Chapter 2

Data Mapping and Exchange
Part I

Email:begna19mrblgo@gmail.com
visit: https://begnafrique.wordpress.com/

Metadata I

- Metadata is structured information that describes, explains, locates, or otherwise makes it easier to retrieve, use, or manage information resource.
- Metadata is often called data about data or information about information.
- Metadata (Meta content) is denied as the data providing information about one or more aspects of the data, such as:

  - Means of creation of the data
  - Purpose of the data
  - Time and date of creation,
  - Creator or author of the data,
  - Location on a computer network where the data were created.

Metadata II

Metadata types

(i) Structural metadata:

- It is used to describe the structure of database objects such as tables, columns, keys and indexes.
- Indicates how compound objects are put together, eg how pages are ordered to form chapters.

(ii) Guide metadata:

- It is used to help humans find specific items and is usually expressed as a set of keywords in a natural language.
- Describes a resource for purposes such as discovery and identifcation.
- It can include elements such as title, abstract, author, and keywords.

(iii) Administrative metadata:

- It provides information to help manage a resource, such as when and how it was created, file type and ot technical information, and who can access it.

Data Representation and Encoding I

- Before natural language data can be written to a computer recording device like disk, tape or memory it needs to be put in a format that the computer recognizes.
- For example, to record data blocks on the surface of the disk the data needs to be represented as a string of pulses, where each pulse is in either one of two states: positive or negative polarity.
- Since there can be only two states, we refer to this as binary notation.
- The direction of the polarity (i.e. + or -) determines if the data is interpreted as a binary one or a binary zero.

Computer Data types:

- Numeric Data: Consists of only numbers 0,1,..,9
- Alphabetic Data: Consists of only the letters A-Z, in both uppercase
- and lowercase , and blank character.
- Alphanumeric Data: is a string of symbols where a symbol may be one
- of the letters A-Z in either uppercase or lowercase, or one of the digits
- 0-9 ,or special characters such as, + - * / , . () = etc.

Data Representation and Encoding II

- So how do we tell the computer to store the letter "A", specifically a capital A?
- Computer is a digital system and can only deal with l's and 0s.
- That means digital computers use the binary system to represent and manipulate numeric values.
- So to deal with letters and symbols they use alphanumeric codes
- Well, in order to represent a human readable character in other than a one or zero, computer designers came up with various coding schemes consisting of a string of ones and zeros to represent many of the common characters needed by computer users.
- Computer codes are used for internal representation of data in the computers.
- As computers use binary numbers for internal data representation, computer codes use binary coding schemas.

Data Representation and Encoding III

- In binary coding, every symbol that appears in the data is represented by a group of bits.
- The group bits used to represents a symbol is called a byte.
- There are three very popular coding schemes in use today:

  -ASCII

  -EBCDIC

  -Unicode
- These coding schemes made it practical for us to record and process natural language characters on "two-state" or binary computing devices.

Data Representation and Encoding IV

ASCII

- The American Standard-Code for Information Interchange (ASCII) pronounced "as-kee" is a 7 bit coc based on the ordering of the English alphabets.
- It was developed by American National Standards Institute (ANSI).
- It is a standard code to represent alphanumeric data.
- It can represents:

  -Latin alphabet,

  -Arabic numerals

  -standard punctuation characters

  -Plus small set of accents and other European special characters
- The first ASCII code was 7-bit code.
- The 7-bit code system can represent 128 characters which means only 7 bits are required to represer an ASCII character.

Data Representation and Encoding V

- They include:
  - Printable characters including 26 upper-case letters (A to Z)
  - 26 lowercase letters (a - z),
  - 10 numerals (0 to 9) and 33 special characters such as mathematical symbols
  - space character etc.
  - It also denes codes for 33 non-printing obsolete characters except for carriage return and/or line feed.
    - However, since the smallest size representation on most computers is a byte, a byte is used to store an ASCII character.
    - **Then ASCII 7-bit code system was extended to 8-bit code.**
    - The 8-bit code system can represent 256 characters.
    - The MSB of an ASCII character is 0.

Data Representation and Encoding VI

Lets take a look at how the character "A" is encoded in ASCII.

An ASCII capital "A" is a Hex "41" and Hex "C1" in EBCDIC

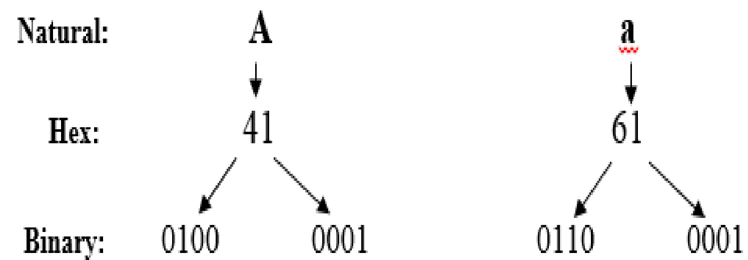
	ASCII	EBCDIC
A	41	C1
B	42	C2
C	43	C3
D	44	C4
E	45	C5
F	46	C6
G	47	C7
H	48	C8
I	49	C9
J	4A	D1
K	4B	D2
L	4C	D3
M	4D	D4
N	4E	D5
O	4F	D6
P	50	D7
Q	51	D8
R	52	D9
S	53	E2
T	54	E3
U	55	E4
V	56	E5
W	57	E6
X	58	E7
Y	59	E8
Z	5A	E9

	ASCII	EBCDIC
a	61	81
b	62	82
c	63	83
d	64	84
e	65	85
f	66	86
g	67	87
h	68	88
i	69	89
j	6A	91
k	6B	92
l	6C	93
m	6D	94
n	6E	95
o	6F	96
p	70	97
q	71	98
r	72	99
s	73	A2
t	74	A3
u	75	A4
v	76	A5
w	77	A6
x	78	A7
y	79	A8
z	7A	A9

	ASCII	EBCDIC
0	30	F0
1	31	F1
2	32	F2
3	33	F3
4	34	F4
5	35	F5
6	36	F6
7	37	F7
8	38	F8
9	39	F9
space	20	40

Data Representation and Encoding VII

- Using the above ASCII conversion chart we see that a capital "A" is a hexadecimal 41.
- If we convert this hexadecimal number to its 8-bit binary representation we get "01000001".
- So the disk surface will have the polarity changed to record the following string of 8 bits: 01000001
- Lets look at this again.
- This time lets also convert a lowercase "a" as well:



