMENT name (#PCDATA)>
<!ELEMENT email (#PCDATA)>
<!ELEMENT url EMPTY>
<!ATTLIST url href CDATA #REQUIRED>
```

Department.xml

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

```
 <employee id="AP1201">
  <name>Shashank</name>
  <email>shashank@gmail.com</email>
 </employee>

 <employee id="AP1202">
  <name>Srinandhan</name>
  <email>srinandan@gmail.com</email>
 </employee>

 <employee id="AP1203">
  <name>Vishnu</name>
  <url href="www.cmrcet.org"/>
 </employee>
</department>
```

OUTPUT:



```
<?xml version="1.0" ?>
<!DOCTYPE department (View Source for full doctype...)>
- <department>
 - <employee id="AP1201">
     <name>Shashank</name>
     <email>shashank@gmail.com</email>
   </employee>
 - <employee id="AP1202">
     <name>Srinandhan</name>
     <email>srinandan@gmail.com</email>
   </employee>
 - <employee id="AP1203">
     <name>Vishnu</name>
     <url href="www.cmrcet.org" />
   </employee>
  </department>
```

EditPlus - [Browser - E:\CMRCOLLEGE_NOTES\WebTechnologies\Yellaswamy_WT\WT_UNIT_3\DTDEXAMPLES\EmpEx\department.xml]
File Edit View Search Browser Project Tools Window Help
Address file:///E:/CMRCOLLEGE_NOTES/WebTechnologies/Yellaswamy_WT/WT_UNIT_3/DTDEXAMPLES/EmpEx/department.xml
department.xml
E:\CMRCOLLEGE_N(
student.xml
E:\CMRCOLLEGE_N(
department.dtd
Done
Done
EN
12:46 PM
6/30/2014

Combination of Internal and External:

<!DOCTYPE root_element_name SYSTEM “external dtd file path” [internal DTD code]>

XML Document Structure

<?	?>	Processing Instruction Tag
<!	>	Instruction Tag
<!--	-->	Comment Instruction Tag
<	>	opening tag
</	>	closing Tag
<	/>	Self ending tag

Structure of XML

```
<?xml version=”1.0” encoding=”UTF-8” standalone=”no”?>
<!DOCTYPE …….>
<root_element_name>
</root_element_name>
```

- XML processing instruction tags is optional but recommended to be used
- If used should be the first element (even comment is not allowed before this)
- If used **version** attribute is mandatory this takes the XML version where the current version is 1.0 **encoding** is optional if not given takes the system encoding format **standalone** is also optional takes yes or No if not given by default it takes No.This attribute indicates whether this document depends on any external resources or not(if it is depending it should be given as ‘no’ if not ‘yes’

UNIT-3

Lecture-21

XML Schema:

Is used to declare the elements of the Markup Language and Grammar rules i.e an alternative to DTD

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

An XML Schema:

- Defines elements that can appear in a document
- Defines attributes that can appear in a document
- Defines which elements are child elements
- Defines the number of child elements
- Defines whether an element is empty or can include text
- Defines data types for elements and attributes
- Defines default and fixed values for elements and attributes

Differences Between DTD and XML Schema

- DTD uses a small language to define the rules where as xml schema is xml document.XML schema documents are more descriptive than compared to DTD
- With DTD &XML Schemas we have a provision to declare complex types but with DTD the type name and the element name should be same which is not required in XML Schema
- With DTD we don't have a support to specify a particular occurrence for a element i.e MIN and MAX occurrence(We were allowed to given MIN as 0 or 1 and MAX 1 or more) where as with XML Schema we can specify the required Max and Min occurrences.
- DTD doesn't supports all the common types(i.e it considers numbers.. all as text #PCDATA) where as with XML Schema we can specific type like String,char,number,double,float,Boolean
- XML schema supports NameSpace.Since XML Schema document is also an XML document it can be generated/written using any tool which supports

XML Schemas are the Successors of DTDs

We think that vey soon XML Schemas will be used in most Web Applications as a replacement for DTDs.

Here are Some reasons:

- XML Schemas are extensible to future additions
- XML Schemas are richer and more useful than DTDs
- XML Schemas are written in XML
- XML Schemas support dat types
- XML Schemas support namespaces

XML Schema has support for Data Types

One of the greatest strengths of XML Schema is the Support for data types

With the support for data types:

- It is easier to describe permissible document content
- It is easier to validate the correctness of data
- It is easier to work with data from a database
- It is easier to define facets(restrictions on data)
- It is easier to define data patterns
- It is easier to convert data between different data types

XML Schemas Secure Data Communication:

When data is sent from sender to a receiver it is essential that both parts have the same "expectations" about the content.

With XML Schemas,the sender can describe the data in way that the receiver will understand.

Well-Formed is not enough

A well-formed XML document is a document that conforms to the XML syntax rules:

- Must begin with the XML declaration
- Must have one unique root element
- All start tags must match end tags
- XML tags are case sensitive
- All elements must be closed
- All elements must be properly nested
- All attribute values must be quoted
- XML entities must be used for special characters

Even if documents are well-Formed they can still contain errors and those errors can have serious consequences. Think of this situation: you order 5 gross of laser printers, instead of 5 laser printers. With XML Schema most of these errors can be caught by your validating software.

A simple XML Document

“note.xml”

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

A simple DTD
This simple DTD file called “**Note.dtd”** that defines the elements of the XML document above(“note.xml”)

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

The <schema> Element:
The <schema> Element is the root element of every XML Schema

Syntax:
```
<?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://www.w3schools.com" xmlns="http://www.w3schools.com"
elementFormDefault="qualified">
--
---

