Document Type Definition
DTDs

Document Type Definition

- A DTD defines the legal elements of an XML document.
- In simple words we can say that a DTD defines the document structure with a list of legal elements and attributes.
- Actually DTD and XML schema both are used to form a well formed XML document.
- We should avoid errors in XML documents because they will stop the XML programs.
- **A single DTD can be used in many XML files.**

Document Type Definition

- Data sent along with a DTD is known as valid XML.
  In this case, an XML parser could check incoming data against the rules defined in the DTD to make sure the data was structured correctly.
- Data sent without a DTD is Known as well-formed XML.
  Here an XML-based document instance , such as hierarchically structured weather data shown ,can be used to implicitly describe itself..
- With both valid and well-formed XML,XML encoded data is self-describing since descriptive tags are intermixed with the data.
- DTD's help ensure that different people and programs can read each other's files.
- The DTD defines exactly what is and is not allowed to appear inside a document

XML Parsers

- A *parser* is a piece of program that takes a physical representation of some data and converts it into an in-memory form for the program as a whole to use.
- One of the ways to classify XML parsers is as below :
  - non-validating**:** the parser does not check a document against any DTD (Document Type Definition); only checks that the document is *well-formed* (that it is properly marked up according to XML syntax rules)
  - validating**:** in addition to checking well-formedness, the parser verifies that the XML document conforms to a specific DTD i.e. the xml file is written as per defined by the DTD.

XML Parsers

- An xml document is well formed if it follows all the rules for well formedness.
- It is valid if it is well formed and follows a DTD.
- So,
- *All valid xml documents are well formed but all well formed documents may not be valid.*

Document Type Definition

- **They have a file extension .dtd**
- **A DTD is a document which serves the following purposes:**
  - Specify the valid tags that can be used in a XML document.
  - Specify the valid tag sequence/arrangements.
  - Specifies whether whitespace is significant or ignorable.
- **The DTD used by a XML document is declared after the prolog.**
- **DTD defines the structure of xml data and is similar to class template.**

DTD for Our Simple XML

A DTD Consists of a left square bracket character ([ ) followed by a series of markup declarations , followed by a right square bracket character ( ]).

```
<?xml version=“1.0” standalone=“yes”?>
<!DOCTYPE  message
[
<!ELEMENT message ANY>
]
>
<message> This is most simplest XML document I have ever seen
</message>
```

This example shows:

The document type is defined with the root element named as “message".

The element “message" is defined to have "any" sub elements, text content, and attributes.

Main features of DTD:

- ➢DTD is a simple language with only 4 types of statements: DOCTYPE, ELEMENT, ATTLIST, and ENTITY.
- ➢One DOCTYPE statement defines one document type.
- ➢Within the DOCTYPE statement, one or more ELEMENT statements, some ATTLIST statements and some ENTITY statements are included to define details of the document type.
- ➢DTD statements that define the document type can be included inside the XML file.
- ➢DTD statements that define the document type can be stored as a separate file and linked to the XML file.

DTD Declarations

DOCTYPE is a DTD statement included in an XML file to declare that this XML document has been linked to DTD document type and to specify the name of the root element of this XML document.

Valid syntax formats for DOCTYPE statement are:

```
<!DOCTYPE root_element [
  internal_DTD_statements
]>
<!DOCTYPE root_element SYSTEM "DTD_location">
<!DOCTYPE root_element SYSTEM "DTD_location" [
  internal_DTD_statements
]>
<!DOCTYPE root_element PUBLIC "DTD_name" "DTD_location">
<!DOCTYPE root_element PUBLIC "DTD_name" "DTD_location"
  internal_DTD_statements
]>
```

DTD Types

- Internal DTD :
  - Xml and DTD are in the same file
  - <! DOCTYPE root-element [element-declarations] >
- External DTD :
  - A dtd that resides in a file (.dtd extension) other than the xml file
  - Advantage of an external dtd is reusability
  - Can be
    - Private : shared by a group of people.
      - <!DOCTYPE root-element SYSTEM "filename“>
      - In place of filename you can specify an url also
    - Public : when the dtd is intended for public use
      - <!DOCTYPE root-element PUBLIC “dtd_name” “dtd_URL”>

Internal Dtd

- **Sample xml file with an internal dtd :**

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

<!DOCTYPE book [
<!ELEMENT book (title,noofpages,price,edition)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT noofpages (#PCDATA)>
<!ELEMENT price (#PCDATA)>
<!ELEMENT edition (#PCDATA)>
]>

<book>
<title>Xml Pocket reference</title>
<noofpages>150</noofpages>
<price>100.00</price>
<edition>second</edition>
</book>
```

External dtd

- Sample xml file with external dtd

```
<?xml version="1.0"?>
<!DOCTYPE book SYSTEM “test.dtd”>
<book>
<title>Xml Pocket reference</title>
<noofpages>150</noofpages>
<price>100.00</price>
<edition>second</edition>
</book>
```

Contents of File test.dtd

```
<!ELEMENT book (title,noofpages,price,edition)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT noofpages (#PCDATA)>
<!ELEMENT price (#PCDATA)>
<!ELEMENT edition (#PCDATA)>
```

Having internal and external dtds

- You can declare some elements in internal dtd and remaining in external dtd.
- But declaring same element both in internal and external dtds may not be acceptable by some processors while some may allow internal dtds to take precedence.

General rules about the DOCTYPE statement:

- It must appear right after the "xml" processing instruction and before the root element.

- "root_element" names the root element of the XML document.

- Key word "SYSTEM" indicates that the external DTD file specified as "DTD_location" is private.

- Key word "PUBLIC" indicates that the external DTD file specified as "DTD_location" is public.

- "DTD_location" is a URL of the external DTD file.

DTD Declarations

- Element type declarations
- Attribute-list declarations
- Entity declarations
- Notation declarations
- Processing Declarations
- Comments
- Parameter entity references

Xml building blocks

- Any xml file can contain the following types of elements:
  - An empty element :

```
<letter></letter> or  <letter/>
```

  - Element with attribute :

```
<letter Date=“2009/11/11”></letter>
```

  - Element with text :

```
<letter>This is the letter content</letter>
```

  - Element with child elements :

```
<letter>
        <sender>Rahul</sender>
</letter>
```

  - Element with child elements and Contents.