Data Representation and Encoding VIII

EBCDIC(Extended Binary Coded Decimal Interchange Code)

- EBCDIC, pronounced "ebb-c-dick", was deployed in the early 1960s by IBM when they announced a new computer series that became known as System 360.
- It turned out that EBCDIC followed a direction totally different from ASCII, where the heritage of paper tape was clearly established.
- So EBCDIC and ASCII are not compatible and require a translation to move data from an EBCDIC machine to an ASCII machine and vice versa.
- It uses 8 bits to represent a symbol.
- It can represent 256 different characters.
- Zoned decimal numbers are used to represent numeric values (Positive, negative, unsigned)

Data Representation and Encoding IX

- A sign indicator (C for plus, D for minus and F for unsigned) is used in the zoned position of the rightmost digit.
- It is used primarily in the larger computer environments, specifically mainframes and some mid-frame computing platforms.

Data Representation and Encoding X

Unicode(Universal Code)

- Both EBCDIC and ASCII were built around the Latin alphabet.
- As such, they are restricted in their abilities to provide data representation for the non Latin alphabets used by the majority of the worlds population.
- As all countries began using computers, each was devising codes that would most effectively represent their native languages.
- ASCII and EBCDIC worked fine for English and the Romance languages but didn't have enough character combinations to support the alphabets of languages from Eastern Europe, Asia and Africa.
- No single encoding system supports all languages Different encoding systems conflict.

Data Representation and Encoding XI

Unicode features:

- Provides a consistent way of encoding multilingual plain text
- Defines codes for characters used in all major languages of the world
- Defines codes for special characters, mathematical symbols, technical symbols, and diacritics ....
- Capacity to encode as many as a million characters
- Assigns each character a unique numeric value and name
- Reserves a part of the code space for private use
- Affords simplicity and consistency of ASCII, even corresponding characters have same code
- Species an algorithm for the presentation of text with bi-directional behavior

Data Representation and Encoding XII

- Encoding Forms

  UTF-8, UTF-16, UTF-32.
- With 16 bits Unicode can support over 65,000 characters.
- The first 256 Unicode characters are the same as ASCII.
- Unicode is required by web users and modern standards XML, Java, ECMAScript (JavaScript), LDAP, CORBA 3.0, WML.
- Can you think of any issues with trying to distribute data in a distributed computing environment going from one coding scheme to another?

Data Representation and Encoding XII

Two Important Management Issues to Remember

- There are two very important principles to remember when moving data in a heterogeneous computing environment.
- Data moved from one computer to another may be using different coding schemes i.e. ASCII to EBCIDIC or vise versa.
- This data movement will require a conversion to the coding scheme on the target platform.
- For small data sets this may not be an issue; but for very large data sets found on most enterprises this data conversion will be noticeable overhead that affects performance.
- Notice that the collating (sort) sequence is different between ASCII and EBCIDIC.
- Numeric and lower case characters sort in different sequences in each.
- For example in ASCII numbers sort before letters but in EBCDIC letters sort before numbers.

XML Technologies

Part II

What is XML? I

- XML stands for Extensible Markup Language
- XML is a markup language much like HTML
- XML was designed to describe data, not to display data
- XML tags are not predefined.
- You must dene your own tags XML is designed to be self-descriptive
- XML is a W3C Recommendation.
- XML was designed to describe data.
- HTML was designed to display data.

What is XML? II

Difference between XML and HTML

- XML is not a replacement for HTML; XML is a complement to HTML.
- XML is a software- and hardware-independent tool for carrying information.
- XML was designed to describe data, with focus on what data is
- HTML was designed to display data, with focus on how data looks.
- HTML is about displaying information, while XML is about carrying information.

What is XML? III

- With XML You Invent Your Own Tags
- The tags (like <to> and <from>) are not predefined in any XML standard.
- These tags are "invented" by the author of the XML document.
- That is because the XML language has no predefined tags.
- The tags used in HTML are predefined.
- HTML documents can only use tags defined in the HTML standard (like <p>, <h1>, etc.).

What is XML? IV

The power of XML

- XML is used in many aspects of web development, often to simplify data storage and sharing.
- As general,Instead of serving as a language for displaying information, XML is a language for **storing and carrying information.**
- XML can also be used to share data between disparate systems and organizations.
- The reason for this is that an XML document is simply a text file and nothing more.
- It is well-structured, easy to understand, easy to parse, easy to manipulate, and is considered human-readable.
- Finally, XML is a non-proprietary specification and is free to anyone who wishes to use it.

What is XML? V

(i) XML Separates Data from HTML

- If you need to display dynamic data in your HTML document, it will

take a lot of work to edit the HTML each time the data changes.

- With XML, data can be stored in separate XML files.
- This way you can concentrate on using HTML/CSS for display and layout, and be sure that changes in the underlying data will not require any changes to the HTML.
- With a few lines of **JavaScript** code, you can read an external XML file and update the data content of your web page.

What is XML? VI

(ii) XML Simplifies Data Sharing

- In the real world, computer systems and databases contain data in incompatible formats.
- XML data is stored in plain text format.
- This provides a software- and hardware-independent way of storing data.
- This makes it much easier to create data that can be shared by different applications.

(iii) XML Simplifies Data Transport

- One of the most time-consuming challenges for developers is to exchange data between incompatible systems over the Internet.

What is XML? VII

- Exchanging data as XML greatly reduces this complexity, since the data can be read by different incompatible applications.

(iv) Internet Languages Written in XML

- Several Internet languages are written in XML.
- Here are some examples: XHTML, XML Schema, SVG, WSDL and RSS.

