Document Type Definition

Agenda

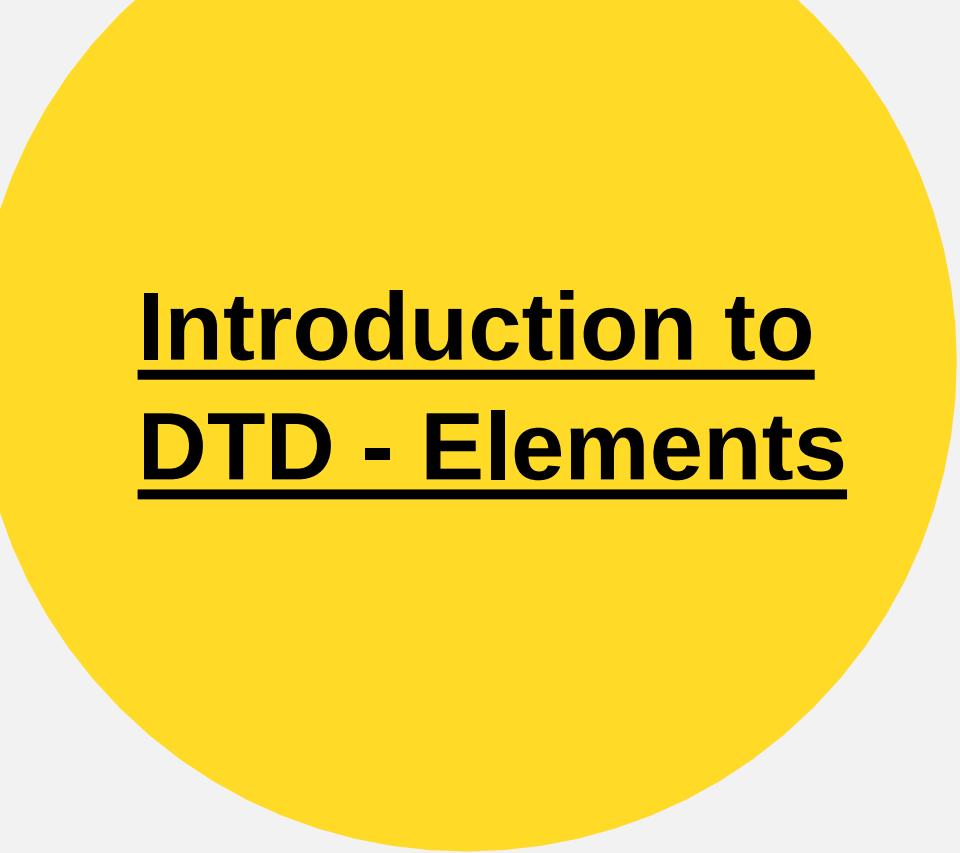
1. Introduction to DTD – Elements
2. Attribute Declarations
3. Entities and Notation

Objectives

At the end of this module, you will be able to

- Create DTDs
- Use internal and external DTDs
- Declare elements
- Declare attributes
- Declare entities
- Use predefined entities
- Use external entities containing text
- Use external entities containing non-xml data
- Declare notations in DTDs

Introduction to DTD - Elements



Valid XML Documents

- An XML document is considered well-formed if it follows XML Syntax rules
- An XML document is valid if it
  - Is well-formed
  - Declares a DTD (Document Type Definition)
  - Conforms to the constraints specified in that DTD
  - Document Type Definitions and XML Schemas provide descriptions of document structures.

- The purpose of a DTD is to define legal building blocks of an XML document
- Few alternatives to DTDs are XML Schemas and RELAX NG

Document Type Definition (DTD)

- A means to validate XML documents
- A DTD can be included in XML, but usually created as a separate document
- Syntax for DTDs is built into the XML 1.0 specification
  - Defines what tags are legal
  - Where they can occur in XML document

- Provides grammar for one or more XML documents
  - Contains necessary rules that XML code must follow
  - Describes a model of the structure of the content of an XML document

Syntax of a DTD declaration

- A DTD is declared in a document by using a document type declaration

- A **<!DOCTYPE>** element is used to create a DTD and the DTD appears in that element

