

SWE 363: Web Engineering & Development

Module 8-1

Data Description and Transformation – XML



Objectives

- ❑ Learn the basics of XML
- ❑ Learn about the role of DTD and XML Schemas XSD
- ❑ Learn how to manipulate and transform XML document

- ❑ What is XML?
- ❑ How XML differs from HTML
- ❑ XML – Document Structure
- ❑ Namespaces
- ❑ Defining XML Data Formats
 - Document Type Definition (DTD)
 - XML Schemas (XSD)
- ❑ XML Parser
- ❑ XML DOM
- ❑ XML Querying and Transformation
 - XSLT
 - XPath
- ❑ XML Processing

Introduction

❑ Programming languages

- to **build** applications, games, etc..
- Examples: Java, C++, etc.

❑ Query Languages

- to communicate with **DB**
- Example: SQL, XQuery

❑ Scripting Languages

- to write programs on a special run-time environment that **automate** the execution of tasks that could alternatively be executed one-by-one by a human operator
- Example: JavaScript, etc.

❑ Markup Languages

- to **annotate** text and embed tags in accurately styled electronic documents include HTML, **XML**, and XHTML



What is XML?

- ❑ XML = e**X**tensible **M**arkup **L**anguage
- ❑ A portable technology for **data representation**, **storage**, **processing** and **exchange**
- ❑ XML plays an important role in the **exchange of a wide variety of data** on the web
- ❑ XML defines a set of rules **for encoding documents** which is both human-readable and machine-readable
- ❑ All rules are defined in XML 1.0 specification developed by W3C an open standard
- ❑ XML documents are typically files with the *.xml extension*

- ❑ Many **parsers** or APIs (Application Programming Interface) are available to process the XML data
- ❑ An *XML parser* is responsible for
 - **identifying** components of XML documents and
 - then **storing** those components in a **data structure** for manipulation

XML Data Model: Example

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

The diagram illustrates the XML data model for the provided code. It shows four elements highlighted with blue boxes and labeled "Element":

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

The root element `<note>` and the closing tag `</note>` are not highlighted.

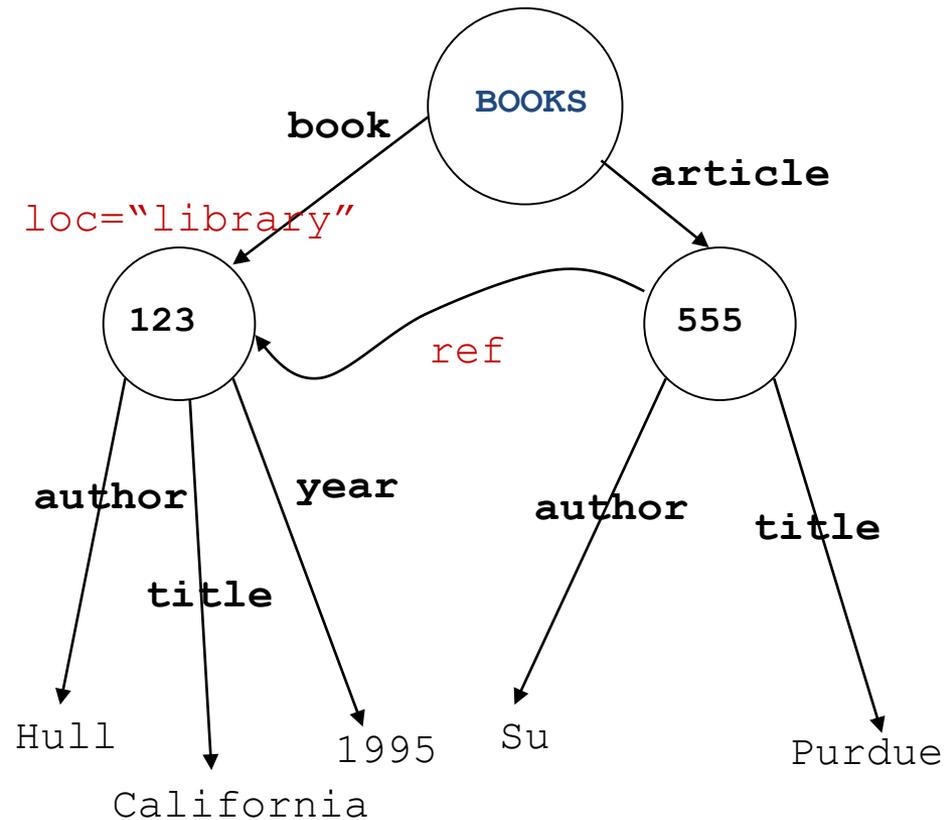
XML Data Model: Example



XML Data Model: Example..

```
<BOOKS>
  <book id="123" loc="library">
    <author>Hull</author>
    <title>California</title>
    <year> 1995 </year>
  </book>

  <article id="555" ref="123">
    <author>Su</author>
    <title> Purdue</title>
  </article>
</BOOKS>
```



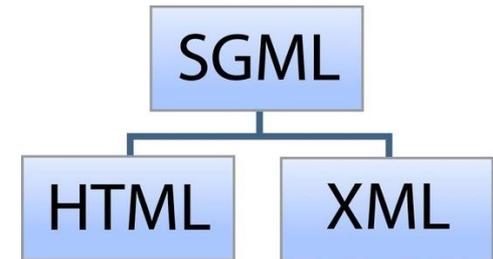
A common way of processing XML documents is to read them into memory in a **tree structure** - **arbitrary tree**, not binary

One way to process the in-memory XML document is to use a **traversal algorithm**

XML differs from HTML

- ❑ HTML and XML look similar because they are both **SGML** (Standard Generalized Markup Language)

- Both HTML and XML use elements enclosed in tags
- Both use tag attributes



- ❑ XML is **NOT** a replacement for HTML

- XML was designed to **structure, store** and **transport** data and to focus on what data is (not how to display)
- HTML was designed to **display** data and to focus on how data looks.

