

X m l

# **X.M.L.** Extnesible **M**arkup **L**anguage

*Pellizzaro Massimiliano*

**Milano, Settembre 2003**

© Copyright IBM Corporation 2002

## Agenda

---

- Ø XML history
- Ø XML components
- Ø XML DTD vs Schema
- Ø Parsing XML: SAX e DOM
- Ø XSL
- Ø FO

## History of XML

---

- Ø XML is **not** a markup language
- Ø XML **is** a **set of rules** for creating **new** markup languages (XML document vs. XML compliant document)
- Ø XML is a subset of **SGML** (Standard Generalized Markup Language), that specifies rules for creating markup languages (es: HTML)
- Ø Generic **markup** separates the logical structure of a document from its content
- Ø The markup describe the **meta-data** of the document
- Ø XML was born for sharing data on the Web without extending HTML
- Ø XML 1.0 was made by W3C on February 1998

## Benefits XML

---

- Ø Data Independence from application (Java vs platform)
- Ø Improve communication
- Ø One data source, multiples views
- Ø Improving data searches
- Ø Simpler application development
- Ø .....

## XML Components

---

*There are six types of markup defined in XML*

Ø **Elements**. An element is composed of start and end tag

Ø `<STREET> 4296 Razon Hill Road </STREET>`

Ø **Attributes**. Name/values pairs are placed after start tag name

Ø `<APPLET with="100" height="200">`

Ø **Comments**. It is a free text ignored by XML processor

Ø `<!-- free text placed here -->`

Ø **Processing instruction**. Used to pass information to a processing application

Ø `<?application data ?>`

Ø Examples: attaching XSL stylesheets to a document `<?xml:stylesheet href="http://.../memo.xsl" type="text/xsl" >`

Ø Examples: XML declaration `<?xml version="1.0" encoding="UTF-16" ?>`

Ø **Entity Reference**. They are used to put reserved characters in markup.

Ø Example `&lt;`; stand for `<`

Ø **CDATA Section**. The data written in CDATA section should not be processed but passed as is to the application.

## XML Specification vs HTML

---

- Ø XML elements (composed of a start and end tag) must be strictly nested
  - Ø `<B> <l> xxxxx </l> </B>`
- Ø Every tag must have an end tag
- Ø An empty tag must have an ending “slash”
  - Ø `<IMG SRC="picture.jpg" />`
- Ø XML document allows only **one** root element
- Ø Attributes must be surrounded by quotes
  - Ø `<TEST PARAM="10">`
- Ø XML tags are case sensitive
- Ø Whitespace are relevant in XML

## XML Namespaces

---

- Ø W3C Recommendation on January 14, 1999 to solve the problem of proliferating of names used in XML document
- Ø To create an “universal” name, an XML name is separated into two parts: a namespace prefix and a local part
- Ø Declaring namespaces
  - Ø `<ROOT xmlns:ht="http://.... ">`
  - Ø `<ht:NAME ht:ATTR1="value1"> </ht:NAME>`
- Ø It is possible to define default namespaces without declare the namespace
  - Ø `<ht xmlns="http://.... ">`

## Valid document vs well formed document

---

- Ø A **well formed** document means that the document has not any “syntax error”. It can be read by any xml parser.
- Ø A **valid** document means that the particular document follows special rules specified in a different section - file
  - Ø XML - DTD
  - Ø XML - Schema

## XML-DTD

- Ø It states which tag we can use in a xml-document
- Ø A document specifies its DTD in a document type declaration
- Ø There are two forms of document declaration: internal or external

- Ø `<!DOCTYPE JDATA [ ..... ]>`

- Ø `<!DOCTYPE JDATA SYSTEM "javadata.dtd">`

EMPTY  
 ANY  
 Mixed  
 Element

- Ø Element declaration:

- Ø `<!ELEMENT elemName content-spec>`

- Ø Attribute declaration:

- Ø `<!ATTLIST elemName attName attType default-decl>`

- Ø Indicators:

- Ø **?** zero or one time
- Ø **\*** zero or more times
- Ø **+** one or more times

