Introduction of XML

Leo Huang
黃嘉輝

Agenda

- XML and E-Commerce
- XML Introduction
- XML Syntax
- Document Type Definition (DTD)
- Document Object Model (DOM)
- Simple API for XML (SAX)

What is XML?

XML is a text-based markup language that is fast becoming the standard for **Data Interchange** on the Web. As with HTML, you identify data using tags .

But unlike HTML, XML tags tell you what the data **means**, rather than how to display it. Where an HTML tag says something like "display this data in bold font" (<b>...</b>), an XML tag acts like a field name in your program. It puts a label on a piece of data that identifies it (for example: <message>...</message>).

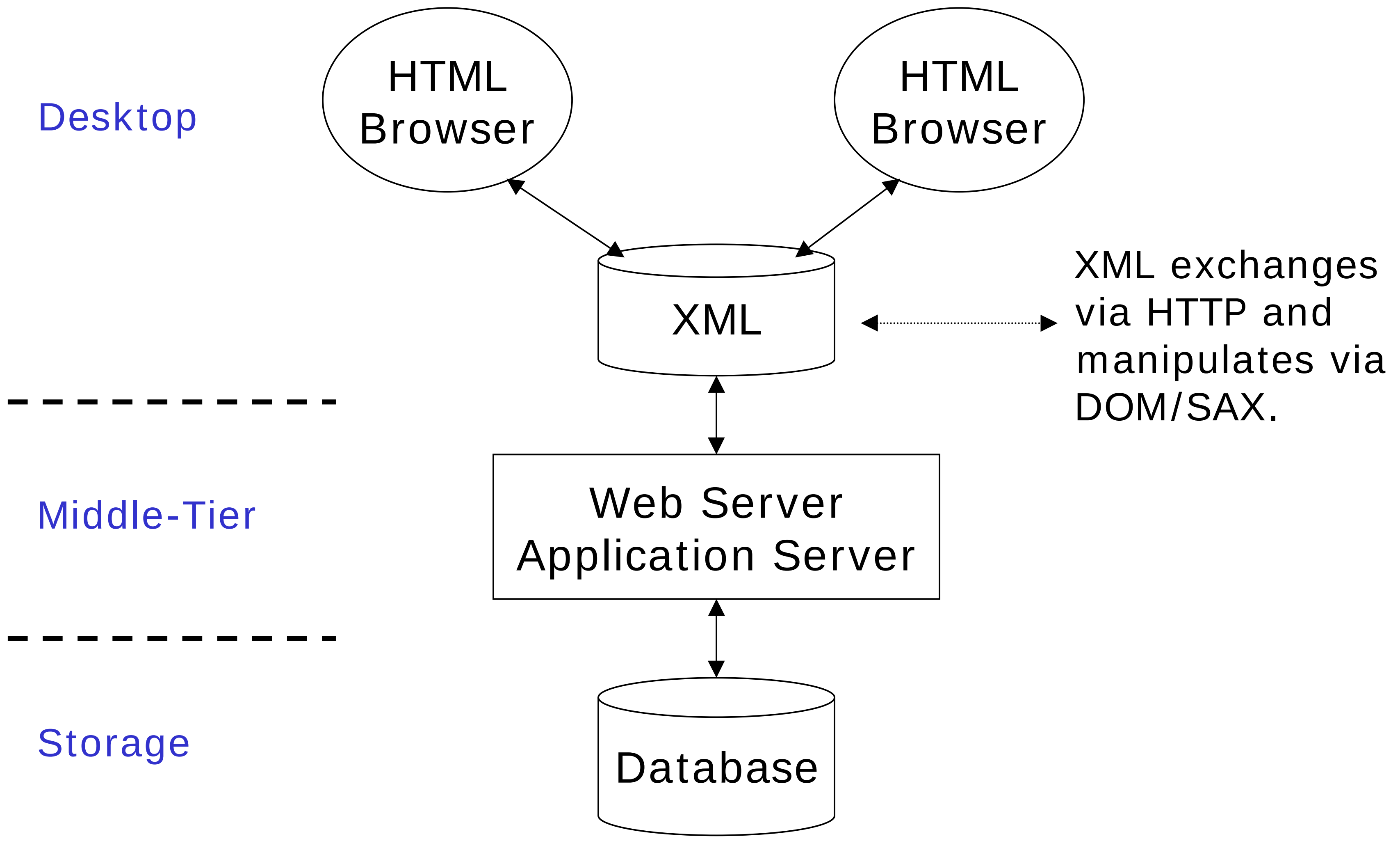
HTML v.s. XML

HTML is a language designed to displaying information: headings <head>, titles <title>, fonts <font> and so on. It is document structure- and presentation-oriented.

HTML Drawbacks:

- HTML is not extensible.
- HTML is display-centric.
- HTML is not directly reusable.
- HTML only provides one “view” of data.
- HTML has little or no semantic structure.

XML Architecture



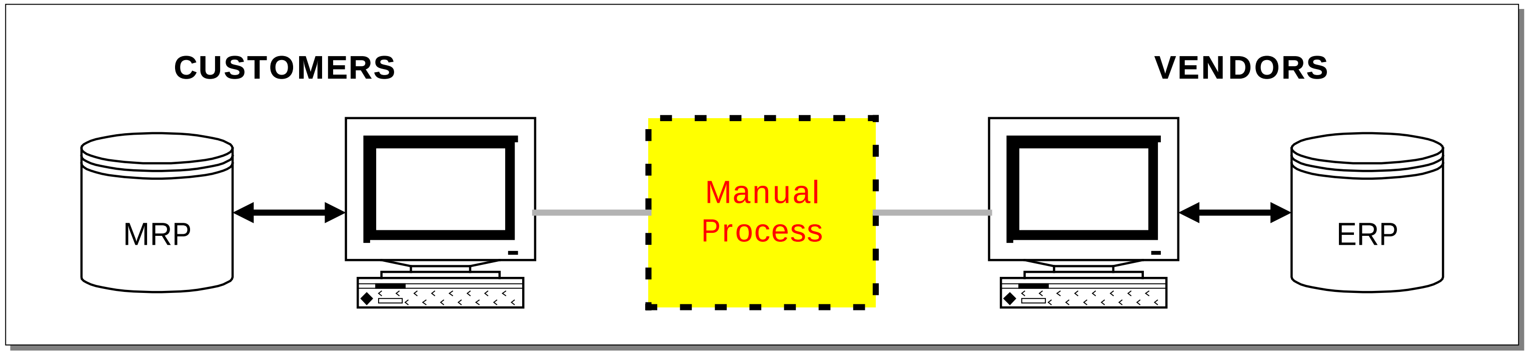
XML exchanges via HTTP and manipulates via DOM/SAX.

XML and E-Commerce

B2B Commerce

The objective of e-commerce is to eliminate the manual processes by allowing internal applications of different companies to directly exchange information.

