DTD Attributes

Elements can have zero or more attributes, which are declared using the !ATTLIST tag. Unlike element definitions, attribute definitions do not impose order on when the attributes occur. Furthermore, if several attributes are declared for the same element type, they can be declared using multiple !ATTLIST tags.
In the DTD, XML element attributes are declared with an ATTLIST declaration.

Syntax :
<!ATTLIST element-name attribute-name attribute-type default-value>

DTD Example :
<!ATTLIST payment type CDATA "check">
XML Example :
<payment type="check" />

Each attribute in a declaration has three parts: a name, a type, and a default value.

ATTRIBUTE DEFAULT VALUES

Default Value	Description
value	The default value of the attribute
#REQUIRED	The attribute value must be included in the element
#IMPLIED	The attribute does not have to be included
#FIXED value	The attribute value is fixed

Attribute default values can be value, optional (#IMPLIED), required (#REQUIRED), or fixed (#FIXED).

Default attribute value
Syntax:
<!ATTLIST element-name attribute-name CDATA "default-value">
DTD example:
<!ATTLIST payment type CDATA "check">
XML example:
<payment type="check">

Specifying a default value for an attribute, assures that the attribute will get a value even if the author of the XML document didn't include it.

Implied attribute
Syntax:
<!ATTLIST element-name attribute-name attribute-type #IMPLIED>
DTD example:
<!ATTLIST contact fax CDATA #IMPLIED>
XML example:
<contact fax="555-667788">

Use an implied attribute if you don't want to force the author to include an attribute and you don't have an option for a default value either.

Required attribute

Syntax:
<!ATTLIST element-name attribute_name attribute-type #REQUIRED>
DTD example:
<!ATTLIST person number CDATA #REQUIRED>
XML example:
<person number="5677">

Use a required attribute if you don't have an option for a default value, but still want to force the attribute to be present.

Fixed attribute value

Syntax:
<!ATTLIST element-name attribute-name attribute-type #FIXED "value">
DTD example:
<!ATTLIST sender company CDATA #FIXED "Microsoft">
XML example:
<sender company="Microsoft">

Use a fixed attribute value when you want an attribute to have a fixed value without allowing the author to change it. If an author includes another value, the XML parser will return an error. An attribute declaration may specify that an attribute has a fixed value. In this case, the attribute is not required, but if it occurs, it must have the specified value. If it is not present, it will appear to be the specified default. One use for fixed attributes is to associate semantics with an element.

Example of a DTD	**Example 1 of XML using this DTD**
<!ELEMENT book EMPTY> <!ATTLIST book title CDATA #REQUIRED author CDATA 'anonymous' weight CDATA #IMPLIED format (paper-back | hard-back) 'paper-back' >	<?xml version="1.0" standalone="no"?> <!DOCTYPE book SYSTEM "book.dtd"> <book title="False Pretences" author="Margaret Yorke" format="hard-back" />

ATTRIBUTE TYPE

Attributes can have these types:
CDATA, Enumerated attribute value, ID, IDREF, IDREFS, NMTOKEN, NMTOKENS, ENTITY, ENTITIES, NOTATION

CDATA

CDATA attributes are strings, any text is allowed. Don't confuse CDATA attributes with CDATA sections.

Enumerated attribute values

Syntax:
<!ATTLIST element-name attribute-name (eval|eval|..)>
DTD example:
<!ATTLIST payment type (check|cash)>
XML example:
<payment type="check">
or
<payment type="cash">
Use enumerated attribute values when you want the attribute values to be one of a fixed set of legal values.

ID

The value of an ID attribute must be a name. All of the ID values used in a document must be different. IDs uniquely identify individual elements in a document. Elements can have only a single ID attribute.

IDREF

The IDREF type allows the value of one attribute to be an element elsewhere in the document provided that the value of the IDREF is the ID value of the referenced element. A validating parser verifies that no two elements have the same *ID* value. Thus the attribute can be used as a *key* of the element. Other elements can then have *IDREF* attributes (defined in a DTD) that "reference" an element by its *ID* attribute. Note that a validating parser makes sure that all *IDREF* attributes refer to an existing *ID* attribute in the XML document.

XML example:

```
<author ref="myorke">Margaret Yorke</author>
...
<book author="myorke">False Pretences</book>
```

is based on the DTD:

```
<!ELEMENT author (#PCDATA)>
<!ATTLIST author ref ID #REQUIRED>
<!ELEMENT book (#PCDATA)>
    <!ATTLIST book author IDREF #IMPLIED>
```

The disadvantages are that the *ID* values are not restricted to an element. That is, an *ID* attribute has to be unique in an XML document regardless of the element in which it is defined. This is troublesome because if an XML document includes a *student* element and a *course* element, both with their own *ID* attribute, then the *id* (or key) of a student cannot be the same of the *id* of a course. This issue has to be considered when designing the DTD and the understanding of the *id* in elements that contain it.

Another disadvantage is that the *semantics* of an *ID/IDREF* cannot be formally specified. For example, a *student* element with an *IDREF* attribute that is intended to be a relationship to a course *id* cannot be specified in the DTD rules. A validating parser only validates that an *IDREF* has a value for an *id* that exists within the document, it cannot validate that such value has to exist "in a specific element". That is, the following basic rule cannot be specified in the DTD: the value for the attribute "takes" of a *student* element refers to an *id* of an existing *course* element.