</xs:schema>
```

A simple XML Schema

This simple XML Schema file called “Note.xsd” that defines the elements of the XML document above(“note.xml”)

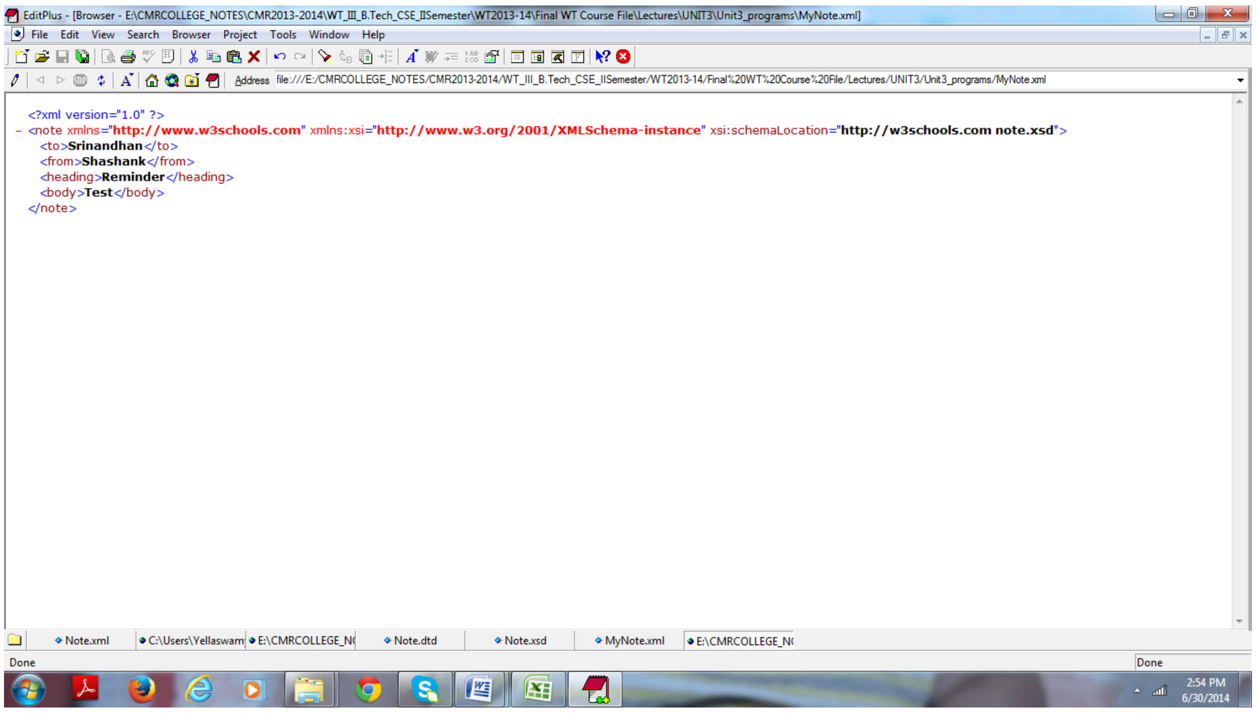
“Note.xsd”

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.w3schools.com" xmlns="http://www.w3schools.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>
```

A reference to an XML Schema:

```
          <?xml version="1.0"?>
          <note xmlns="http://www.w3schools.com" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
          instance" xsi:schemaLocation="http://w3schools.com note.xsd">
          <to>
          Srinandhan
          </to>
          <from>Shashank</from>
          <heading>Reminder</heading>
          <body>Test</body>
</note>
```

Output:



EditPlus - [Browser - E:\CMRCOLLEGE_NOTES\CMR2013-2014\WT_III_B.Tech_CSE_IISemester\WT2013-14\Final WT Course File\Lectures\UNIT3\Unit3_programs\MyNote.xml]

Address file:///E:/CMRCOLLEGE_NOTES/CMR2013-2014/WT_III_B.Tech_CSE_IISemester/WT2013-14/Final%20WT%20Course%20File/Lectures/UNIT3/Unit3_programs/MyNote.xml

```
  <?xml version="1.0" ?>
- <note xmlns="http://www.w3schools.com" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://w3schools.com note.xsd">
    <to>Srinandhan</to>
    <from>Shashank</from>
    <heading>Reminder</heading>
    <body>Test</body>
  </note>
```

Namespace:

Namespace is used to make the element/attribute unique

i.e this is most required when multiple markup language elements are used in one document in such a case if the element names are same from both the markup languages then a small prefix can represent a element uniquely describing that the element is of a particular markup language.

Types of Namespaces

1. General Namespace
2. Default Namespace

Declaring a Namespace:

Namespace is declared as an attribute in the element.

Where the value will be the unique URI given by the markup language provider.

Namespace declared has a scope within that element including that element i.e the namespace declared can be used for that element and its child and its Childs.

But not applicable for its parent or siblings

To declare General namespace:

xmlns:<namespace_name>="<namespace uri>"

Where

<namespace_name> can be any name without special characters and space this is used as a prefix for the elements/attributes.

<Namespace uri>---is given by the markup language provider whose elements we wanted to refer.

To Declare Default Namespace:

xmlns="<namespace uri>"

in this case we dont have any prefix and if default namespace is declared then all the unqualified elements (i.e. the elements without any prefix) within the scope will be considered under the default namespace.

