Lecture 9: Introduction to XML

Web Application Design and Development
(COMP 3421)

Prof. Dan Wang
dan.wang@polyu.edu.hk

THE HONG KONG POLYTECHNIC UNIVERSITY
香港理工大學

Department of Computing
電子計算學系

Outline

- Part I: Motivation and Applications
  - Motivation and History
  - Comparison with HTML
  - XML Core Technologies
  - Case Application - OFX
- Part II: XML Technologies
  - XML Syntax
  - XML Namespace
  - XML DTD

Part I: Motivation and Applications

A Brief History of XML

- XML builds on the “evolution” of TWO major extremes of markup languages: HTML & SGML
  - A **markup language** is a modern system for annotating a text in a way that is syntactically distinguishable from that text.
- The generalized markup concept emerged in early 60’s and come to a standard - SGML by ISO in 1986
- However, SGML standard was too COMPLICATED to be implemented, a simplified version of SGML was needed for the market
- Popularity of the Internet in mid 90’s resulted in the rapid adoption of HTML standard. However, HTML is on the other extreme: its backbone is too SIMPLE to be further developed. HTML is for display markup only.
- The pressure to create a new standard increase ⇨ Launch of XML version 1.0 on Feb. 1998.

What is XML?

- HTML
  - A **simple** way / *standard* to handle electronic documents over the Internet
    - Including: texts, images & graphics, video & audio clippings, ...
    - display-oriented language vs. general markup language
    - Human-centric vs. application-centric
- HTML $\Rightarrow$ **XML** $\Leftarrow$ SGML
  - A **more** concrete / structural / flexible standard to provide such a platform (compared with HTML)
  - A **realistic** solution for the provision of “generalized markup language”
  - “XML” - acronym for “e**X**tensible **M**arkup **L**anguage”
  - Language / standard not owned / dominated by any **commercial** body
  - Developed by W3C

A Brief History of XML

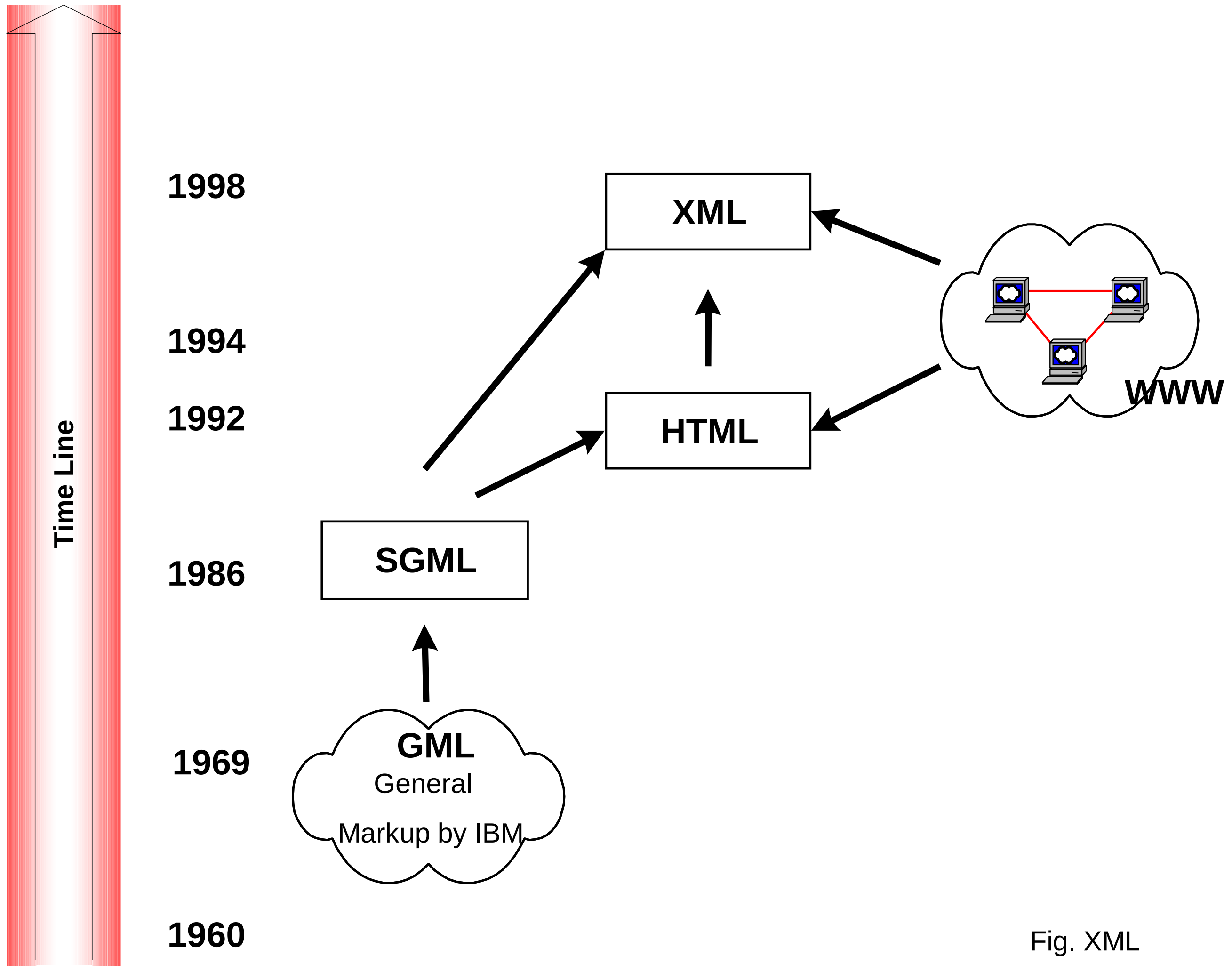


Fig. XML

XML Overview

- ***Flexible*** markup language for storing structured and semi-structured data ⇨ information
  - Structured information
    - Table format representation
    - Financial data / figures
    - Weather information: e.g. temperatures / RH of various cities over the world
    - Database records e.g. book records in library system
  - Semi-structured information
    - Tree format representation
    - Presentation Report, main sections, subsections, references
    - Spatial data, building> floors > rooms > equipment, people
- ASCII and open specification provide unified (non-proprietary) "channel" for information dissemination over the Internet

Design Philosophy

- Straightforward to use over Internet
- Support many applications
- Compatible with SGML
  - Easy to port existing SGML documents
  - Based on well tested technology
- Easy to write programs for processing XML documents, e.g. C, Java, Pascal
- Formal and concise design with minimal optional features
  - Easy to express the syntax and semantics to communicate across developers. No ambiguity
- Easy to create and human readable

XML versus HTML

- XML defines data semantics

  `<date>July 7, 2000</date>`

  - a date, but no indication how it is displayed
  - can define how to display the date using a stylesheet (CSS or XSL)

- HTML defines data display

  `<p><b>July 7, 2000</b></p>`

  - displayed in boldface within a paragraph
  - how to know it is a date?