Well-Formed XML I

- To the purist there is no such thing as well-formed XML; a document is either XML and therefore, by definition, well-formed, or its just text.
- But in common parlance **well-formed XML** means a document that follows the W3Cs XML Recommendation with all its rules governing the following:

  -How the content is separated from the metadata (markup)

  -What is used to identify the markup

  -What the constituent parts are

  - In what order and where these parts can appear

Well-Formed XML II

(i) Forbidden Characters

- The first thing to know before writing XML is that a few restrictions exist on what characters are permitted in an XML document.
- These rules vary slightly depending on whether you're using version 1.0 or 1.1, the latter being a bit more permissive.
- Both versions forbid the use of null in a document; this is the character represented by 0x0 in hexadecimal.
- In version 1.0 you are also forbidden to use the characters represented by the hexadecimal codes between 001 and 019, except for three: the tab (09), the newline (0A), and the carriage return (0D).
- For example, you cannot use the character 07, known as the bell character, because it sounds a bell or a beep on some systems.

Well-Formed XML III

- In version 1.1 you can use all these control characters although their use is a little unusual.
- You see how to specify which version you are using in the next section.
- A few characters in the Unicode specification also can’t be used but you’re unlikely to come across these.
- You can find the full list in the W3C0s XML Recommendation.

(ii) XML Prolog

- The first part of a document is the prolog.
- It is optional so you won’t see it every time, but if it does exist it must come first.
- The prolog begins with an XML declaration which, in its simplest form, looks like the following: **<?xml version=1.0?>**

Well-Formed XML IV

- This declaration contains only one piece of information, the version number, and currently this will always be either 1.0 or 1.1.
- Sometimes the declaration may also contain information about the encoding used in the document:

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

- When an XML processor reads a document, it has to know which encoding was used; but, its a **chicken-and-egg** situation if it does not know the encoding how can it read what you have put in the declaration?

Well-Formed XML V

- The simple answer to this lies in the fact that the first few bytes of a file can contain a byte order mark, or BOM.
- This helps the parser enough to be able to read the encoding specified in the declaration.
- Once it knows this it can decode the rest of the document.
- Two main encoding systems use Unicode: UTF-8 and UTF-16.
- **UTF** stands for **UCS** Transformation Format, and **UCS** itself means Universal Character Set.
- The number refers to how many bits are used to represent a simple character, either 8 or 16 (one or two bytes, respectively).

Well-Formed XML VI

- The reason UTF-8 manages with only one byte whereas UTF-16 needs two is because UTF-8 uses a single byte to represent the more commonly used characters and two or three bytes for the less common ones.
- UTF-16 uses two bytes for the majority of characters and three bytes for the rest.
- All XML processors are mandated to understand UTF-8 and UTF-16 even if those are the only encodings they can read.
- UTF-8 is the default for documents without encoding information.

Well-Formed XML VII

(iii) Completing the Declaration

- Now that you have specified the type of encoding you are using, you can finish the declaration.
- The final part of the declaration is determining whether the document is considered to be standalone:

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

- Sometimes there are a few additional, elements to the XML prolog.
- These optional parts include comments and processing instructions.

Well-Formed XML VIII

- Comments are usually meant for human consumption and are not supposed to be part of the actual data in a document.
- They are initiated by the sequence <! -- and terminated by -- >.
- Following is example.xml with a comment added:

```
<?xml version="1.0" encoding="utf-8"?>
<!--This is a comment in XML declaration -->
```

- Once the XML prolog is finished you need to create the root element of the document.
- XML documents form a tree structure that starts at "the root" and branches to "the leaves".
- The following section details elements and how to create them:

Well-Formed XML IX

```xml
1  <?xml version="1.0" encoding="utf-8"?>
2  <message>
3     <to> IT 4th Year </to>
4     <from> Your Instructor </from>
5     <subject> Test Notification</subjcet>
6     <body> There will be a Test next week </body>
7  </message>
```

- The first line is the XML declaration.
- It defines the XML version (1.0).
- The next line describes the root element of the document (like saying:
- "this document is a message"):
- The next 4 lines describe 4 child elements of the root.

Well-Formed XML X

Creating Elements

- Elements are the basic building blocks of XML and all documents will have at least one.
- All elements are denfined in one of two ways.
- At its simplest, an element with content consists of a start tag, which is a left angle bracket (<) followed by the name of the element, such as myElement,and then a right angle bracket (>).
- So a full start tag might be < myElement >.
- To close the element the end tag starts with a left angle bracket, a forward slash, and then the name of the element and a right angle bracket.
- So the end tag for < *myElement* > would be </myElement >.
- You can add spaces after the name in a start tag, such as <myElement >, but not before the name as in < *myElement*>.

Well-Formed XML XI

```
1 <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2 <!-- XML comment -->
3 <myElement></myElement>
```

XML Element Naming Rules

(i) Naming Styles

- In addition to the two ways to define an element, there are a few different naming styles for elements and, as in many things IT-related, people can get quite evangelical about them.
- The one thing almost everyone agrees on is to be consistent; choose a style for the document and stick with it.

Well-Formed XML XI

- Following are the main contenders for how you should name your elements the main idea is how you distinguish separate words in an element name:

Naming Style

- **Pascal-casing**: This capitalizes separate words including the first: <MyElement/>.
- **Camel-casing**: Similar to Pascal except that the first letter is lowercase:<myElement /> .
- **Underscored names**: Use an underscore to separate words:<my_element />.
- **Hyphenated names**: Separate words with a hyphen: <my-element />.

Well-Formed XML XI

Naming Specifications

- XML has certain specific rules governing which names you can use for its markup object.
- An element name can begin with either an underscore or an uppercase or lowercase letter from the Unicode character set.This means you can use the Roman alphabet used by English and many other Western languages, the Cyrillic one used by Russian and its language relatives, characters from Greek, or any of the other numerous scripts, such as Thai or Arabic, that are defined in the Unicode standard.