UNIT-3

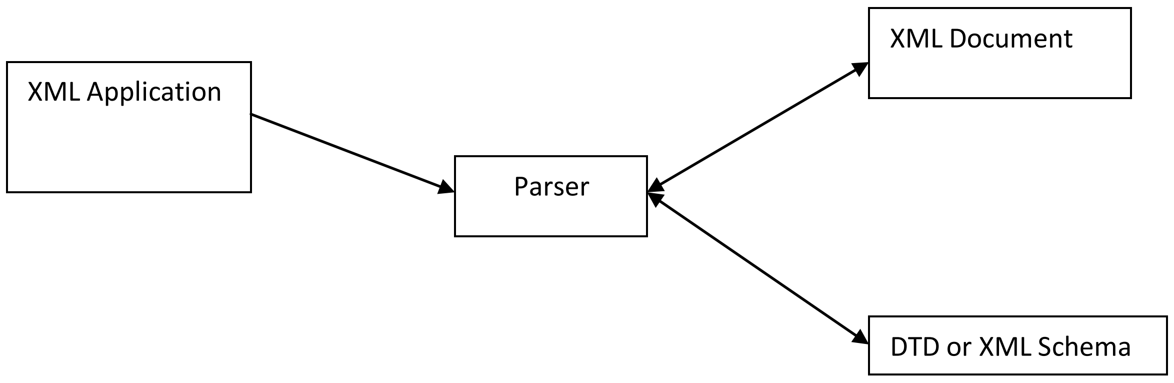
Lecture-22

XML Parsers:

Parser is a standard abstraction between the xml application and xml document.

The responsibility of parser is:

- should read the given xml document
    - if available read the schema definition(i.e DTD or XML Schema)
- check the XML document (i.e validation)
- Makes the XML Content available to the application



Types of Parsers:

1. Based on the type definition syntax it understands
    a. DTD Validator
    b. XML Schema Validator
2. Based on the Validation
    a. Non Validating Parser
    b. validating parser
3. Based on the approach of making the data available to the application
    a. Tree based approach(object based)
    b. Event based approach

Non Validating Parsers:

These type of parsers checks only wellformness of the Document.

where if xml document follows the following rules then it is said to be well formed document.

Rules

1. only one root element is allowed.
2. every opening tag should have a closing tag.
3. All the elements should be properly arranged in tree structure i.e, first child tag has to be closed and then parent tag.
4. All the attributes values should be in quotes
   i.e single or double quotes
5. All the entities used in the document should be declared.

Validating Parsers:

These parsers checks for wellformness and then if it is well formed then it checks the xml document following the grammar rules given under the DTD or XML Schema.

JAXP (Java API for XML Processing)

JAXP API

The Main JAXP API are defined in the javax.xml.parsers package This package contain vendor neutral factory classes

- SAXParserFactory
- DocumentBuilderFactory
- TransformerFactory

javax.xml.parsers:

The JAXP API,which provides a common interface for different vendors SAX and DOM Parsers

org.w3c.dom:

Defines the Document class as well as classes for all the components of a DOM

org.xml.sax:

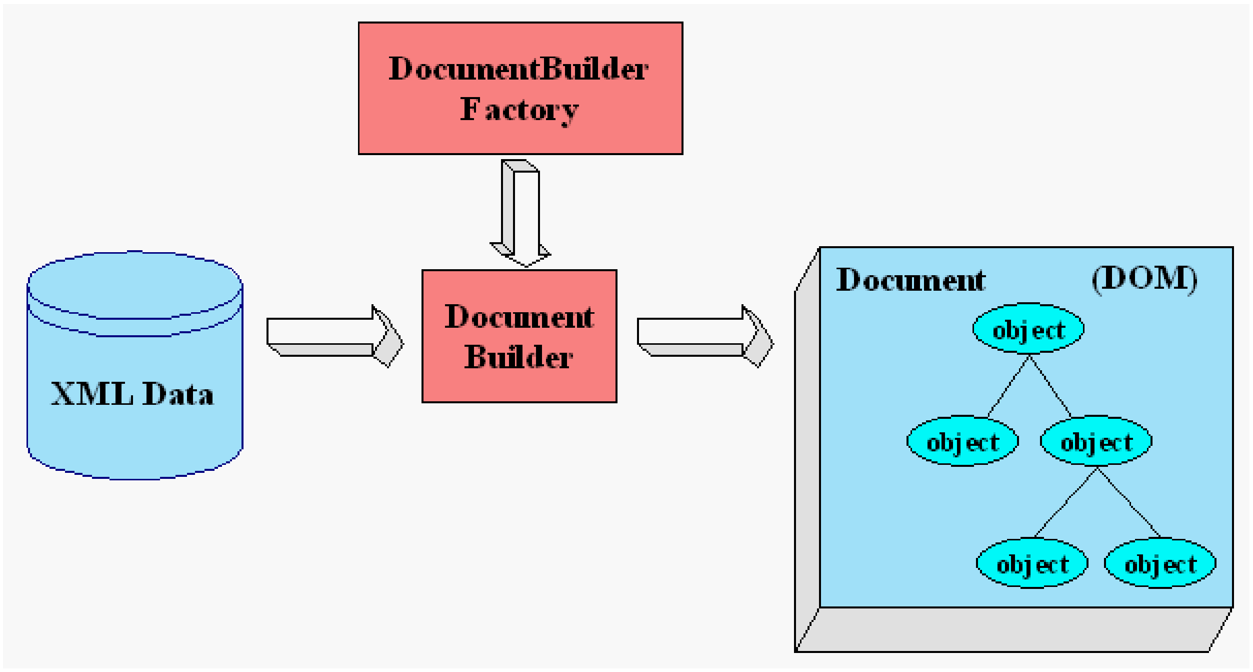
Defines the basic SAX API

javax.xml.transform:

Defines the XSLT API that let you transform XML into other forms

DOM (Document Object Model)

- Is a Specification for w3c
- is a validating parser
- DOM is a DTD Validator and DOM Level 3 parser supports XML Schema also
- is a tree based i.e it follows tree based approach(Makes the complete object tree available to the application)
- for each part of the xml document it prepares an object and construct an object tree representing the xml document and org.w3c.dom.Node is the super most type for all the types in DOM Specification
- These Specifications are implemented by 3rd party vendors



You use the `javax.xml.parsers.`DocumentBuilderFactory class to get a DocumentBuilder instance, and use that to produce a Document (a DOM) that conforms to the DOM specification. The builder you get, in fact, is determined by the System property, `javax.xml.parsers.DocumentBuilderFactory`, which selects the factory implementation that is used to produce the builder. (The platform's default value can be overridden from the command line.)