```
<chapter> introduction to xml
            <NoofPages> 20 </NoofPages>
</chapter>
```

An element in an DTD file is represented as :

<!ELEMENT ELEMENT_NAME CONTENT_MODEL>

Name : is the name of the element you are defining.

Content model : specifies what the xml element can contain. Following are the different types of content models

CONTENT_MODEL : ANY

- The first type of content model is :
  - Any : indicates that the element can have any type of content.
    - which means the element content will not be validated by the xml parser.
    - **<!ELEMENT PRODUCT ANY>**
    - Removes the syntax checking associated with the element.
    - If you put any child elements in the element of ANY content type, those child elements must be declared in the dtd.

CONTENT_MODEL : CHILD ELEMENTS

- Child elements : you can specify any element to contain other XML elements.
- Ex: The XML element BOOK is as below :

```
<BOOK>
   <CHAPTER>introduction</CHAPTER>
 </BOOK>
```

- The DTD declaration for a BOOK element is

<!ELEMENT BOOK (CHAPTER)>

- If BOOK contains more than one child elements

  Ex :
```
<BOOK>
        <CHAPTER>one</CHAPTER>
        <PRICE>20.99</PRICE>
    </BOOK>
<!ELEMENT BOOK (CHAPTER,PRICE)>
```

- The order of the chapter and price must be followed as written in the DTD definition.
- Illegal Definition

```
<BOOK>
    <PRICE>20.99</PRICE>
    <CHAPTER>one</CHAPTER>
</BOOK>
```

CONTENT_MODEL : CHILD ELEMENTS Contd.

- For defining multiple occurrences of child elements
  - (CHAPTER)+ : indicates one or more occurrences of chapter element. **<!ELEMENT BOOK (CHAPTER)+>**
  - (CHAPTER)* : indicates zero or more occurrences of chapter element. **<!ELEMENT BOOK (CHAPTER)*>**
  - (CHAPTER)? : optional i.e zero or one occurrences of chapter.

    <!ELEMENT BOOK (CHAPTER)?>
  - CHAPTER | APPENDIX : zero or more repeats of either of the two can occur.

    <!ELEMENT BOOK (CHAPTER|APPENDIX)>

CONTENT_MODEL : PCDATA

- An element can contain text, for which the content model used would be :
- #PCDATA : is the non markup text.
- The parsed character data(PCDATA) is the actual content of the xml document, the plain text.
- PCDATA will allow an element to contain only text but NOT other child elements.
- PCDATA is not able to distinguish between the types of data like numbers, characters, floats etc

Content Model : PCDATA

- <!ELEMENT BOOK(CHAPTER)>

  <!ELEMENT CHAPTER(#PCDATA)>
- For the above dtd declaration a valid xml would be

```
<BOOK>
   <CHAPTER>
    Introduction to xml dtds
   </CHAPTER>
</BOOK>
```

Some dtd and xml examples

DTD

- <!ELEMENT BOOK (CHAPTER+,APPPENDIX?) >

  <!ELEMENT CHAPTER (#PCDATA) >

  <!ELEMENT APPPENDIX(#PCDATA) >

XML

- <BOOK>

  <CHAPTER>1</CHAPTER>

  <APPENDIX> E</APPENDIX>

  </BOOK>
- **OR**
- <BOOK>

  <CHAPTER>1</CHAPTER>

  <CHAPTER>2</CHAPTER>

  <APPENDIX> E</APPENDIX>

  </BOOK>
- **OR**
- <BOOK>

  <CHAPTER>1</CHAPTER>

  <CHAPTER>2</CHAPTER>

  </BOOK>

DTD **XML**

- Subsequences using parenthesis :

  <!ELEMENT CHAPTER (NAME, (PAGENO,REFERENCE*)+) >

- <CHAPTER>
  <NAME>JIM</NAME>
  <PAGENO> 23</PAGENO>
  <REFERENCE>a </REFERENCE>
  </CHAPTER>
- OR
- <CHAPTER>
  <NAME>JIM</NAME>
  <PAGENO> 23</PAGENO>
  <REFERENCE>a </REFERENCE>
  <PAGENO> 29 </PAGENO>
  <REFERENCE>C </REFERENCE>
  </CHAPTER>
- OR
- <CHAPTER>
  <NAME>JIM</NAME>
  <PAGENO> 23</PAGENO>
  <PAGENO> 90</PAGENO>
  </CHAPTER>

DTD

- Choice :

  <!ELEMENT CHAPTER (NAME, PAGENO, (REFERENCE|FOOTNOTE))

XML

- <CHAPTER>

  <NAME>JIM</NAME>

  <PAGENO> 23</PAGENO>

  <REFERENCE>a </REFERENCE>

  </CHAPTER>

  OR

- <CHAPTER>

  <NAME>JIM</NAME>

  <PAGENO> 23</PAGENO>

  <FOOTNOTE>FT</FOOTNOTE>

  </CHAPTER>

DTD

```
<!ELEMENT person     (name,
profession*)>
<!ELEMENT name       (first_name,
last_name)>
<!ELEMENT first_name (#PCDATA)>
<!ELEMENT last_name  (#PCDATA)>
<!ELEMENT profession (#PCDATA)>
```

Invalid XML

Valid XML