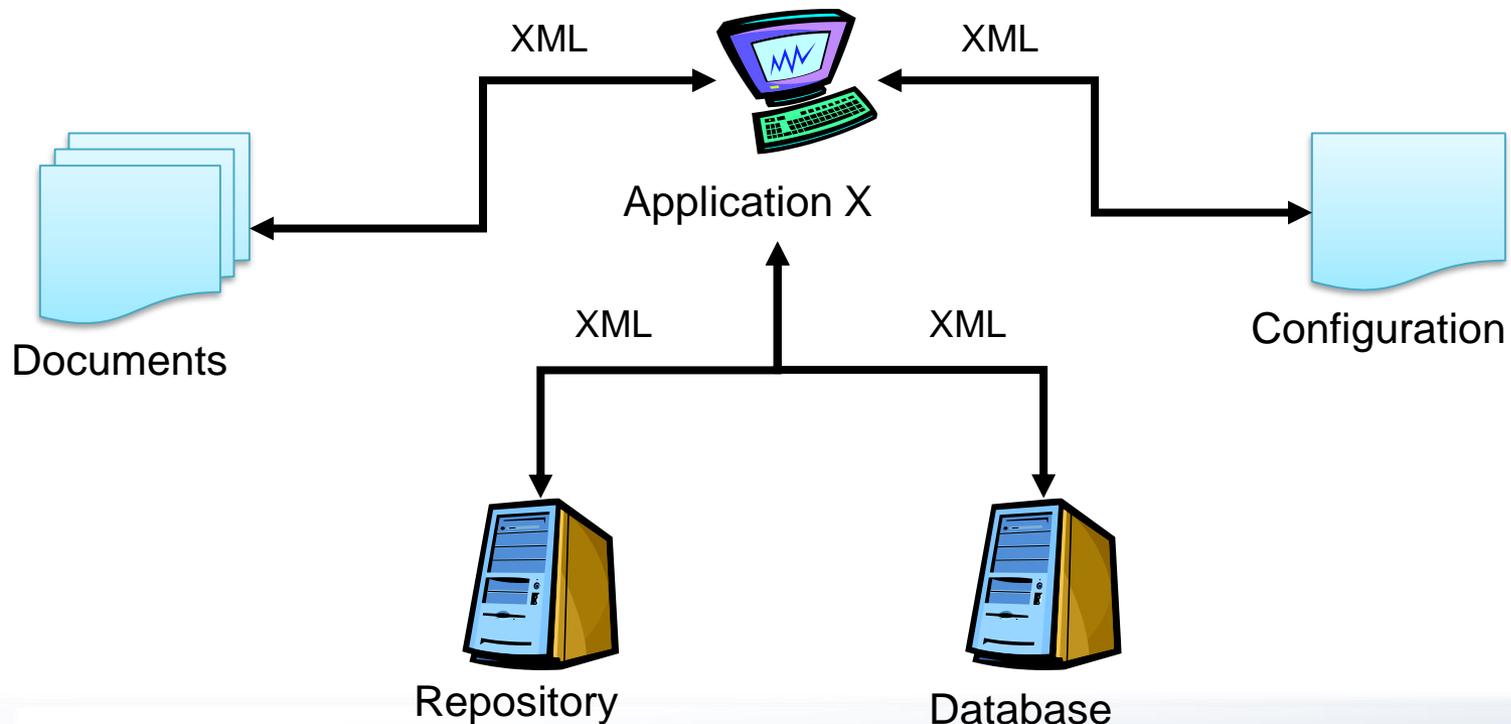
- ❑ HTML uses a fixed set of tags, whereas in XML, **you make up your own tags**

- XML tags are not predefined
 - XML allows the author to define his own tags and his own document structure
- HTML uses a fixed, predefined, unchangeable set of tags
 - The author of HTML documents can only use tags that are defined in the HTML standard

- ❑ HTML is for **humans**, while XML is for **computers**

XML Usage

- ❑ XML is **everywhere**
- ❑ Allows for high degree of interoperability, i.e. representation of data across **heterogeneous environments** (Cross platform)



XML Usage

- ❑ XML documents are used to **transfer/exchange data** often over the Internet.
- ❑ XML can be used for **offloading** and **reloading** of databases.
- ❑ XML can easily be **merged with style sheets** to create almost any desired output.
- ❑ Virtually, any **type of data can be expressed as an XML document**.

- ❑ **XML is a meta-language** >> used to create new web languages. A lot of Web languages are created with XML such as
 - WSDL (Web Services Description Language) for describing available web services
 - RSS (Really Simple Syndication) languages for news feeds
 - RDF (Resource Description Framework) and OWL (Web Ontology Language) for describing resources and ontology
 - SMIL (Synchronized Multimedia Integration Language) for describing multimedia for the web

XML Document - Example

```
<?xml version="1.0" encoding="UTF-8"?>
<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>
```

XML – Document Structure

- ❑ Every XML-document is **text-based**
 - => sharing data between different computers
 - => sharing data in internet
 - => platform independence

- ❑ An XML document is a structured **collection of text format markup tags**
- ❑ Each tag either defines:
 - some information used to describe how the document is to be interpreted
 - some data contained within the document

- ❑ Tags in XML documents fall into the following categories:
 - **Comments:**
 - XML, like Java and HTML, has a means of adding comments to a document
`<!-- comments in XML look like this -->`

XML – Document Structure..

➤ **XML Declaration:**

- It identifies the document as an XML document.

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

- The XML declaration is not required by browsers, but is required by most **XML processors** (>> so include it!)
- If present, the XML declaration must be at **top** of the XML document -not even whitespace should precede it
- `version="1.0"` is required
- encoding can be `"UTF-8"` (ASCII) or `"UTF-16"` (Unicode), or something else, or it can be omitted
- `standalone` tells whether there is a separate DTD

➤ **DTD declarations**

- DTD (Document Type Definition) specifies the semantic constraints

```
<!DOCTYPE addressBook SYSTEM .....path\addressBook.dtd">
```

XML – Document Structure..

