Document Type Definition (DTD)

Asst. Prof. Dr. Kanda Runapongsa
(krunapongsa@kkuni.ac.th)
Dept. of Computer Engineering
Khon Kaen University

Overview
- Document type declaration
- Element type declaration
- Element type content specification
- Attribute-list declaration
- Entity declaration
- Notation declaration
- Internal and external DTDs
Document Type Declaration (1/3)

- Document type declaration declares the document type that is in use in the document
- We use the `<!DOCTYPE>` element to create a document type declaration
- It should be placed before the root element

Document Type Declaration (2/3)

- The element `<!DOCTYPE>` can take many different forms
  - `<!DOCTYPE rootname [DTD]>`
  - `<!DOCTYPE rootname SYSTEM URI>`
  - `<!DOCTYPE rootname PUBLIC FPI URI>`
Document Type Declaration (3/3)

- With the SYSTEM keyword, the DTD is for private use by an organization of individuals
- With the PUBLIC keyword, the DTD is for public use which needs a formal public identifier (FPI)
  - To use the PUBLIC keyword, we must also create a FPI

Rules for FPIs (1/3)

- FPIs must follow a specific syntax. This syntax is
  “Owner//Keyword Description//Language”
- Owner
  - This indicates the owner of the FPI
  - If this string start with “ISO” then this is an ISO owned FPI. For example, ISO:8879:1986 is the ISO number of the SGML standard
  - Otherwise, this string will either look like -//Owner or +//Owner
Rules for FPIs (2/3)

- FPIs must follow a specific syntax. This syntax is “Owner/Keyword Description/Language”
- Owner
  - If the string starts with - then the owner information is unregistered
  - If the string starts with + then the owner information identifies it as being registered

Rules for FPIs (3/3)

- FPIs must follow a specific syntax. This syntax is “Owner/Keyword Description/Language”
- Keyword
  - It indicates the type of document
- Description
  - Any description you want to supply for the contents of this file. This may include version numbers or any short text that is meaningful to you
FPI Examples

- <!DOCTYPE DOCUMENT PUBLIC "-//abc//MyXML Version 1.0//EN" "http://www.abc.com/MyXML.dtd">
- <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN" "http://www.w3.org/TR/REC-html4.0/strict.dtd">

XML with Document Type Declaration

```xml
<?xml version="1.0" standalone="yes"?>
<!DOCTYPE nation [
  <!ELEMENT nation (name, location)>  
  <!ATTLIST nation id ID #REQUIRED>  
  <!ELEMENT name (#PCDATA)>  
  <!ELEMENT location (#PCDATA)>  
]>
<nation id="th">  
  <name>Thailand</name>  
  <location>Southeast Asia</location>  
</nation>
```
Document Type Declaration Examples

- With the document type definition in an external file
- In file “nation.xml”
  ```xml
  <!DOCTYPE nation SYSTEM “nation.dtd”>
  <nation>…</nation>
  ```
- In file “nation.dtd”
  ```xml
  <!ELEMENT nation (name, location)>
  ...
  <!ELEMENT location (#PCDATA)>
  ```

Element Type Declaration

- Element type declaration must
  - Start with the string “<!ELEMENT”
  - Followed by the name
  - Then followed by content specification
- Element type names are XML names
- Each element type declaration must use a different name
  - Because a particular element type cannot be declared more than once
**Element Type Content Specification**

- **EMPTY content**
  - May not have content

- **ANY content**
  - May have any content

- **Mixed content**
  - May have character data or a mix of character data and sub-elements

- **Children content**
  - May have only sub-elements

---

**EMPTY Content**

- For an element type that can never have any content, we would give it a content specification of EMPTY

- In DTD:
  - `<!ELEMENT br EMPTY>`

- In XML:
  - `<br/>`
ANY Content

- For an element with the content model of ANY, the declared element can contain any type of content
  - In DTD:
    ```xml
    <!ELEMENT misc ANY>
    ```
  - In XML:
    ```xml
    <misc>car</misc>
    <misc><house/></misc>
    ```

Mixed Content

- Element types with mixed content are allowed to hold
  - Character data alone
  - Character data with child elements interspersed
- A paragraph is a good example of a typical mixed content element
  - It might have character data with some mixed in emphasis and quotation sub-elements
Mixed Content with Only Characters