XML versus HTML

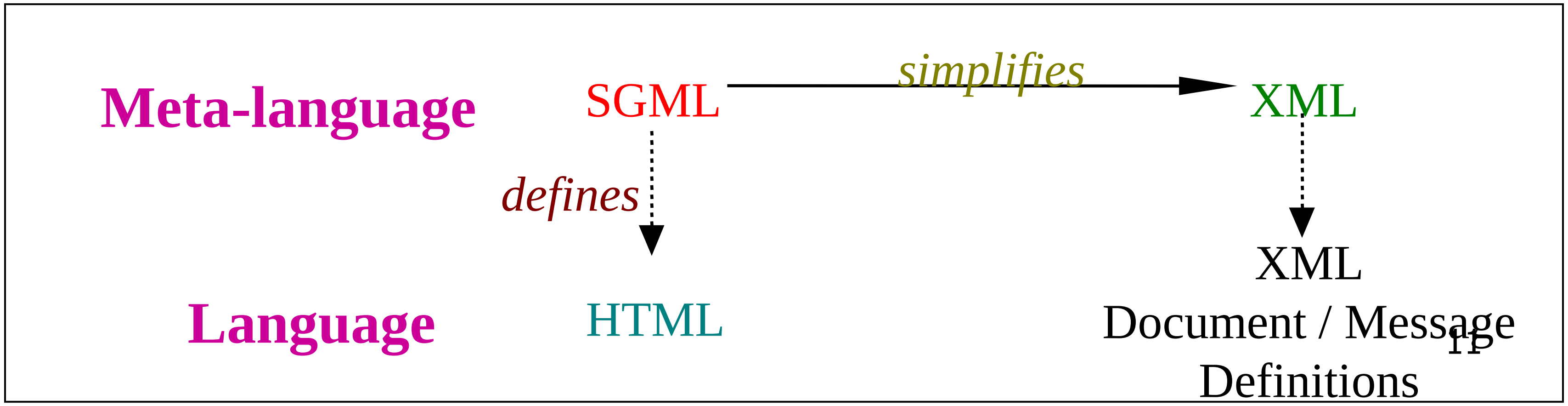
- Provide a set of rules to define semantic tags
- Tag an element with value between begin tag and end tag
- XML has a basic set of tags
- XML allows new tags to be defined

```
<dt> Where are we going?
<dd> by Jacky Chan
<ul>
<li> Producer: Jacky Chan
<li> Publisher: Polygram
<li> Length: 5:00
<li> Written: 1988
<li> Artist: Jacky Chan

 HTML mark up
```

```
<SONG>
   <TITLE>Where are we
          going?</TITLE>
   <VOCAL>Jacky Chan</VOCAL>
   <PRODUCER>Jacky Chan</PRODUCER>
   <PUBLISHER>Polygram</PUBLISHER>
   <LENGTH>5:00</LENGTH>
   <YEAR>1988</YEAR>
   <ARTIST>Jacky Chan</ARTIST>
</SONG>

XML mark up
```

XML versus HTML

- Misconception:
  - XML is a generalized HTML
- Fact:
  - XML is simplified from SGML but inherits the generalized behavior, i.e. language to create other languages
  - HTML is a specific *definition* of SGML
- Compare XML and HTML in the context of their popularity and future!
- XHTML is a derivation from XML. Which quadrant should it be?

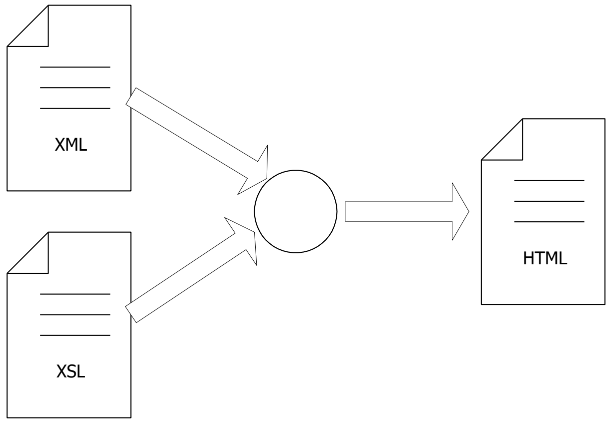


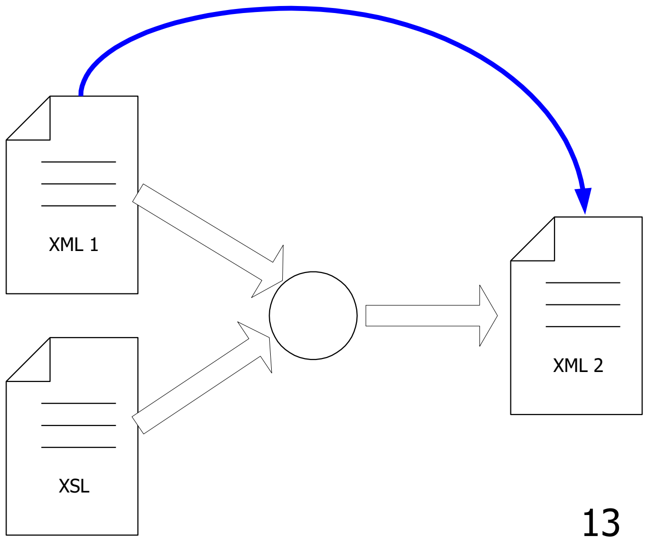
Bridging XML and HTML

- Are they really different and should be separated?
  - XML often called a meta-language to create other languages
  - HTML is a markup for display and hyperlink
- Fact today: browsers understand HTML-based technology more
- CSS (Cascading Style Sheet)- Stylesheet used for HTML, which can be adapted to XML
- XSL (eXtensible Stylesheet Language)- Stylesheet that specifies how XML document should be displayed. XML-based.
- XHTML- Derivation of HTML from XML
- DTD (Document Type Definition)- Set the rules for the structure of document

Additional to XML-XSL

- XML is not one technology but a group of technologies
- XSL- eXtensible Style Language.
    - XML is only concerned with data formatting, XSL define the style language for rendering XML, e.g. to HTML
    - Derived from DSSSL (Document Style Semantics and Specification Language) for SGML and CSS (Cascading stylesheet for HTML).
    - XSL is more general such that it can be used for XML to XML translation.
    - XSL itself is derived from XML