➤ **Elements:**

- An XML element is everything from the element's start tag to the element's end tag. **It may Contain data values or other XML elements.**
- Elements are defined between “<” and “>” characters
- XML Elements have **simple naming rules**

➤ **Attributes:**

- Attributes are **name/value** pairs associated with elements
- Data can be stored in child elements or in attributes
- Should you avoid using attributes?

Here are some of the problems using attributes:

- attributes **cannot contain multiple values** (child elements can)
- attributes are **not easily expandable** (for future changes)
- attributes **cannot describe structures** (child elements can)
- attributes are **more difficult to manipulate** by program code
- attribute values are **not easy to test against a DTD** - which is used to define the legal elements of an XML document

Example: XML Attributes

Using attributes:

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

Using elements:

```
<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 Namespaces

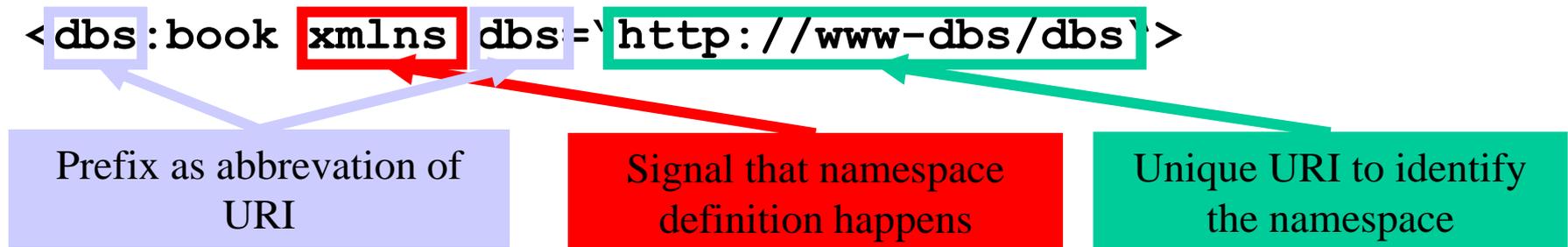
```
<library>
  <description>Library of the CS Department </description>
  <book bid="HandMS2000">
    <title> Principles of Data Mining</title>
    <description>
      Short introduction to <em>data mining</em>, useful
      for the IRDM course
    </description>
  </book>
</library>
```

Semantics of the **description** element is ambiguous
Content may be defined differently Renaming may be impossible (standards!)

An XML instance may contain element or attribute names from more than one XML vocabulary. If each vocabulary is given a **namespace**, the ambiguity between identically named elements or attributes can be resolved.

XML Namespaces..

- ❑ XML Namespaces provide a method to **avoid element name conflicts**.
- ❑ Namespaces identify that elements in the document come from different sources:
 - This often **results in a conflict when trying to mix XML documents** from different XML applications.
 - you can mix different namespaces in the same document
 - namespace **appears as prefix on element/attribute**
- ❑ At the top of each document, you need to declare the namespaces used by that document



XML Namespaces..

- ❑ Namespaces are first declared at the root element containing the elements belonging to the namespace.

- eg:

- `<Address : addressBook`

- `xmlns:Address="http://www.xyz.com/addressBook">`

- ❑ Elements in an XML document can be marked with a namespace.

- ❑ The namespace is a **prefix to the element name** –

- the format for the element tag is:

- `<namespace : elementName>`

- `Data`

- `</namespace : elementName>`

XML Namespaces: example

With namespace

```
<?xml version="1.0" encoding="UTF-8"?>
<Address:addressBook
  xmlns:Address="http://www.zyx.com/addressBook">
<Address:addressBook>
  <Address:entry list="personal">
    <Address:name>Ms Smith</Address:name>
    <Address:address>1 Central Rd, Sydney</Address:address>
    <Address:phone>555 5555</Address:phone>
  </Address:entry>
  <Address:entry list="business">
    <Address:name>Mr Suit</Address:name>
    <Address:address>1 George St, Sydney</Address:address>
    <Address:phone>555 6666</Address:phone>
  </Address:entry>
</Address:addressBook>
```

Without namespace

```
<?xml version="1.0" encoding="UTF-8"?>
<addressBook>
  <entry list="personal">
    <name> Ms Smith </name>
    <address> 1 Central RD, Sydney </address>
    <phone>555 5555</phone>
  </entry>
  <entry list="business">
    <name> Mr Suit </name>
    <address> 1 George St, Sydney </address>
    <phone>555 6666</phone>
  </entry>
</addressBook>
```

What else..

- ❑ As a human, it's easy for you to see the structure:
 - Elements, Attributes, Entity references, Comments, Processing instructions, CDATA sections

- ❑ But to be any real use, XML has to be read by a computer application
 - not just read it as a single chunk of ASCII text
 - need to parse XML document to recognize the structure

- ❑ Human beings are quite good at discovering the meaning in written text.
 - E.g. Give to x the value 5 \equiv Give the value 5 to x (same word, different order)

- ❑ XML is a well-structured format that can easily be parsed by computer programs.

Defining XML Data Formats

Document Type Definitions

- ❑ Often you need to add a *grammar* (Rules) to structure a XML document
 - to define allowed tags, elements, attributes and their order and data types
 - to allow validation of XML documents to check its conformity to the grammar expressed by the schema

- ❑ Well formed vs Valid
 - "Well Formed" XML document follows the basic syntax of XML
 - "Valid" XML document conforms to the extra rules contained in a the corresponding DTD (or associated XML Schema) file.
 - e.g. each book can have many authors, but only one publisher

- ❑ An XML document validated against a DTD is both "Well Formed" and "Valid".

❑ Linking DTD and XML Docs

- `<!DOCTYPE note SYSTEM "Note.dtd">`

keywords

Root element

URI for the DTD

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

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

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

Document Type Definitions..

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

PCDATA: parsed
character data

- ❑ The purpose of a DTD is to define the structure of an XML document. It defines the structure with a list of legal elements:
 - `!DOCTYPE note` defines that the root element of the document is `note`
 - `!ELEMENT note` defines that the note element must contain the 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`"

- ❑ Key DTD's limitations:
 - It doesn't support basic **data types**: int, doubles, dates, ... (everything is text)
 - **No structured**, self-definable data types
 - It doesn't use **XML syntax**
 - It doesn't provide enough support for **namespace**
 - Very limited for **reusability** and **extensibility**

Alternative ⇒ XML Schema

- Expressed in **XML** (use XML syntax)
- More **expressive** than DTDs
- Supporting **Namespace** and **import/include**
- More **data types**
- Able to create **complex data**
- Usable by **various** XML applications
- More ...

Simplified XML Schema Example

book.xsd

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="book">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="title" type="xs:string"/>
        <xs:element name="author" type="xs:string"/>
        <xs:element name="qualification" type="xs:string"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

