



**INTERNET & WEB**  
**APPLICATION DEVELOPMENT**  
**SWE 444**

Fall Semester 2008-2009 (081)

**Module 4 (V): Document Type  
Definitions (DTDs)**

**Dr. El-Sayed El-Alfy**

Computer Science Department  
King Fahd University of Petroleum and Minerals  
alfy@kfupm.edu.sa

## Objectives/Outline

### • Objectives

- Understand the role of DTDs
- Learn how to write and use DTDs

### • Outline

- What are DTDs?
- Why DTDs?
- DTD Syntactic Elements
  - ELEMENT
  - ATTRIBUTE
  - ENTITY
  - Types
- Examples
- Validation

## What is DTD?

- Document Type Definition (DTD) is a grammar that describes the structure of a class of XML documents
  - structure of the documents is described via
    - element and attribute-list declarations.
- Element declarations
  - name the allowable set of elements within the document, and
  - specify whether and how declared elements and runs of character data may be contained within each element.
- Attribute-list declarations
  - name the allowable set of attributes for each declared element, including the type of each attribute value, if not an explicit set of valid value(s).
- DTDs are written in EBNF-like notation
  - EBNF (Extended Backus–Naur Form) used to express context-free grammars, i.e. a formal way to describe computer programming languages and other formal languages

## Why DTDs?

- XML documents are designed to be processed by computer programs
  - If you can put just any tags in an XML document, it's very hard to write a program that knows how to process the tags
  - A DTD specifies what tags may occur, when they may occur, and what attributes they may (or must) have
- Applications can use a standard DTD to verify that the data you receive from the outside world is valid
- With a DTD, independent groups of people can agree to use a standard DTD for interchanging data
  - produce consistent XML documents
- You can also use a DTD to verify your own data

## Parsers

- An XML *parser* is an API that reads the content of an XML document
  - Currently popular APIs are **DOM** (Document Object Model) and **SAX** (**S**imple **A**PI for **X**ML)
- A *validating parser* is an XML parser that compares the XML document to a DTD and reports any errors

## Building Blocks of XML Documents

- From a DTD point of view, all XML documents (and HTML documents) are made up by the following building blocks:
  - **Elements** -- the main building blocks of both XML and HTML documents
  - **Attributes** -- provide extra information about elements, placed inside the opening tag of an element, come in name-value pairs
  - **Entities** -- are a way to include fixed text
  - **PCDATA** – Parsed Character DATA (the text that WILL be parsed by a parser; tags inside the text will be treated as markup and entities will be expanded)
  - **CDATA** -- Character DATA (i.e. the text found between the start tag and the end tag of an XML element and will NOT be parsed by a parser; tags inside the text will NOT be treated as markup and entities will not be expanded)

## An XML example

```
<novel >
  <foreword>
    <paragraph>This is a great novel .
    </paragraph>
  </foreword>
  <chapter number="1">
    <paragraph>It was a dark and stormy
    night.</paragraph>
    <paragraph>Suddenly, a shot rang
    out!</paragraph>
  </chapter>
</novel >
```

- An XML document contains (and the DTD describes):
  - Elements, such as `novel` and `paragraph`, consisting of tags and content
  - Attributes, such as `number="1"`, consisting of a name and a value
  - Entities (not used in this example)

## A DTD example

```
<!DOCTYPE novel [
  <!ELEMENT novel (foreword, chapter+)>
  <!ELEMENT foreword (paragraph+)>
  <!ELEMENT chapter (paragraph+)>
  <!ELEMENT paragraph (#PCDATA)>
  <!ATTLIST chapter number CDATA #REQUIRED>
]>
```

- A `novel` consists of a `foreword` and one or more `chapter`s, in that order
  - Each `chapter` must have a `number` attribute
- A `foreword` consists of one or more `paragraph`s
- A `chapter` also consists of one or more `paragraph`s
- A `paragraph` consists of parsed character data (text that cannot contain any other elements)

## DTD ELEMENT

- In a DTD, elements are declared with an ELEMENT declaration.