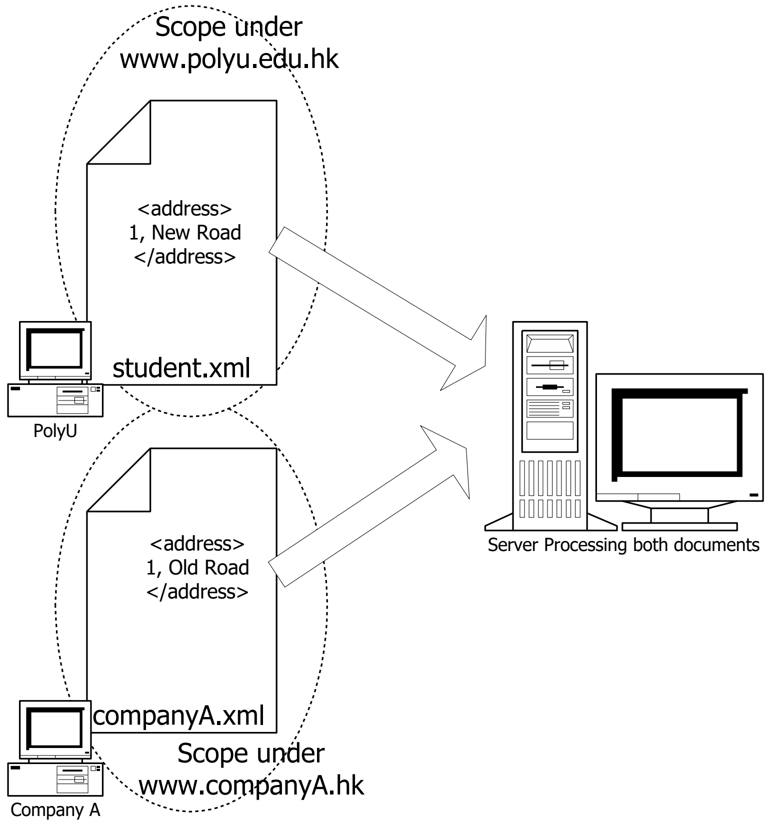


Additional to XML – XLL

- **XLL** (eXtensible Linking Language) is a powerful linking language
- HTML linking using <A href =“abc.com”> provide simple single directional link to another page.
- XLL Supports
  - Bi-direction linking of pages
  - Create external links by decoupling links from document
  - Support range.
- Specify in three working documents:
  - XPath: To specify the actual addressing of parts rather than whole XML document. Uses the notion of “path notation”.
  - XLink: uses XML syntax to specify linking properties such as single/bi-direction, multi-ended link and typed link
  - XPointer: Builds on XPath to support addressing to internal structures of XML document.

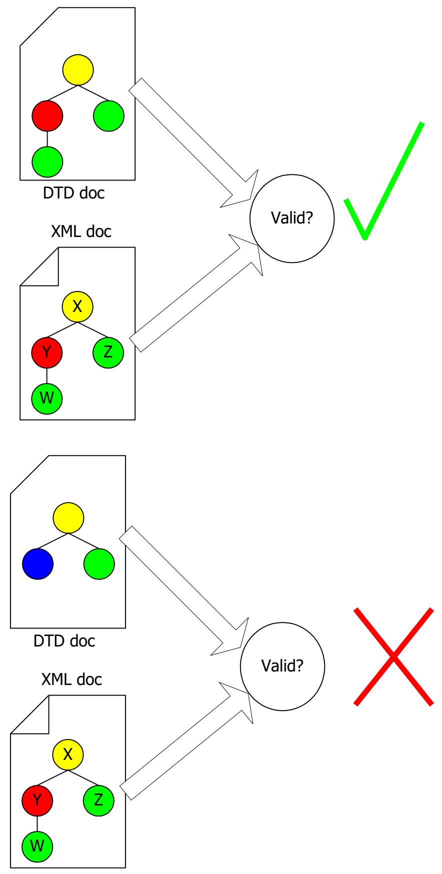
Additional to XML – Namespace

- XML is flexible to create other XML-based languages with no constraint on the structure of element names
- We may have two different XML-based languages using the same element names, e.g. <address> from student.xml and <address> from companyA.xml
- Namespace associates a unique identifier to the document to scope the context and avoid collision. For example, we can use URI



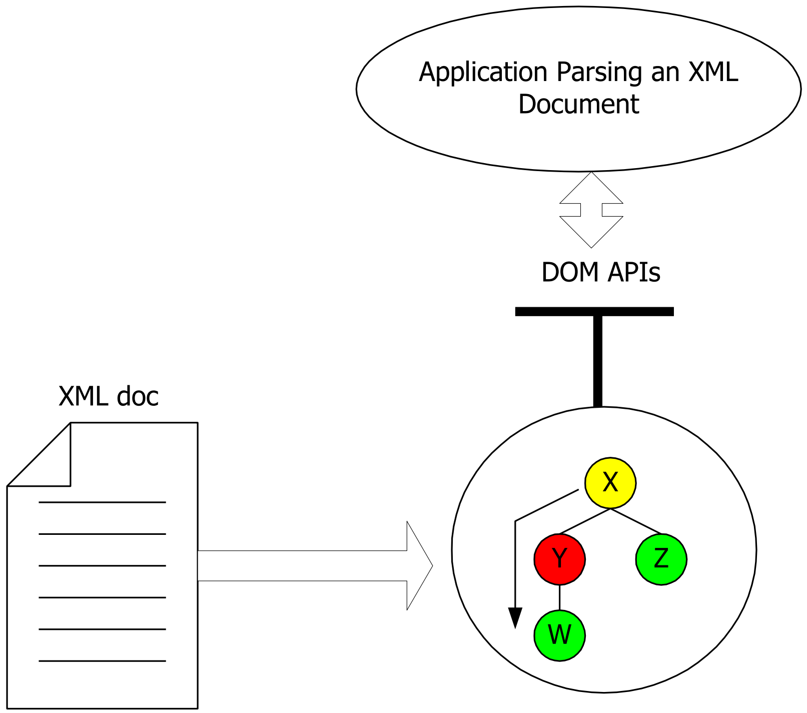
Additional to XML – DTD

- Document Type Declaration provides a mean to specify the validity of an XML document
- DTD allows us to specify the structure and the type declarations of a valid XML
- Receiver processing an XML document can check its validity against the associated DTD for that document.



Additional to XML – DOM

- Document Object Model provide a standard platform and language-neutral application interface (API) to process XML document
- It abstracts an XML document into an object based on a tree model
- APIs provided to allow application to parse document by traversing the tree nodes.
- Support read and write.



Why XML?