Well-Formed XML XIII

- Names can contain letters, numbers, and other characters
- A name consists of at least one letter: a,z or A to Z.
- Subsequent characters can also be a dash (-)or a digit.
- Names are case-sensitive, so the start and end tags must match exactly.
- Names cannot contain spaces
- Names beginning with the letters XML, either in uppercase or lowercase, are reserved, and shouldn’t be used (although many parsers allow them in practice)

Well-Formed XML XIV

The following Table provides some examples of correctly and incorrectly formed elements:

LEGAL ELEMENT	REASON	ILLEGAL ELEMENT	REASON
`<myElement></myElement>`	Spaces are allowed after a name.	`<my Element />`	Names cannot contain spaces.
`<my1stElement/>`	Digits can appear within a name.	`<1stName />`	Names cannot begin with a digit.
`<myElement />`	Spaces can appear between the name and the forward slash in a self-closing element.	`< myElement />`	Initial spaces are forbidden.
`<my-Element />`	A hyphen is allowed within a name.	`<-myElement/>`	A hyphen is not allowed as the first character.
`<όνομα />`	Non-roman characters are allowed if they are classified as letters by the Unicode specification. In this case the element name is *forename* in Greek.	`<myElement></MyElement>`	Start and end tags must match case-sensitively.

Well-Formed XML XV

Root Element

- The next step after writing the prolog is creating the root element.
- All documents must have one and only one root element.
- Everything else in the document lies under this element to form a hierarchical tree.
- One example is when using XML as a logging format.
- A typical log file might look like this:

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <entry type="audit"> Failed logon attempt</entry>
3 <entry type="audit"> Successful logon attempt</entry>
4 <entry type="information"> Successful folder synchronisation</
     entry>
```

Well-Formed XML XVI

- The problem with this format, though, is that there isn't a unique root element; you have to add one to make it well-formed:

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <log>
3    <entry type="audit">Failed logon attempt</entry>
4    <entry type="audit">Successful logon attempt </entry>
5    <entry type="information"> Successful folder synchro </entry
         >
6 </log>
```

Well-Formed XML XVII

XML Elements can be Empty

- An alternative syntax can be used for XML elements with no content:
- Instead of writing a book element (with no content) like this:

<book></book> .

- It can be written like this: <book/> .
- This sort of element syntax is called **self-closing.**

Well-Formed XML XVIII

XML Elements are Extensible

- XML elements can be extended to carry more information.
- One of the beauties of XML, is that it can be extended without breaking applications.
- If you want to add more data, perhaps a middle name for example, to the application users data, you can do that easily by creating a new attribute,middle Name:

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <applicationUsers>
3    <user firstName="Joe" middleName="John" lName="Fawcett" />
4    <user firstName="Dany" middleName="Bekele" lName="Ayers" />
5 </apllicationUsers
```

Well-Formed XML XIX

Other Elements

- Underneath the root element can lie other elements that follow the same rules for naming and attributes and, as you saw earlier, there can also be free text.
- These nested elements can be used to show individual or repetitive items of data depending on what you are trying to represent.
- For example, your root element could be <person> and the elements underneath could show the persons characteristics, such as:

  <biography> and <address>.

Well-Formed XML XX

Attributes

- Attributes provide additional information about an element.
- Elements are one of the two main building blocks of XML the other one is attributes.
- Attributes are name-value pairs associated with an element.
- You can add a couple of attributes to the example document like so:

```
1  <?xml version="1.0" encoding="UTF-8? standalone="yes"?>
2  <!-- This is a comment that follows the XML declaration -->
3  <myElement myFirstAttribute="One" mySecondAttribute="Two"></
       myElement>
```

Well-Formed XML XXI

- A number of rules also govern attributes exist:
- Attributes consist of a name and a value separated by an equals sign.
- The name, for example, myFirstAttribute, follows the same rules as element names.
- The attribute value must be in quotes.
- You can use either single or double quotes, the choice is entirely yours.
- You can use single on some attributes and double on others, but you can’t mix them in a single attribute.
- There must be a value part, even if its just empty quotes.
- You can’t have something like <option selected> as you might in HTML.
- Attribute names must be unique per element.
- If you use double quotes as the delimiter you can’t also use them as part of the value.
- The same applies for single quotes.

Well-Formed XML XXII

Element and Attribute Content

- Attribute values and elements can both contain character data (called text in normal parlance).
- You've already seen examples of attributes in earlier code snippets.
- A similar example for an element would be:

```
<myElement> Here is some character content</myElement>
```

- There are only two more restrictions to follow regarding character content.
- Two characters cannot appear in attribute values or direct element content: the ampersand (&) and the left angle bracket (<).
- You cannot use the latter because its used to delimit elements and it can confuse the parser.
- You cannot use the former because its used to begin entity and character references.

Well-Formed XML XIX

Elements Versus Attributes

- On many occasions you will have a choice whether to represent data as an element or an attribute.
- There are no fixed rules regarding whether you should use one form or the other, but the following are some things to consider when making.

```
1  <applicationUsers>
2    <user firstName="Joe" lastName="Fawcett" />
3    <user firstName="Danny" lastName="Ayers" />
4    <user firstName="Catherine" lastName="Middleton" />
5  </applicationUsers>
```

Cont..

```
1    <applicationUsers>
2      <user>
3        <firstName>Joe</firstName>
4        <lastName>Fawcett</lastName>
5    </user>
6 <applicationUsers>
```

There are no rules about when to use attributes or when to use element.

XML Attributes for Metadata

Sometimes ID references are assigned to elements.

These IDs can be used to identify XML elements in much the same way as the id attribute in HTML.

This example demonstrates this

Cont..

```
1 <books>
2    <ISBN = "12578858">
3       <title>Network Design</title>
4       <author>Unknown</author>
5       <year>2012</year>
6    </user>
7 <books>
```

- The id attributes above are for identifying the different books.
- It is not a part of the book itself.
- Metadata (data about data) should be stored as **attributes**, and the data itself should be stored as **elements.**
- Information contained in an attribute is generally considered metadata; that is, information about the data in the element, as opposed to the data itself.
- An element can have as many attributes as desired, as long as each has a unique name.

XML Tree Structure I

- Another area where XML-formatted data flourishes over simple text files is when representing a hierarchy; for instance a file system.
- This scenario needs a root with several folders and files; each folder then may have its own subfolders, which can also contain folders and files.
- This can go on indefinitely.
- If all you had was a text file, you could try something like this, which has a column representing the path and one to describe whether its a folder or a file:

XML Tree Structure II