- The element can take different forms

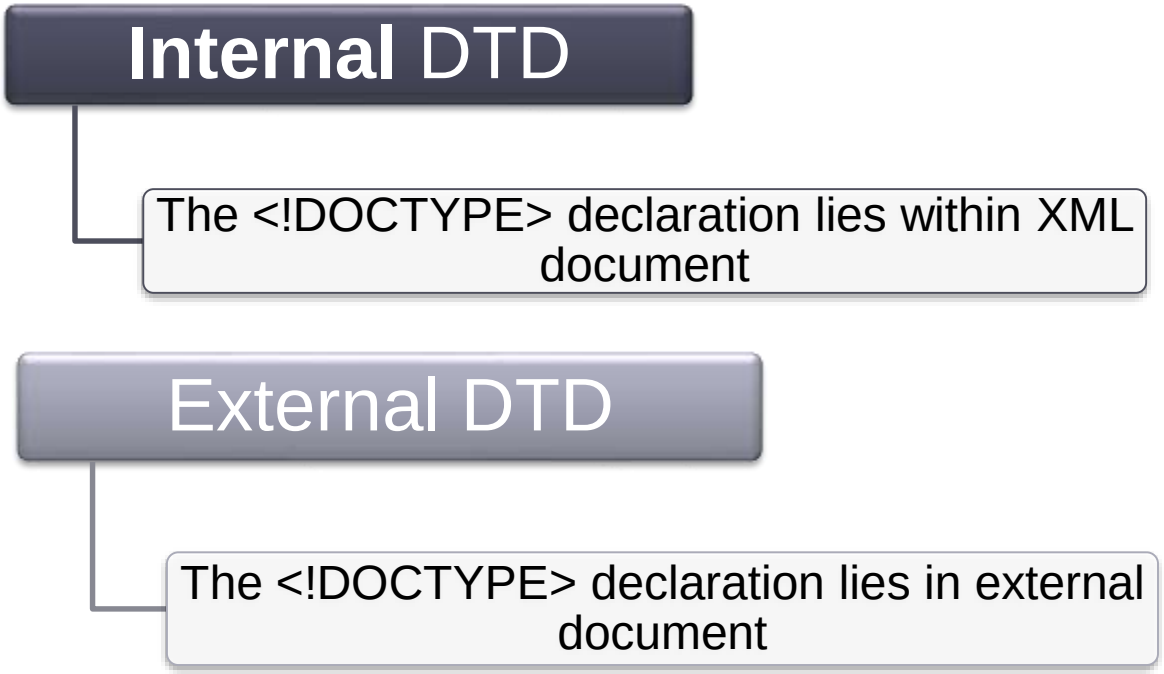
```
<!DOCTYPE rootname [ DTD ]>
<!DOCTYPE rootname SYSTEM URI >
<!DOCTYPE rootname PUBLIC identifier URI >
```

rootname is the name of the root element

URI is the URI of a *DTD* outside the current *XML* document

Types of DTDs

A DTD can be declared inline in your XML document, or as an external reference



External DTD

- External DTDs are also known as external subset. The DTD portion of the document need not always have to be stored inside the related XML document. Instead, it can be saved in a file for reference by one document or by several different documents.
- **Syntax:**

```
< !DOCTYPE  Root element   System "filename">
```

If the DTD is an external DTD, then we have to include in the xml document the following statement which uses SYSTEM or PUBLIC identifiers.

- **Ex : <!DOCTYPE BOOK SYSTEM “output.dtd”>**

External DTD : Example

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT library ANY>
<!ELEMENT book (#PCDATA)>
<!ELEMENT magazine (#PCDATA)>
```

An XML Source file **library.xml**

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

<magazine>Quest</magazin
e>
   <book>Java</book>
</library>
```

A external DTD file **library.dtd**

Note: Keywords like ANY, #PCDATA will be discussed shortly.

Internal DTD

- Internal DTDs are also known as internal subset.
- If the DTD is included in your XML source file, it should be wrapped in a DOCTYPE definition, with the following syntax :

- **Syntax:**

```
< !DOCTYPE  Rootname    [element declaration] >
```

Since the declarations are defined in the internal DTD subset, the XML processor need not have to read and process external documents.