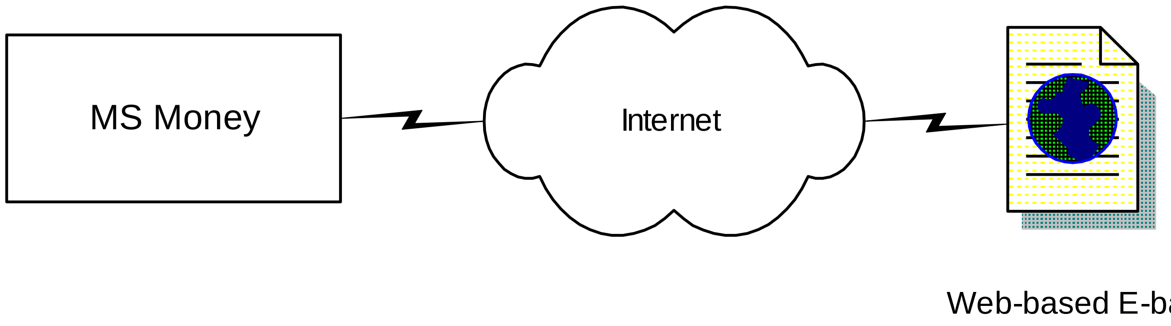
- Describe structures. Brings meaning to data rather than just display friendly.
- Self describing data. By looking at plain XML we can deduce its semantics….cf. HTML
- Structured and integrated data. Allows complex hierarchy and relationships to be built using DTD.
- Data interchange. True openness and the least common denominator for (ASCII representation) inter-operation
- Extension of markup capability

Applications of XML

- Structuring of complex documents. Can use for semi-structured data representation.
- Presenting database contents across system, especially semi-structured
- XML meta-extensions for different application-oriented data
- Data interchange for E-commerce
- For protocol interactions such as SOAP (web services), XML-RPC and publishing with CDF (compound document format)

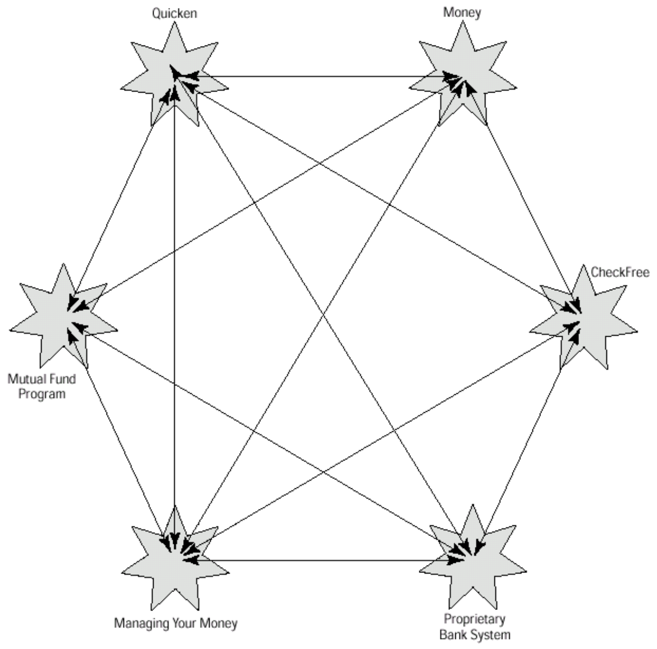
Case Study: Open Financial Exchange (OFX)

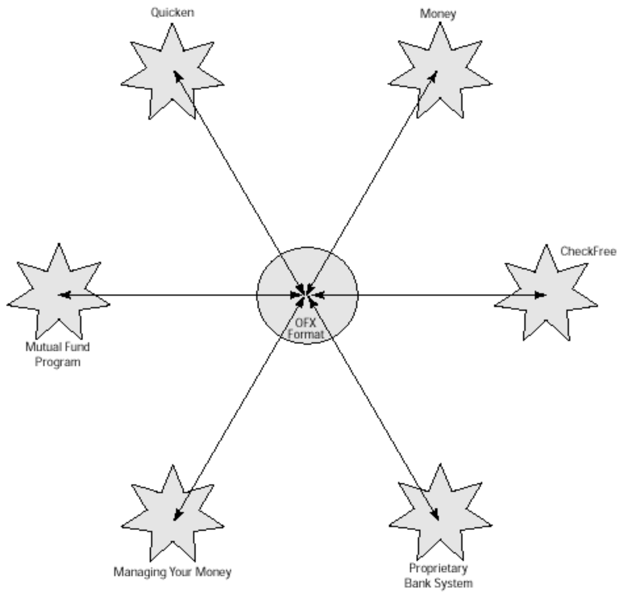
- Many financial packages available such as Quicken, Microsoft Money and even Web-based e-banking
- To exchange data between software is difficult: formats are different
    - Convert the data format using bridging software
    - Re-enter information
- Any chance for MS Money to co-ordinate with your web-based e-banking?



OFX

- OFX is an application of XML to describe financial data
- OFX is open and can be used as a common format for exchange of data between applications
- Allows local applications such as MS Money and Quicken, as well as online web applications to exchange





Hello XML

- XML documents can either be:
  - **Well-formed**: document requires only to conform to minimum criteria. E.g. tags must be properly formed, even for empty ones.
  - **Valid document**: needs to conform to a pre-defined set of data structure as spelled out in DTD.

```
<?xml version="1.0" standalone="yes"?>
<GREETINGS>
     Hello XML!
</GREETINGS>
```

- Alternatively,

```
<?xml version="1.0" standalone="yes"?>
<DOCUMENT>
     Hello XML!
</DOCUMENTS>
```

In HTML like,

```
<?xml version="1.0" standalone="yes"?>
<P>
     Hello XML!
</P>
```

Hello XML

Markup tags can have three kinds of meaning: structure, semantics and style.

- *Structure* expresses the form of the document. Just like when we write a business letter, there must be a standard format to adhere.
- *Semantics*: the tags can also brings out the semantics or meaning of its existence. This meaning must be in agreement with programs or people reading the document, e.g. what does GREETINGS mean?
- *Style* specifies how the content of a tag is to be displayed

Hello HTML

- So far, we see how XML provides ways to introduce structure to a document. How to display it?
- Style sheet provides instructions to browser on how to display data contained within it.
- Two ways CSS (Cascading Style Sheet) and XSL (Extensible Style Language). More XSL to come in following weeks.
- CSS is HTML-based
- E.g. greetings.css

```
GREETINGS {display: block; font-size: 48 pt; font-weight: bold;}
```

- Greetings.xml

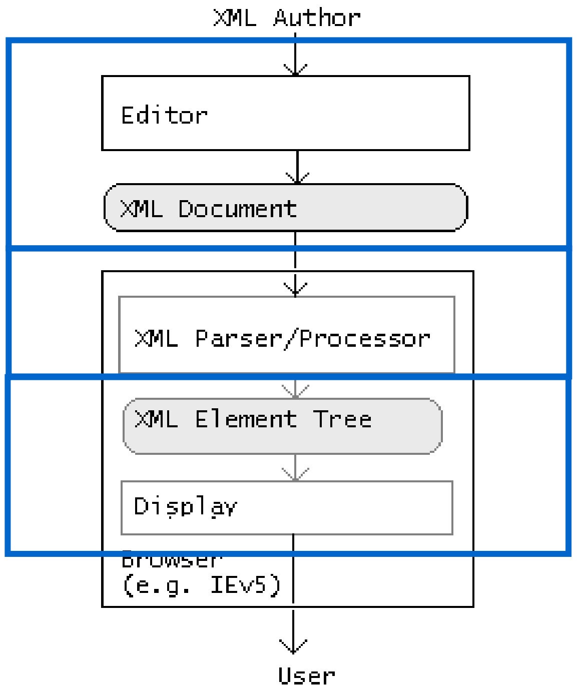
```
<?xml version=“1.0” standalone=“yes”?>
<?xml-stylesheet type=“text/ccs” href=greetings.css”?>
<GREETINGS>
   Hello XML!
</GREETINGS>
```

Run it!