Traditional Commerce:



Approaches of E-Commerce

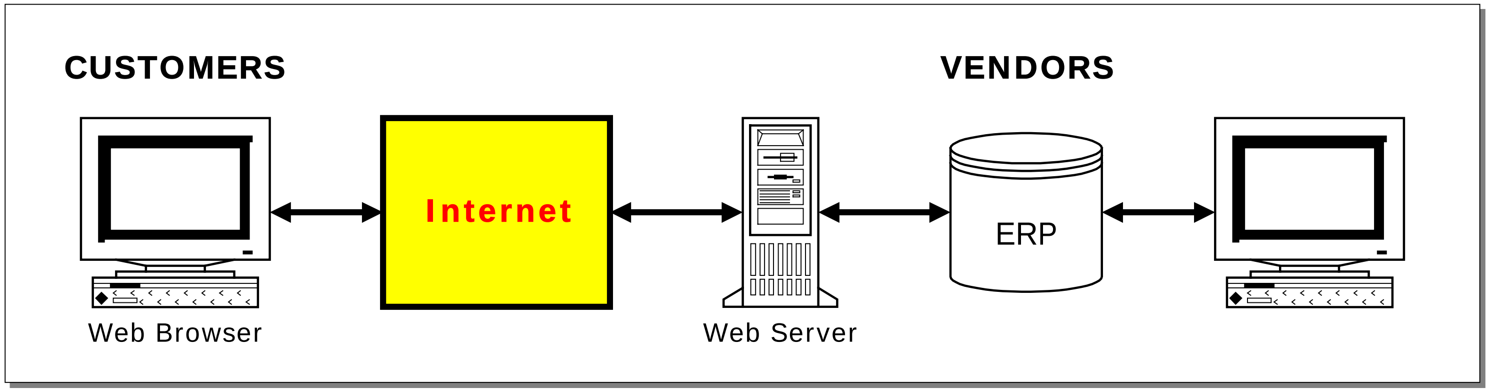
There are three approaches of E-Commerce currently:

- Web Storefronts (Selling direct to customers).
- E-Commerce Portals (B2B transactions).
- E2E E-Commerce (Information sharing).

Web Storefronts

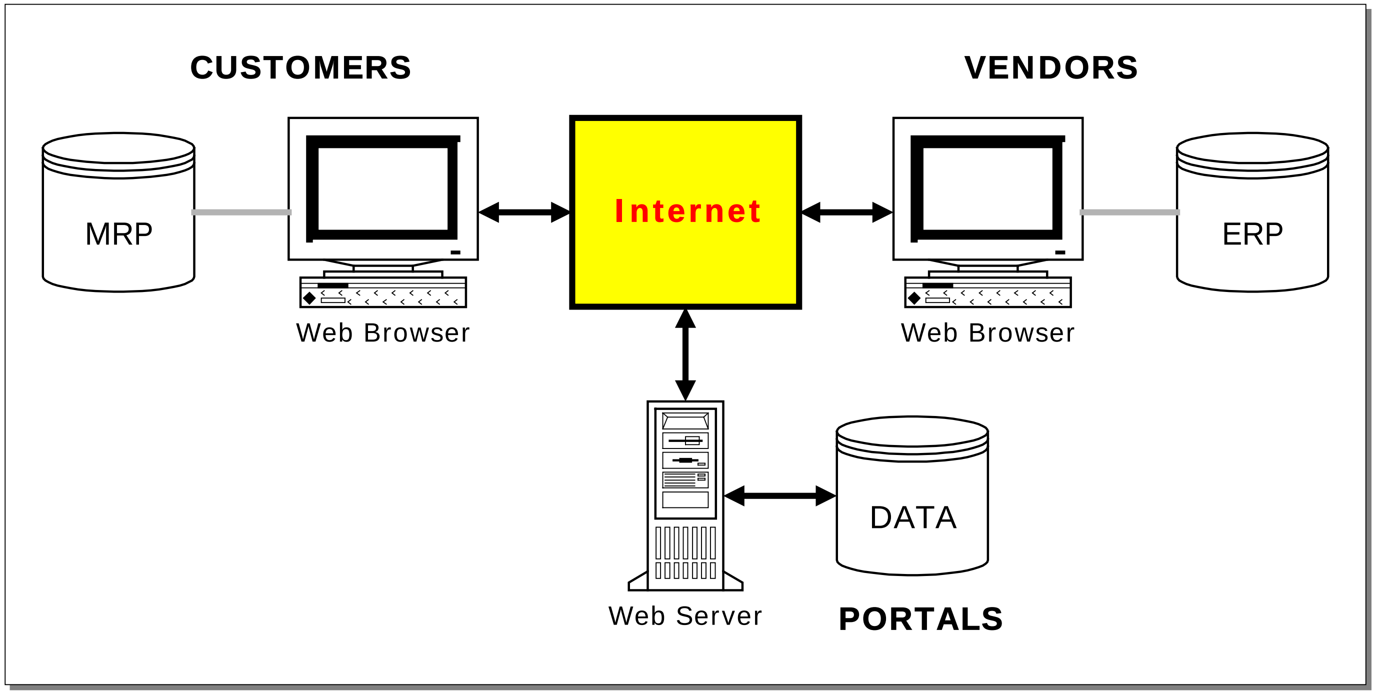
Web Storefronts provide a web interface to a vendor's catalog of products or services and integrate the placing of an order over the web with an internal processing system (ERP).

This may be an acceptable solution for B2C e-commerce but it is inadequate for true B2B e-commerce.



E-Commerce Portals

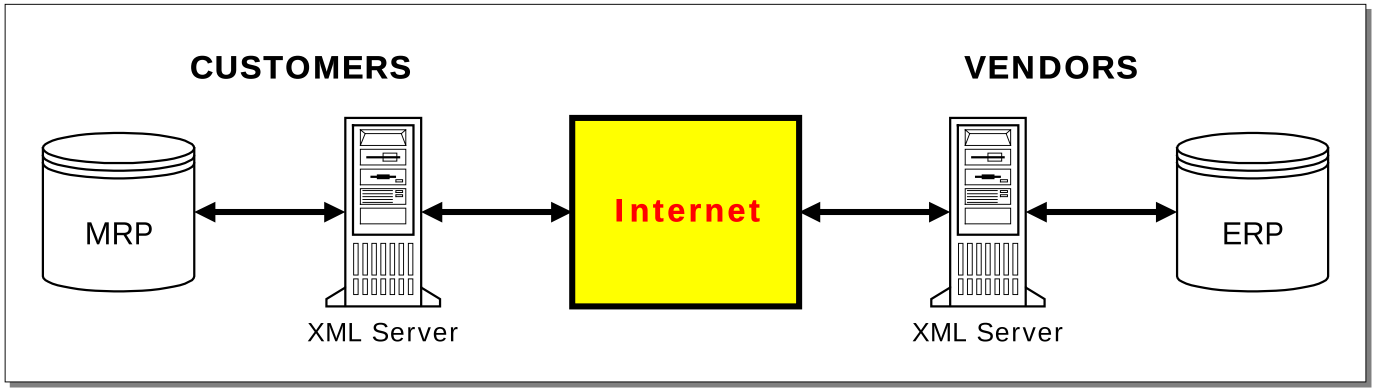
Utilizing e-commerce portals, customers can browse numerous vendor catalogs and place orders only visiting one portal web site. Vendors go to the same portal web site to view and fill customer orders.