```
<person>
  <name>
    <first_name>Alan</first_name>
    <last_name>Turing</last_name>
  </name>
</person>
```

Invalid XML

```
<! – invalid because it omits name -- >
<person>
  <profession>computer
scientist</profession>
  <profession>mathematician</profession>
  <profession>cryptographer</profession>
</person>
```

CONTENT_MODEL : MIXED

- An element can contain both child elements and text, in which case the content model would be mixed.
- It does not allow the user to specify the order or number of occurrences of child elements.
- <!ELEMENT chapter (#PCDATA | NoofPages)*>
- The corresponding Xml will be

```
<chapter>                                        <chapter>
<NoofPages> 20 </NoofPages>   or         <NoofPages> 20 </NoofPages>
introduction to xml                              </chapter>
        </chapter>
```

- The element can contain either or both of the mixed content for any number of times.

CONTENT_MODEL : EMPTY

- Empty elements do not have any text or child element BUT they may have attributes;
- Ex : <Book_id></Book_id>
- In the dtd,
- <!ELEMENT Book_id EMPTY>

CDATA Sections

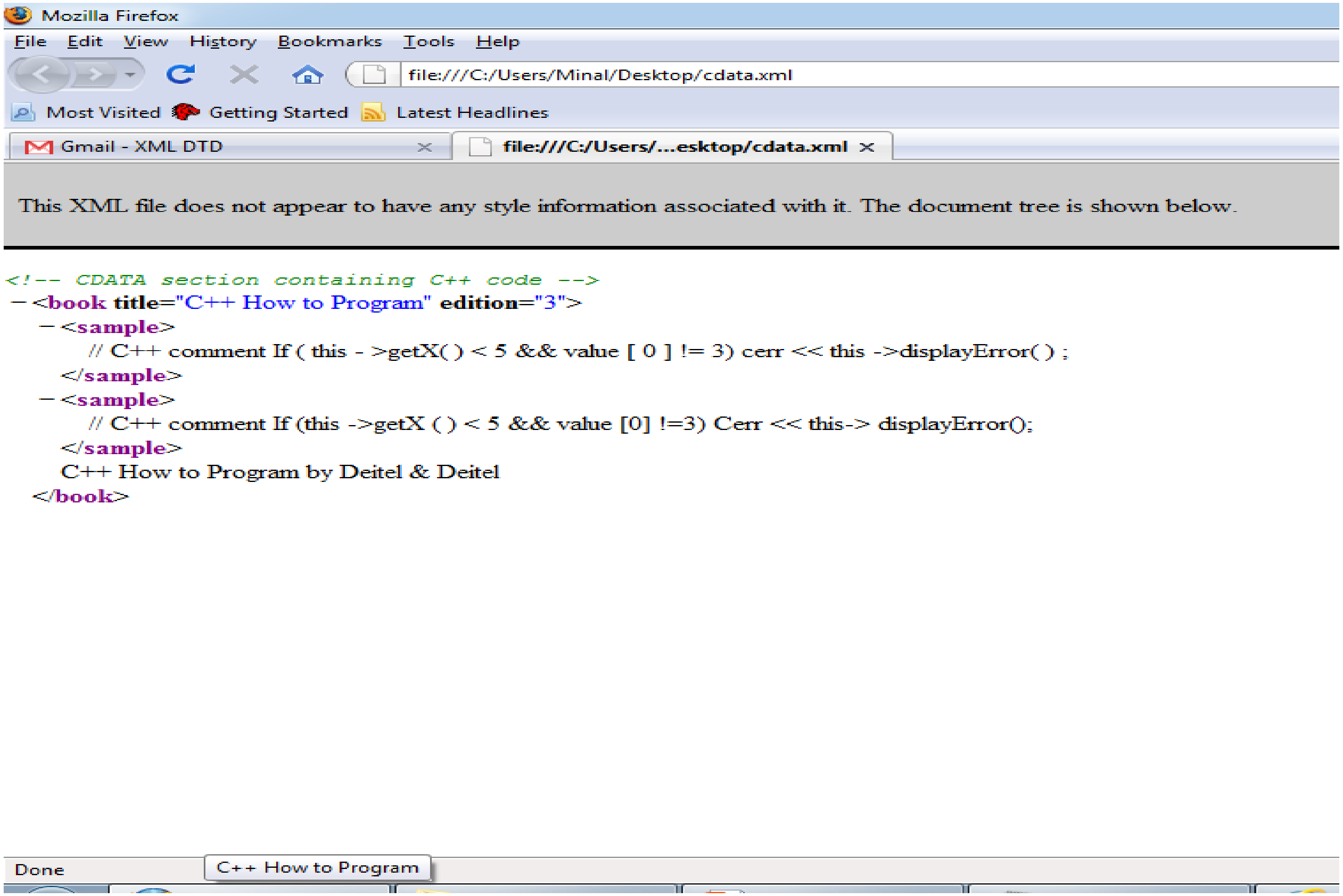
- May contain text, reserved characters and whitespace
  - Reserved characters need not be replaced by entity references
- Not processed by XML parser
- Commonly used for scripting code (e.g JavaScript)
- Begin with <![CDATA[
  Terminate with]]>

Example – CDATA Section