Part II: XML Technologies

Life Cycle of XML Document

- XML document can be created by specialized XML author, via editor
  - alternatively created by programs
- XML document must be verified to be correct in syntax, via XML parser/processor
- Elements in XML document are then displayed, via browser



```
<?xml version='1.0'?>
<book>
  <isbn>123-456</isbn>
  <title>Internet Computing</title>
  <chapters>
    <chapter num='1'> Introduction </chapter>
    <chapter num='2'>XML Syntax</chapter>
  </chapters>
</book>
```

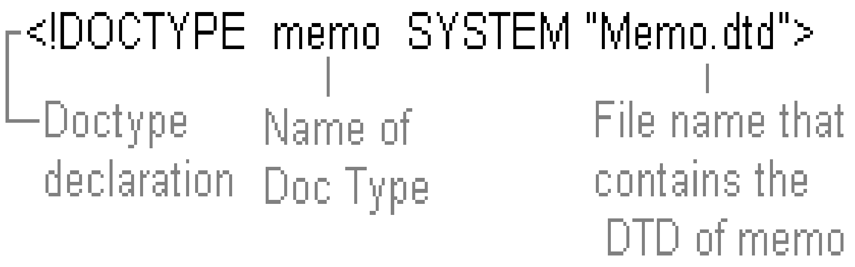
Test it

XML Syntax: Prolog

- Prolog is opening section
  - defines the language to be used for the document
  - declaration for XML includes a version number
  - <? *** ?> defines a processing instruction. Provides info to be used by software application.

  <?noisemaker noise="sound.wav"?>
  - unlike HTML, XML is case-sensitive, e.g. <title>, <TITLE>
  - DTD used to validate XML document. It can be specified either internally and externally (either public or user-defined).

```
<?xml version="1.0"?>
<!DOCTYPE HTML PUBLIC "//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/html/html40.dtd ">
```


<!DOCTYPE memo SYSTEM "Memo.dtd">
Doctype declaration
Name of Doc Type
File name that contains the DTD of memo

XML Syntax: Document

- XML documents made up of many elements
- Each XML document must have a single root element that contains all other elements
  - called the root element
  - no specific tag as with HTML document (<html>)
- XML content defined with
  - elements
  - attributes
  - comments
  - processing instructions

XML Syntax: Document

- Elements
  - primary logical components of a document
  - must have start and end tags
    - e.g. <*element*>something here</*element*>
  - null content elements can be condensed
    - e.g. <br/> stands for <br></br> on XML
- Attributes
  - a property of an element
  - each attribute has a name and a value
  - expressed as <*element attribute*=value>
    - e.g. <book title=“Using XML” year=“1999”>book1</book>
    - Any example in HTML?
    - Other alternatives?

XML Syntax: Document

- Comments
  - To add clarity to the XML document
  - begin with markup declaration open delimiter <! and followed by comment open delimiter --
  - close with comment close delimiter -- and markup declaration close delimiter >
    - e.g. <!-- this is a sample comment line -->
- Processing Instructions
  - provide information to applications
  - enclosed by <? and ?> pairs
    - e.g. <?exec project="main.java" level="2"?>
  - XML parser passes data to the application (e.g. exec)

XML Namespaces

- Elements and attributes names are freely assigned e.g. <name> in customer.xml and <name> in product.xml

```
<customer>
  <name>Steven Lam</name>
</customer>
```

```
<product>
  <name>Sony Player<name>
</product>
```

- Tag sets from different applications may conflict
- To see the confusion, lets assume the customer orders a Sony player.

```
<customer>
     <name>Steven Lam</name>
     <order>
       <product>
          <name>Sony Player</name>
       </product>
     </order>
</customer>
```

- As human we can differentiate the difference in the <name> tags, but not the parser
- With namespace parser can easily tell the difference between the elements

XML Namespaces

- Namespaces can be declared using one of two methods:
  - Default declaration
  - Explicit declaration
- Default namespace specifies a namespace to use for ALL child elements under the current element
- Here, we create a default declaration for <customer> element by using xmlns attribute

```
<customer xmlns=“http://www.e-store.com/customer”>>
    <name>Steven Lam</name>
    <order xmlns=“http://www.e-store.com/order”>
      <product>
         <name>Sony Player</name>
      </product>
    </order>
</customer>
```

- All elements under <customer> then inherits the namespace of http://www.e-store.com/customer
- The inheritance is interrupted by the xmlns attribute of order element

XML Namespaces

- Sometimes it may not be easy to create nested default declaration, i.e. order overriding customer namespace.
- Explicit declaration uses prefix to associate with the xmlns attribute. Then uses this prefix to associate with the namespace of the element.

```
<cust:customer xmlns:cust="http://www.e-store.com/customer"
 xmlns:ord ="http://www.e-store.com/order" >
    <cust:name>Steven Lam</name>
     <ord:order>
      <ord:product>
          <ord:name>Sony Player</ord:name>
            <cust:visa>4977 6767 1234 2345</cust:visa>
      </ord:product>
     </ord:order>
</cust:customer>
```

- By identifying the namespace, parser can use different rules to be applied for customer names versus product names.

XML Namespaces

XML: Name Space

Name Conflict

```
<customer>
        <name>
         Steven Lam
        </name>
        <order>
         <product>
            <name>
          Sony Player
            </name>
         </product>
        </order>
</customer>
```

Define name space using xmlns

```
<customer xmlns=“customer”>
        <name>
          Steven Lam
        </name>
        <order xmlns=“order”>
          <product>
            <name>
              Sony Player
            </name>
         </product>
        </order>
</customer>
```

Use explicit names with xmlns

```
<cs:customer xmlns:cs=“customer”>
        <cs:name>
          Steven Lam
        </cs:name>
        <or:order xmlns:or=“order”>
          <or:product>
            <or:name>
              Sony Player
            </or:name>
         </or:product>
        </or:order>
</cs:customer>
```

Part II: XML Technologies – DTD

Review for the Previous Lecture

XML: A meta-language to generate mark-up languages

XML documents consist of: *Elements, Attributes, Comments, Processing Instructions.*

Well-formed

Well-Formedness:

1. **There is only one root element that contains all other elements.**
2. **Each nonempty element has both start and end tags.**
3. **Elements nest correctly.**