E2E E-Commerce

The first integrated approach aimed at solving B2B e-commerce was called Electronic Data Interchange (EDI). Companies are building Enterprise-to-Enterprise e-commerce using XML which allows the internal applications of different companies to share information directly.

The tremendous advantage XML holds over EDI is that XML is both machine and human readable while EDI is only machine-readable.



EDI v.s. XML

The creators of EDI were concerned about the size of their messages. Bandwidth for EDI networks is very expensive, thus, EDI are very compressed and use codes to represent complex values. The complexity of EDI makes EDI programmers hard to train and expensive to keep. Complexity drives cost.

On the other hand, XML messages are rich in metadata, making them easy to read and debug. Good XML strives to be self-describing. The simplicity of XML makes XML programmers easy to train and less expensive to keep, in turn making XML applications less expensive to buy and maintain.

Convert EDI to XML

Header:

```
ISA*00* *00* *08*61112500TST *01*DEMO WU000003
*991231*1039*U00302000009561*0*P?
GS*PO*6111250011*WU000003 *970911*1039*9784*X*003020
ST*850*397822
BEG*00*RE*1234** 991231
REF*AH*M109
REF*DP*641
REF*IA*000100685
DTM*010*970918
N1*BY*92*1287
N1*ST*92*87447
N1*ZZ*992*1287
```

Segment:

```
PO1*1*1*EA*13.33**CB*80211*IZ*364*UP*718379271641
PO1*1*2*EA*13.33**CB*80211*IZ*382*UP*718379271573
PO1*1*3*EA*13.33**CB*80213*IZ*320*UP*718379271497
PO1*1*4*EA*13.33**CB*80215*IZ*360*UP*718379271848
PO1*1*5*EA*13.33**CB*80215*IZ*364*UP*718379271005
CTT*25
SE*36*397822
GE*1*9784
```

Trailer:

```
IEA*1*000009561
```

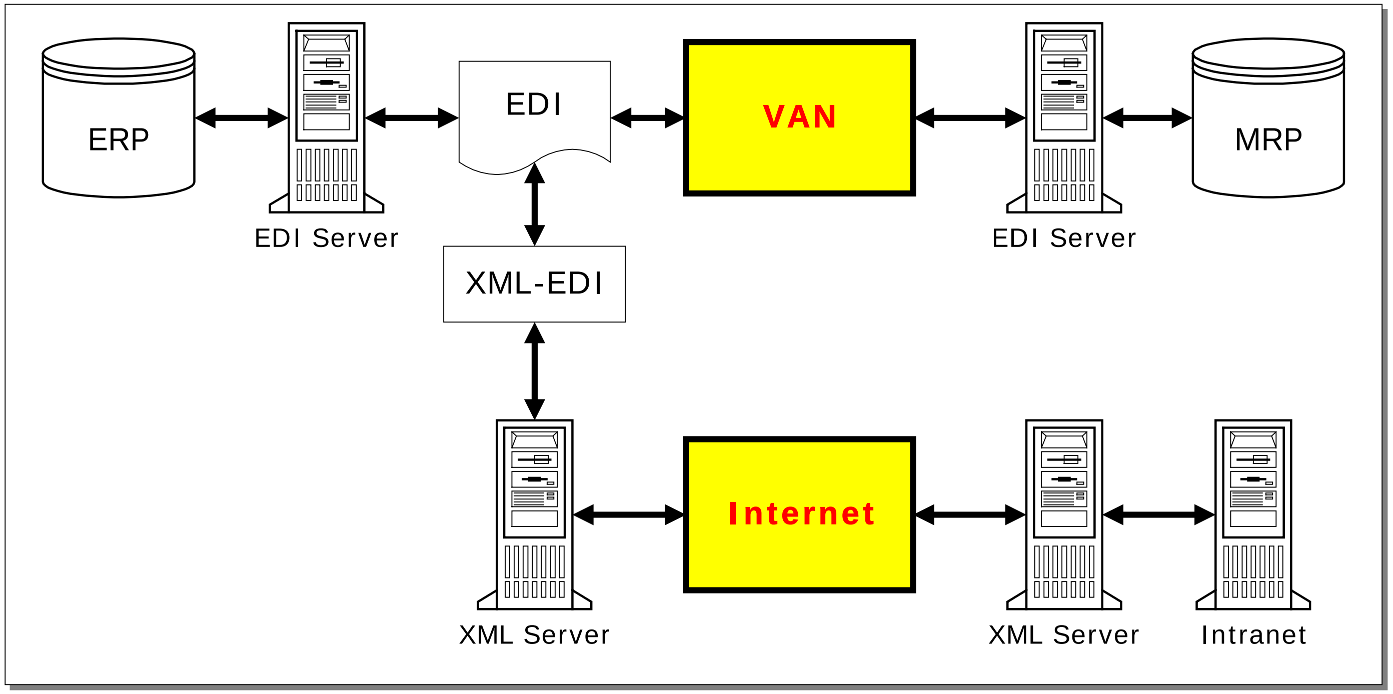
Convert EDI to XML

Header:

Segment:

```
<?xml version="1.0" ?>
<purchase-order>
    <header>
        <po-number>1234</po-number>
        <date>1999-12-31</date>
    </header>
    <order items="1" >
        <item>
            <part-no>097251</part-no>
            <description>XML</description>
            <quantity>4</quantity>
            <unit-price>11.99</unit-price>
            <price>47.96</price>
        </item>
        <tax type="sales" >
            <tax-unit>VA</tax-unit>
            <calculation>0.045</calculation>
            <amount>2.16</amount>
        </tax>
    </order>
    ...
```

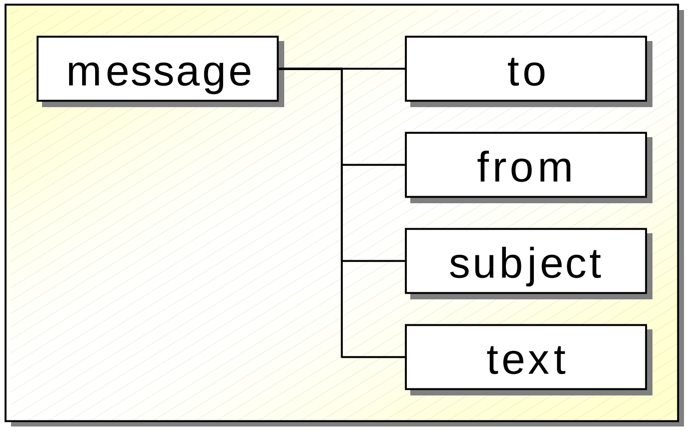
XML-EDI Trading System



XML Syntax

XML Example

```
<?xml version="1.0"?>
<message>
    <to>to@xml.com</to>
    <from>from@xml.com</from>
    <subject>XML Subject</subject>
    <text>XML Text</text>
</message>
```



XML Elements

Elements are the basic building blocks of XML. They contain:

- Other Elements
- Character Data
- Character References
- Entity References
- Comments
- Process Instructions
- CDATA Sections

Document Rules

A well-formed XML document must follow the rules:

- No unclosed tags.

```
<to>to@xml.com
```

- No overlapping tags.

```
<message><to>to@xml.com</message></to>
```

- Attribute values must be enclosed in quotes(").