You can also use the DocumentBuilder `newDocument()` method to create an empty Document that implements the
org.w3c.dom.Document interface. Alternatively, you can use one of the builder's parse methods to create a Document
from existing XML data. The result is a DOM tree like that shown in the diagram.

Example:

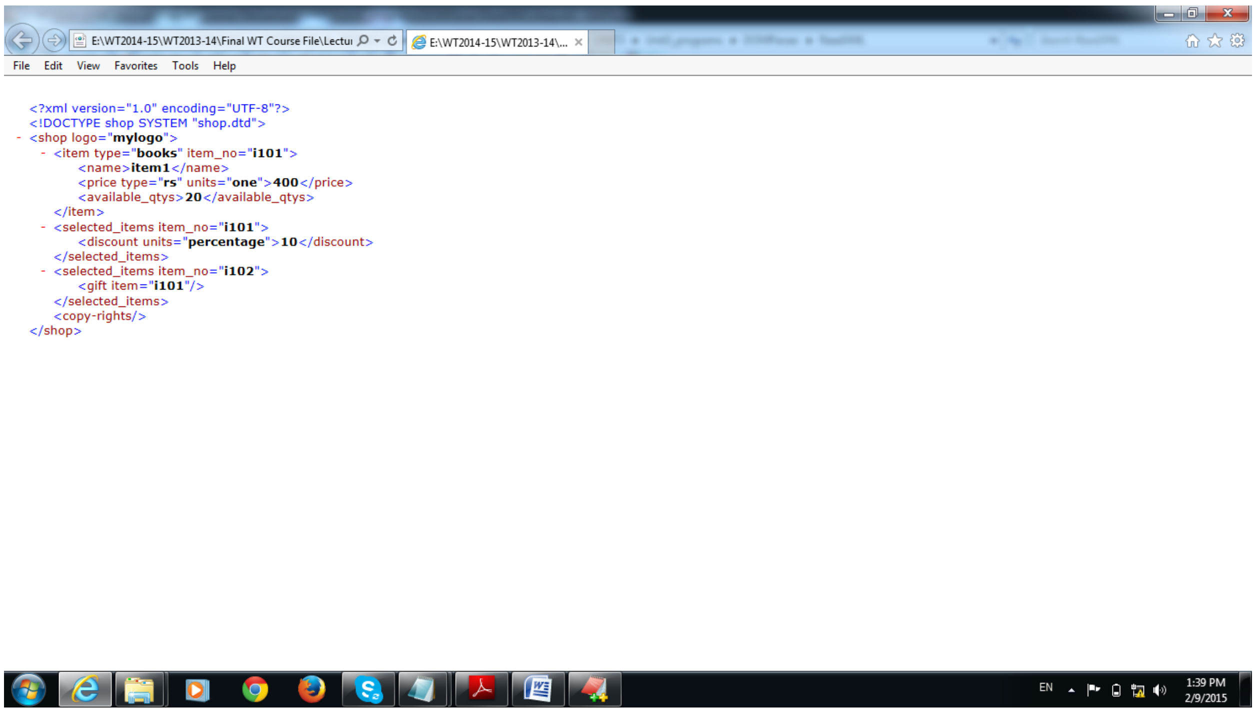
Shop.dtd

```
<!ENTITY copyrights "copyrights Myshop 2012-2013">
<!NOTATION gif SYSTEM "image|gif">
<!ENTITY mylogo SYSTEM "shoplogo.gif" NDATA gif>
<!ELEMENT shop (item+,selected_items*,copy-rights)>
<!ELEMENT item (name,price,available_qtys)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT price (#PCDATA)>
<!ELEMENT available_qtys (#PCDATA)>
<!ELEMENT selected_items (discount?,gift*)>
<!ELEMENT discount (#PCDATA)>
<!ELEMENT gift EMPTY>
<!ELEMENT copy-rights (#PCDATA)>
<!ATTLIST shop logo ENTITY #IMPLIED>
<!ATTLIST item item_no ID #REQUIRED>
<!ATTLIST item type CDATA #IMPLIED>
<!ATTLIST discount units CDATA #REQUIRED>
<!ATTLIST price units (one|kg|meter) 'one' type NMTOKEN  #IMPLIED>
<!ATTLIST selected_items item_no IDREFS #REQUIRED>
<!ATTLIST gift item IDREF #REQUIRED>
```