   ……..

```
<?xml version="1.0"?>
<!-- The root element -->
<Document>
      <GREETINGS level="Excited">
            Hello XML!
      </GREETINGS>
      <MESSAGE>
            Welcome to the XML world!
      </MESSAGE>
 </Document>
```

NO Root elements.

```
<?xml version="1.0"?>
<GREETINGS level="Excited">
            Hello XML!
 </GREETINGS>
 <MESSAGE>
            Welcome to the XML world!
<GIFT>
 </MESSAGE>
            A new job with more money.
</GIFT>
```

✗

Why we need DTD?

```
<?xml version="1.0"?>
<Document>
   <GREETINGS level="Excited">
         Hello XML!
    </GREETINGS>
   <MESSAGE>
        Welcome to the XML world!
   </MESSAGE>
</Document>
```

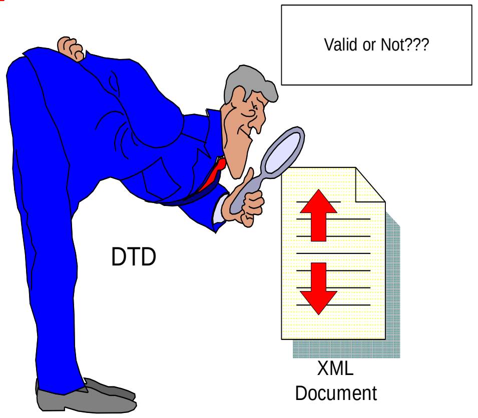
```
<?xml version="1.0"?>
<Document>
   <GREETINGS level="Excited">
            Hello XML!
    </GREETINGS>
   <MESSAGE>
            Welcome to the XML world!
   </MESSAGE>
   <Message>
            No matter you like or not,
            you have to learn XML.
   </Message>
</Document>
```

- **Both are well-formed XML documents.**
- **Which one is correct or both are correct/wrong?**
- **Based on well-formedness, without a structure definition, we can not decide.**

We need a method to define *the document structure* so we can validate it later.

Document Type Definitions (DTD)

- Previously describe about the syntax of XML and how to create a well-formed structured XML document
- DTD is used to define the legal building blocks of an XML document. It defines the document structure with a list of legal elements.
- XML document is compared against a DTD to check for its validity
- In other words, DTD is like a dictionary that defines how a valid XML document should be formatted and the valid elements, tags, attributes.



Document Type Definitions (DTD)

- DTD appears after the XML declaration but before the root element.

```
<?xml version="1.0" standalone="yes"?>
<!DOCTYPE GREETING [
 <!ELEMENT GREETING (#PCDATA)>
]>
<GREETING>
Hello XML!
</GREETING>
```

XML Declaration

DTD

XML Document Contents

- It says, it contains one element and the element may contain Parsed Character Data, i.e. any text that is not the markup text.

- Begins with <!DOCTYPE followed by the root element of the document, i.e., GREETING.

- Within the brackets ([..]) are a list of declarations of the elements, attributes, etc containing within the xml

Document Type Definition (DTD)

Valid or Not!

```
<GREETING>
            <SAY> Hello World</SAY>
</GREETING>
```

✕

```
<GREETING>

            <GREETING>

                        HelloWorld

            </GREETING>

</GREETING>
```

✕

There are many validating parsers available on the web for free. A simple online one is https://www.truugo.com/xml_validator/

Internal DTD & External DTD

- DTD can be put within an XML document (Internal DTD) or outside of an XML document (External DTD).

```
<?xml version="1.0" standalone="yes"?>
<!DOCTYPE GREETING [
  <!ELEMENT GREETING (#PCDATA)>
]>
<GREETING>
Hello XML!
</GREETING>
```

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE GREETING  SYSTEM ="x.dtd">
<GREETING>
Hello XML!
</GREETING>

x.dtd
<!ELEMENT GREETING (#PCDATA)>
```

- How to validate an XML document with an external DTD?
  - Put the DTD file online (In the example, x.dtd is put into my unix home directory).
  - In the XML document, refer it using URL

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE GREETING  SYSTEM  "http://www.comp.polyu.edu.hk/~csdwang/x.dtd">
<GREETING>
Hello XML!
</GREETING>
```

Declaration of Elements in DTD

- A DTD section begins with !DOCTYPE declaration, which specifies the root tag of the document, e.g.

```
<!DOCTYPE SEASON[
    elements declaration
]>
```

- XML defines four types of contents for elements
  - **Empty**: no content
  - **Any**: any type of content. Avoid using if possible
  - **Element**: only contents made up of other elements
  - **#PCDATA**: contents contain parsed characters
  - **Mixed**: contents from other elements or text

- Specify the structure of elements
  - **?** means optional (zero or one)
  - ***** means optional or more (zero or more)
  - **+** means required or more (one or more)
  - **,** means all of the list
  - **|** means choice of any one

Element Structures

```
<!DOCTYPE WORK_SCHEDULE [

        <!ELEMENT WORK_SCHEDULE (PEOPLE*)>
        <!ELEMENT PEOPLE (NAME,DAY+)>

        <!ELEMENT NAME (#PCDATA)>
        <!ELEMENT DAY (DATE, (HOLIDAY|SLOT+))>

        <!ELEMENT DATE (#PCDATA)>
        <!ELEMENT HOLIDAY (#PCDATA)>

        <!ELEMENT SLOT (TIME,TITLE,DESCRIPTION?)>

         <!ELEMENT TIME (#PCDATA)>
        <!ELEMENT TITLE (#PCDATA)>
        <!ELEMENT DESCRIPTION (#PCDATA)>

        <!ATTLIST NAME POSITION CDATA #REQUIRED>

 ]>
```

```
<?xml version="1.0" standalone="yes"?>
<!DOCTYPE WORK_SCHEDULE [
 …
]>
<WORK_SCHEDULE>
</WORK_SCHEDULE>
```

```
<?xml version="1.0" standalone="yes"?>
<!DOCTYPE WORK_SCHEDULE [
…
]>
<WORK_SCHEDULE>
  <PEOPLE>
     <NAME POSITION=“faculty"> Dan </NAME>
     <DAY>
        <DATE> Dec. 25, 2001 </DATE>
        <HOLIDAY> Christmas </HOLIDAY>
      </DAY>
  </PEOPLE>
</WORK_SCHEDULE>
```

Element Structures

```
<!DOCTYPE WORK_SCHEDULE [

        <!ELEMENT WORK_SCHEDULE (PEOPLE*)>
        <!ELEMENT PEOPLE (NAME,DAY+)>

        <!ELEMENT NAME (#PCDATA)>
        <!ELEMENT DAY (DATE, (HOLIDAY|SLOT+))>

        <!ELEMENT DATE (#PCDATA)>
        <!ELEMENT HOLIDAY (#PCDATA)>

        <!ELEMENT SLOT (TIME,TITLE,DESCRIPTION?)>

         <!ELEMENT TIME (#PCDATA)>
        <!ELEMENT TITLE (#PCDATA)>
        <!ELEMENT DESCRIPTION (#PCDATA)>

        <!ATTLIST NAME POSITION CDATA #REQUIRED>

 ]>
```