```
1    Path Type
2       C:\folder
3       C:\pagefile.sys file
4       C:\Program Files folder
5       C:\Program Files\desktop.ini file
6       C:\Program Files\Microsoft folder
7       C:\Program Files\Mozilla folder
8       C:\Windows folder
9       C:\Windows\System32 folder
10      C:\Temp folder
11      C:\Temp\~123.tmp file
12      C:\Temp\~345.tmp file
```

- As you can see, this is not pretty and the information is hard for us humans to read and quickly assimilate.
- Comparatively, now look at one possible XML version of the same information:

XML Tree Structure III

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

    <folder name="C:\">
        <folder name="Program Files">
            <folder name="Microsoft"> </folder>
            <folder name="Mozilla"> </folder>
        </folder>
        <folder name="Windows">
            <folder name="System32"></folder>
        </folder>
        <folder name="Temp">
            <files>
                <file name="~123.tmp"></file>
                <file name="~345.tmp"></file>
            </files>
        </folder>
        <files>
            <file name="pagefile.sys"></file>
        </files>
    </folder>
```

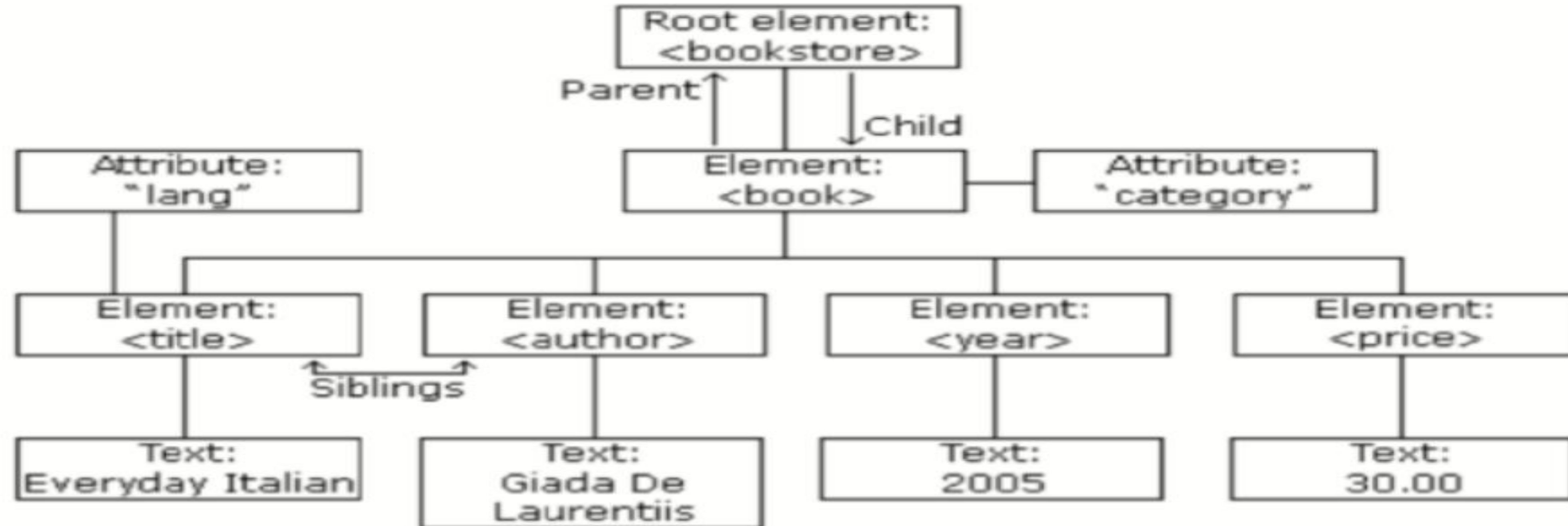
This hierarchy is much easier to appreciate.

XML Tree Structure IV

- There is less repetition of data and it would be fairly easy to parse XML documents must contain a root element.
- This element is "the parent" of all other elements.
- The elements in an XML document form a document tree.
- The tree starts at the root and branches to the lowest level of the tree.
- All elements can have sub elements (child elements).
- The terms parent, child, and sibling are used to describe the relationships between elements.
- Parent elements have children.
- Children on the same level are called siblings (brothers or sisters).
- All elements can have text content and attributes (just like in HTML).

XML Tree Structure V

Example



XML Syntax Rules I

(i) All XML Elements Must Have a Closing Tag

- Every element must have a closing tag.
- In HTML, some elements do not have to have a closing tag: <p> This is a paragraph
- In XML, it is illegal to omit the closing tag.
- All elements must have a closing tag: <p> This is a paragraph.</p>

(ii) XML Tags are Case Sensitive

- The tag <Letter> is different from the tag <letter>.
- Opening and closing tags must be written with the same case:

```
1    <Message>This is incorrect</message>
2    <message>This is correct</message>
```

XML Syntax Rules II

(v) XML Attribute Values Must be enclosed in Quoted

(vi) White-space is preserved in XML

(vii) Entity References

- Some characters have a special meaning in XML.
- If you place a character like "<" inside an XML element, it will generate an error because the parser interprets it as the start of a new element.

```
1    This will generate an XML error: <message>if salary < 1000
         then</message>
```

- To avoid this error, replace the "<" character with an entity reference:
- There are 5 predefined entity references in XML:
- Entities are a kind of auto text; a way of entering text into an XML document without typing it all out.

XML Syntax Rules III

<	<	Less than
>	>	Greater than
&	&	ampersand
'	‘	Apostrophe
"	“	Quotation mark

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

XML DTD and XML Schema

How does an XML processor check your xml document?

There are two main checks that XML processors make:

1. Checking that your document is well-formed (Syntax rule)
2. Checking that it's valid (syntax-check your XML either in XML DTD or XSD)

- ***DTD- Document Type Definition***
- ***XSD-XML Schema Definition***

Why need XML Validator

–Use our XML validator to syntax-check your XML.

–Errors in XML documents will stop your XML applications unlike HTML browser

XML DTD I

–An XML document with correct syntax is called "Well Formed".

–An XML document validated against a DTD is "Well Formed" and "Valid".

–The purpose of a DTD is to define the structure of an XML document and a list of legal elements.