```
<?xml version="1.0"?>
<!– CDATA section containing C++ code -->
<book title="C++ How to Program"  edition="3">
<sample>
// C++ comment
If  ( this - &gt;getX( )  &lt; 5 &amp;&amp; value  [ 0 ] != 3)
cerr  &lt;&lt; this -&gt;displayError( ) ;
</sample>
<sample>
<![CDATA[
// C++ comment
If (this ->getX ( ) < 5 && value [0] !=3)
Cerr << this-> displayError();
]]>
</sample>
C++  How to Program by  Deitel &amp; Deitel
</book>
```

Entity references required if not in CDATA section

XML Does not process CDATA Section


Mozilla Firefox
File Edit View History Bookmarks Tools Help
file:///C:/Users/Minal/Desktop/cdata.xml
Most Visited Getting Started Latest Headlines
Gmail - XML DTD
file:///C:/Users/...esktop/cdata.xml
Done
C++ How to Program

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
<!-- CDATA section containing C++ code -->
 - <book title="C++ How to Program" edition="3">
   - <sample>
       // C++ comment If ( this - >getX( ) < 5 && value [ 0 ] != 3) cerr << this ->displayError( ) ;
     </sample>
   - <sample>
       // C++ comment If (this ->getX ( ) < 5 && value [0] !=3) Cerr << this-> displayError();
     </sample>
     C++ How to Program by Deitel & Deitel
   </book>
```

Online Validation for DTD

1) http://validator.w3.org/#validate_by_upload

2) http://www.stg.brown.edu/service/xmlvalid/

3) http://www.xmlvalidation.com/index.php

Exercise:

1) Create a DTD and XML Document For music collection which need to have entries for CD collection. At least one CD collection required.

2) Create a DTD and XML Document For
- list
  - recipe
  - author
  - recipe name
  - meal
  - ingredients
    - It Contains collection of Different item , but at least one item.
  - directions

3) Create a DTD and XML Document For a page which must have title and content and optional comment.

Answers To Exercise

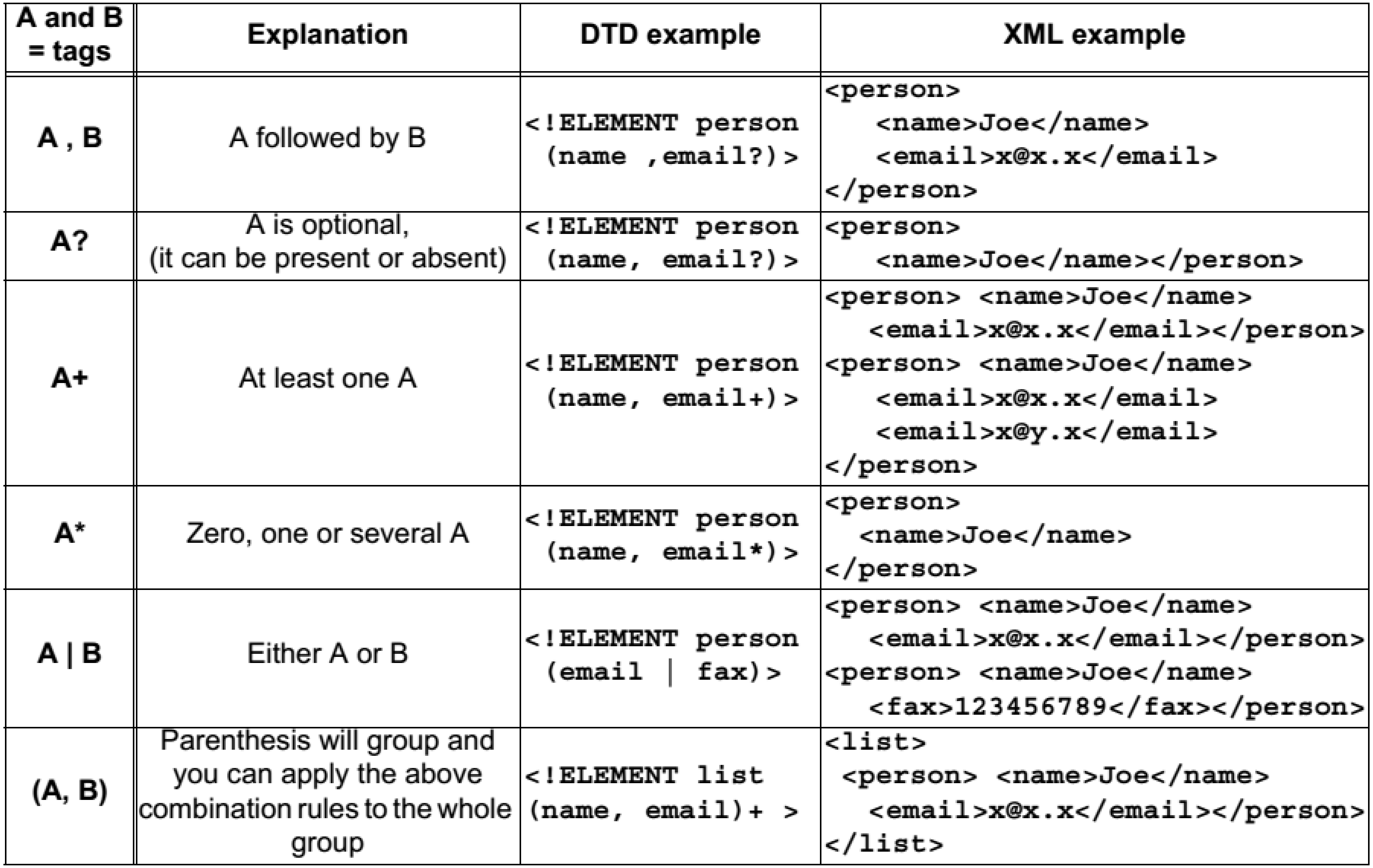
1)
```
<?xml version="1.0"?>
<!DOCTYPE cdCollection [
 <!ELEMENT cdCollection (cd+)>
  <!ELEMENT cd (title, artist, year)>
    <!ELEMENT title (#PCDATA)>
    <!ELEMENT artist (#PCDATA)>
    <!ELEMENT year (#PCDATA)>
 ]>
 <cdCollection>
  <cd>
    <title>Dark Side of the Moon</title>
    <artist>Pink Floyd</artist>
    <year>1973</year>
  </cd>
 </cdCollection>
```