Shop.xml

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE shop SYSTEM "shop.dtd">

<shop logo="mylogo">
<item item_no="i101" type="books">
<name>item1</name>
<price units="one" type="rs">400</price>
<available_qtys>20</available_qtys>
</item>
<selected_items item_no="i101">
<discount units="percentage">10</discount>
</selected_items>
<selected_items item_no="i102">
<gift item="i101"/>
</selected_items>
<copy-rights>&copyrights;</copy-rights>
</shop>
```

Output:


E:\WT2014-15\WT2013-14\Final WT Course File\Lectu
E:\WT2014-15\WT2013-14\...
File Edit View Favorites Tools Help
EN
1:39 PM
2/9/2015

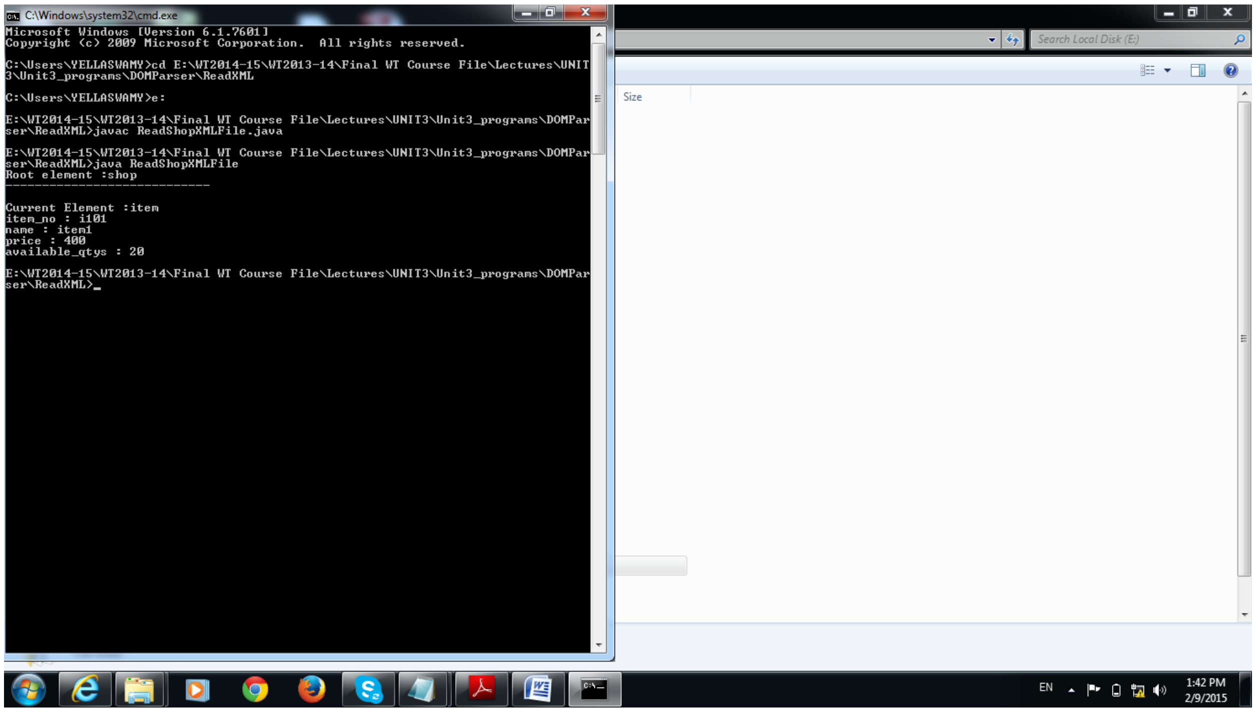
```
  <?xml version="1.0" encoding="UTF-8"?>
  <!DOCTYPE shop SYSTEM "shop.dtd">
- <shop logo="mylogo">
  - <item type="books" item_no="i101">
        <name>item1</name>
        <price type="rs" units="one">400</price>
        <available_qtys>20</available_qtys>
    </item>
  - <selected_items item_no="i101">
        <discount units="percentage">10</discount>
    </selected_items>
  - <selected_items item_no="i102">
        <gift item="i101"/>
    </selected_items>
    <copy-rights/>
  </shop>
```

ReadShopXMLFile.java

```
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.DocumentBuilder;
import org.w3c.dom.Document;
import org.w3c.dom.NodeList;
import org.w3c.dom.Node;
import org.w3c.dom.Element;
import java.io.File;

public class ReadShopXMLFile
	{

  public static void main(String argv[]) {

    try {

		File fXmlFile = new File("shop.xml");
	DocumentBuilderFactory dbFactory = DocumentBuilderFactory.newInstance();
	DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();
	Document doc = dBuilder.parse(fXmlFile);


	doc.getDocumentElement().normalize();

	System.out.println("Root element :" + doc.getDocumentElement().getNodeName());

	NodeList nList = doc.getElementsByTagName("item");

	System.out.println("----------------------------");

	for (int temp = 0; temp < nList.getLength(); temp++)
		{

		Node nNode = nList.item(temp);

		System.out.println("\nCurrent Element :" + nNode.getNodeName());

		if (nNode.getNodeType() == Node.ELEMENT_NODE) {

			Element eElement = (Element) nNode;

			System.out.println("item_no : " + eElement.getAttribute("item_no"));
			System.out.println("name : " +
eElement.getElementsByTagName("name").item(0).getTextContent());
			System.out.println("price : " +
eElement.getElementsByTagName("price").item(0).getTextContent());
			System.out.println("available_qtys : " +
eElement.getElementsByTagName("available_qtys").item(0).getTextContent());
```

```
                                        //System.out.println("Salary : " +
//eElement.getElementsByTagName("salary").item(0).getTextContent());

                                }
                        }
   } catch (Exception e) {
            e.printStackTrace();
   }
  }