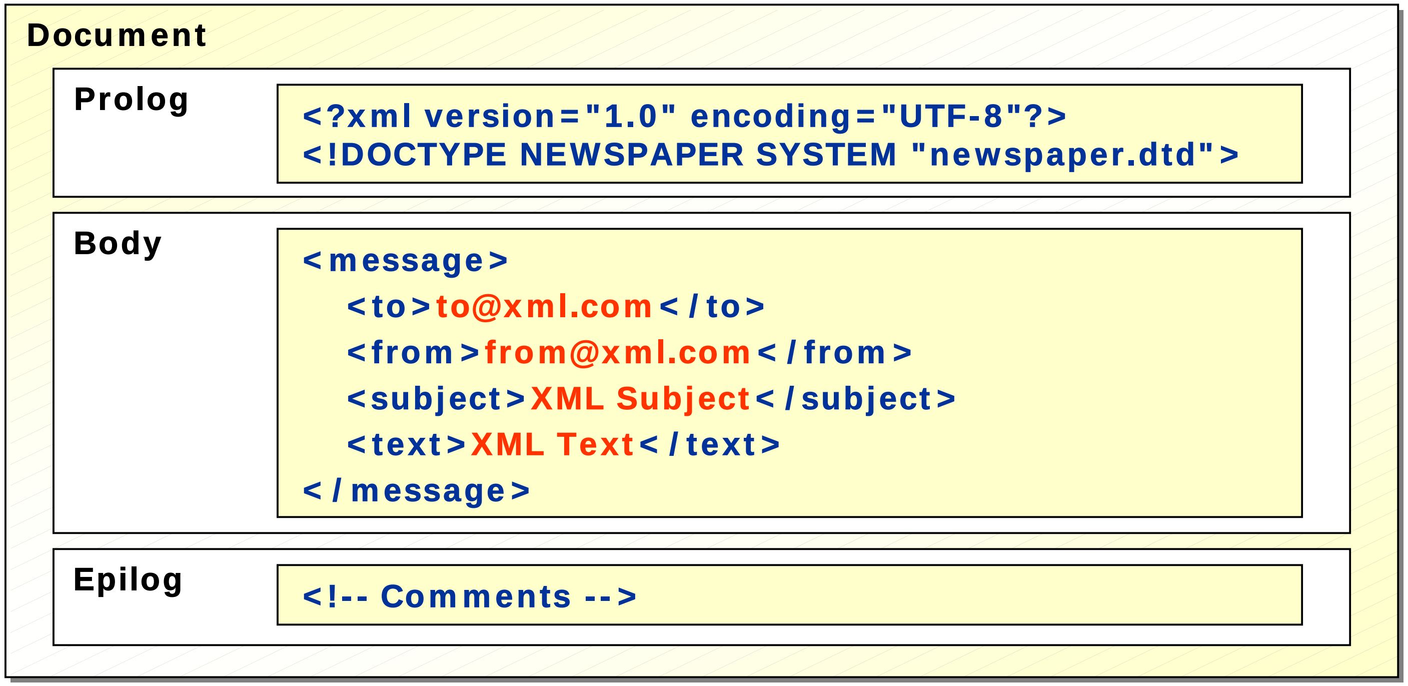
```
<customer ID="AB1" Index="1">
```

- The text characters (<), (>) and (") must be represented by "character entities".

Document Parts

A well-formed XML document is included:

- An optional **prolog**.
- **Body** (elements and character data).
- An optional **epilog**.



```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE NEWSPAPER SYSTEM "newspaper.dtd">
```

```
<message>
   <to>to@xml.com</to>
   <from>from@xml.com</from>
   <subject>XML Subject</subject>
   <text>XML Text</text>
</message>
```

```
<!-- Comments -->
```

Prolog

The prolog contains a declaration that identifies the document as an XML document.

```
<?xml version="1.0" encoding="UTF-8" standalone=“yes” ?>
```

Attributes:

- **version:** The version of XML. (Not optional)
- **encoding:** The character set.
- **standalone:** To reference an external entity or DTD. If no references, then “yes”.

Tags

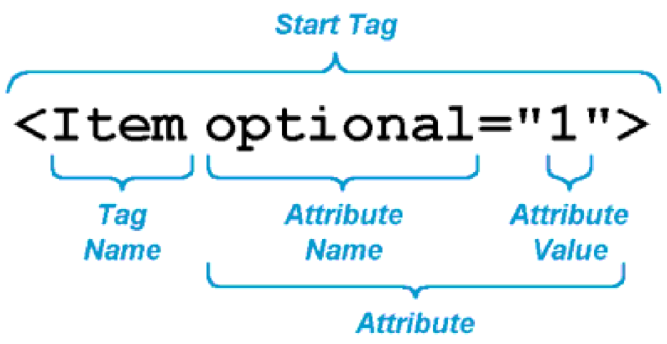
- Start Tag and End Tag

<subject>XML</subject>

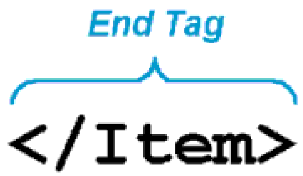
- Empty Tag

<log></log>

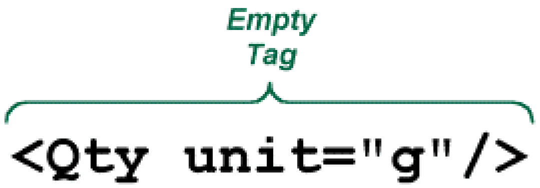
<log/>



```
<Item optional="1">
```



```
</Item>
```

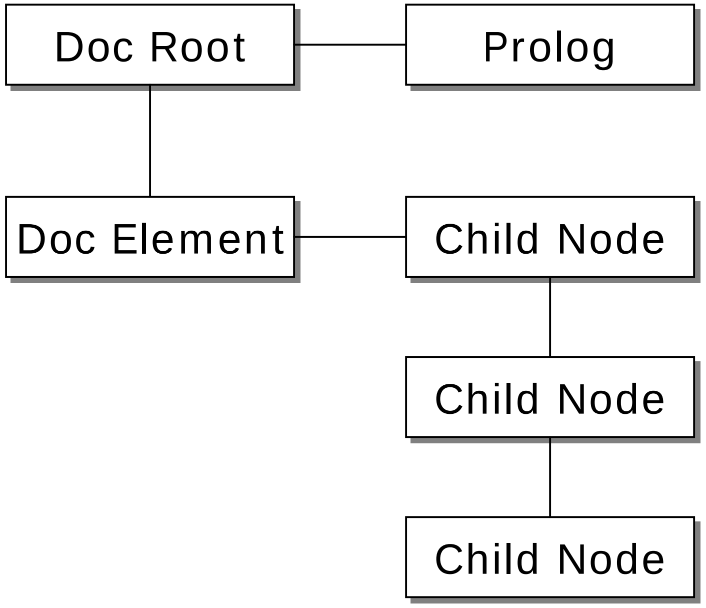


```
<Qty unit="g"/>
```

Hierarchical Tree

Well-formed XML is defined as being in the form of a **Hierarchical** tree:

- Document Root.
- Document Element.
- Child Node



Hierarchical Tree

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

<message>
   <to>to@xml.com</to>
   <from>from@xml.com</from>
   <subject>XML Subject</subject>
   <text>XML Text</text>
</message>

<!-- This is comment -->
```

```
DOCUMENT                                        *
|---XMLDECL
|   |---ATTRIBUTE version "1.0"
|---ELEMENT message
|   |---ELEMENT to
|   |   +---PCDATA "to@xml.com"
|   |---ELEMENT from
|   |   +---PCDATA "from@xml.com"
|   |---ELEMENT subject
|   |   +---PCDATA "XML Subject"
|   +---ELEMENT text
|       +---PCDATA "XML Text"
+---COMMENT
```

*: The tree format is parsed by Microsoft command line XML tool - MSXML.exe

Attributes

Attributes: A qualifier on an XML tag that provides additional information to an element without making the attributes a part of the content of that element.

```
<Login>
  <log Date="2000/3/28" User=”guest"/>
  <log Date="2000/3/29" User=”xml"/>
</Login>
```

Character References

Character References represent a displayed character preceded by **&#** (decimal) or **&#x** (hexadecimal) followed by **;**.

&#169; or **&#xA9;** represents **©**

&#174; or **&#xAD;** represents **®**

Entity References

Entity References: A substitution for the reference when the XML document is parsed. It may reference a predefined entity like &lt; or it may reference one that is defined in the DTD*.

Entity	Usage
&amp;	& (ampersand)
&apos;	' (apostrophe)
&lt;	< (opening angle bracket)
&gt;	> (closing angle bracket)
&quot;	" (double quotation mark)

AT&amp;T represents **AT&T**

Comments

Text in an XML document that is ignored.

A comment is enclosed in a comment tag.

```
<!-- This is a comment -->
```

Process Instructions

An XML file can also contain processing instructions that give commands or information to an application that is processing the XML data.

Processing instructions have the following format:

<?target instructions?>

<?xml-stylesheet?>

CDATA Sections

CDATA Sections are a method of including text that would be recognized as markup.

CDATA sections are typically used to show examples of XML syntax.

```
<![CDATA[<test>A sample</test>]]>
```

represent:

```
<test>A sample</test>
```

Summary of XML Syntax

Syntax	Component
<?xml version="1.0" encoding="UTF-8" standalone=“yes” ?>	XML declaration
<tagname>	Start Tag
<tagname attribute=“value”>	Start Tag with attribute
</tagname>	End Tag
<tagname/>	Empty Tag
<?xml-stylesheet?>	Process Instruction
<!-- comment -->	Comment
<![CDATA[string]]>	Unparsed character data
&#decimal;	Character number by decimal
&#xHEX;	Character number by hexadecimal
&ref;	Pre-defined XML entity in DTD*

Document Type Definition (DTD)

Document Type Definition

The DTD specifies constraints on the valid tags and tag sequences that can be in the document.

The DTD has a number of shortcomings which has led to various schema proposals.

The DTD includes the Internal subset, defined inside the XML file, and the External subset, which consists of the definitions contained in external **`*.dtd`** files that are referenced in the local subset using a parameter entity.

Internal Subset DTD

The part of the DTD that is defined within the current XML file.

```
<?xml version="1.0"?>
<!DOCTYPE EMAIL [
	<!ELEMENT EMAIL (FROM, TO, SUBJECT)>
	<!ELEMENT FROM (#PCDATA)>
	<!ELEMENT TO (#PCDATA)>
	<!ELEMENT SUBJECT (#PCDATA)>
]>
<EMAIL>
	<FROM>from@xml.com</FROM>
	<TO>to@xml.com</TO>
	<SUBJECT>DTD</SUBJECT>
</EMAIL>
```

DTD1.XML

External Subset DTD

The part of the DTD that is defined by references to external *.dtd files.

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

<!DOCTYPE EMAIL SYSTEM "DTD2.dtd">

<EMAIL>
    <FROM>from@xml.com</FROM>
    <TO>to@xml.com</TO>
    <SUBJECT>External DTD</SUBJECT>
</EMAIL>
```

DTD2.XML

External Subset DTD

- **SYSTEM:** to specify the location of the DTD file. (The path is relative to the location of the XML document)

```
<!DOCTYPE EMAIL SYSTEM "DTD2.dtd">
<!DOCTYPE EMAIL SYSTEM "http://xml/dtd2.dtd">
```

- **PUBLIC:** to specify the DTD file using a **URI** (Uniform Resource Identifier) or **unique name** which parser would be able to resolve it.

```
<!DOCTYPE EMAIL PUBLIC "URI/DTD">
<!DOCTYPE EMAIL PUBLIC "URI/DTD" "http://xml/dtd2.dtd">
```

DTD2.XML
DTD3.XML

DTD Markup Declarations

There are three types of DTD markup declarations used in DTD.

- **ELEMENT:** XML element type.
- **ATTLIST:** Attributes assigned to a specific element type.
- **ENTITY:** Declaration of reusable content.

ELEMENT Declaration

The soul of XML is the element. Element types are declared in DTD using the **ELEMENT** tags.

The ELEMENT consists of the combination of child element names, operators and the #PCDATA.

The three categories of Element types: empty, any and mixed.

```
<!ELEMENT EMAIL (FROM+, TO*, SUBJECT?)>
```

Operator (Mix-Element)

The **Operators** are used to denote cardinality and indicate how elements and character data may be combined.

Operator	**Meaning**
,	Strict sequence
|	Choice (Or)
?	Optional
*	Zero or more
+	One or more

Operator (Mix-Element)

- **Strict sequence:** <!ELEMENT EMAIL (FROM, TO)>

```
<EMAIL>
      <FROM>...</FROM>
      <TO>...</TO>
</EMAIL>
```

- **Choice:** <!ELEMENT EMAIL (FROM, (TO | CC))>

```
<EMAIL>
      <FROM>...</FROM>
      <TO>...</TO>
</EMAIL>
```

or

```
<EMAIL>
      <FROM>...</FROM>
       <CC>...</CC>
 </EMAIL>
```

Operator (Mix-Element)

- **Optional:** The element may or may not appear.

  `<!ELEMENT EMAIL (FROM?)>`

- **Zero or more:** `<!ELEMENT EMAIL (FROM, TO*)>`

```
<EMAIL>
      <FROM>...</FROM>
</EMAIL>
```

or

```
<EMAIL>
      <FROM>...</FROM>
       <TO>...</TO>
       <TO>...</TO>
 </EMAIL>
```

Operator (Mix-Element)

- **One or more:** <!ELEMENT EMAIL (FROM, TO**+**)>

```
<EMAIL>
    <FROM>...</FROM>
    <TO>...</TO>
</EMAIL>
```

or

```
<EMAIL>
    <FROM>...</FROM>
    <TO>...</TO>
    <TO>...</TO>
</EMAIL>
```

Operator (Example)

```
<!ELEMENT EMAIL (FROM, TO+, (CC | BCC)*, SUBJECT?)>
```

This declaration indicates that:

- The FROM element must appear exactly once.
- The TO element is required and can appear more than once.
- Either CC or BCC element can be chose. If it appears, it can appear once or more.
- The Subject element is optional, but it can appear only once if included.