2)
```
<?xml version="1.0"?>
 <!DOCTYPE list [
 <!ELEMENT list (recipe+)>
 <!ELEMENT recipe (author,recipe_name,meal,ingredients,directions)>
<!ELEMENT author (#PCDATA)>
<!ELEMENT recipe_name (#PCDATA)>
<!ELEMENT meal (#PCDATA)>
<!ELEMENT ingredients (item+)>
<!ELEMENT item (#PCDATA)>
<!ELEMENT directions (#PCDATA)>
]>
 <list>   <recipe>  <author>Carol Schmidt</author>
   <recipe_name> Chocolate Chip Bars</recipe_name>
   <meal>Dinner</meal>
   <ingredients>
   <item>2/3 C butter</item>      <item>2 C brown sugar</item>
   <item>1 tsp vanilla</item>     <item>1 3/4 C unsifted all-purpose flour</item>
     <item>1 1/2 tsp baking powder</item>
     <item>1/2 tsp salt</item>      <item>3 eggs</item>
     <item>1/2 C chopped nuts</item>
     <item>2 cups (12-oz pkg.) semi-sweet choc. chips</item>
    </ingredients>
```

```
<directions>
Preheat oven to 350 degrees.
Melt butter; combine with brown sugar and vanilla in large mixing bowl.
Set aside to cool. Combine flour, baking powder, and salt; set aside.
Add eggs to cooled sugar mixture; beat well.
Stir in reserved dry ingredients, nuts, and chips.
Spread in greased 13-by-9-inch pan.
Bake for 25 to 30 minutes until golden brown; cool.  Cut into squares.
   </directions>
 </recipe>
</list>
```

3)
```
<?xml version="1.0"?>
 <!DOCTYPE page[
 <!ELEMENT page (title,content,comment*)>
 <!ELEMENT title (#PCDATA)>
 <!ELEMENT content (#PCDATA)>
 <!ELEMENT comment (#PCDATA)>
 ]>
 <page>
 <title>Hello friend</title>
 <content>Here is some content</content>
<!-- <comment>Written by Tester</comment> -->
</page>
```

Combination rules for elements



A and B = tags	Explanation	DTD example	XML example

A , B	A followed by B	`<!ELEMENT person (name ,email?)>`	`<person>`
			`<name>Joe</name>`
			`<email>x@x.x</email>`
			`</person>`

A?	A is optional,
	(it can be present or absent)
		`<!ELEMENT person (name, email?)>`	`<person>`
			`<name>Joe</name></person>`

A+	At least one A	`<!ELEMENT person (name, email+)>`	`<person> <name>Joe</name>`
			`<email>x@x.x</email></person>`
			`<person> <name>Joe</name>`
			`<email>x@x.x</email>`
			`<email>x@y.x</email>`
			`</person>`

A*	Zero, one or several A	`<!ELEMENT person (name, email*)>`	`<person>`
			`<name>Joe</name>`
			`</person>`

A | B	Either A or B	`<!ELEMENT person (email | fax)>`	`<person> <name>Joe</name>`
			`<email>x@x.x</email></person>`
			`<person> <name>Joe</name>`
			`<fax>123456789</fax></person>`

(A, B)	Parenthesis will group and you can apply the above combination rules to the whole group	`<!ELEMENT list (name, email)+ >`	`<list>`
			`<person> <name>Joe</name>`
			`<email>x@x.x</email></person>`
			`</list>`

DTD Attributes

- Attributes provide supplementary information about a particular element and appear in name/value pairs.
- <book id="123"></book>
- In dtd, you define an attribute as below :
  - <!ATTLIST ELEMENT_NAME

```
        ATTRIBUTE_NAME   TYPE   DEFAULT_VALUE
        ATTRIBUTE_NAME   TYPE   DEFAULT_VALUE
        >
```

- <!ATTLIST book id CDATA "123">
- where ELEMENT_NAME and ATTRIBUTE_NAME is the name of the element and its attributes resp.

Attribute Default values

- You can specify the actual default value in DTD:
  - <!ATTLIST PAGE AUTHOR CDATA “AUTHOR_ONE”>
- The corresponding xml element would be
  - <PAGE AUTHOR=“AUTHOR_ONE” />

  Or

  - <PAGE AUTHOR=“Dan Brown” />

#IMPLIED

- The default placeholder can also be substituted by one of the following :
  - #IMPLIED : indicates
    - This is not a mandatory attribute of the xml element and that the you do not want to give any default value
  - <!ATTLIST Page author CDATA #IMPLIED>
  - Valid xml for above attribute declaration :
  - <Page />

Or

  - <Page author="Dan Brown" />

#REQUIRED

- #REQUIRED : indicates
  - That you are not specifying the actual default value but it is mandatory for the author of xml file to supply a value
- <!ATTLIST Page author CDATA #REQUIRED>
- Valid xml for above attribute declaration :
  - <Page author="Dan Brown" />

    Or
  - <Page author="J.K.Rowling" />
- Used for mandatory attributes like author of document, date of creation of xml etc.

#FIXED

- #FIXED : used when you want to fix the value of an attribute.
- It is like creating constants.
- <!ATTLIST Page author CDATA #FIXED “Dan Brown”>
- Valid xml for above attribute declaration :
  - <Page author=“Dan Brown” />

    Or
  - <Page />
- Even if the attribute is not supplied in the xml, the fixed value is passed to any application that reads this xml.
- Also, <Page author=“ABC” /> is an invalid declaration.