```
<?xml version="1.0"?>
<book xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xs:noNamespaceSchemaLocation="book">
  <title> Being a Dog Is a Full-Time Job</title>
  <author>Charles M. Schulz</author>
  <qualification> extroverted beagle </qualification>
</book>
```

XML document

XML Schema data types

- XML Schemas have predefined data types

Data Type	Description	Facets
string	Represents character strings.	length, pattern, maxLength, minLength, enumeration, whiteSpace
boolean	Represents Boolean values, which are either true or false.	pattern, whiteSpace
decimal	Represents arbitrary precision numbers.	enumeration, pattern, totalDigits, fractionDigits, minInclusive, maxInclusive, maxExclusive, whiteSpace
float	Represents single-precision 32-bit floating-point numbers.	pattern, enumeration, minInclusive, minExclusive, maxInclusive, maxExclusive, whiteSpace
More....		

Sample XSD

```
<xs:element name="Customer_dob"  
  type="xs:date"/>
```

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

```
<xs:element name="OrderID"  
  type="xs:int"/>
```

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

Sample XML

```
<Customer_dob>  
  2000-01-12T12:13:14Z  
</Customer_dob>
```

```
<Customer_address>  
  99 London Road  
</Customer_address>
```

```
<OrderID>  
  5756  
</OrderID>
```

```
<Body>  
  (a type can be defined as  
  a string but not have any  
  content; this is not true  
  5. of all data types, however).  
</Body>
```

An XML Document

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<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>
```

- root element, "**shiporder**"
- required attribute called "**orderid**"

- **xmlns:xsi="....."** tells the XML parser that this document should be validated against a schema.
- **xsi:noNamespaceSchemaLocation="shiporder.xsd"** specifies WHERE the schema resides

- 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" elements.

How to create an XML Schema

- simply follow the structure in the XML document and define each element as you find it.

```
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
...
</xs:schema>
```

- `xs` is the **standard namespace**, and the associated URI is the **Schema language definition**, which has the standard value: <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.

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

The child elements of the "`shiporder`" element is surrounded by a **`xs:sequence`** element that defines an ordered sequence of sub elements

how to create an XML Schema..

- ❑ "orderperson" element is defined as a **simple type** (because it does not contain any attributes or other elements).

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

- ❑ Next, define two elements that are of the **complex type**: "shipto" and "item"

```
<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.

how to create an XML Schema..

- ❑ Now you can define the "item" 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>
```

- ❑ To declare the **attribute** of the "shiporder" element. Since this is a required attribute → specify use="required".

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

- Note: The attribute declarations must always come last:

```
<?xml version="1.0" encoding="UTF-8" ?>
<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>
```

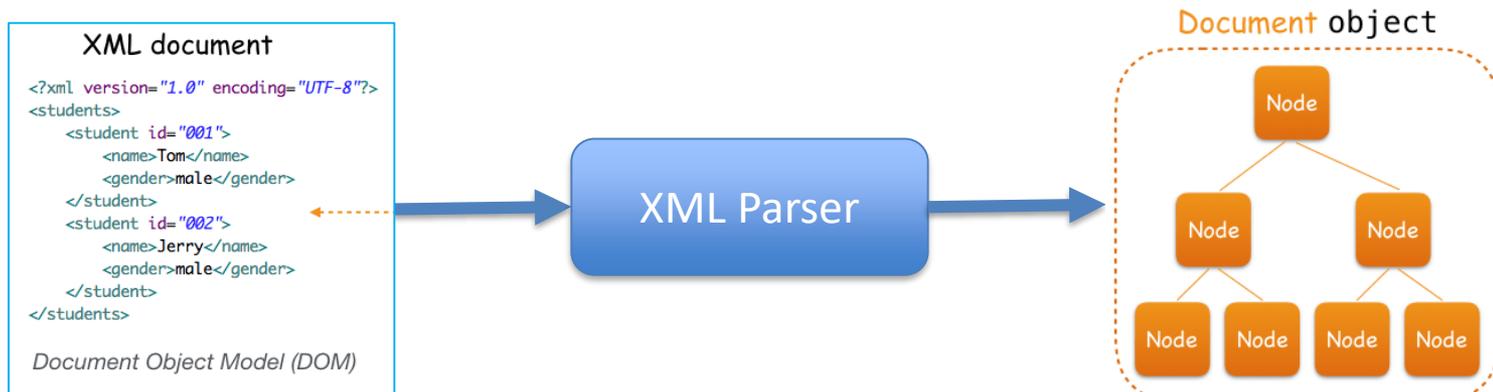
Restrictions for Datatypes

Constraint	Description
enumeration	Defines a list of acceptable values
fractionDigits	Specifies the maximum number of decimal places allowed. Must be ≥ 0
length	Specifies the exact number of characters or list items allowed. Must be ≥ 0
maxExclusive	Specifies the upper bounds for numeric values (the value must be $<$ this value)
maxInclusive	Specifies the upper bounds for numeric values (the value must be \leq this value)
maxLength	Specifies the maximum number of characters or list items allowed. Must be ≥ 0
minExclusive	Specifies the lower bounds for numeric values (the value must be $>$ this value)
minInclusive	Specifies the lower bounds for numeric values (the value must be \geq this value)
minLength	Specifies the minimum number of characters or list items allowed. Must be ≥ 0
pattern	Defines the exact sequence of characters that are acceptable
totalDigits	Specifies the exact number of digits allowed. Must be > 0
whiteSpace	Specifies how white space (line feeds, tabs, spaces, and carriage returns) is handled

XML Parser

XML Parser

- ❑ All major browsers have a built-in **XML parser** to access and manipulate XML.
- ❑ **XML parsing** is the process of **reading** an XML document and **providing** an interface for client applications to work with XML documents.
- ❑ **XML parser** is a software library or a package that provides such interface
 - It **checks** for proper format of the XML document and may also **validate** the XML documents.
 - The goal of a parser is to **transform XML into a readable code**.