- **How you add a DTD to our XML document**

1. DTDs can be separate documents (or)
2. They can be built into an XML document using a special element named **<!DOCTYPE>.**

XML DTD II

An XML Document with a DTD (example1.xml)

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/css" href="css1.css"?>
<!DOCTYPE document
[ <!ELEMENT document (heading, message)> <!ELEMENT
heading (#PCDATA)> <!ELEMENT message (#PCDATA)> ]>
<document>
<heading> Hello From XML </heading> <message> This is an
XML document! </message>
</document>
```

XML Schema I

- Another way of validating XML documents: using XML schemas.
- The XML Schema language is also referred to as **XML Schema Definition (XSD),** describes the structure of an XML document.
- defines the legal building blocks (elements and attributes) of an XML document like DTD.
- defines which elements are child elements
- defines the number and order 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

XML Schema II

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 powerful than DTDs
- XML Schemas are written in XML
- XML Schemas support data types and namespaces

•Creating XML Schemas by Using XML Schema-Creation Tools

- –HiT Software
- –xmlArchitect
- –XMLspy
- –XML Ray
- –Microsoft Visual Studio .NET

XML Schema example

Example (shiporder.xml):

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<shiporder orderid="889923">
        <orderperson>John Smith</orderperson>
        <shipto>
           <name>Ola Nordmann</name>
           <address>Langgt 23</address>
           <city>4000 Stavanger</city>
           <country>Norway</country>
        </shipto>
</shiporder>
```

XML Schema example cont...

Example "shiporder.xsd":

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="shiporder">
          <xs:complexType>
                    <xs:sequence>
                              <xs:element name="orderperson" type="xs:string"/>
                              <xs:element name="shipto">
                              <xs:complexType>
                              <xs:sequence>
                                        <xs:element name="name" type="xs:string"/>
                                        <xs:element name="address" type="xs:string"/>
                                        <xs:element name="city" type="xs:string"/>
                                        <xs:element name="country" type="xs:string"/>
                              </xs:sequence>
                              </xs:complexType>
                              </xs:element>
                    </xs:sequence>
                    <xs:attribute name="orderid" type="xs:string" use="required"/>
          </xs:complexType>
</xs:element>
</xs:schema>
```

Part 3

XML Namespace

XML Namespace I

- At their simplest, namespaces are a way of grouping elements and attributes under a common heading in order to differentiate them from similarly-named items.

For example it could be:

- Someone discussing a dinner party with their spouse and they need a bigger dining table.
- A database developer who is been asked to design a system to store user preferences on a website a new database table.
- An HTML developer who has been told to display some extra information on the users account page an HTML table.
- You can tell only if you know the context, or if the complete names are used dining table, database table, or HTML table.
- This is how namespaces work with elements and attributes.

XML Namespace II

Why Namespaces

- You have details about your company employees stored as XML and you want to be able to include a brief biography in the form of some HTML within the document.

```
1  <?xml version="1.0" encoding="UTF-8? standalone="yes"?>
2  %<!-- This is a comment that follows the XML declaration
       -->
3   <employees>
4       <employee id="001">
5          <firstName>Joe</firstName>
6          <lastName>Fawcett</lastName>
7          <title>Mr</title>
8          <dateOfBirth>1962-11-19</dateOfBirth>
9          <dateOfHire>2005-12-05</dateOfHire>
0          <position>Head of Software Development</position>
1          <biography><!-- biography here --></biography>
2       </employee>
3  <!-- more employee elements can be added here -->
4  </employees>
```

XML Namespace IV

- This document doesn't use namespaces, and it still works fine.
- Now say you want to add the biography and you're going to use XHTML that doesn't declare any namespaces and which illustrates the problem:

```
1   <?xml version="1.0" encoding="UTF-8? standalone="yes"?>
2   <employees>
3     <employee id="001">
4     <firstName>Joe</firstName>
5     <lastName>Fawcett</lastName>
6     <title>Mr</title>
7     <dateOfBirth>1962-11-19</dateOfBirth>
8     <dateOfHire>2005-12-05</dateOfHire>
9     <position>Head of Software Development</position>
0     <biography>
```

XML Namespace V

```
11                     <html>
12                          <head>
13                             <title>Joes Biography</title>
14                          </head>
15                       <body>
16                          <p>After graduating from the University of Life
17                          Joe moved into software development,
18                          originally working with COBOL on mainframes in
                                the 1980s.
19                          </p>
20                       </body>
21                     </html>
22                  </biography>
23               </employee>
24               <!-- more employee elements -->
25            </employees>
```

XML Namespace VI

- The way to get around this is to group the two sets of information the employee data and the biographical information into two different namespaces.

```
1      <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2    <employees>
3      <employee id="001">
4        <firstName>Joe</firstName>
5        <lastName>Fawcett</lastName>
6        <title>Mr</title>
7        <dateOfBirth>1962-11-19</dateOfBirth>
8        <dateOfHire>2005-12-05</dateOfHire>
9        <position>Head of Software Development</position>
0        <biography>
1          <html xmlns="http://www.w3.org/1999/xhtml">
2            <head>
3              <title>Joes Biography</title>
4            </head>
5            <body>
```

XML Namespace VII

```
16                  <p>
17                     After graduating from the University of Life
18                     Joe moved into software development, originally
                          working          with COBOL on mainframes in
                          the 1980s.
19                  </p>
20               </body>
21            </html>
22         </biography>
23      </employee>
24      <!-- more employee elements can be added here -->
25        </employees>
```

XML Namespace VIII

HOW TO DECLARE A NAMESPACE

- You can declare a namespace in two ways, depending on whether you want all the elements in a document to be under the namespace or just a few specific elements to be under it.
- If you want all elements to be included, you can use the following style:

* xmlns="http://wrox.com/namespaces/applications/hr/config"

- For example