Empty-Element Declaration

To declare that an element can not contain any content, you can use the keyword EMPTY in the element declaration, as shown here:

```
<!ELEMENT EmptyType EMPTY>
```

As defined:

```
<EmptyType/>
```

Any-Element Declaration

The type of element can contain any content allowed by the DTD in any order. It is usually used for the root element of a general-purpose XML document.

```
<!ELEMENT AnyType ANY>
```

ATTLIST Declaration

In addition to defining the structure of an element and the kind of content it contains, you can associate attributes with an element.

In XML, attributes are declared in the DTD using the following syntax:

```
<!ATTLIST ElementName AttributeName Type Default>
```

Attribute Type

Type	Usage
CDATA	Character data (string).
ENTITY	The name of an entity defined in the DTD.
ENTITIES	A space-separated list of entities.
ID	A unique name that no other ID attribute shares.
IDREF	A reference to an ID defined in the DTD.
IDREFS	A series of IDREF delimited by white space.
NMTOKEN	A valid XML name.
NMTOKENS	A series of NMTOKEN delimited by white space.
NOTATION	The name of a DTD-specified notation which describes a non-XML data format. (image files)*
(Enumerated)	Attribute value must match one of the included values.

*: NOTATION is a obsolescing specification followed by W3C.

Attribute Default

Default	Usage
#REQUIRED	The attribute value must be specified in DTD.
#IMPLIED	This attribute is optional.
#FIXED “default”	This attribute must have the default value. If the attribute does not appear, the default value is assumed.
“default”	Identifies a default value for an attribute. If the attribute does not appear, the default value is assumed.

ATTLIST (Example)

```
<!ELEMENT EMAIL (FROM, TO)>
<!ATTLIST EMAIL ENCRYPTED CDATA #IMPLIED>
...
```

Optional

```
<EMAIL ENCRYPTED="128"> ... </EMAIL>
```

or

```
<EMAIL> ... </EMAIL>
```

ATTLIST (Example)

```
<!ELEMENT EMAIL (FROM, TO)>
<!ATTLIST EMAIL PRIORITY (NORMAL|LOW|HIGH) "NORMAL">
...
```

Enumerated

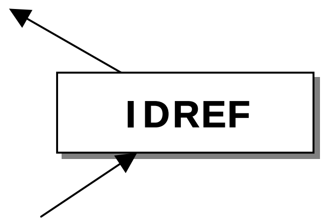
Default

```
<EMAIL PRIORITY="NORMAL"> ... </EMAIL>
    or
<EMAIL PRIORITY="LOW"> ... </EMAIL>
    or
<EMAIL PRIORITY="HIGH"> ... </EMAIL>
```

ATTLIST (Example)

```
<!ELEMENT Person (#PCDATA)>
<!ATTLIST Person PID ID #REQUIRED>
<!ELEMENT Customer EMPTY>
<!ATTLIST Customer CID IDREF #REQUIRED>

<Person PID="11111"> ... </Person>
```



```
<Customer CID="11111”/>
```

ATTLIST (Example)

```
<!NOTATION gif SYSTEM "Iexplore.exe">
<!NOTATION jpg SYSTEM "Iexplore.exe">

<!ELEMENT PICTURE (#PCDATA)>
<!ATTLIST PICTURE Photos ENTITIES #IMPLIED>
<!ENTITY scene1 SYSTEM "scene1.gif" NDATA gif>
<!ENTITY scene2 SYSTEM "scene2.jpg" NDATA jpg>
...

<PICTURE Photos="scene1 scene2"> ... </PICTURE>
```

Document Object Model

XML Parser

A module that reads in XML data from an input source and breaks it up into chunks so that your program knows when it is working with a tag, an attribute, or element data.

- A non-validating parser ensures that the XML data is well formed, but does not verify that it is valid.
- A validating parser is a parser which ensures that an XML document is valid, as well as well-formed.

Approaches of XML Parser

There are two approaches to implementing an XML parser:

- **Tree-Based Parsers:** XML Document Object Model **(XML DOM)**.
- **Event-Driven Parsers:** Simple API for XML **(SAX)**.

XML DOM

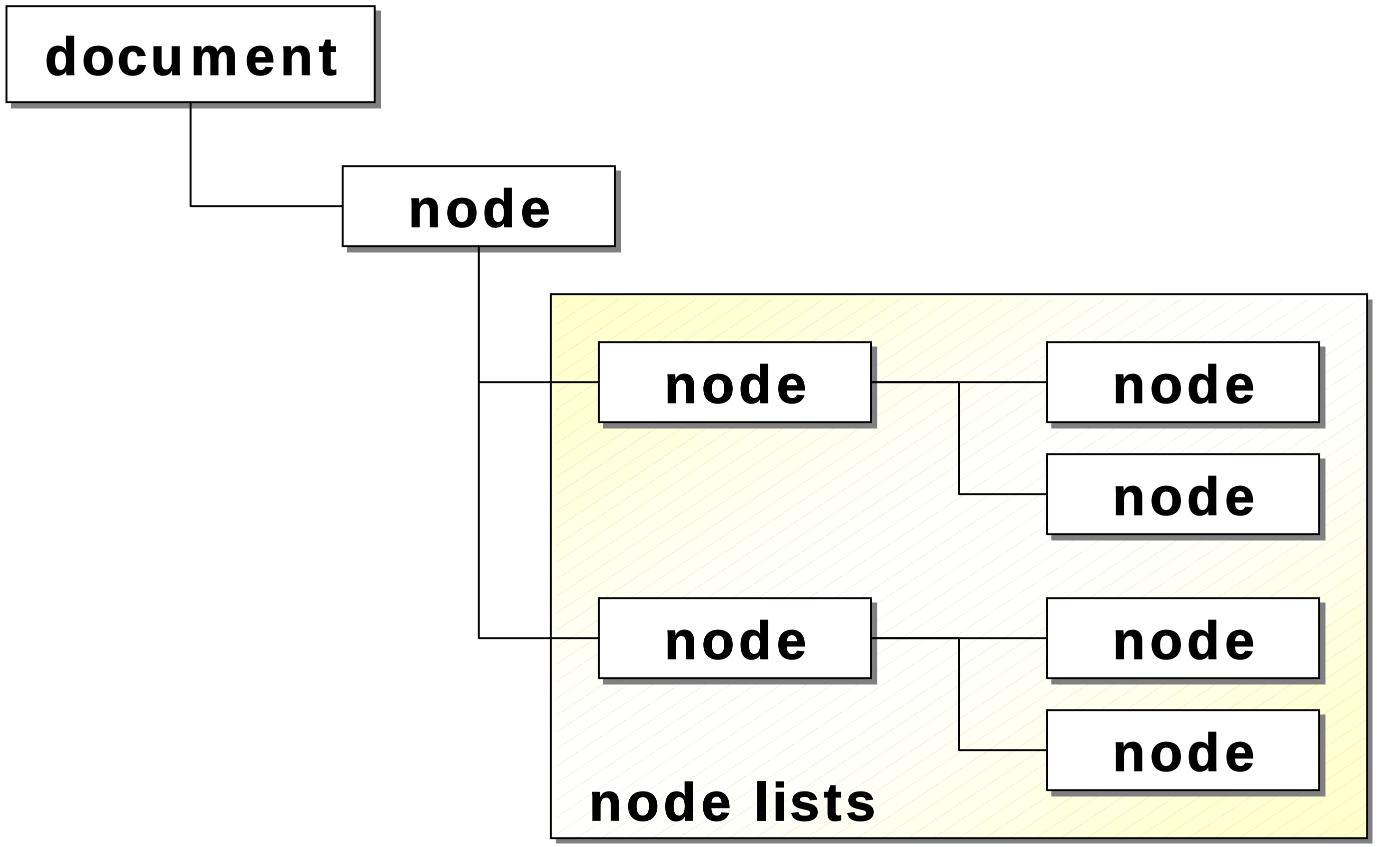
A Document Object Model for XML is an object model that exposes the contents of an XML document. The W3C's DOM Level 1 Specification currently defines what a DOM should expose as Properties, Methods, and Events.