Internal DTD : Example

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE EMAIL [
  <!ELEMENT EMAIL (TO+, FROM, CC*,SUBJECT?, BODY?)>
  <!ELEMENT TO (#PCDATA) >
  <!ELEMENT FROM (#PCDATA) >
  <!ELEMENT CC (#PCDATA) >
  <!ELEMENT SUBJECT (#PCDATA)>
  <!ELEMENT BODY (#PCDATA)>
]>
<EMAIL>
  <TO>anny@gmail.com</TO>
  <TO>john@hotmail.com</TO>
  <FROM>bill@gmail.com</FROM>
  <BODY>Hello, Learning XML!</BODY>
</EMAIL>
```

An XML Source file
email.xml

Declaring Elements in DTDs

- Elements that appear in XML document are declared by **<!ELEMENT>** declaration in DTD
- Syntax of an element in a *DTD:*

- A **content model** indicates what content the element is allowed to have
  - Use ANY keyword to allow any content
  - Use EMPTY keyword to allow empty element
  - You can allow child elements or text data

ELEMENT descriptions

- **Suffixes:**

Suffix	Meaning	Example
?	Zero or Once (Optional)	Subject?
+	One or More	TO+
*	Zero or More	FROM*

- The absence of suffix used for an operator means that the element or content must appear exactly once.
- For example: Consider

<!ELEMENT email (TO+, FROM, CC*, Subject?, Body?)>

- The + sign in this example associated to “TO” element declares that the child element “TO" can occur 1 or more times inside the “email" element.
- The child element “FROM" must occur once, and only once inside the “EMAIL" element.
- The * sign declares that the child element “CC" can occur 0 or more times.
- The ? sign declares that the child element “SUBJECT and BODY" can occur 0 or one time.

ELEMENT descriptions

- **Separators**

Separator	Meaning	Example
,	Both, in order(sequence)	TO+, FROM
|	Or, but not both (Choice)	A|B

- **Grouping**

Group	Meaning	Example
()	Grouping	(A|B)+

- In the previous example:
- **<!ELEMENT email (TO+, FROM, CC*, Subject?, Body?)>**
- The , (comma) specifies that TO, FROM, CC, Subject, Body elements should occur in the same order in the XML document.
- Grouping of the elements are done by ()

Handling text data - #PCDATA

- Parsed character data (#PCDATA) is a text examined by parser for entities and markup
  - Should not contain any characters such as &, <, or >
  - These need to be represented by & < and > entities, respectively
- Character data is the text found between start and end tag of an XML element
- Example of PCDATA

```
<A>The character data within this element is PCDATA</A>
```

- Given the element

```
<Bookstore>Barnes &amp; Noble
Bestsellers</Bookstore>
```

The parsed character data is Barnes & Noble Bestsellers

Handling any content - ANY

- Specifying content model of an element with ANY allows any content
  - Means any elements and/or any character data
- **Example**

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE employee [
<!ELEMENT employee ANY>
<!ELEMENT benefit (#PCDATA)>
]>
<employee>James
    <benefit>Medical</benefit>
    <benefit>Loan</benefit>
</employee>
```

- If the user is not sure about the content he can specify the ANY content in the DTD file; whereas the sub-elements occurring in the enclosing elements also need to be declared explicitly.
- Since the whole point of using a DTD is to define the structure of a document, ANY should be avoided wherever possible

Empty Elements - EMPTY

- Specifying content model of an element with EMPTY
  - Elements do not contain character data
  - Elements do not contain child elements

```
<?xml version="1.0"?>
<!DOCTYPE employee [
<!ELEMENT employee ANY>
<!ELEMENT fulltime EMPTY>
]>
<employee>James
    <fulltime></fulltime>
</employee>
```

← Is semantically same as <fulltime/>

- It might contain any number of attributes, e.g.: <Temperature scale="C"/>

Element's content model – Child elements

There are a number of options for declaring an element that can contain child elements

- Containing a single child element

  <!ELEMENT company (employee)>

- Containing a list of child elements

  <!ELEMENT company (employee, contractor, partner)>

- Allowing zero or more occurrences of <employee> elements <!ELEMENT company (employee)*>

Elements with Mixed Content Model (Contd.).