NOTES:

1. **Element definitions can be in any orders in DTD.**
2. **Elements in XML *MUST* follow the orders defined in DTD.**

```
<?xml version="1.0" standalone="yes"?>
<!DOCTYPE WORK_SCHEDULE [
…
]>
<WORK_SCHEDULE>
  <PEOPLE>
   <DAY>
      <DATE> Dec. 25, 2001 </DATE>
      <HOLIDAY> Christmas </HOLIDAY>
   </DAY>
   <NAME POSITION=“faculty" > Dan </NAME>
  </PEOPLE>
</WORK_SCHEDULE>
```

Element Structures

```
<!DOCTYPE WORK_SCHEDULE [

        <!ELEMENT WORK_SCHEDULE (PEOPLE*)>
        <!ELEMENT PEOPLE (NAME,DAY+)>

        <!ELEMENT NAME (#PCDATA)>
        <!ELEMENT DAY (DATE, (HOLIDAY|SLOT+))>

        <!ELEMENT DATE (#PCDATA)>
        <!ELEMENT HOLIDAY (#PCDATA)>

        <!ELEMENT SLOT (TIME,TITLE,DESCRIPTION?)>

         <!ELEMENT TIME (#PCDATA)>
        <!ELEMENT TITLE (#PCDATA)>
        <!ELEMENT DESCRIPTION (#PCDATA)>

        <!ATTLIST NAME POSITION CDATA #REQUIRED>

 ]>
```

```
<?xml version="1.0" standalone="yes"?>
<!DOCTYPE WORK_SCHEDULE [
…
]>
<WORK_SCHEDULE>
  <PEOPLE>
    <NAME POSITION="faculty" > Dan </NAME>
    <DAY>
      <DATE> Dec. 25, 2001 </DATE>
      <HOLIDAY> Christmas </HOLIDAY>
      <SLOT>
       <TIME> 9am-10am </TIME>
       <TITLE> Lab </TITLE>
     </SLOT>
    </DAY>
  </PEOPLE>
</WORK_SCHEDULE>
```

Element Structures

```
<!DOCTYPE WORK_SCHEDULE [

        <!ELEMENT WORK_SCHEDULE (PEOPLE*)>
        <!ELEMENT PEOPLE (NAME,DAY+)>

        <!ELEMENT NAME (#PCDATA)>
        <!ELEMENT DAY (DATE, (HOLIDAY|SLOT+))>

        <!ELEMENT DATE (#PCDATA)>
        <!ELEMENT HOLIDAY (#PCDATA)>

        <!ELEMENT SLOT (TIME,TITLE,DESCRIPTION?)>

         <!ELEMENT TIME (#PCDATA)>
        <!ELEMENT TITLE (#PCDATA)>
        <!ELEMENT DESCRIPTION (#PCDATA)>

        <!ATTLIST NAME POSITION CDATA #REQUIRED>

 ]>
```

```
<?xml version="1.0" standalone="yes"?>
<!DOCTYPE WORK_SCHEDULE [
…
]>
<WORK_SCHEDULE>
  <PEOPLE>
    <NAME POSITION=“faculty" > Dan </NAME>
   <DAY>
      <DATE> Dec. 25, 2001 </DATE>
     <SLOT>
       <TIME> 9am-10am </TIME>
       <TITLE> Lab </TITLE>
    </SLOT>
   <SLOT>
       <TIME> 10am-11am </TIME>
       <TITLE> XML Course </TITLE>
       <DESCRIPTION>
               XML is beautiful.
       </DESCRIPTION>
    </SLOT>
   </DAY>
 </PEOPLE>
</WORK_SCHEDULE>
```

Element Content Types: Empty & Any

- **EMPTY**: declare <!ELEMENT email EMPTY> in the DTD:
  - only have attribute-value attached
  - e.g. <email date="1.2.2000"/>

- **ANY**: declare <!ELEMENT testElement ANY> in the DTD:
  - anything can be allowed in the element
  - e.g. <testElement>A sample element</testElement>
  - <testElement><test>Hi</test></testElement>

Element Content Types: Elements & PCDATA

- **Elements**: only elements are allowed
  - <!ELEMENT pizza (seafood | meat | vegetable | mixed)>
  - <!ELEMENT pizza (vegetables*, red meat+, cheese?, pepperoni)>
  - Elements in XML documents must follow the order defined in the brackets.

- **PCDATA** (Parsed Character Data):   the text will be examined by the parser for entities and markup. PCDATA should not contain any &, <, or > characters; these need to be represented by ***the predefined general entities***: &amp; &lt; and &gt; respectively.

e.g., <!ELEMENT GREETING (#PCDATA)>

```
<GREETING>
        This text is inside the &lt; GREETING &gt; element;
</GREETING>
```

Element Content Type: Mixed

- **Mixed**: text plus elements

<!ELEMENT textSub (#PCDATA | subElement)*>

```
<!ELEMENT myMessage(#PCDATA | message)*>
<myMessage> Here is some text, some
  <message> other text </message> and
  <message> even more text </message>
</mymessage>
```

Entities

- An entity is simply an XML way to refer to a data item.

- There are two kinds of entities:
  - Reference entities: To be used in XML documents for data reference.
    - Internal entity (general entity)
      Define in DTD: <!ENTITY name “refer_data”>
      e.g. <!ENTITY today “10/4/08">
    - External entity:
      Define in DTD: <!ENTITY name SYSTEM/PUBLIC “file_name” >
      e.g. <!ENTITY today “data.xml”>

    Use in XML document (reference): &today;

  - Parameter entities: To be used inside DTD itself for data reference.
    Define in DTD: <!ENTITY % record “(Name, Birthday, ID)” >
    Use in DTD: %record;

Reference (General) Entities

- Five **predefined general entities**:

  &lt;→ <   &gt;→ >   &amp;→&   &quot;→ "   &apos; → ‘

- Define an **internal general entity**: <!ENTITY *name* “*definition*”>

  e.g. <!ENTITY TODAY **“April 10, 2008”>**

  Use in the XML document: <DATE> &TODAY; </DATE>

- Define an **external general entity**:

  <!ENTITY *name* SYSTEM/PUBLIC “*file_name*”>

  e.g. <!ENTITY TODAY SYSTEM “data.xml”>

  data.xml then can store April, 10, 2008

  Use in the XML document: <DATE> &TODAY; </DATE>

Parameter Entities

- Parameter entities: To be used inside DTD itself for data reference.