CDATA  
 ENUMERATION  
 tokenized types

#REQUIRED  
 #IMPLIED  
 #FIXED  
 default

## Ex.1 - a

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XML Spy v4.3 U (http://www.xmlspy.com) by max (ibm) -->
```

```
<!ELEMENT ADDRESS_BOOK (ADDRESS)+>
<!ELEMENT ADDRESS (NAME, STREET+, CITY, STATE, ZIP)>
<!ELEMENT NAME (#PCDATA)>
<!ELEMENT STREET (#PCDATA)>
<!ELEMENT CITY (#PCDATA)>
<!ELEMENT STATE (#PCDATA)>
<!ELEMENT ZIP (#PCDATA)>
<!ATTLIST STREET
    TYPE (street | suiteno | aptno | other) #IMPLIED
>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ADDRESS_BOOK SYSTEM "ablm.dtd">
<ADDRESS_BOOK>
    <ADDRESS>
        <NAME>Micheal Daconta</NAME>
        <STREET>4296 Razon Hill Road</STREET>
        <CITY>Beleton</CITY>
        <STATE>VA</STATE>
        <ZIP>22712</ZIP>
    </ADDRESS>
    <ADDRESS>
        <NAME>Max Pellizzaro</NAME>
        <STREET>3256 West 8th</STREET>
        <CITY>Larned</CITY>
        <STATE>KS</STATE>
        <ZIP>67775</ZIP>
    </ADDRESS>
</ADDRESS_BOOK>
```

## XML Schema

---

- ∅ Define and describe XML document using XML-compliant markup
- ∅ There are two parts of the XML Schemas: **Structures** and **Data Types**
  - ∅ **Structures**: describe a replacement syntax for describing XML document with a finer granularity than DTD
  - ∅ **Data Types**: defines primitives primitive data types and other XML specification (xsl)
- ∅ Structure:
  - ∅ Schema `<schema> </schema>`
  - ∅ Simple type definition see Data Types
  - ∅ Complex type definition `<type> ... </type>`
  - ∅ Element type declaration `<element>.. </element>`
  - ∅ Attribute declaration `<attribute> ..</attribute>`
  - ∅ .....
- ∅ Data Types
  - ∅ String, boolean, binary, long, int, short, byte....
  - ∅ ....

## Ex.1 - b

---