}
```

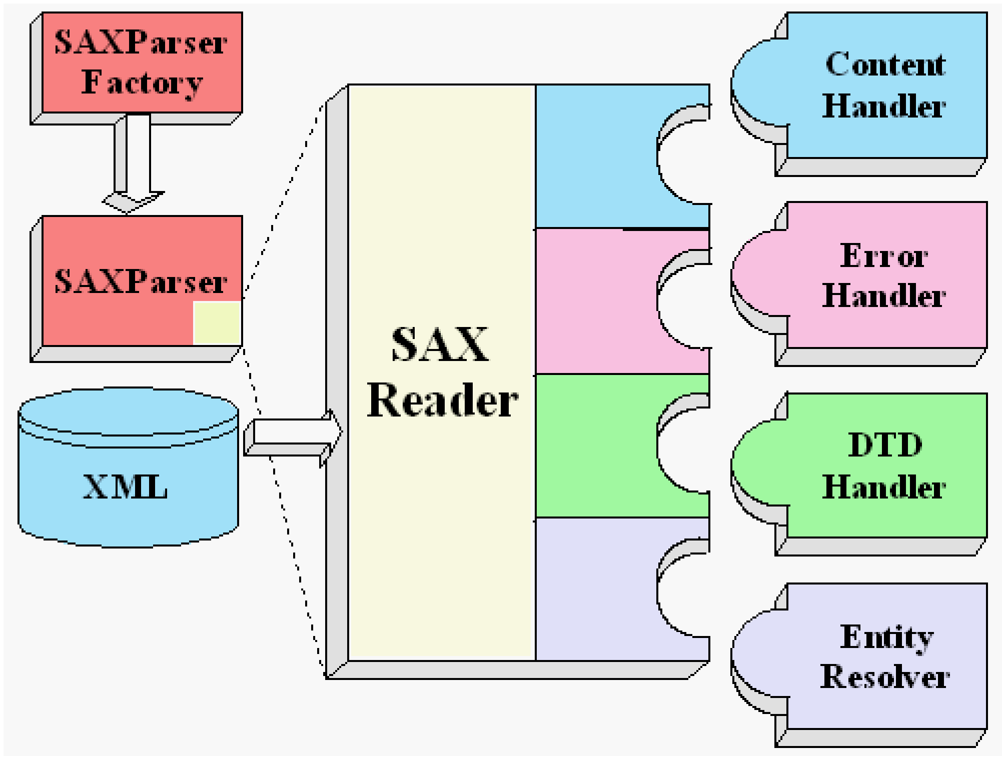
Output:



```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\YELLASWAMY>cd E:\WT2014-15\WT2013-14\Final WT Course File\Lectures\UNIT
3\Unit3_programs\DOMParser\ReadXML

C:\Users\YELLASWAMY>e:

E:\WT2014-15\WT2013-14\Final WT Course File\Lectures\UNIT3\Unit3_programs\DOMPar
ser\ReadXML>javac ReadShopXMLFile.java

E:\WT2014-15\WT2013-14\Final WT Course File\Lectures\UNIT3\Unit3_programs\DOMPar
ser\ReadXML>java ReadShopXMLFile
Root element :shop
----------------------------

Current Element :item
item_no : i101
name : item1
price : 400
available_qtys : 20