- General Syntax

`<! ELEMENT el element-name category>`

or

`<! ELEMENT el element-name (el element-content)>`

## Elements without children

- The syntax is `<! ELEMENT name category>`
  - *name* is the element name used in start and end tags
  - *category* should be `EMPTY`
  - Example
    - In the DTD: `<! ELEMENT br EMPTY>`
    - In the XML: `<br></br>` or just `<br />`
  - In the XML, an empty element may not have any content between the start tag and the end tag
  - An empty element may (and usually does) have attributes

## Elements with unstructured children

- The syntax is `<! ELEMENT name category>`
  - *category* may be **ANY**
    - This indicates that any content -- character data, elements, even undeclared elements -- may be used
    - Since the whole point of using a DTD is to define the structure of a document, **ANY should be avoided wherever possible**
  - *category* may be **(#PCDATA)**, indicating that only parsed character data may be used
    - Example
      - In DTD: `<! ELEMENT paragraph (#PCDATA)>`
      - In XML: `<paragraph>A shot rang out! </paragraph>`
    - Notes:
      - The parentheses are required!
      - In **(#PCDATA)**, whitespace is kept exactly as entered
      - Elements may not be used within parsed character data
      - Entities are character data, and may be used

## Elements with children

- An element may have one or more children
- The syntax is `<! ELEMENT element-name (child1)>`
  - Or `<! ELEMENT element-name (child1, child2, ...)>`
  - Example `<! ELEMENT note (to, from, heading, body)>`
    - Parentheses are required, even if there is only one child
    - A space must precede the opening parenthesis
    - Commas (,) between elements mean that all children must appear, and must be in the order specified
    - All child elements must themselves be declared
    - Children may have children
    - “|” separators means any one child may be used
    - Suffixes can be used, e.g. `<! ELEMENT novel (foreword, chapter+)>`
    - Parentheses can be used for grouping, e.g. `<! ELEMENT novel (foreword, (chapter+|section+))>`

## ELEMENT descriptions

### ➤ Suffixes:

?	optional (zero or one)	foreword?
+	one or more	chapter+
*	zero or more	appendix*

### ➤ Separators

,	both, in order	foreword?, chapter+
	or	section chapter

### ➤ Grouping

( )	grouping	(section chapter)+
-----	----------	--------------------

## Elements with mixed content

- **#PCDATA** describes elements with only character data
- **#PCDATA** can be used in an “or” grouping, e.g.  
`<!ELEMENT note (#PCDATA|message)*>`
  - This is called *mixed content*
  - Certain (rather severe) restrictions apply:
    - **#PCDATA** must be first
    - The separators must be “|”
    - The group must be *starred* (meaning zero or more)

## Names and namespaces

- All names of elements, attributes, and entities, in both the DTD and the XML, are formed as follows:
  - The name must begin with a letter or underscore
  - The name may contain only letters, digits, dots, hyphens, underscores, and colons
- The DTD doesn't know about namespaces -- as far as it knows, a colon is just part of a name
  - The following are different (and both legal):
    - `<!ELEMENT chapter (paragraph+)>`
    - `<!ELEMENT myBook:chapter (myBook:paragraph+)>`
  - Avoid colons in names, except to indicate namespaces

## An expanded DTD example

```
<!DOCTYPE novel [  
  <!ELEMENT novel  
    (foreword, chapter+, biography?, critical Essay*)>  
  <!ELEMENT foreword (paragraph+)>  
  <!ELEMENT chapter (section+|paragraph+)>  
  <!ELEMENT section (paragraph+)>  
  <!ELEMENT biography(paragraph+)>  
  <!ELEMENT critical Essay (section+)>  
  <!ELEMENT paragraph (#PCDATA)>  
>
```



## Attributes and entities

- In addition to elements, a DTD may declare attributes and entities
- An attribute describes information that can be put within the start tag of an element
  - In XML:  
`<car name= "Toyota" model = "2001"></car>`
  - In DTD:  
`<! ATTLIST car  
          name CDATA #REQUIRED  
          model CDATA #IMPLIED >`
- An entity describes text to be substituted
  - In XML: `&copyri ght;`
  - In DTD: `<! ENTI TY copyri ght "Copyri ght KFUPM" >`

## Attributes

- The format of an attribute is:  
`<! ATTLIST element-name  
          name type requirement  
          name type requirement>`  
where the *name-type-requirement* may be repeated as many times as desired
- Only spaces separate the parts, so careful counting is essential
- The *element-name* tells which element may have these attributes
- The *name* is the name of the attribute
- Each attribute has a *type*, such as `CDATA` (character data)
- Each attribute may be *required*, *optional*, or *fixed*
- In XML, attributes may occur in any order

## Important attribute types

- There are ten attribute types
- These are the most important ones:
  - **CDATA**                   The value is character data
  - **(man|woman|child)**       The value is one from this list
  - **ID**                        The value is a unique identifier
    - ID values must be legal XML names and must be unique within the document
  - **NMTOKEN**                The value is a legal XML name
    - This is sometimes used to disallow whitespace in the name
    - It also disallows numbers, since an XML name cannot begin with a digit
- The other seven, less frequently used, are:
  - **IDREF, IDREFS, NMTOKENS, ENTITY, ENTITIES, NOTATION, xml:**