- Define in DTD: **<!ENTITY % *name* "*refer_data*" >**

  e.g. <!ENTITY % record "(Name, Birthday, ID)" >

  Use in DTD: %record;

- Note that:

  In an *internal* DTD, parameter entities can **not** be used inside an element declaration;

  In an *external* DTD, parameter entities can be used anywhere.

An Example for Parameter Entities

```
<?xml version="1.0"
standalone="yes"?>
<!DOCTYPE document [
<!ENTITY  % br "<!ELEMENT
BR EMPTY>" >
<!ELEMENT document
((student|staff), BR)* >
<!ELEMENT student (name,id) >
<!ELEMENT staff (name, id) >
<!ELEMENT name (#PCDATA) >
<!ELEMENT id (#PCDATA) >
%br;
]> ✓

<document>
  <student>
      <name> Jason </name>
     <id>  12dfdf1    </id>
  </student>
  <BR />

  <student>
      <name> Jack  </name>
      <id>  198881 </id>
  </student>
  <BR />
</document>
```

parameter_entity.dtd

```
<!ENTITY  % record "(name,id)" >
<!ELEMENT document
((student|staff), BR)* >
<!ELEMENT student (%record;) >
<!ELEMENT staff (%record;) >
<!ELEMENT name (#PCDATA) > ✓
<!ELEMENT id (#PCDATA) >

<!ELEMENT BR EMPTY>
```

```
<?xml version="1.0"
standalone="no"?>

<!DOCTYPE document SYSTEM
"http://www.comp.polyu.edu.hk/~csd
wang/parameter_entity.dtd">

<document>
  <student>  <name>Jason </name>
                  <id> 12dfdf1 </id>
  </student>
  <BR />
  <student><name>Jason</name>
                 <id>12dfdf1</id>
  </student>
  <BR />
</document>
```

```
<?xml version="1.0"
standalone="no"?>
<!DOCTYPE document [
<!ENTITY  % record "(name,id)" >
<!ELEMENT document
((student|staff), BR)* >
<!ELEMENT student (%record;) > ✗
<!ELEMENT staff (%record;) >
<!ELEMENT name (#PCDATA) >
<!ELEMENT id (#PCDATA) >
<!ELEMENT BR EMPTY>
]>
<document>
  <student>
      <name>Jason</name>
      <id>12dfdf1</id>
  </student>
  <BR />

  <student>
       <name>Jason</name>
      <id>12dfdf1</id>
  </student>
  <BR />

</document>
```

Declaration of Attributes in DTD

- Attributes are extra information associated with an element

- Each attribute having a name/value pair and separated by space

- Example:

```
<GREETING LANGUAGE="English">
 Hello XML!
</GREETING>
```

- The LANGUAGE is a useful information of the content, but is itself not part of the content.

Declaring Attributes in DTD

- Declare using <!ATTLIST> with the form of:
  - **<!ATTLIST Element_name Attribute_name Type Default_value**>
  - <!ATTLIST GREETING LANGUAGE CDATA “English”>
  - CDATA is essentially the same as PCDATA for elements

- Instead of specifying explicit default value, you can force a value to the attribute:
  - **#REQUIRED** forces that attribute to be mandatory
  - **#IMPLIED** optional attribute
  - **#FIXED** fixes the attribute value without allowing document to change it!

```
<AUTHOR NAME=“James” COMPANY=“Amazon” FAX=“123456”>
<!ELEMENT AUTHOR>
<!ATTLIST AUTHOR NAME CDATA #REQUIRED>
<!ATTLIST AUTHOR COMPANY CDATA #FIXED “Amazon”>
<!ATTLIST AUTHOR FAX CDATA #IMPLIED>
```

Types of Attributes

- Several types of attributes
  - **CDATA** string types: a string of characters. May contain any string of text except less-than sign (<) or quotation marks (“). Can still be inserted by entitiy references ($lt and $quot)

    <RECTANGLE LENGTH=“7” WIDTH=“8”/>

  - **NMTOKEN** tokenized types: prevent insertion of whitespaces. Use to restrict a value to a valid XML name, e.g., if you want document to use abbreviation HK rather than Hong Kong

    <!ATTLIST ADDR COUNTRY NMTOKEN #REQUIRED>

  - **NMTOKENS** plural form of NMTOKEN. Consist of multiple XML names separated by whitespace e.g.

    <!ATTLIST ADDRESS STATES NMTOKENS #REQUIRED>
    <ADDRESS STATES=“MI NY LA CA”>

Types of Attributes

- **ID** type uniquely identifies the element in document. Must be valid XML name. Use to uniquely identify an element within the same element name.

```
<!ATTLIST P PNUMBER ID #REQUIRED>
<P PNUMBER=“p1”>This is a test</P>
<P PNUMBER=“p2”>This is a test</P>
```

- **IDREF** is the ID of another element in the document

```
<?xml version="1.0" standalone="yes"?>
<!DOCTYPE DOCUMENT [
  <!ELEMENT DOCUMENT (PERSON*)>
  <!ELEMENT PERSON (#PCDATA)>
  <!ATTLIST PERSON PNUMBER ID #REQUIRED>
  <!ATTLIST PERSON FATHER IDREF #IMPLIED>
  <!ATTLIST PERSON MOTHER IDREF #IMPLIED>
]>
<DOCUMENT>
  <PERSON PNUMBER="a1">Susan</PERSON>
  <PERSON PNUMBER="a2">Jack</PERSON>
  <PERSON PNUMBER="a3" MOTHER="a1" FATHER="a2">Chelsea</PERSON>
  <PERSON PNUMBER="a4" MOTHER="a1" FATHER="a2">David</PERSON>
</DOCUMENT>
```

Types of Attributes

- **Enumerated types**: a unique choice from a set of values, separated by vertical bar (|) e.g., <!ATTLIST MASTER PROGRAM (MScIS|MScIT|MScST|MScEC) “MScIT”>
- **Entity** enables linking of external binary data, i.e. unparsed entity into document, e.g. image

```
<!ELEMENT IMAGE EMPTY>
<!ATTLIST IMAGE SOURCE ENTITY #REQUIRED>
<!ENTITY LOGO SYSTEM “logo.gif”>
```

The desired image can be inserted using the IMAGE element tag:

```
<IMAGE SOURCE=“LOGO”/>
```

- **Entities** plural form of Entity. Allows attribute to point to multiple unparsed entities separated by whitespace.

```
<!ELEMENT SLIDESHOW EMPTY>
<!ATTLIST SLIDESHOW SOURCES ENTITIES #REQUIRED>
<!ENTITIY PIC1 SYSTEM “cat.gif”>
<!ENTITY PIC2 SYSTEM “dog.gif”>
<!ENTITY PIC3 SYSTEM “cow.gif”>
```

To insert in the document,

```
<SLIDESHOW SOURCES=“PIC1 PIC2 PIC3”>
```