```
<schema>
  <element name="ADDRESS_BOOK" type="ADDRESS_BOOK_TYPE" />
  <type name="ADDRESS_BOOK_TYPE">
    <element name="ADDRESS" type="ADDRESS_TYPE" minOccurs="1"
maxOccurs="*" />
  </type>
  <type name="ADDRESS_TYPE">
    <element name="NAME" type="string" />
    <element name="STREET" type="string" />
    <element name="CITY" type="string" />
    <element name="STATE" type="string" />
    <element name="ZIP" type="string" />
  </type>
</schema>
```

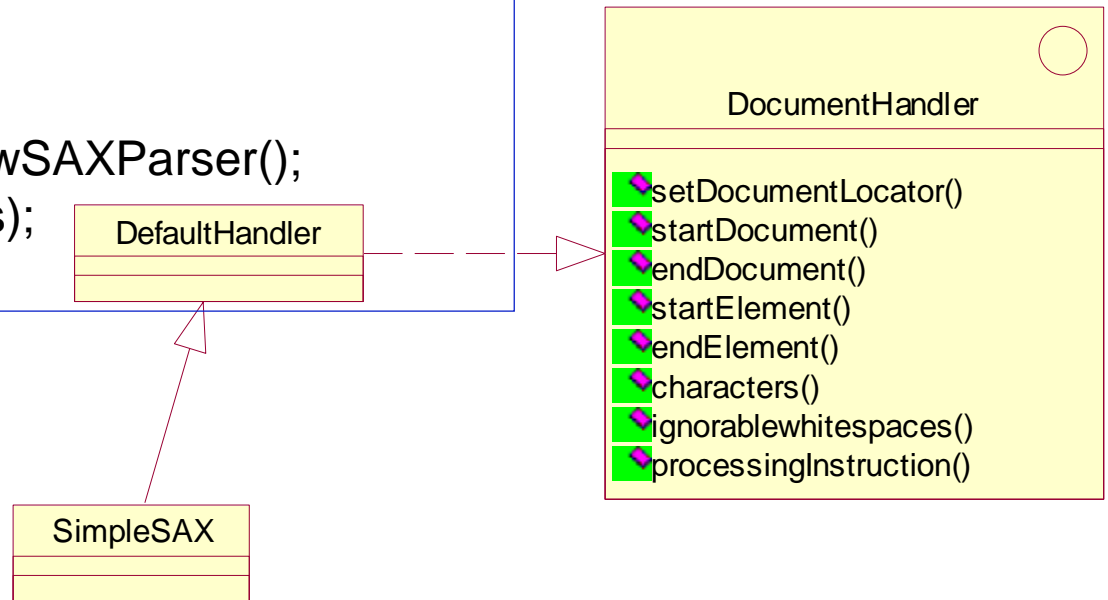
## Parsing XML documents

---

- Ø There are **two** parsing methods: event-based vs. Tree-based
- Ø **Event-based**: an event based parser reads the XML document and reacts to the elements you are interested in. It does not “consume” memory because it does not remember what has been parsed
- Ø **Tree-based**: a tree based parser produces a tree data structure, such as the W3C’s Document Object Model (DOM). This is an in-memory representation of the XML text document. It is possible to navigate in the document using methods like getChilds(), getParent()....
- Ø In order to parse the XML document you can choose between two parsers:
  - Ø SAX API Parser
  - Ø DOM API Parser

## SAX api a simple example

```
try
{
  SAXParserFactory factory =
  SAXParserFactory.newInstance();
  SAXParser saxParser = factory.newSAXParser();
  saxParser.parse( new File(file), this);
}
```



To run the example in the classpath it must be present classes for the SAXParser (jxap.jar) and for DocumentHadler (xalan.jar) Oracle xmlparser2.jar does it.

## DOM api a simple example

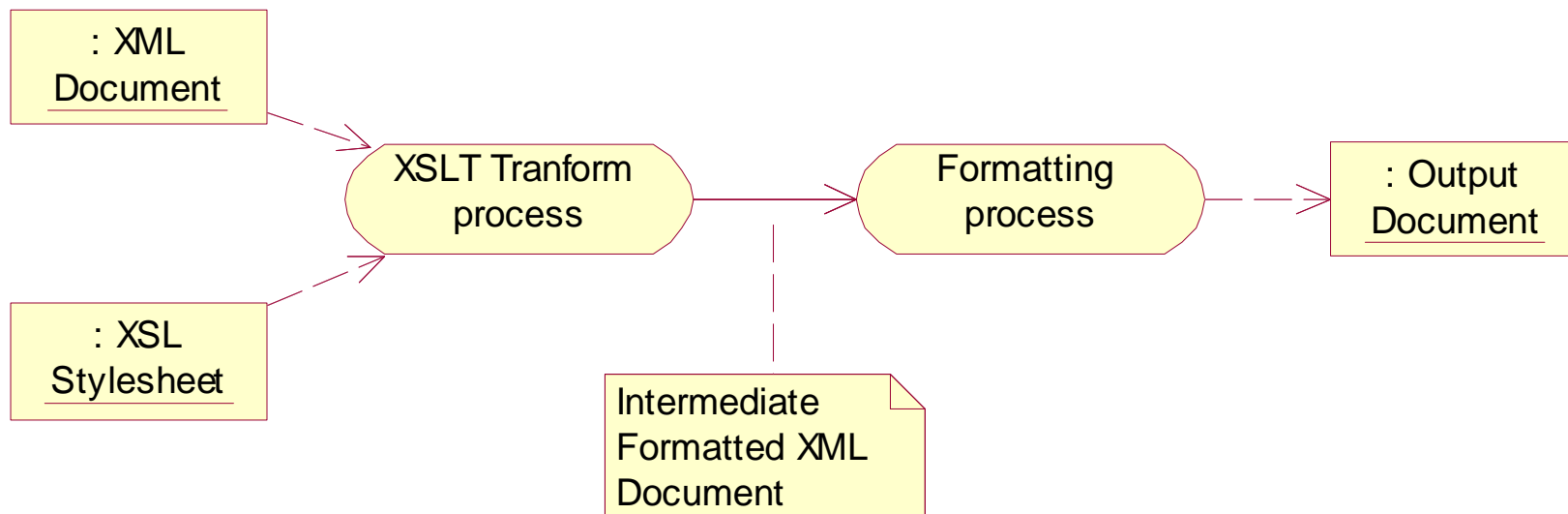
```
try
{
    DocumentBuilderFactory factory =
DocumentBuilderFactory.newInstance();
    DocumentBuilder builder = factory.newDocumentBuilder();
    Document doc = builder.parse(new File(args[0]));
    Element root = doc.getDocumentElement();
    printNodes(root);
}
```