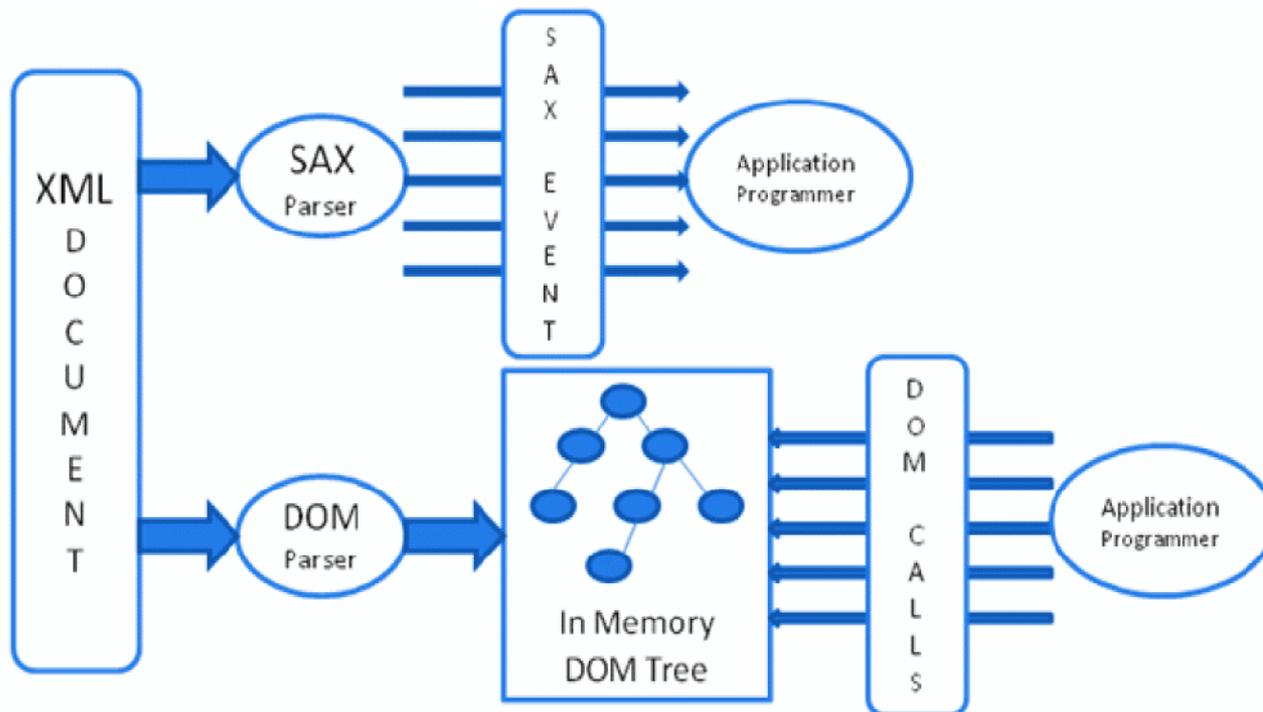


XML Parser

XML-Parsing Standards

SAX (Simple API for XML)

DOM (Document Object Model)



- ❑ **Event-based: SAX** (Simple API for XML)
 - Originally a Java-only API.
 - Developed by XML-DEV mailing list community
 - No tree is built
 - The parser **reads** the file and **triggers** events as it finds elements/attribute/text in the XML doc
- ❑ XML is read sequentially
- ❑ When a **parsing event happens**, the parser invokes the corresponding method of the corresponding *handler*
- ❑ The *handlers* are programmer's implementation of standard Java API (i.e., interfaces and classes)
- ❑ Similar to an I/O-Stream, goes in one direction