## Requirements

- Recall that an attribute has the form  
`<!ATTLIST element-name name type requirement>`
- The requirement is one of:
  - A default value, enclosed in quotes
    - Example: `<!ATTLIST degree CDATA "PhD">`
  - **#REQUIRED**
    - The attribute must be present
  - **#IMPLIED**
    - The attribute is optional
  - **#FIXED "value"**
    - The attribute always has the given value
    - If specified in the XML, the same value must be used

## Entities

- Entities are a way to include fixed text (sometimes called “boilerplate”)
- Example of use in the XML:  
`This document is &copy; 2002.`
- There are exactly five predefined entities:  
`&lt;`, `&gt;`, `&amp;`, `&quot;` and `&apos;`;
- Additional entities can be defined in the DTD:  
`<! ENTITY copyri ght "Copyri ght KFUPM" >`
- Entities can be defined in another document:  
`<! ENTITY copyri ght SYSTEM "MyURI " >`
- Entities should not be confused with character references, which are numerical values between `&` and `#`
  - Example: `&233#;` or `&xE9#;` to indicate the character é

## Another example: XML

```
<?xml version="1.0"?>
<!DOCTYPE myXmlDoc SYSTEM
  "http://www.mysite.com/mydoc.dtd">
<weatherReport>
  <date>05/29/2002</date>
  <location>
    <city>Philadelphia</city>, <state>PA</state>
    <country>USA</country>
  </location>
  <temperature-range>
    <high scale="F">84</high>
    <low scale="F">51</low>
  </temperature-range>
</weatherReport>
```

## The DTD for this example

```
> <!ELEMENT weatherReport (date, location,
                             temperature-range)>
<!ELEMENT date (#PCDATA)>
<!ELEMENT location (city, state, country)>
<!ELEMENT city (#PCDATA)>
<!ELEMENT state (#PCDATA)>
<!ELEMENT country (#PCDATA)>
<!ELEMENT temperature-range
      ((low, high) | (high, low))>
<!ELEMENT low (#PCDATA)>
<!ELEMENT high (#PCDATA)>
<!ATTLIST low scale (C|F) #REQUIRED>
<!ATTLIST high scale (C|F) #REQUIRED>
```

## Inline DTDs

- If a DTD is used only by a single XML document, it can be put directly in that document, as follows:

```
<?xml version="1.0">
<!DOCTYPE myRootElement [
  <!-- DTD content goes here -->
]>
<myRootElement>
  <!-- XML content goes here -->
</myRootElement>
```

- An inline DTD can be used only by the document in which it occurs

## External DTDs

- External DTDs are almost always preferable to inline DTDs, since they can be used by more than one document
- The file extension for an external DTD is `.dtd`
  - External DTDs can only be referenced with a URL
- An external DTD (a DTD that is a separate document) is declared with a `SYSTEM` or a `PUBLIC` command:

```
<!DOCTYPE myRootElement SYSTEM  
"http://www.mysite.com/mydoc.dtd">
```

  - The name that appears after `DOCTYPE` (in this example, `myRootElement`) must match the name of the XML document's root element
  - Use `SYSTEM` for external DTDs that you define yourself, and use `PUBLIC` for official, published DTDs

## Validators

- Opera 5 and Internet Explorer 5+ can validate your XML against an internal DTD
  - IE provides (slightly) better error messages
  - Opera apparently just ignores external DTDs
  - IE considers an external DTD to be an error
- jEdit with the XML plugin will check for well-structuredness and (if the DTD is inline) will validate your XML each time you do a Save
  - <http://www.jedit.org/>
- Validate [Using Inline DTD]  
<http://www.stg.brown.edu/service/xmlvalid/>

## Limitations of DTDs

- DTDs are a very weak specification language
  - You can't put any restrictions on element contents
  - It's difficult to specify:
    - All the children must occur, but may be in any order
    - This element must occur a certain number of times
  - There are only ten data types for attribute values
- But most of all: DTDs aren't written in XML!
  - If you want to do any validation, you need one parser for the XML and another for the DTD
  - This makes XML parsing harder than it needs to be
  - There is a newer and more powerful technology: *XML Schemas*
  - However, DTDs are still very much in use

## Q & A



## References

- Some useful links with examples and other resources:
  - W3School DTD Tutorial
    - <http://www.w3schools.com/dtd/default.asp>
  - MSXML 4.0 SDK
  - <http://www.topxml.com>
  - <http://www.xml.org>
  - <http://www.xml.com>
  - Several online presentations