- The simplest mixed content specifications allow data only and start with "(" followed by the string #PCDATA and ended with a ")"
  - In DTD
    `<!ELEMENT emph (#PCDATA)>`
  - In XML
    `<emph>Thailand</emph>`

- The declaration above create element types that cannot contain sub-elements

Mixed Content (Elements and Character Data)

- We can extend the DTD to allow a mix of elements and character data by using * and |
  - `<!ELEMENT paragraph (#PCDATA|emph)*>`
- The * (0 to many) is required to allow a mix of character data and elements
- The | is required to indicate “or” operation between items
- The format of the mixed content element declaration must be (#PCDATA|e1|e2)*
Mixed Content Element Example

<paragraph>
  We live in
  <emph>Thailand</emph>
  which is located in Asia<br/>
  We use <emph>Thai</emph>
  language as a national language
</paragraph>

Children Content Model: Single

- A simple children content model could have a single sub-element type
  - <!ELEMENT book (title)>  
  - The above declaration indicates that a book must have a single title within it 
  - “book” must have one and only one “title”
### Related XML Examples

<table>
<thead>
<tr>
<th>Invalid XML</th>
<th>Valid XML</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;book&gt;</code></td>
<td><code>&lt;book&gt;</code></td>
</tr>
<tr>
<td><code>&lt;title&gt;</code>XML and Web Services&lt;/title&gt;`</td>
<td><code>&lt;title&gt;</code>XML and Web Services&lt;/title&gt;`</td>
</tr>
<tr>
<td><code>&lt;title&gt;</code>PHP Web Services&lt;/title&gt;`</td>
<td><code>&lt;title&gt;</code>PHP Web Services&lt;/title&gt;`</td>
</tr>
<tr>
<td><code>&lt;/book&gt;</code></td>
<td><code>&lt;/book&gt;</code></td>
</tr>
</tbody>
</table>

### Children Content Model: Sequence

- To indicate multiple sub-elements while the order among those elements are important, we use “,” between them
  - `<!ELEMENT memo (from, to, subject, body)>`
  - The above memo element consists of a sequence of elements
Related XML Examples

- Invalid XML
  ```xml
  <memo>
  <to>Students</to>
  <from>Teachers</from>
  <subject>Greeting</subject>
  <body>Hello</body>
  </memo>
  ```

- Valid XML
  ```xml
  <memo>
  <from>Teachers</from>
  <to>Students</to>
  <subject>Greeting</subject>
  <body>Hello</body>
  </memo>
  ```

Children Content Model: Choice

- Sometimes we want to have a choice rather than a sequence. In this case we use “|” which indicates that the author can choose between the element types

  - `<!ELEMENT figure (graphic|code)>`
  - A figure can contain either a graphic element or a code element
Related XML Examples

- Invalid XML
  ```xml
  <figure>
  <graphic>
  flower.jpg
  </graphic>
  </figure>
  ```

- Valid XML
  ```xml
  <figure>
  <code>
  123456
  78910
  </code>
  </figure>
  ```

Children Content Model: Combine

- We may also combine choices and sequences using parenthesis
  - `<!ELEMENT figure (caption, (table|flow-chart))>`
  - The above figure element is made up of a sequence of two content particles
Related XML Examples

<table>
<thead>
<tr>
<th>Invalid XML</th>
<th>Valid XML</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;figure&gt;</code></td>
<td><code>&lt;figure&gt;</code></td>
</tr>
<tr>
<td><code>&lt;caption&gt;</code> Fig 1</td>
<td><code>&lt;caption&gt;</code> Fig 1</td>
</tr>
<tr>
<td><code>&lt;/caption&gt;</code></td>
<td><code>&lt;/caption&gt;</code></td>
</tr>
<tr>
<td><code>&lt;table&gt;</code> t1</td>
<td><code>&lt;table&gt;</code> t1</td>
</tr>
<tr>
<td><code>&lt;/table&gt;</code></td>
<td><code>&lt;/table&gt;</code></td>
</tr>
<tr>
<td><code>&lt;flow-char&gt;</code> f1</td>
<td><code>&lt;/flow-char&gt;</code></td>
</tr>
<tr>
<td><code>&lt;/flow-chart&gt;</code></td>
<td><code>&lt;/flow-chart&gt;</code></td>
</tr>
<tr>
<td><code>&lt;/figure&gt;</code></td>
<td><code>&lt;/figure&gt;</code></td>
</tr>
</tbody>
</table>