SAX Parser: Example

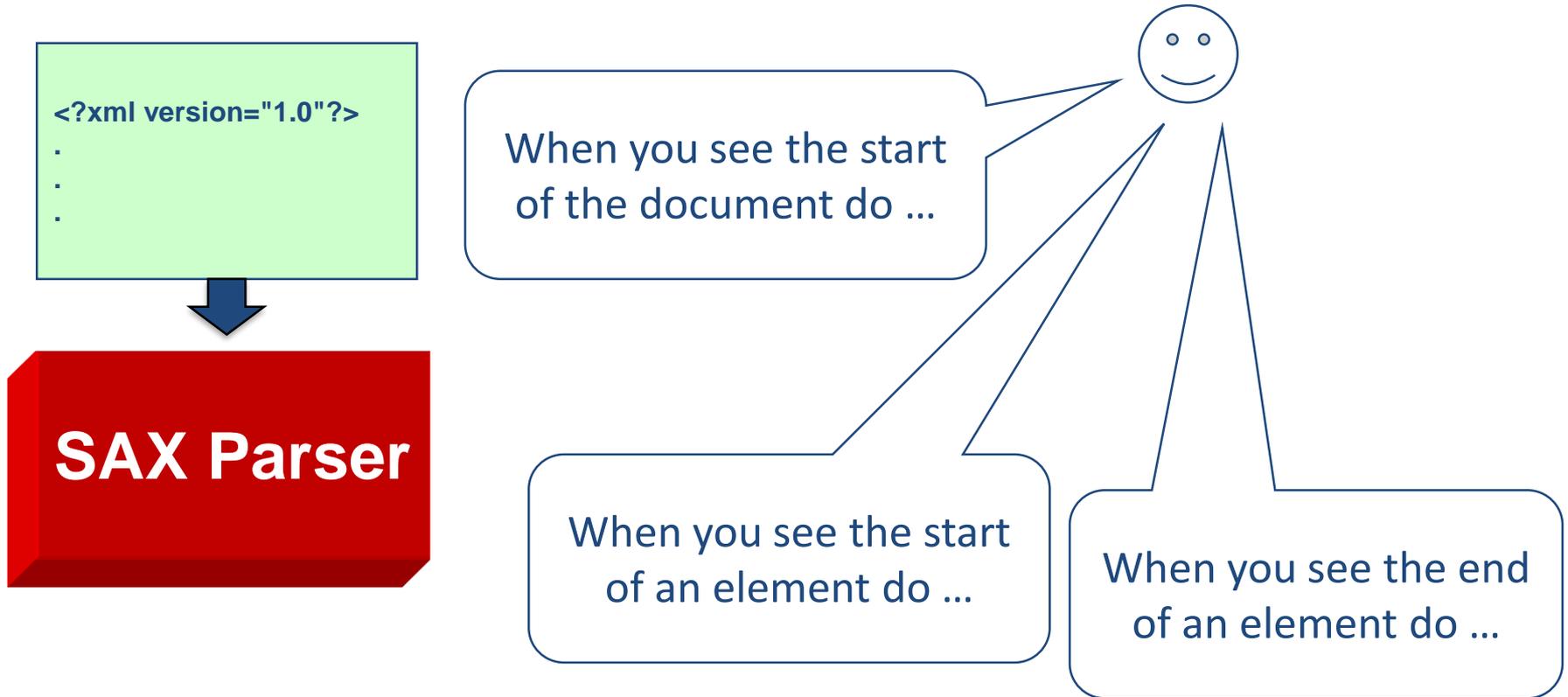
Parsing Event

Example: Orders Data in XML:

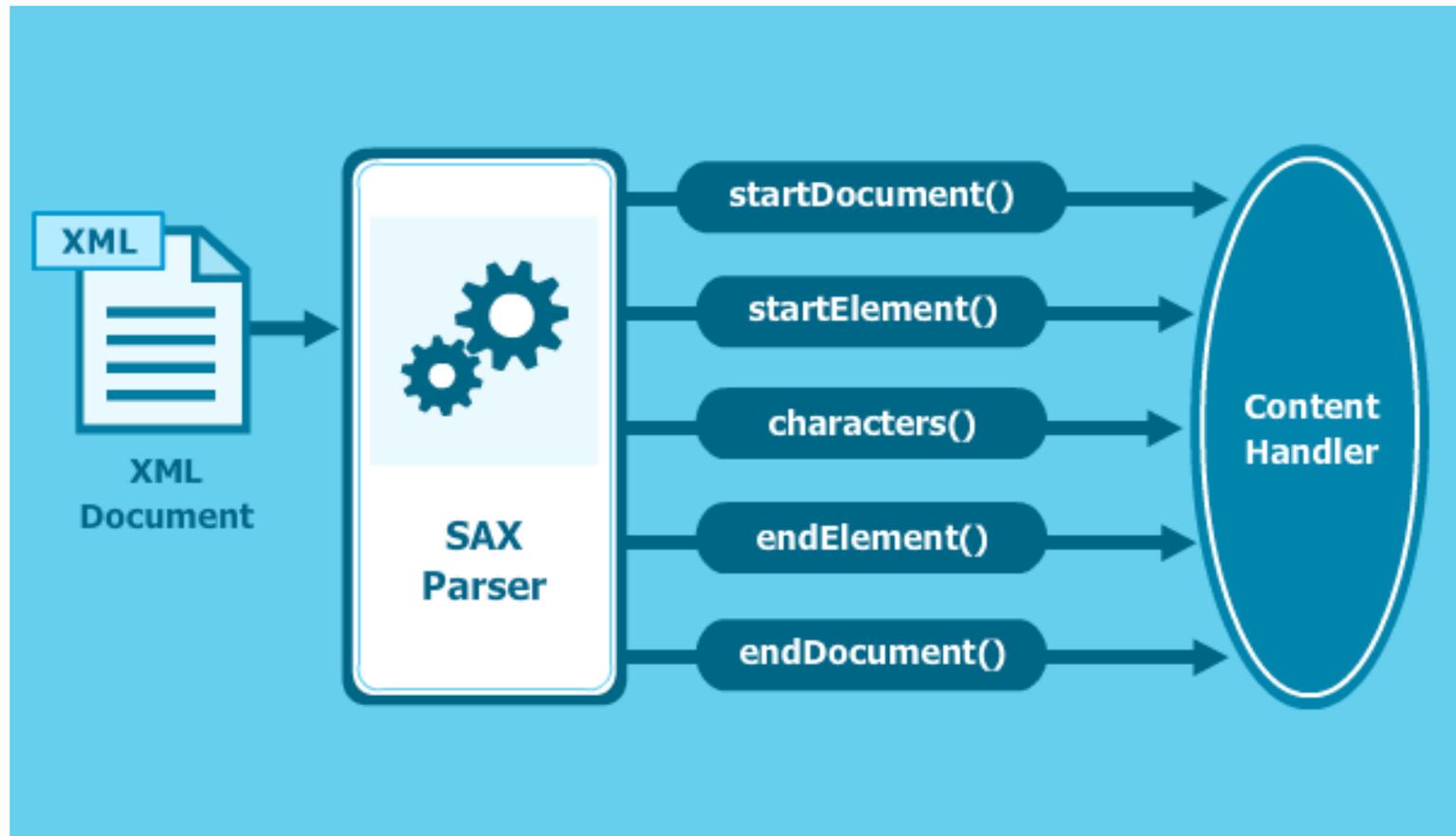
- several orders, each with several items
- each item has a part number and a quantity

```
<orders> startDocument
<order>
  <onum>1020</onum>
  <takenBy>1000</takenBy>
  <customer>1111</customer>
  <recDate>10-DEC 94</recDate>
  <items>
    <item>
      <pnum>10506</pnum>
      <quantity>1</quantity>
    </item>
    <item> startElement
      <pnum>10507</pnum>
      <quantity>1</quantity>
    </item> endElement
    <item>
      <pnum>10508</pnum>
      <quantity>2</quantity>
    </item>
    <item>
      <pnum>10509</pnum>
      <quantity>3</quantity>
    </item>
  </items>
</order>
...
</orders> endDocument
```

SAX Parser..