- Combination of elements and PCDATA

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE company [
<!ELEMENT company (#PCDATA | employee)*>
<!ELEMENT employee (#PCDATA | benefit)*>
<!ELEMENT benefit (#PCDATA)>
]>
<company>ABC Inc.
    <employee>Jane
        <benefit>Medical</benefit>
    </employee>
    <employee>Anny</employee>
</company>
```

This is mixed content

Elements with Mixed Content Model(Contd.).

- Elements in the DTD can contain a mix of PCDATA or other elements. #PCDATA describes elements with only character data.

- Use #PCDATA, “|” and “*” signs in combination for a mixed content model.

- For example, #PCDATA that can be used in an “or” grouping: <!ELEMENT company (#PCDATA|employee)*> is called mixed content.
- However, certain restrictions apply:
  - #PCDATA must be first
  - The separators must be “|”
  - The group must be starred (meaning zero or more)

Attribute Declarations

What is an Attribute?

- The property of an element
- Expressed as name - value pairs
- Used in a start tag or an empty tag to provide additional information for an element

- **Example:** XML Code

```
<employee supervisor="yes">
<name>Harry</name>
</employee>
```

Attribute Declaration

- Specifies attribute list of an element
- Use ATTLIST attribute list declaration
- The format of an attribute (where name-type-requirement may be repeated as desired) is:

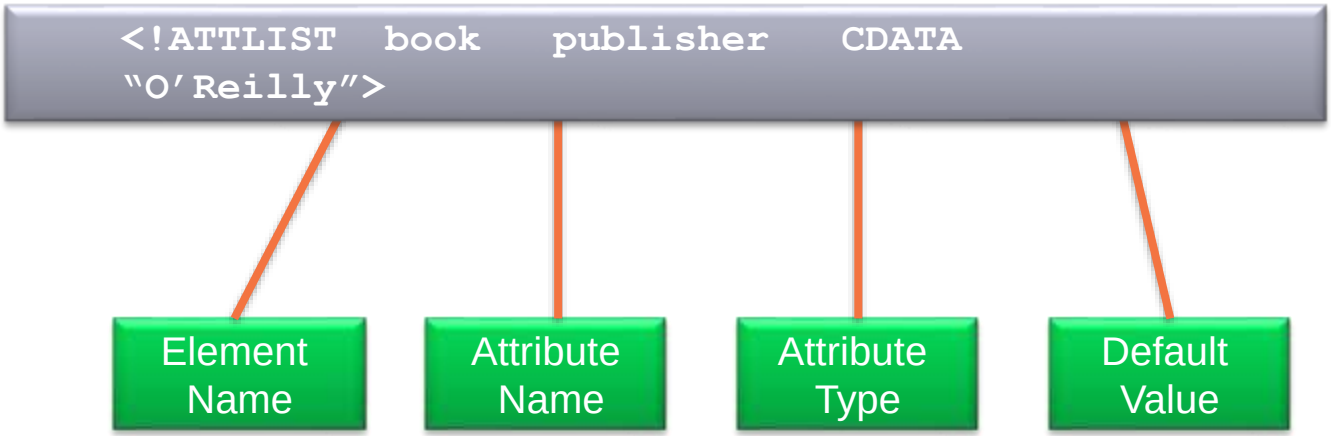
<!ATTLIST *element-name*
name type requirement
name type requirement>

- In XML, attributes may occur in any order

Attribute Declaration - Example

There are 4 parts in the syntax

- The first ATTLIST is the syntax
- Second part is the Element Name for which the attribute has to be declared
- Third the Attribute Name, fourth the Datatype of the attribute since the attribute stores values as name - value pairs
- The last is the Default value of the attribute.



Attribute Types

The attribute type in attribute declaration can be one of the following:

Type	Description of attribute value
CDATA	Character data
(en1 | en2)	One from an enumerated list
ID	A unique id
IDREF	The id of another element
IDREFS	A list of other ids
NMTOKEN	A valid XML name
NMTOKENS	A list of valid XML names separated by whitespace
ENTITY	An entity
ENTITIES	A list of entities
NOTATION	A name of a notation

Types of default values

The default value (*requirement)* in attribute declaration can be one of the following:

Value	Description
Explicit value	The default value of the attribute
#REQUIRED	The attribute must be present
#IMPLIED	The attribute is optional
#FIXED value	The attribute value is fixed

Using explicit default value

- If the attribute is not used in XML document then it automatically takes default value

- In DTD