IDREFS

The value is a list of other ids. An IDREF attribute's value must be the value of a single ID attribute on some element in the document. The value of an IDREFS attribute may contain multiple IDREF values separated by white space.

ENTITY or ENTITYS

An ENTITY attribute's value must be the name of a single entity . The value of an ENTITIES attribute may contain multiple entity names separated by white space.

Values of type ENTITY must match the Name production, values of type ENTITIES must match Names; each Name must match the name of an unparsed entity declared in the DTD.

```
<!ATTLIST MYPHOTO FILENAME ENTITY #REQUIRED>
<!ENTITY employeephoto "images_employees" >
<!ELEMENT employee (name, sex, title, years) >
<!ATTLIST employee pic ENTITY #IMPLIED >
```

...
<employee pic="employeephoto">

NMTOKEN or NMTOKENS

"NMTOKEN" implying that the value must conform to XML identifier name specifications. Name token attributes are a restricted form of string attribute. In general, an NMTOKEN attribute must consist of a single word. But there are no additional constraints on the word; it doesn't have to match another attribute or declaration. The value of an NMTOKENS attribute may contain multiple NMTOKEN values separated by white space.

The characters of an NMTOKEN value must be a letter, digit, '.', '-', '_', or ':'. It may not include white space.

<!ELEMENT student_name (#PCDATA)>
<!ATTLIST student_name student_no NMTOKEN #REQUIRED>
example:
<student_name student_no="9216735">Jo Smith</student_name>

<!ATTLIST stud s_no NMTOKENS #REQUIRED>
example:
<stud stu_no="9216735 err">Jo Smith</stud>

Notation Type

specifies that the names must match a particular notation name. Notation declarations identify specific types of external binary data. This information is passed to the processing application, which may make whatever use of it it wishes. A typical notation declaration is:
<!NOTATION> tag is used to define a notation in DTD. Here GIF87A mp and st are notation.
<! NOTATION GIF87A SYSTEM "GIF">
<!NOTATION mp SYSTEM "mplay32.exe">
<!NOTATION st SYSTEM "soundtool">

<!ATTLIST sound player NOTATION #REQUIRED>
Example:
<sound player="mp">
Attribute player of sound tag can store only the names of defined notation.

Figure 2. A short bibliography in XML. This example demonstrates the use of attributes as well as referencing using the id and idref attributes.

```
<?xml version="1.0" standalone="yes">
<!– This is an example bibliography. –>
<BIB>
<BOOK nickname="Dragon book">
<AUTHOR id="aho"> Aho, A. V. </AUTHOR>
<AUTHOR id="sethi"> Sethi, R. </AUTHOR>
<AUTHOR id="ullman"> Ullman, J. D. </AUTHOR>
<TITLE> Compilers: Principles, Techniques, and Tools </TITLE>
<PUBLISHER> Addison-Wesley </PUBLISHER>
<YEAR> 1985 </YEAR>
</BOOK>
<BOOK>
<AUTHOR idref ="ullman"/>
```

```
<TITLE> Principles of Database and Knowledge-Base Systems, Vol.1 </TITLE>
</BOOK>
...
</BIB>
```

Figure 3. A DTD for the bibliography example. The DTD defines a grammar for documents.

```
<!DOCTYPE bib [
<!ELEMENT BIB (BOOK+)>
<!ELEMENT BOOK (AUTHOR+, TITLE, PUBLISHER?, YEAR?)>
<!ATTLIST BOOK isbn CDATA #IMPLIED nickname CDATA #IMPLIED>
<!ELEMENT AUTHOR (#PCDATA)>
<!ATTLIST AUTHOR id ID #IMPLIED idref IDREF #IMPLIED>
<!ELEMENT TITLE (#PCDATA)>
<!ELEMENT PUBLISHER (#PCDATA)>
<!ELEMENT YEAR (#PCDATA)>
]>
```

Weaknesses of the DTD:

- DTD has a limited capability for specifying data types
- DTD requires its own language, Not in XML syntax
- DTD provides incompatible set of data types with those found in databases. e.g. “PCDATA,” “CDATA”
- Example: DTD do not allow to specify element **day** and **month** of Type *Integer* and within a certain *Range*: <day>32</day><month>13</month>
- Do not support XML namespace
- Not expressed in XML syntax - can’t process them using XML tools (such as parsers)

XML Schemas Overcome most Limitations of DTD

- Richer capabilities than a DTD
  - Describes the possible arrangement of tags and text in a valid document
  - 44+ datatypes (as opposed to 10 in DTDs)
  - Allows attribute grouping
  - Supports use of namespaces
  - Supports object-oriented design
    - “Custom” data types can be derived from other data types and inherit their attributes *
  - Can specify element constraints and valid values

XML Schema “Limitations”

- Not all developers and standards bodies use Schema the same way
  - There are at least 20 ways to declare attributes
    - From local attribute with default value to global attribute with fixed value
  - At least 15 ways to declare elements
    - From local element with default value to global element with complex type