EXAMPLE OF Attributes

```
<?xml version="1.0"?>
<!DOCTYPE videolibrary[
<!ELEMENT videolibrary (film,class,(hero|director|heroine)+)>
<!ELEMENT film (#PCDATA)>
<!ATTLIST film color CDATA #IMPLIED language CDATA  #FIXED
"Hindi" year CDATA #REQUIRED>
<!ELEMENT class (#PCDATA)>
<!ELEMENT hero (#PCDATA)>
<!ELEMENT director (#PCDATA)>
<!ELEMENT heroine (#PCDATA)>
]>
<videolibrary>
<film year="1994">HUM APKE HAIN KAUN</film>
<class>Love Story</class>
<heroine>Madhuri Dixit</heroine>
</videolibrary>
```

Attribute Types

- Type specifies the type of the attribute.
- Some important attribute types are :
  - CDATA : plain text/character data not including any markup.
    - <!ATTLIST book id CDATA “123”>
  - Enumerated : a list of possible values out of which any one would be the attribute value.
    - <!ATTLIST emp employment_status (contract | regular) “regular”>

Attribute Types

- NMTOKEN : is a type which allows any valid xml name to be the value of the attribute.
  - A name token is same as an xml name except that a token can begin with number, hyphen and period.
  - It is similar to CDATA but with the restriction of the naming rules important being prohibition of white space.
  - The character # is not permitted in attributes of type NMTOKEN and NMTOKENS

- NMTOKENS : the value of an attribute can consist of multiple name tokens. NMTOKENS can contain the same characters as NMTOKEN plus whitespaces. White space consists of one or more space characters, carriage returns, line feeds, or tabs.

Example of NMTOKEN and NMTOKENS

```
<!ELEMENT attributes (#PCDATA)>
<!ATTLIST attributes
      aaa CDATA  #IMPLIED
      bbb NMTOKEN  #REQUIRED
      ccc NMTOKENS  #REQUIRED>


<attributes aaa="d1" bbb="a1:12" ccc=" 3.4 div    -4"/>
```

Attribute Types

- ID : the type ID is used to uniquely identify each element in the document.
  - An attribute value of type ID must be a valid xml name. i.e. The attribute value of ID type can contain only characters permitted for NMTOKEN and must start with a letter.
  - A name may not be used as an id attribute of more than one tag/element.
  - Each element can have only one attribute of ID type.
    - <!ATTLIST Book b_id ID #REQUIRED>
    - <Book b_id="b1"/>
    - <!ATTLIST NoofPages page_id ID #REQUIRED>
    - <NoofPages page_id="b1">35346</NoofPages>
  - The ID type cannot be used with #FIXED.

Attribute Types

IDREF :

IDREF is also an identifier type and it should also point to one element only. we can use IDREF attributes to refer to an element from other elements.

IDREFS :

This attribute takes multiple element Id's as an it's value, where individual IDREF values are separated by white space. It is used to point to a list of related elements in an XML document.

Example :

```
<!ATTLIST topic topicid ID #REQUIRED>
<!ATTLIST topic prev IDREF  #IMPLIED>
<!ATTLIST topic next IDREF  #IMPLIED>
<!ATTLIST topic Xref IDREFS  #IMPLIED>

<topic topicid="topic4" prev="topic3" next="topic8" Xref="topic1 topic2">
<!– topics 4-7 are missing -->
The Topic is XML
</topic>
```

-Entity /Entities : also can be an attribute type.

Attribute Types

- NOTATION : A notation is used to specify the format of non-XML data / non-textual data .
- Format for a notation is:
  <!NOTATION *name* system "*external_ID*“>
- The name identifies the format used in the document, and the external_id identifies the notation - usually with MIME-types.
- For example, to include a GIF image in your XML document:
  <!NOTATION GIF system "image/gif">
- The attribute of NOTATION type allows you to use a value that has been declared as a *notation* in the DTD.
- An element type MUST NOT have more than one **NOTATION** attribute specified
- For compatibility with SGML, an attribute of type **NOTATION** MUST NOT be declared on an element declared **EMPTY**.

Overview of Attributes

Type	Attribute types
CDATA	"Character Data" - Text data
NMTOKEN	A single word (no spaces or punctuations)
ID	Unique identifier of the element.
IDREF	Reference to an identifier.
IDREFS	Reference to one or more identifiers
(A|B|C|..)	List of values (from which the user must choose)

	Type Definition
#IMPLIED	Attribute is optional
#REQUIRED	Attribute is mandatory)
#FIXED Value	Attribute has a fixed value (user can't change it)

Examples

DTD rule	example XML
<!ATTLIST person first_name CDATA #REQUIRED>	<person first_name="Joe">
<!ATTLIST person gender (male|female) #IMPLIED>	<person gender="male">
<!ATTLIST form method CDATA #FIXED "POST">	<form method="POST">
<!ATTLIST list type (bullets|ordered) "ordered">	<list type="bullets">
<!ATTLIST sibling type (brother|sister) #REQUIRED>	<sibling type="brother">
<!ATTLIST person id ID #REQUIRED>	<person id="N1004">

Attributes Vs Elements

• There are some design rules that may help you decide whether using an element or an attribute
• In case of doubt, always use elements ...

Rather use child elements in side an element (information block):
• if order is important (attributes can’t be ordered)
• if you plan to use the same kind of information block with different parents
• if a future version of DTD may specify sub-components of an information block
• if the information block represents a "thing" (an object in OO programming)
• if the DTD is text-centric, because an author must see contents she/he edits and attributes are often hidden away in XML editors; only use attributes to qualify properties like style !