```
<!ATTLIST BOOK publisher CDATA “O’Reilly”>
```

- In XML

```
<BOOK>XML in a Nutshell</BOOK>
```

Using #IMPLIED

- Indicates existence of an attribute for an element and its usage is optional

- In DTD

```
<!ATTLIST CONTACT phone CDATA #IMPLIED >
```

- In XML

```
<CONTACT phone="080-4421380" />
<CONTACT />
```

Using #REQUIRED

- Indicates existence of an attribute for an element and its usage is mandatory

- In DTD

```
<!ATTLIST PERSON pid CDATA #REQUIRED>
```

- In XML

```
<PERSON pid="172386" />
```

Using #FIXED

- It appears in the DTD with an attribute having a fixed value

- In DTD

```
<!ATTLIST COMPANY name CDATA #FIXED “ABC Inc.”>
```

- In XML

```
<COMPANY name=“ABC Inc.” />
```

Attribute type - CDATA

- Specifies character data which is just text without markup

- Characters like "<" and "&" are illegal in CDATA attribute values since they are markup
  - "<" generates an error as parser interprets it as start of a new element
  - "&" generates an error as parser interprets it as start of a character entity

- In DTD: `<!ATTLIST BOOK title CDATA #IMPLIED>`

- In XML: `<BOOK title="Web Designing" />`

Attribute types – NMTOKEN / NMTOKENS

- NMTOKEN is used for strings that can contain characters, digits, underscores, colons, periods and dashes
- Use NMTOKENS if string has white spaces as attribute value
- In DTD

```
<!ATTLIST test
            a CDATA #IMPLIED
            b NMTOKEN #REQUIRED
            c NMTOKENS #REQUIRED>
```

- In XML

```
<test a=">;d&lt;1>" b="a1:12" c="3.4 div -4"/>
```

Enumerated Attribute Type

- A list of possible values that an attribute can take
- Each possible value must be a valid XML name
- Example: The supervisor attribute has two possible values—"yes" and "no” and a default value of "no”

```
<!ATTLIST employee supervisor (yes | no) "no">
```

- Example: The gender attribute can have either value M or F and F is default value

```
<!ATTLIST person gender ( M | F ) “F”>
```

Attribute type – ID

- IDs are unique identifiers for elements
- ID attributes can have default values of #REQUIRED or #IMPLIED
- In DTD

```
<!ELEMENT customer (name)>
<!ELEMENT name (#PCDATA)>
<!ATTLIST customer cid ID #REQUIRED>
```

- In XML