E:\WT2014-15\WT2013-14\Final WT Course File\Lectures\UNIT3\Unit3_programs\DOMPar
ser\ReadXML>_
```

SAX (Simple API for XML)

- is used for Simple Search operations
- Follows Event Based Approach
- SAX1.0 is a non-validating parser where as 2.0 can be used for validating also
- These parsers are used for read-only purpose
- in this case at movement when parser reads the data it pushes the data in to the application allowing the application to take decision to store the data or not.
- It reads the content in a forward direction
- once the XML document submitted to the parser it validates the document.
- if its valid then starts processing where it gives the notification of each of the part of the document which it is reading.like startElement,endElement.



Here is a summary of the key
SAX APIs:
`SAXParserFactory`
A SAXParserFactory object creates an instance of the parser determined by the system property, `javax.xml.parsers.SAXParserFactory`.

`SAXParser`

The `SAXParser` interface defines several kinds of `parse()` methods. In general, you pass an XML data source and a DefaultHandler object to the parser, which processes the XML and invokes the appropriate methods in the handler object.

SAXReader

The SAXParser wraps a SAXReader. Typically, you don't care about that, but every once in a while you need to get hold of it using SAXParser's `getXMLReader()`, so you can configure it. It is the SAXReader which carries on the conversation with the SAX event handlers you define.

DefaultHandler
Not shown in the diagram, a DefaultHandler implements the `ContentHandler`, `ErrorHandler`,`DTDHandler`, and `EntityResolver` interfaces (with null methods), so you can override only the ones you're interested in.

ContentHandler

Methods like `startDocument`, `endDocument`, `startElement`, and `endElement` are invoked when an XML tag is recognized. This interface also defines methods `characters` and `processingInstruction`,which are invoked when the parser encounters the text in an XML element or an inline processing instruction,respectively.

ErrorHandler
Methods `error`, `fatalError`, and `warning` are invoked in response to various parsing errors. The default error handler throws an exception for fatal errors and ignores other errors (including validation errors). That's one reason you need to know something about the SAX parser, even if you are using the DOM. Sometimes, the application may be able to recover from a validation error. Other times, it may need to generate an exception. To ensure the correct handling, you'll need to supply your own error handler to the parser.

DTDHandler
Defines methods you will generally never be called upon to use. Used when processing a DTD to recognize and act on declarations for an *unparsed entity*.

EntityResolver
The `resolveEntity` method is invoked when the parser must identify data identified by a URI. In most cases,a URI is simply a URL, which specifies the location of a document, but in some cases the document may be identified by a URN -- a *public identifier*, or name, that is unique in the web space. The public identifier may be specified in addition to the URL. The `EntityResolver` can then use the public identifier instead of the URL to find the document, for example to access a local copy of the document if one exists.
A typical application implements most of the `ContentHandler` methods, at a minimum. Since the default implementations of the interfaces ignore all inputs except for fatal errors, a robust implementation may want to implement the ErrorHandler methods, as well.

The SAX Packages

The SAX parser is defined in the following packages.

Package Description

org.xml.sax Defines the SAX interfaces. The name "`org.xml`" is the package prefix that was settled on by the group that defined the SAX API.

org.xml.sax.ext

Defines SAX extensions that are used when doing more sophisticated SAX processing, for example, to process a document type definitions (DTD) or to see the detailed syntax for a file.

org.xml.sax.helpers

Contains helper classes that make it easier to use SAX -- for example, by defining a default handler that has null-methods for all of the interfaces, so you only need to override the ones you actually want to implement.

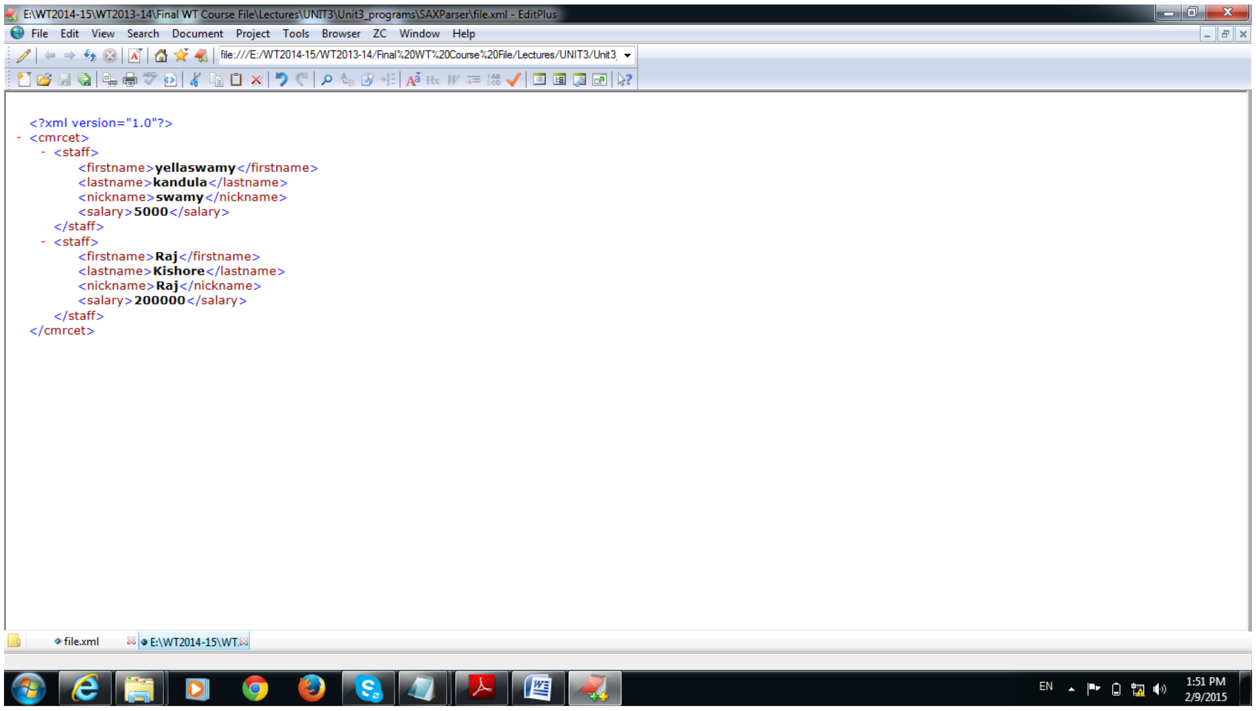
javax.xml.parsers Defines the `SAXParserFactory` class which returns the SAXParser. Also defines exception classes for reporting errors.

Example:

File.xml

```
<?xml version="1.0"?>
<cmrcet>
        <staff>
                <firstname>yellaswamy</firstname>
                <lastname>kandula</lastname>
                <nickname>swamy</nickname>
                <salary>5000</salary>
        </staff>
        <staff>
                <firstname>Raj</firstname>
                <lastname>Kishore</lastname>
                <nickname>Raj</nickname>
                <salary>200000</salary>
        </staff>
</cmrcet>
```

Output:



```
<?xml version="1.0"?>
- <cmrcet>
  - <staff>
        <firstname>yellaswamy</firstname>
        <lastname>kandula</lastname>
        <nickname>swamy</nickname>
        <salary>5000</salary>
    </staff>
  - <staff>
        <firstname>Raj</firstname>
        <lastname>Kishore</lastname>
        <nickname>Raj</nickname>
        <salary>200000</salary>
    </staff>
  </cmrcet>
```

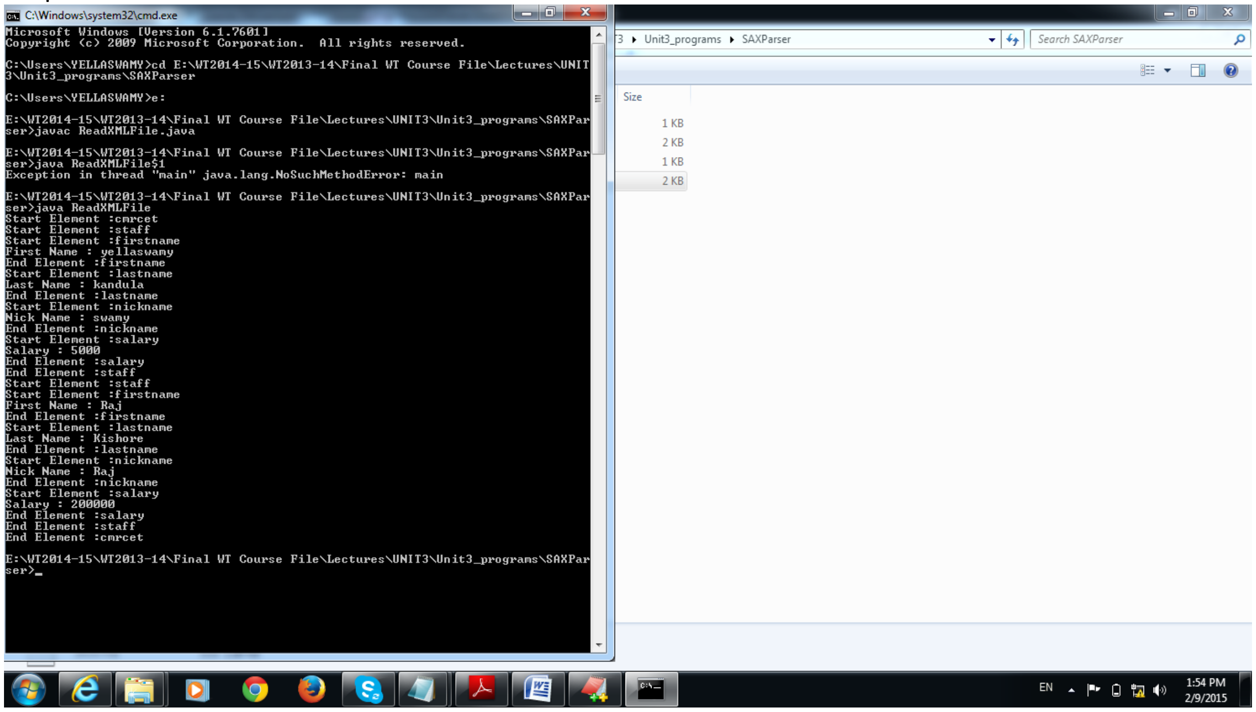
ReadXMLFile.java

```
import javax.xml.parsers.SAXParser;
import javax.xml.parsers.SAXParserFactory;
import org.xml.sax.Attributes;
import org.xml.sax.SAXException;
import org.xml.sax.helpers.DefaultHandler;

public class ReadXMLFile {

  public static void main(String argv[]) {

    try {
 //Step1
            SAXParserFactory factory = SAXParserFactory.newInstance();
            SAXParser saxParser = factory.newSAXParser();
 //Step2  set the dcoument handler
            DefaultHandler handler = new DefaultHandler() {

            boolean bfname = false;
            boolean blname = false;
            boolean bnname = false;
            boolean bsalary = false;

            public void startElement(String uri, String localName,String qName,
                Attributes attributes) throws SAXException {
```

```
		System.out.println("Start Element :" + qName);

		if (qName.equalsIgnoreCase("FIRSTNAME")) {
			bfname = true;
		}

		if (qName.equalsIgnoreCase("LASTNAME")) {
			blname = true;
		}

		if (qName.equalsIgnoreCase("NICKNAME")) {
			bnname = true;
		}

		if (qName.equalsIgnoreCase("SALARY")) {
			bsalary = true;
		}

	}

	public void endElement(String uri, String localName,
		String qName) throws SAXException {

		System.out.println("End Element :" + qName);

	}

	public void characters(char ch[], int start, int length) throws SAXException {

		if (bfname) {
			System.out.println("First Name : " + new String(ch, start, length));
			bfname = false;
		}

		if (blname) {
			System.out.println("Last Name : " + new String(ch, start, length));
			blname = false;
		}

		if (bnname) {
			System.out.println("Nick Name : " + new String(ch, start, length));
			bnname = false;
		}

		if (bsalary) {
```

```
                                System.out.println("Salary : " + new String(ch, start, length));
                                bsalary = false;
                        }

                }

            };

             saxParser.parse("file.xml", handler);

            } catch (Exception e) {
             e.printStackTrace();
            }

          }

        }
```

Output:



```
C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\YELLASWAMY>cd E:\WT2014-15\WT2013-14\Final WT Course File\Lectures\UNIT
3\Unit3_programs\SAXParser

C:\Users\YELLASWAMY>e:

E:\WT2014-15\WT2013-14\Final WT Course File\Lectures\UNIT3\Unit3_programs\SAXPar
ser>javac ReadXMLFile.java

E:\WT2014-15\WT2013-14\Final WT Course File\Lectures\UNIT3\Unit3_programs\SAXPar
ser>java ReadXMLFile$1
Exception in thread "main" java.lang.NoSuchMethodError: main

E:\WT2014-15\WT2013-14\Final WT Course File\Lectures\UNIT3\Unit3_programs\SAXPar
ser>java ReadXMLFile
Start Element :cmrcet
Start Element :staff
Start Element :firstname
First Name : yellaswamy
End Element :firstname
Start Element :lastname
Last Name : kandula
End Element :lastname
Start Element :nickname
Nick Name : swamy
End Element :nickname
Start Element :salary
Salary : 5000
End Element :salary
End Element :staff
Start Element :staff
Start Element :firstname
First Name : Raj
End Element :firstname
Start Element :lastname
Last Name : Kishore
End Element :lastname
Start Element :nickname
Nick Name : Raj
End Element :nickname
Start Element :salary
Salary : 200000
End Element :salary
End Element :staff
End Element :cmrcet

E:\WT2014-15\WT2013-14\Final WT Course File\Lectures\UNIT3\Unit3_programs\SAXPar
ser>_
```