A DOM is a tree structure, where each node contains one of the components from an XML structure. The XML object model consists of four basic objects:

- **document** object—the XML data source
- **node** object—a parent node or one of its child nodes
- **nodeList** object—a list of "sibling" nodes
- **parseError** object—a non-content-bearing object used for retrieving parsing error information

For more information about XML DOM: http://www.w3.org/TR/REC-DOM-Level-1/

XML Object Model



Microsoft XML DOM

Microsoft exposes the XML DOM via a set of standard COM interfaces in **Msxml.dll** (version 2.0 shipped with IE 5.0). To reference MS XML DOM in:

- Visual Basic:

```
Dim xmlDoc As MSXML.DOMDocument
Set xmlDoc = New MSXML.DOMDocument
```

- VBScript in ASP:

```
Set xmlDoc = Server.CreateObject("Microsoft.XMLDOM")
```

- JavaScript:

```
var xmlDoc = new ActiveXObject("Microsoft.XMLDOM");
```

- Applet:

```
<APPLET CODE=com.ms.xml.dso.XMLDSO.class ID=xmlDoc>
```

Process XML Document

To load an XML document, first create an instance of the MSXML.DOMDocument class or Microsoft.XMLDOM. Once obtain a valid reference, open a file using the **Load** method. The MSXML parser can load XML documents from a local disk, over the network using UNC references, or via a URL.

```
Dim xmlDoc As MSXML.DOMDocument
Set xmlDoc = New MSXML.DOMDocument

If xmlDoc.Load("XML.xml") Then
      ' The document loaded successfully.
Else
      ' The document failed to load.
End If
```

Process XML Document

It is important to examine a document's **ReadyState** property to ensure a document is ready before you start to examine its contents. The ReadyState property returns one of five possible values:

Value	State
0	Uninitialized: loading has not started
1	Loading: while the load method is executing
2	Loaded: load method is complete
3	Interactive: User can interact with the object even though it is not fully loaded.
4	Completed: data is loaded, parsed and available for read/write operations.

Process XML Document

```
<script language="JavaScript">
    var xmlDoc = new ActiveXObject("microsoft.xmldom");
    xmlDoc.load("XML.xml");

    function loadXML()
    {
        if (xmlDoc.ReadyState == "4")
            ...
        else
            window.setTimeout("loadXML()", 5000);
    }
</script>
```

XMLDOM.htm

Retrieving Information

- **documentElement** property: Identifies the root node.

  ```
  rootNode = xmlDoc.documentElement
  ```

- **childNodes** property: Returns a node list containing all the available children of the current node.

  ```
  objNodeList = rootNode.childNodes
  ```

- **length** property: Returns the number of items in a collection.

  ```
  rootNode.childNodes.length
  ```

- **item** method: Accesses an individual node in a document tree.
- **text** property: Gets or sets the text for a node.

  ```
  rootNode.childNodes.item(i).text
  ```

- **hasChildNodes** method: Returns true if the specified node has one or more children

  ```
  if objNodeList.hasChildNodes then
  ```

Retrieving Information

IXMLDOMNode interface to read/write to individual node elements.

- NodeType: exposes an enumeration of DOMNodeType items.
- NodeTypeString:retrieve a textual string for the node type.

DOMNodeType	Example
NODE_ELEMENT	<EMAIL>...</EMAIL>
NODE_ATTRIBUTE	<BOOK ISDN="1234">
NODE_TEXT	<NOTE>Text.</NOTE>
NODE_PROCESSING_INSTRUCTION	<?xml version="1.0"?>
NODE_DOCUMENT_TYPE	<!DOCTYPE EMAIL SYSTEM "XML.dtd">

Example

```
<SCRIPT LANGUAGE="JavaScript">
    var xmlDoc = new ActiveXObject("microsoft.xmldom");
    xmlDoc.load("XMLDOM.xml");

    function start()
    {
      var newHTML = "";
      rootNode= xmlDoc.documentElement;
      for (el=0; el<rootNode.childNodes.length; el++)
      {
        if (el != rootNode.childNodes.length-1)
          {
          newHTML = newHTML + "<SPAN STYLE='font-weight:bold;font-size:16'>" +
            rootNode.childNodes.item(el).nodeName + ": </SPAN><SPAN STYLE='font-weight:normal'>" +
            rootNode.childNodes.item(el).text + "</SPAN><P>";
          }
        else
          {
          newHTML = newHTML + "<HR><SPAN STYLE='font-weight:normal'>" +
            rootNode.childNodes.item(el).text + "</SPAN><P>";
          }
        }
      content.innerHTML = newHTML;
      }
  </SCRIPT>
```