```
<customer cid="A114">
   <name>Jason</name>
</customer>
```

Attribute type – IDREF

Use IDREF to tie an element to another element, using the other element's ID value as a reference

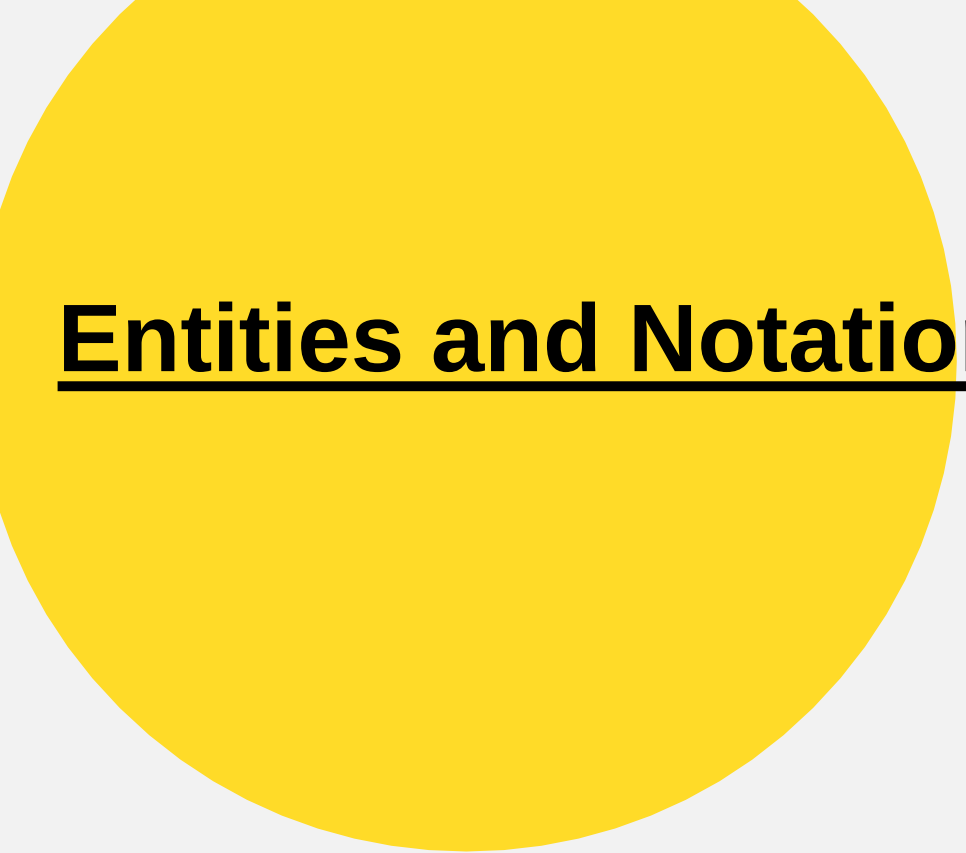
DTD

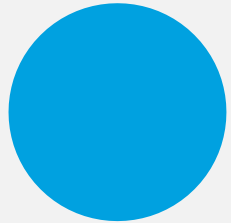
```
<!ATTLIST employee
   id ID #REQUIRED
   supervisor IDREF #IMPLIED>
                ]>
```

XML

```
<employee id="E234" supervisor="E567">
            <name>Harry</name>
            </employee>
            <employee id="E567“>
                        <name>John</name>
            </employee>
```

Entities and Notation





What is an Entity ?

- An entity in XML is a data item
- Symbolic representation of information
- Used as a container for content
- The content can reside inside or outside the XML document
- Deal with both text data and binary data
- Most entities must be declared in DTD (except predefined entities)

Types of Entities

Entities can be:

Internal Entities	Definitions found within the document's DTD
External Entities	Definitions found outside the document's DTD
Parsed Entities	Contents parsed by XML parser
Unparsed Entities	Contents not parsed by XML parser. E.g.: Sound files, Image files etc.

Types of Entities(Contd.).

- Entities may be either parsed or unparsed. A parsed entity's contents are referred to as its replacement text.

- An unparsed entity is a resource whose contents may or may not be text and if text, may be other than XML. Each unparsed entity has an associated notation, identified by name. Beyond a requirement that an XML processor makes the identifiers for the entity and notation available to the application, XML places no constraints on the contents of unparsed entities.

- Parsed entities are invoked by name using entity references. Unparsed entities are invoked by name given in the value of ENTITY or ENTITIES attributes.

Uses of Entities

Entities are used for:

- Reducing code in the DTD by bundling declarations into entities
- Denoting special markups such as > and < tag
- Managing binary files and other data which is not native to XML
- Repeating frequently used text, which guarantees consistency in spelling and usage

Pre-defined Entities

- Pre-defined entities are built-in entities that can be invoked without declaration

- There are exactly five predefined entities: <, >, &, " , and '

Declaring Entities

- Entities must be declared in DTD before using them
- Declared with the "ENTITY" declaration

```
<!ENTITY name definition>
```

  - *name*: an entity name used to refer to the entity
  - *definition*: an entity's definition can take several different forms

- Exact format of the declaration distinguishes between internal and external entities

Declaring Internal Entities

- An internal entity declaration has the following form:

```
<!ENTITY name "replacement text">
```

- Example:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE newspaper [
<!ENTITY ADTEXT "An XML training">
<!ELEMENT newspaper (#PCDATA)>
]>

<newspaper>&ADTEXT;</newspaper>
```

Entity defined in DTD

This is an entity reference that will be replaced with the text

Declaring External Entities

External entity declarations come in two forms

- 1st form: An external entity defined in another document containing text

```
<!ENTITY name SYSTEM URI>
```