Rather use attributes for an element
• if an attribute refers to an other element
• <pet_of owner_name="lisa" pet_type="cat") would refer to <animal category="cat">
• to declare usage/type/etc. of an element:
<address usage="prof"> ... </address>
• if you wish to list all possible values a user can enter
• if you want to restrict datatype of the attribute value (e.g. require a single word)

Entities

- In real world xml applications, the content of an xml file comes from more than one source.
- Each such source of content or data storage unit that makes up a part a of the xml document is an entity.
- Entities can be considered as abbreviations for some other content.
- A file , database table, record etc may be an entity.
- Every xml document has at least one entity which is the document entity itself which may have references to other entities in turn.
- Entities can be classified as
  - Internal : defined and used completely within the dtd.
  - External : the actual content of entity is residing outside the document.

Types of Entities

- ➢ Internal (to a doc) vs. External (use URI)

- ❑ Syntax of an internal entity definition:
  `<!ENTITY entity_name "content">`
- ❑ Syntax of an external entity definition:
  `<!ENTITY entity_name SYSTEM URI>`
- ❑ Syntax of using an entity:
  `&entity_name;`

- ➢ General (in XML doc) vs. Parameter (in DTD)

- ❑ Syntax for Parameter Entities:

  `<!ENTITY %entity_name "content">`

  `<!ENTITY %entity_name SYSTEM "URI">`

- ➢ Parsed (XML) vs. Unparsed (non-XML)

Parsed and unparsed entities

- Other way of classifying entities could be
  - Parsed entities are entities containing XML data that is processed by an XML application. There are two fundamental types of parsed entities:
    - General entities
    - Parameter entities
  - Unparsed entities : are entities containing non XML data.

Internal General Entity

- Used for commonly required text chunks or data
- For defining an entity in the dtd ,
  - <!ENTITY name “the actual content”>
  - <!ENTITY des “This is the index page”>
- For using it in the xml document,
  - <page>&des;</page>
- The entity value may contain one or more elements also.
- But it cannot contain a part of an element, like only starting or ending tag
- The advantage is you can centralize commonly needed data like headers, footers etc when sharing a single dtd between many xml files.

Example of Entity

```
<?xml version="1.0"?>
<!DOCTYPE document[
<!ENTITY ELTP "ENTRY LEVEL TRAINING PROGRAM">
<!ELEMENT document (title,course)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT course (course_code,date)>
<!ELEMENT course_code (#PCDATA)>
<!ELEMENT date (#PCDATA)>
]>
<document>
<title>&ELTP;</title>
<course>
<course_code>MSC_CA_12</course_code>
<date>Nov 27,2012</date>
</course>
</document>
```

External General entities

- The actual content of the entity is outside the main file.
- It allows you to construct one single xml file from multiple independent files.
- In DTD, <!ENTITY name SYSTEM “URI”>
- The external entity could be an xml or even an simple text file.
- The mozilla in built parser is not able to resolve external entities.
- The document must be well-formed even after the entity is expanded.

Example:

Contents of ext.xml

```
<?xml version="1.0"?>
<!DOCTYPE document SYSTEM "ext-entity.dtd">
<document>
<title>&ELTP;</title>
<course>
<course_code>MSC_CA_12</course_code>
<date>Nov 27,2012</date>
</course>
</document>
```

Contents of ext-entity.dtd

```
<?xml version="1.0"?>
<!ENTITY ELTP "ENTRY LEVEL TRAINING PROGRAM">
<!ELEMENT document (title,course)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT course (course_code,date)>
<!ELEMENT course_code (#PCDATA)>
<!ELEMENT date (#PCDATA)>
```

Predefined entities

- Following characters cannot be used as text in an xml file as they can be mistaken for markup.
- Hence, the predefined entities

• &amp;	&
• &lt;	<
• &gt;	>
• &quot;	“
• &apos;	‘

Parameter entities

- General entities are defined in the dtd but they appear/are used in the xml document itself.
- But parameter entities are meant for use within the dtd only.
- Parameter entity references begin with the “%” sign instead of &.
- <!ENTITY % name “actual content ”>
- EX :
  - Definition : <!ENTITY % data “(#PCDATA)”>
  - Usage : <!ELEMENT item %data;>

External parameter entities

- Used for larger and complex dtds, they allow us to build a complex dtd by combining smaller, individual dtds.
- <!ENTITY % name SYSTEM “URI” >
- EX :

  -
    ```
    <!ELEMENT  class(courseName,strength,subjects)>
    <!ELEMENT  courseName  (#PCDATA) >
    <!ELEMENT  strength  (#PCDATA) >
    <!ENTITY  %  sub  SYSTEM “second.dtd” >
    % sub;
    ```

Example of Internal Parameterised Entity

```
<?xml version = "1.0" encoding="UTF-8" standalone="yes"?>
<!DOCTYPE document [
<!ENTITY % project "<!ELEMENT project (product, id, price)>">
<!ELEMENT document (employee)*>
<!ELEMENT employee (name, hiredate, projects)>
<!ELEMENT name (lastname, firstname)>
<!ELEMENT lastname (#PCDATA)>
<!ELEMENT firstname (#PCDATA)>
<!ELEMENT hiredate (#PCDATA)>
<!ELEMENT projects (project)*>
%project; <!--parameter entity reference %project; in the DTD,
it will be replaced with the text "<!ELEMENT project (product, id, price)>".-
->
<!ELEMENT product (#PCDATA)>
<!ELEMENT id (#PCDATA)>
<!ELEMENT price (#PCDATA)>
<!ATTLIST employee supervisor CDATA #IMPLIED>
]>
```

Example of Internal Parameterised Entity (Continued)

```
<document>
    <employee supervisor="no">
        <name>
            <lastname>Kelly</lastname>
            <firstname>Grace</firstname>
        </name>
        <hiredate>October 15, 2005</hiredate>
        <projects>
            <project>
                <product>Printer</product>
                <id>111</id>
                <price>$111.00</price>
            </project>
            <project>
                <product>Laptop</product>
                <id>222</id>
                <price>$989.00</price>
            </project>
        </projects>
    </employee>
</document>
```

Example of External Parameterised Entity