```
1 <?xml version="1.0" encoding="UTF-8? standalone="yes"?>
2 <applicationUsers xmlns="http://wrox.com/namespaces/applications/hr
       config">
3    <user firstName="Joe" lastName="Fawcett" />
4    <user firstName="Danny" lastName="Ayers" />
5    <user firstName="Catherine" lastName="Middleton" />
6 </applicationUsers>
```

XML Namespace VIII

- This is known as declaring a default namespace, which is associated with the element on which it is declared, in this case <applicationUsers>, and any element contained within it.
- The namespace is said to be in scope for all these elements.
- Attributes, such as firstName are not covered by default namespace.
- To declare a namespace explicitly you have to choose a prefix to represent it.
- This is partly because it would be very onerous having to use the full name every time you created a tag or an attribute.
- The prefix can be more or less whatever you like; it follows the
- same naming rules as an element or attribute, but cannot contain a colon (:).
- Say you decide to use hr as your prefix.

XML Namespace IX

- You would then declare your namespace using the slightly modified form:

xmlns:hr="http://wrox.com/namespaces/applications/hr/cong

```
1	<?xml version="1.0" encoding="UTF-8? standalone="yes"?>
2	<applicationUsers xmlns:hr="http://wrox.com/namespaces/
	    applications/hr/config">
3	    <user firstName="Joe" lastName="Fawcett" />
4	    <user firstName="Danny" lastName="Ayers" />
5	    <user firstName="Catherine" lastName="Middleton" />
6	</applicationUsers>
```

However, this just means that you have a namespace URI that is identified by a prefix of hr; so far none of the elements or attributes are grouped in that namespace.

XML Namespace X

- To associate the elements with the namespace you have to add the prefix to the elements tags.

```
1  <hr:applicationUsers xmlns:hr="http://wrox.com/namespaces/
       applications/hr/config">
2    <user firstName="Joe" lastName="Fawcett" />
3    <user firstName="Danny" lastName="Ayers" />
4    <user firstName="Catherine" lastName="Middleton" />
5  </hr:applicationUsers>
```

- If you want the attributes in the document to be also in the hr namespace you follow a similar procedure

XML Namespace XI

```
1       <hr:applicationUsers xmlns:hr="http://wrox.com/namespaces/
            applications/hr/config">
2         <user hr:firstName="Joe" hr:lastName="Fawcett" />
3         <user hr:firstName="Danny" hr:lastName="Ayers" />
4         <user hr:firstName="Catherine" hr:lastName="Middleton" />
5       </hr:applicationUsers>
```

Summary on XML namespaces

- The primary purpose of namespaces is to group related elements and to differentiate them from elements with the same name that were designed to represent different types of data.
- How to declare a namespace using xmlns.
- There are two ways of specifying namespaces: default and prefixed.
- Documents can have more than one namespace; this makes it possible for software applications to treat the two content types differently.

Part 4

XSL, XSLT and XPath

XSL

- XSL stands for E**X**tensible **S**tylesheet **L**anguage.

–It is an XML-based Stylesheet Language.

–XSL describes how the XML document should be displayed

–XSL consists of three parts:

–XSLT - a language for transforming XML documents

–XPath - a language for navigating in XML documents

–XSL-FO - a language for formatting XML documents

XSLT

XSLT stands for XSL Transformations,

- XSLT transforms an XML source-tree into an XML result-tree.
- XSLT transforms an XML document into another XML document, recognized by a browser, like HTML and XHTML.
- Add/remove elements and attributes to or from the output file.
- Rearrange and sort elements, perform tests and make decisions about which elements to hide and display, and a lot more.
- XSLT uses XPath to find information in an XML document.
- XPath is used to navigate through elements and attributes in XML documents.

XSLT Cont..

How Does it Work?

–In the transformation process, XSLT uses XPath to define parts of the source document that should match one or more predefined templates.

–When a match is found, XSLT will transform the matching part of the source document into the result document.

–All major browsers such as Internet Explorer ,Chrome, Firefox, Safari and Opera supports XML, XSLT, and XPath

XSLT Cont..

XSLT <xsl:stylesheet> Element

–defines that this document is an XSLT style sheet document (along with the version number and XSLT namespace attributes).

•**XSLT <xsl:template> Element**

–An XSL style sheet consists of one or more set of **rules** that are called **templates**.

–A template contains rules to apply when a specified node is matched.

–The <xsl:template> element is used to build templates.

XSLT Cont..

- **XSLT match attribute**

–The **match** attribute is used to associate a template with an XML element.

–The match attribute can also be used to define a template for the entire XML document.

–The value of the match attribute is an XPath expression (i.e. match="/“ defines the whole document).

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

XSLT Cont..

- **XSLT <xsl:value-of> Element**

–used to extract the value of an XML element and add it to the output stream of the transformation

•**XSLT select attribute**

–contains an XPath expression. An XPath expression works like navigating a file system; a forward slash (/) selects subdirectories.

•**XSLT <xsl:for-each> and <xsl:sort> Element**

–**<xsl:for-each>** element to loop through the XML elements, and display all of the records.

XSLT Cont..

The **<xsl:sort>** element is used to sort the output.

–To sort the output, simply add an <xsl:sort> element inside the <xsl:foreach> element in the XSL file:

```
<xsl:for-each select="catalog/cd">
<xsl:sort select="artist"/>
<xsl:value-of select="title"/>
<xsl:value-of select="artist"/>
</xsl:for-each>
```

XSLT Cont..

The XML File

```
The XML File
<?xml version="1.0" encoding="UTF-8"?>
<catalog>
        <cd>
                <title>Empire Burlesque</title>
                <artist>Bob Dylan</artist>
                <country>USA</country>
                <company>Columbia</company>
                <price>10.90</price>
                <year>1985</year>
        </cd>
        .
        .
</catalog>
```

XSLT Cont..

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
<html>
        <body>
            <h2>My CD Collection</h2>
            <table border="1">
                <tr bgcolor="#9acd32">
                        <th>Title</th>
                        <th>Artist</th>
                </tr>
                <xsl:for-each select="catalog/cd">
                        <xsl:sort select="artist"/>
                        <tr>
                                <td><xsl:value-of select="title"/></td>
                                <td><xsl:value-of select="artist"/></td>
                        </tr>
                </xsl:for-each>
             </table>
        </body>
</html>
</xsl:template>
</xsl:stylesheet>
```

XSLT Cont..

XSLT <xsl:if> Element

–The <xsl:if> element is used to put a conditional test against the content of the XML file.