```
private static void printNodes(Node n)
{
    if ( n== null)
        return ;
    System.out.println("Node name: " + n.getNodeName() + ", value: " + n.getNodeValue());
    NodeList list = n.getChildNodes();
    if (list==null)
        return;
    else
    {
        int len = list.getLength();
        for (int i=0; i<len; i++)
            printNodes(list.item(i));
    }
}
```

## The XSL Language

Ø XSL (Extensible Stylesheet Language) is actually two languages in one: one for *transforming* and one for *formatting*

- Ø Transforming -> use **xsl:** namespace
- Ø Formatting -> use **fo:** namespace



## XSL a simple example

```
<xsl:stylesheet  
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"  
version="1.0">
```

Root Element

```
<xsl:template match="/">  
  <html><body>  
    <xsl:apply-templates/>  
  </body></html>  
</xsl:template>
```

XSL Top-Level  
Element

```
<xsl:template match="Header">  
  <head><title><xsl:value-of select="."/></title></head>  
  <center><h1><xsl:value-of select="."/></h1></center>  
</xsl:template>
```

```
</xsl:stylesheet>
```

## xsl:template

---

- Ø Templates defined rules to be applied when matching criteria are found
- Ø Matching criteria:
  - Ø `match="/"` Match the root element
  - Ø `match="**"` Match any element
  - Ø `match="ElementName"` Match any 'Element Name'
  - Ø `match="@attribute"` Match any source element with the named attribute
- Ø Templates **are recursive**
  - Ø `<xsl:apply-templates/>`
- Ø Selecting values:
  - Ø `<xsl:value-of select="."/>` Selects all elements and its child nodes
  - Ø `<xsl:value-of select="@attribute"/>` Select the value of the attribute
  - Ø `<xsl:value-of select="ChildElement"/>` Select the value of a child node

## More complex criteria

---

Ø It is possible to use Predicates and functions to match elements

Ø `<xsl:template match="Listing[position() = last()]" >`

Ø Useful functions

Ø `count()`, `id(someid)`, `last()`, `position()`

Ø Combining expression

Ø It is possible to use normal operator: `=`, `or`, `&ln;`, `>`

Ø Controlling the flow of events

Ø `<xsl:for-each select="@*" />`

Ø `<xsl:if test = "... " >`

Ø ...

Ø Numeric function

Ø `ceiling()`, `floor()`, `number()`, `sum()`.. Ex: `<xsl:value-of select="round(@ElementName)" />`

Ø String expression

Ø `concat[ ]`, `substring[ ]`, `start-with[ ]`, ....

Ø Node Set expression

## Ex. 2 Processing xml document

---

- Ø Using jclark : set in the classpath: xp.jar, xt.jar, sax.jar
- Ø Run this command:

```
java -Dcom.jclark.xsl.sax.parser=com.jclark.xml.sax.CommentDriver
```

```
com.jclark.xsl.sax.Driver
```

```
REListing.xml
```

```
RETable.xsl
```

```
output.html
```

Input  
xml

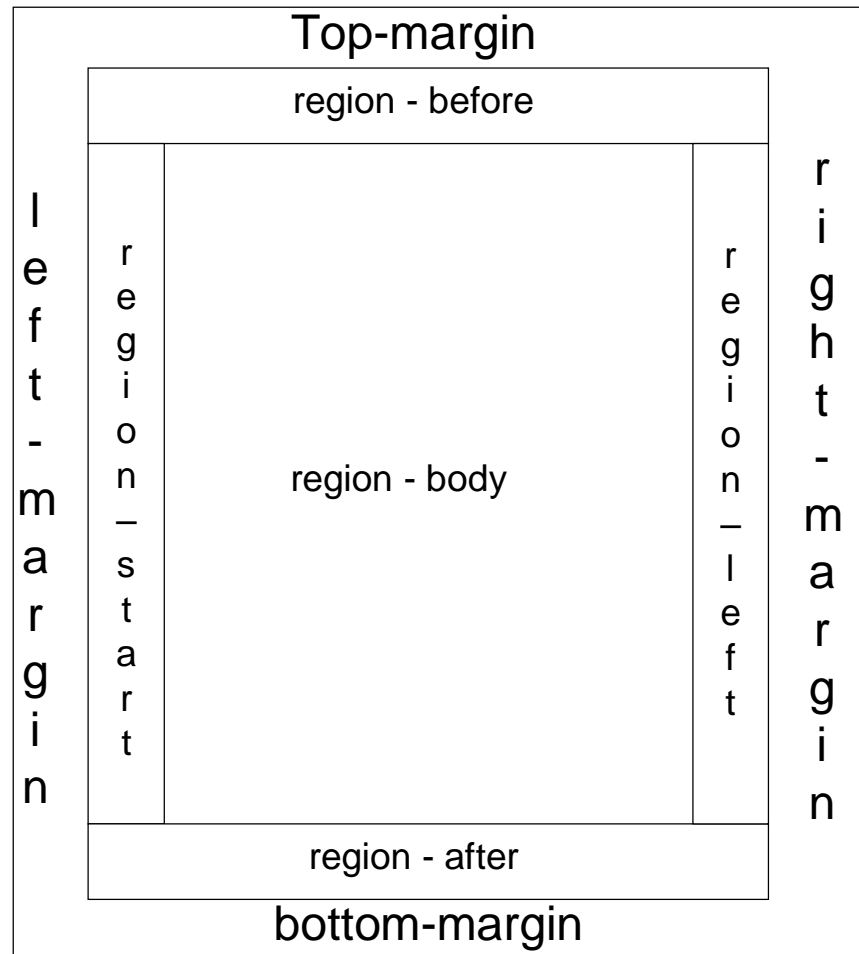
Input  
xsl

Output

```
graph LR; A[REListing.xml] --- B[Input xml]; C[RETable.xsl] --- D[Input xsl]; E[output.html] --- F[Output]
```

## Formatting Objects

- Ø FO is a specification for defining the layout of a document
- Ø FO is based on a **page model**



## Writing FO files

---

```
<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">  
  
<fo:layout-master-set  
  
<fo:simple-page-master  
  page-master-name="cheap" height="8.5in" width="11in"  
  margin-top="0.5in" margin-bottom="0.5in" margin-left="1in" margin-right="1in">  
  
  <fo:region-before extent="1in"/>  
  <fo:region-body margin-top="1.25in"/>  
  <fo:region-after extent=".75in"/>  
</fo:simple-page-master>  
  
</fo:layout-master-set>  
  
<fo:flow>  
  <fo:block> .... </fo:block>  
</fo:flow>  
  
</fo:root>
```

## Running FOP

---

### Ø Using Apache FOP :

- Ø set in the classpath: xerces.jar

- Ø set in the classpath: fop\_bin\_0\_12\_0.jar

### Ø Run this command:

```
java org.apache.fop.apps.CommandLine  
fo1.fob  
fo1.pdf
```