```
<?xml version = "1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE document SYSTEM "expara.dtd">
<document>
    <employee supervisor="no">
        <name>
            <lastname>Kelly</lastname>
            <firstname>Grace</firstname>
        </name>
        <hiredate>October 15, 2005</hiredate>
        <projects>
            <project>
                <product>Printer</product>
                <id>111</id>
                <price>$111.00</price>
            </project>
            <project>
                <product>Laptop</product>
                <id>222</id>
                <price>$989.00</price>
            </project>
        </projects>
```

Example of External Parameterised Entity(Continued)

```
       <supervisorComment>
              <date>23-12-11</date>
              <text>Good Work</text>
              </supervisorComment>
              <customerComment>
              <date>25-12-11</date>
              <text>Satisfied and resolved</text>
              </customerComment>
  </employee>
</document>
```

Contents of expara.dtd file

```
<!ENTITY % record "(date, text)">
<!ELEMENT document (employee)*>
<!ELEMENT employee (name, hiredate,
projects,supervisorComment*,customerComment*,employeeComment*)>
<!ELEMENT name (lastname, firstname)>
<!ELEMENT lastname (#PCDATA)>
<!ELEMENT firstname (#PCDATA)>
<!ELEMENT hiredate (#PCDATA)>
<!ELEMENT projects (project)*>
<!ELEMENT supervisorComment %record;>
<!ELEMENT customerComment %record;>
<!ELEMENT employeeComment %record;>
<!ELEMENT project (product, id, price)>
<!ELEMENT product (#PCDATA)>
<!ELEMENT id (#PCDATA)>
<!ELEMENT price (#PCDATA)>
<!ELEMENT date (#PCDATA)>
<!ELEMENT text (#PCDATA)>
<!ATTLIST employee supervisor CDATA #IMPLIED>
```

Example of General & Parameterised Entity

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE document [
  <!ELEMENT document (quote)+>
  <!ELEMENT quote (#PCDATA)>
  <!ENTITY r 'sachin'>
                                              General Entity
  <!ENTITY s 'tendulkar‘>
  <!ENTITY % y '<!ENTITY x "&r;  &s;">'>   Parameterised Entity
  %y;
]>

<document>
  <quote>
    " My name is &x;, but you can call me Master Blaster."
  </quote>
</document>
```

Unparsed entities and Notations

- An XML entity may contain non-textual information such as pictures, video, or sound in digitized form.
- When such entities are declared, it is essential to indicate that they contain data which an XML parser or processor cannot handle in the same way as the surrounding data — it is no use trying to process entities contain pictures or sound as if they contain text.
- Unparsed entities aren't processed by XML applications and are capable of storing text or binary data.
- Because it isn't possible to embed the content of binary entities directly in a document as text, binary entities are always referenced from an external location, such as a file.
- *Unlike parsed entities, which can be referenced from anywhere in the content of a document, unparsed entities must be referenced using an attribute of type ENTITY or ENTITIES*

Unparsed entities and Notations

- Unparsed external entities are identified using the NDATA keyword in their entity declaration; NDATA simply indicates that the entity's content is not XML data.
- This is accomplished by including an additional keyword in the declaration of such entities, as in the following example:
- <!ENTITY fig1 SYSTEM "figure1.png" NDATA png>
- The keyword *NDATA* indicates that this external entity is *unparsed*: it contains non-XML data which an XML parser should ignore. It is followed by an additional name (png in the example above) which identifies the *notation* used for this data
- NDATA signifies Notation data.
- png in the above example is defined as

Unparsed entities and Notations

- <!NOTATION png SYSTEM “image/png”>
- Notations are used to specify helper information for an unparsed entity and are required of all unparsed entities. Following is an example of a notation that describes the JPEG image type:
- <!NOTATION JPEG SYSTEM "image/jpeg">
  the name of the notation is JPEG, and the helper information is image/jpeg, which is a universal type that identifies the JPEG image format.
- It is expected that an XML application could somehow use this helper information to query the system for the JPEG type in order to figure out how to view JPEG images. So, this information would come into play when an XML application encounters the following image entity:
- <!ENTITY myimage SYSTEM “sample.jpg" NDATA JPEG>
  If you didn't want the XML application to figure out how to view the image on its own, you can get more specific with notations and specify an application, as follows:
- <!NOTATION JPEG SYSTEM "Picasa2.exe">

Unparsed entities

- To embed an unparsed entity in an xml :
  - Declare an element which has an attribute of entity type. The value of this attribute will provide the name of the unparsed entity.

```
<!DOCTYPE  image [
<!NOTATION  JPEG  SYSTEM  "Picasa2.exe ">
<!ENTITY  myimage  SYSTEM  "Sample.jpg“  NDATA JPEG>

<!ELEMENT  image   (#PCDATA)>
<!ATTLIST  image  source   ENTITY   #REQUIRED>
]>
```

Corresponding XML :

- `<image  source=“myimage“></image>`

LIMITATIONS OF DTD’s

Limitation	Explanation
Non-XML Syntax	DTD syntax is quite different from basic XML Syntax.
One DTD Per XML	We cannot use multiple DTDs to validate one XML document. We can include only one DTD reference inside an XML
Weak Data Typing	DTD defines only basic data types.
No inheritance	DTDS are not object-oriented i.e they do not allow the designer to create data types and extend them as desired.
Overriding a DTD	An internal DTD can Override an external DTD.
No Dom Support	DOM is used to parse, read and edit XML Document. It can not be used for DTD’s