XMLDOM.htm

Demonstration

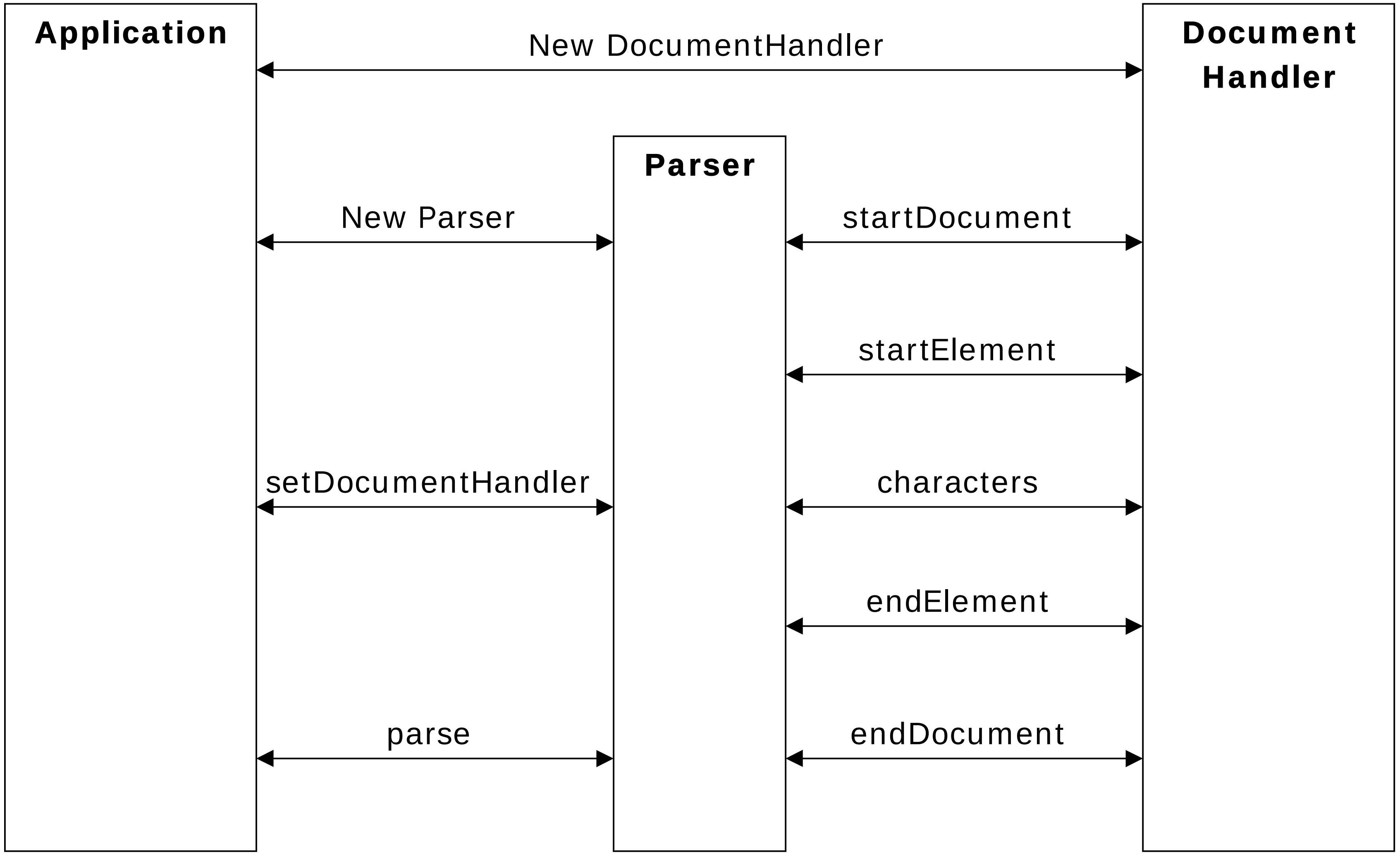
Simple API for XML (SAX)

Event-Driven SAX

The other way to process an XML file in Java is by using the Simple API for XML (SAX). SAX is a different style of interface from DOM (Document Object Model) and a simple Java interface that many Java parsers can use.

With SAX, the parser tells the application what is in the XML document by notifying the application of a stream of parsing Events.

SAX Structure



SAX Parser

A SAX parser is a class that implements the interface **org.xml.sax.Parser**.

To process an XML document, the programmer creates a class that implements interface **org.xml.sax.DocumentHandler**. The Parser object (**org.xml.sax.Parser**) reads the XML from its input source, calling the methods of the DocumentHandler when tags, input strings, and so on are recognized at the input.

SAX API

- **ParserFactory**: Class defines the makeParser() method, which uses the parser specified by the system property, org.xml.sax.parser.

- **Parser**: org.xml.sax.Parser interface defines methods like setDocumentHandler to set up event handlers and parse(URL) to actually do the parsing.

- **DocumentHandler**: 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.

Methods of SAX Parser

The methods of the DocumentHandler interface (org.xml.sax.DocumentHandler) in Java are:

```
public interface DocumentHandler {
  public abstract void setDocumentLocator (Locator locator);
  public abstract void startDocument () throws SAXException;
  public abstract void startElement (String name, AttributeList atts) throws
     SAXException;
  public abstract void endElement (String name) throws SAXException;
  public abstract void endDocument () throws SAXException;
  public abstract void characters (char ch[], int start, int length) throws
     SAXException;
  public abstract void ignorableWhitespace (char ch[], int start, int length)
     throws SAXException;
  public abstract void processingInstruction (String target, String data) throws
     SAXException;
  }
```

Methods of SAX Parser

The methods of the DocumentHandler interface in VB (SAX.DLL) are:

```
Dim WithEvents parser As SAXParser
Set parser = New SAXParser
parser.parseFile txtXML.Text

Sub parser_setLocator(locator)
Sub parser_startDocument()
Sub parser_startElement(ByVal sName As String, ByVal pAttributeList As
    SAXLib.ISAXAttributeList)
Sub parser_characters(ByVal sCharacter As String, ByVal iLength As Long)
Sub parser_comment(ByVal sComment As String)
Sub parser_processingInstruction(ByVal sTarget As String, ByVal sData As
    String)
Sub parser_endElement(ByVal sName As String)
Sub parser_endDocument()
```

Import the Classes

```
import java.io.*;
import org.xml.sax.*;
import org.xml.sax.helpers.ParserFactory;
import com.sun.xml.parser.Resolver;

public class ClassName extends HandlerBase{  ...
```

- org.xml.sax package defines all the interfaces use for the SAX parser.
- ParserFactory class creates the instance.
- Resolver class is needed to create an input source for the parser to operate on.

Setting up the Parser

```
public static void main (String argv []) throws IOException {
        InputSource   input;
        ...
        try {
                ...
                // Turn the filename into an XML input source
                input = Resolver.createInputSource (new File (argv [0]));
                // Get an instance of the non-validating parser.
                Parser   parser;
                parser = ParserFactory.makeParser ("com.sun.xml.parser.Parser");
                parser.setDocumentHandler ( new Echo() );
                // Parse the input
                parser.parse (input);
        } catch (Throwable t) { ... }
        System.exit (0);
}
```

Implementing DocumentHandler

```
public void startDocument ()
    throws SAXException    {    }

public void endDocument ()
    throws SAXException    {    }

public void startElement (String name, AttributeList attrs)
    throws SAXException    {    }

public void endElement (String name)
    throws SAXException    {    }

public void characters (char buf [], int offset, int len)
    throws SAXException    {    }
```

Compiling the Program

Windows:

```
javac -classpath
%XML_HOME%\xml.jar;%JAVA_HOME%\lib\classes.zip <java>
```

UNIX:

```
javac -classpath ${XML_HOME}/xml.jar:${JAVA_HOME}/lib/classes.zip
<java>
```

SAX (Example)

XML:

```
<?xml version="1.0"?>
<EMAIL>
    <SUBJECT>External DTD</SUBJECT>
</EMAIL>
```

SAX Parser:

```
startDocument()
startElement(“EMAIL”)
startElement(“SUBJECT”)
characters(“External DTD”)
endElement(“SUBJECT”)
endElement(“EMAIL”)
endDocument()
```