  - *name*: an entity name used to refer to the entity
  - *SYSTEM*: an external entity that is private
  - *URI*: an external document, most commonly a simple filename

```
<!DOCTYPE library [
<!ENTITY greet SYSTEM "C:\greeting.txt">
<!ELEMENT library (#PCDATA | magazine)*>
<!ELEMENT magazine (#PCDATA)>
]>
<library>
    <magazine>&greet;</magazine>
</library>
```

External entities: Non-xml formats

- 2nd form: External entities that refer to files containing non-XML data

- For example : Graphics stored in odd formats like PNG and GIF must declare that data they contain is not XML

- Achieved by indicating the format of external entity in a notation

```
<!ENTITY entityname SYSTEM URI NDATA notation>
```

- An external entity that refers to GIF image pic1.gif might be declared

```
<!ENTITY myphoto SYSTEM "pic1.gif" NDATA GIF>
```

Entity Attributes

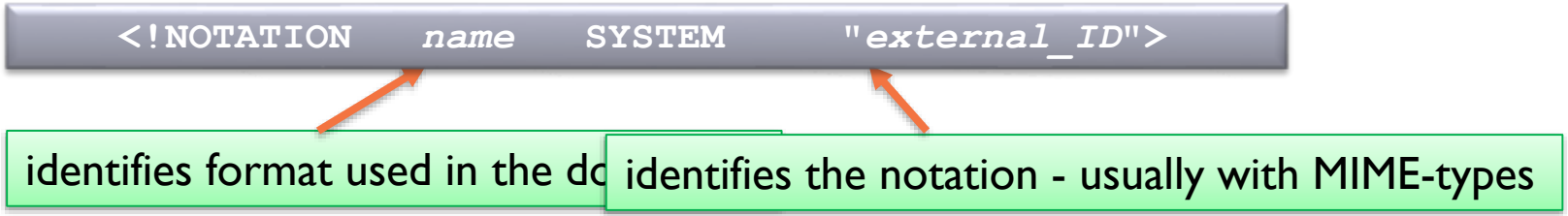
- External entities can be further classified as either "parsed" or "unparsed“
  - Entities which refer to external files that contain text are called "parsed entities"
  - Entities which refer to other types of data, identified by a notation, are "unparsed“
- Parser inserts replacement text of a parsed entity into the document wherever a reference to that entity occurs
- Using entity reference to an unparsed entity directly in an XML document is an error
- Unparsed entities can only be used as attribute values on elements with ENTITY attributes
- Unparsed entities are used most frequently on XML elements that incorporate graphics into a document

Example

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE picture [
<!ELEMENT picture (para|graphic)+>
<!ELEMENT para (#PCDATA)>
<!ELEMENT graphic EMPTY>
<!ATTLIST graphic
        image ENTITY #REQUIRED
        alt   CDATA  #IMPLIED >
<!NOTATION GIF SYSTEM "Graphics Format">
<!ENTITY myphoto SYSTEM "adventure.gif" NDATA GIF>
<!ENTITY tig "Wild Animal">
]>
<picture>
    <para>image incorporated which is declared as "myphoto":
    </para>
    <graphic image="myphoto" alt="A picture of &tig;"/>
</picture>
```

Notations

- Allow to include non-XML data in your documents by describing the format and allow your application to recognize and handle it
- The format for a notation is:



- For example, to include a GIF image in your XML document:

```
<!NOTATION GIF SYSTEM "image/gif">
```

Attribute type - NOTATION

- The NOTATION declaration describes the format of non-XML data within an XML document
- In DTD

```
<!NOTATION  abc SYSTEM "image/jpeg">
<!NOTATION xyz SYSTEM "image/gif">
<!ELEMENT person (#PCDATA)>
<!ATTLIST person
    photo NMTOKEN #IMPLIED
    filetype NOTATION (abc|xyz) #IMPLIED>
```

- In XML

```
<person photo="p1.gif" filetype="xyz">Anny</person>
```

Summary

In this module, you were able to

- Create DTDs
- Use internal and external DTDs
- Declare elements
- Declare attributes
- Declare entities
- Use predefined entities
- Use external entities containing text
- Use external entities containing non-xml data
- Declare notations in DTDs

Thank You