–**Syntax**

```
<xsl:if test="expression">
...some output if the expression is true...
</xsl:if>
```

–To add a conditional test, add the <xsl:if> element inside the <xsl:for-each> element in the XSL file.

–The value of the required **test attribute contains the expression to be evaluated**.

XSLT Cont..

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
<html>
<body>
<h2>My CD Collection</h2>
<table border="1">
<tr bgcolor="#9acd32">
<th>Title</th>
<th>Artist</th>
</tr>
```

XSLT Cont..

```
<xsl:for-each select="catalog/cd">
<xsl:if test="price &gt; 10">
<tr>
<td><xsl:value-of select="title"/></td>
<td><xsl:value-of select="artist"/></td>
<td><xsl:value-of select="price"/></td>
</tr>
</xsl:if>
</xsl:for-each>
</table>
</body>
</html>
</xsl:template>
</xsl:stylesheet>
```

XSLT Cont..

•XSLT <xsl:choose> Element

–The <xsl:choose> element is used in conjunction with <xsl:when> and <xsl:otherwise> to express multiple conditional tests.

–**Syntax**

```
<xsl:choose>
<xsl:when test="expression">
... some output ...
</xsl:when>
<xsl:otherwise>
... some output ....
</xsl:otherwise>
</xsl:choose>
```

–To insert a multiple conditional test against the XML file, add the **<xsl:choose>**, <xsl:when>, and <xsl:otherwise> elements to the XSL file:

XSLT Cont..

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
<html>
<body>
<h2>My CD Collection</h2>
<table border="1">
<tr bgcolor="#9acd32">
<th>Title</th>
<th>Artist</th>
</tr>
<xsl:for-each select="catalog/cd">
```

XSLT Cont..

```
<tr>
<td><xsl:value-of select="title"/></td>
<xsl:choose>
<xsl:when test="price &gt; 10">
<td bgcolor="#ff00ff">
<xsl:value-of select="artist"/></td>
</xsl:when>
<xsl:otherwise>
<td><xsl:value-of select="artist"/></td>
</xsl:otherwise>
</xsl:choose>
</tr>
</xsl:for-each>
</table>
</body>
</html>
</xsl:template>
</xsl:stylesheet>
```

XSLT Cont..

- **XSLT <xsl:apply-templates> Element**

–The <xsl:apply-templates> element ***applies a template to the current element or it's child nodes.***

–If we add a ***select attribute*** *to the <xsl:apply-templates> element* **it will *process only the child element that matches the value of the select attribute*.**

–We can use the select attribute ***to specify the order in which the child nodes are processed.***

–Some time XSL Style Sheet may have multiple matches

```
<xsl:template match="cd">
<p>
<xsl:apply-templates select="title"/>
<xsl:apply-templates select="artist"/>
</p>
</xsl:template>
```

XSLT Cont..

XSLT <xsl:apply-templates> Element Con…

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
<html>
<body>
<h2>My CD Collection</h2>
<xsl:apply-templates/>
</body>
</html>
</xsl:template>
<xsl:template match="cd">
```

XSLT Cont..

```
<p>
<xsl:apply-templates select="title"/>
<xsl:apply-templates select="artist"/>
</p>
</xsl:template>
<xsl:template match="title">
Title: <span style="color:#ff0000">
<xsl:value-of select="."/></span> // attribute to specify the current node
<br />
</xsl:template>
</xsl:stylesheet>
```

XML and XPath

- XPath is a syntax for defining parts of an XML document
- XPath uses path expressions to navigate in XML documents
- XPath contains a library of standard functions
- XPath is also used in XSLT, XQuery, XPointer and XLink
- Without XPath knowledge you will not be able to create XSLT documents.
- XPath is a W3C recommendation

XML and XPath cont..

there are various types of legal XPath expressions:

• **Node sets**-indicates what type of node you want to match

• **Booleans**-use the built-in XPath logical operators to produce Boolean results.

Besides Boolean values, XPath can also work with node sets.

```
<xsl:template match="state[position() > 3]">
<xsl:value-of select="."/>
</xsl:template>
```

XML and XPath cont..

Numbers- use numbers in expressions

```
<xsl:apply-templates select="state[population div area > 200]"/>
```

- **Strings-**XPath functions are specially designed to work on strings
- **Wildcard -** to select element nodes

–* -Matches any element node

–@*-Matches any attribute node

–node() -Matches any node of any kind

XML and XPath cont..

XPath Expression	Result
/bookstore/book[1]	Selects the first book element that is the child of the bookstore element
/bookstore/book[last()]	Selects the last book element that is the child of the bookstore element
/bookstore/book[last()-1]	Selects the last but one book element that is the child of the bookstore element
/bookstore/book[position()<3]	Selects the first two book elements that are children of the bookstore element
//title[@lang]	Selects all the title elements that have an attribute named lang
//title[@lang='eng']	Selects all the title elements that have an attribute named lang with a value of 'eng'
/bookstore/book[price>35.00]	Selects all the book elements of the bookstore element that have a price element with a value greater than 35.00
/bookstore/book[price>35.00]/title	Selects all the title elements of the book elements of the bookstore element that have a price element with a value greater than 35.00

XML and XPath cont..

Path Expression	Result
//book/title | //book/price	Selects all the title AND price elements of all book elements
//title | //price	Selects all the title AND price elements in the document
/bookstore/book/title | //price	Selects all the title elements of the book element of the bookstore element AND all the price elements in the document

Path Expression	Result
/bookstore/*	Selects all the child nodes of the bookstore element
//*	Selects all elements in the document
//title[@*]	Selects all title elements which have any attribute

XML and XPath cont..

Example	Result
child::book	Selects all book nodes that are children of the current node
attribute::lang	Selects the lang attribute of the current node
child::*	Selects all element children of the current node
attribute::*	Selects all attributes of the current node
child::text()	Selects all text node children of the current node
child::node()	Selects all children of the current node
descendant::book	Selects all book descendants of the current node
ancestor::book	Selects all book ancestors of the current node
ancestor-or-self::book	Selects all book ancestors of the current node - and the current as well if it is a book node
child::*/child::price	Selects all price grandchildren of the current node

end!