SAX Parser..



- ❑ **Object-based: DOM** (Document Object Model)
 - The parser **loads** the XML doc into computer memory and **builds a tree** of objects for all elements & attributes
- ❑ The API allows for **constructing**, **accessing** and **manipulating** the structure and content of XML documents
- ❑ User accesses data by **traversing the tree**
- ❑ The **XML DOM** defines the properties and methods for accessing and editing XML.
 - However, before an XML document can be accessed, **it must be loaded into an XML DOM object.**
 - All modern browsers have a built-in XML parser to access and manipulate XML by converting text into an XML DOM object.

Parsing a Text String

```
<html> <body>

<p id="demo"></p>

<script>
var text, parser, xmlDoc;

text = "<bookstore><book>" +
"<title>Everyday Italian</title>" +
"<author>Giada De Laurentiis</author>" +
"<year>2005</year>" +
"</book></bookstore>";

parser = new DOMParser();
xmlDoc = parser.parseFromString(text, "text/xml");

document.getElementById("demo").innerHTML =
xmlDoc.getElementsByTagName("title")[0].childNodes[0].nodeValue;
</script>
</body> </html>
```

Output: Everyday Italian

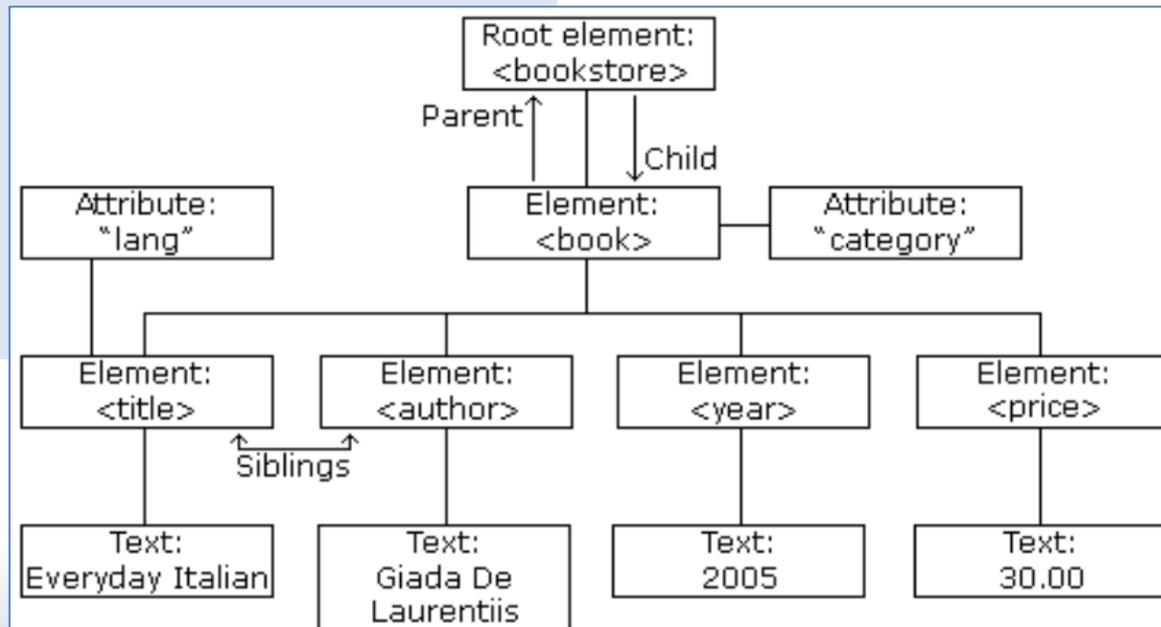
XML DOM

DOM defines a standard for accessing and manipulating documents

>> All XML elements can be accessed through the XML DOM

```
<?xml version="1.0" encoding="UTF-8"?>
<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">XQuery Kick Start</title>
    <author>James McGovern</author>
    <author>Per Bothner</author>
    <author>Kurt Cagle</author>
    <author>James Linn</author>
    <author>Nagarajan</author>
    <year>2003</year>
    <price>49.99</price>
  </book>
</bookstore>
```

It views an XML document as a tree-structure called a *node-tree*.



XML DOM: Examples

- ❑ The JavaScript code to get the text from the first <title> element in books.xml:

```
txt=xmlDoc.getElementsByTagName("title")[0].childNodes[0].nodeValue
```

>> Example Explained:

- `xmlDoc` - the XML DOM object created by the parser.
- `getElementsByTagName("title")[0]` - the first <title> element
- `childNodes[0]` - the first child of the <title> element (the text node)
- `nodeValue` - the value of the node (the text itself)

After the execution of the statement, txt will hold the value **"Everyday Italian"**

Node Navigation: Examples

- ❑ The `getElementsByTagName()` method returns a node list (an array of nodes).

```
xmlDoc=loadXMLDoc("books.xml");  
x=xmlDoc.getElementsByTagName("title");
```

- The `<title>` elements in `x` can be accessed by index number.
- To access the third `<title>` you can write: `y=x[2];`

- ❑ The `length` property defines the length of a node list (the number of nodes).

```
// assume we have loadXMLDoc() function already implemented  
xmlDoc=loadXMLDoc("books.xml");  
x=xmlDoc.getElementsByTagName("title");  
for (i=0;i<x.length;i++)  
{  
    document.write(x[i].childNodes[0].nodeValue);  
    document.write("<br>");  
}
```

Output:
Everyday Italian
Harry Potter
XQuery Kick Start

Node Navigation: Examples

- loops through the children of the root node that have elements

```
xmlDoc=loadXMLDoc("books.xml");
x=xmlDoc.documentElement.childNodes;
for (i=0;i<x.length;i++)
{
  if (x[i].nodeType==1)
    {//Process only element nodes (type 1)
    document.write(x[i].nodeName);
    document.write("<br>");
    }
}
```

Output:

book
book
book

NodeType	Named Constant
1	ELEMENT_NODE
2	ATTRIBUTE_NODE
3	TEXT_NODE
4	CDATA_SECTION_NODE
5	ENTITY_REFERENCE_NODE
6	ENTITY_NODE
7	PROCESSING_INSTRUCTION_NODE
8	COMMENT_NODE
9	DOCUMENT_NODE
10	DOCUMENT_TYPE_NODE
11	DOCUMENT_FRAGMENT_NODE
12	NOTATION_NODE

Which should we use? DOM vs. SAX

- ❑ If you need to manipulate (i.e., change) the XML – use **DOM**
- ❑ If you need to access the XML many times – use **DOM** (assuming the file is not too large)
- ❑ You can save time and effort if you send and receive DOM objects instead of XML files
 - But, DOM object are generally larger than the source
 - Thus If your document is **very large** and you only need to extract only a few elements – use **SAX**
- ❑ Programming with SAX parsers is, in general, more efficient
 - However, in some cases, it is difficult !
 - How can we find, using a SAX parser, elements **e1** with ancestor **e2**?
 - How can we find, using a SAX parser, elements **e1** that have a descendant element **e2**?
 - etc...

XML Processing

XML Processing

- ❑ XML processing in PHP, JavaScript, and other modern development environments is divided into **two** basic styles:
 - The **in-memory approach**, which involves reading the entire XML file into memory into some type of data structure with functions for accessing and manipulating the data.
 - The **event or pull approach**, which lets you pull in just a few elements or lines at a time, thereby **avoiding the memory load of large XML files**.

XML Processing in JavaScript

- ❑ All modern browsers have a **built-in XML parser** and their JavaScript implementations support **an in-memory XML DOM API**, which loads the entire document into memory where it is transformed into a **hierarchical tree data structure**.
- ❑ You can then use the already familiar DOM functions
 - Such as: `getElementById()`, `getElements ByTagName()`, and `createElement()` to access and manipulate the data.

Loading and processing an XML document via JavaScript

```
<script>
if (window.XMLHttpRequest) {
    // code for IE7+, Firefox, Chrome, Opera, Safari
    xmlhttp=new XMLHttpRequest();
}
else {
    // code for old versions of IE (optional you might just decide to
    // ignore these)
    xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
}

```

```
// load the external XML file
xmlhttp.open("GET","art.xml",false);
xmlhttp.send();
xmlDoc=xmlhttp.responseXML;

// now extract a node list of all <painting> elements
paintings = xmlDoc.getElementsByTagName("painting");
if (paintings) {
    // loop through each painting element
    for (var i = 0; i < paintings.length; i++)
    {
        // display its id attribute
        alert("id="+paintings[i].getAttribute("id"));

        // find its <title> element
        title = paintings[i].getElementsByTagName("title");
        if (title) {
            // display the text content of the <title> element
            alert("title="+title[0].textContent);
        }
    }
}
</script>
```

XML Processing in PHP

- ❑ PHP provides several extensions or APIs for working with XML
 - The **DOM extension** - loads the entire document into memory where it is transformed into a hierarchical tree data structure.
 - The **SimpleXML extension** - loads the data into an object that allows the developer to **access** the data via array properties and **modify** the data via methods.
 - The **XML parser** is an **event-based XML extension**. This is sometimes referred to as a SAX-style API
 - The **XMLReader** is a read-only pull-type extension similar to that used with database processing. The **XMLWriter** provides an analogous approach for creating XML files.
- ❑ The **SimpleXML** and the **XMLReader** extensions provide the easiest ways to read and process XML content.
 - reads the entire XML file into memory and transforms into a complex object

```

<?php

$filename = 'art.xml';
if (file_exists($filename)) {
    $art = simplexml_load_file($filename);

    // access a single element
    $painting = $art->painting[0];
    echo '<h2>' . $painting->title . '</h2>';
    echo '<p>By ' . $painting->artist->name . '</p>';
    // display id attribute
    echo '<p>id=' . $painting["id"] . '</p>';

    // loop through all the paintings
    echo "<ul>";
    foreach ($art->painting as $p)
    {
        echo '<li>' . $p->title . '</li>';
    }
    echo '</ul>';
} else {
    exit('Failed to open ' . $filename);
}

?>

```

-> is used in object scope to access methods and properties of an object.

- ❑ Here, the `simplexml_load_file()` function is used to transform XML file into an object
- ❑ The various elements in the XML document can then be manipulated using regular PHP object techniques.

XML: Advantages

- ❑ XML provides a basic syntax that can be used to **share information** between different kinds of computers, different applications, and different organizations.
- ❑ With XML, your data **can be available to all kinds of "reading machines"** (Handheld computers, voice machines, news feeds, etc)
- ❑ XML provides a **gateway for communication between applications**, even applications on wildly different systems. As long as applications can share data (through HTTP, file sharing, or another mechanism)
- ❑ It **supports Unicode**, allowing almost any information in any written human language to be communicated.
- ❑ It can **represent common computer science data structures**: records, lists and trees.
- ❑ Its **self-documenting format** describes structure and field names as well as specific values.
- ❑ It is based on **international standards**.

XML: Disadvantages

- ❑ It is **difficult for the end-user to understand its capabilities**.
- ❑ XML **syntax is redundant or large relative to binary representations** of similar data, especially with tabular data.
- ❑ The **redundancy may affect application efficiency** through higher storage, transmission and processing costs.
- ❑ XML **syntax is verbose**, especially for human readers, relative to other alternative 'text-based' data transmission formats.
- ❑ The **hierarchical model for representation** is limited in comparison to an object oriented graph.
- ❑ XML **namespaces are problematic to use** and namespace support can be difficult to correctly implement in an XML parser.

References

- ❑ “Internet & World Wide Web: How to Program 5th editions”
- ❑ “Fundamentals of Web Development” Book by Randy Connolly and Ricardo Hoar, 2015
- ❑ W3schools: <https://www.w3schools.com/Xml/>