Occurrence Indicators (1/4)

- XML allows us to specify that a content participle is optional or repeatable using an occurrence indicators
  - `?` Optional (0 or 1 time)
  - `*` Optional and repeatable (0 or more times)
  - `+` Required and repeatable (1 or more times)
Occurrence Indicators (2/4)

- `<!ELEMENT image (caption?)>`
  - The above declaration allows the image element to be empty sometimes and contains caption sometimes
  - XML Example 1
    ```xml
    <image>
      <caption>Image 1</caption>
    </image>
    ```
  - XML Example 2
    ```xml
    <image/>
    ```

Occurrence Indicators (3/4)

- `<!ELEMENT book (chapter*)>`
  - The above declaration requires that a book can have none or many chapters
  - XML Example 1
    ```xml
    <book>
      <chapter>Chapter 1</chapter>
      <chapter>Chapter 2</chapter>
    </book>
    ```
  - XML Example 2
    ```xml
    <book/>
    ```
Occurrence Indicators (4/4)

- `<!ELEMENT book (chapter+)>`
  - The above declaration requires that a book must have at least one chapter and can have multiple chapters
  - XML Example
    ```xml
    <book>
    <chapter>Chapter 1</chapter>
    <chapter>Chapter 2</chapter>
    </book>
    ```

Attribute-list Declaration (1/4)

- Attributes are declared for specific element type
- We declare attributes for a particular element type using an attribute-list declaration
- We often see an attribute-list declaration right beside an element type declaration
  - `<!ELEMENT person (#PCDATA)>`
  - `<!ATTLIST person email CDATA #REQUIRED>`
Attribute-list Declaration (2/4)
- Attribute declarations start with the string “<!ATTLIST” and then comes the attribute’s name, its type, and its default
  - <!ATTLIST person email CDATA #REQUIRED>
  - The attribute is named person and is valid on email element. Its value must be character data and it is required

Attribute-list Declaration (3/4)
- We can declare many attributes in a single attribute-list declaration
  - <!ATTLIST person
    email CDATA #REQUIRED
    phone CDATA #REQUIRED
    fax CDATA#REQUIRED>
Attribute-list Declaration (4/4)

- It is possible to have multiple declarations for the same attribute of the same element type
  - The first declaration of the attribute is binding and the rest are ignored
- Two different element types can have attributes with the same name without a conflict

Attribute Types: CDATA and ENTITY

- CDATA: Character data (not include markup)
  - In DTD: `<!ATTLIST person email CDATA #REQUIRED>`
  - In XML: `<person email="manee@thai.com">`
- ENTITY: Names an entity
  - In DTD:
    `<!ENTITY ibm “International Business Machines”>`
    `<!ATTLIST company name ENTITY #REQUIRED>`
  - In XML: `<company name="ibm">`
Attribute Types: ENTITIES

- ENTITIES: Multiple entity names (separated by whitespace)
  - In DTD:
    ```xml
    <!ATTLIST students images ENTITIES #IMPLIED>
    <!ENTITY year1 SYSTEM "studentsy1.gif">
    <!ENTITY year2 SYSTEM "studentsy2.gif">
    ```
  - In XML:
    ```xml
    <students images="year1 year2">
    ```

Attribute Types: Enumerated

- Enumerated: Represents a list of values and any one of them is a legal attribute value
  - In DTD:
    ```xml
    <!ATTLIST customer credit_approved (true | false) "true">
    ```
  - In XML:
    ```xml
    <customer credit_approved="false">
    ```
### Attribute Types: NMTOKEN

- **NMTOKEN:** A name token
  - An attribute of this type can take only values that are made up of one or more letters, digits, hyphens, underscores, colons, and periods
  - In DTD:
    ```xml
    <!ATTLIST shipping ship_state NMTOKEN #REQUIRED>
    ```
  - In XML:
    ```xml
    <shipping ship_state="MI"/>
    ```

### Attribute Types: NMTOKENS

- **NMTOKENS:** To specify that an attribute value must be made up of NMTOKENs separated by whitespace
  - In DTD:
    ```xml
    <!ATTLIST person full_name NMTOKENS #REQUIRED>
    ```
  - In XML:
    ```xml
    <person full_name="Prawase Wasi"/>
    ```
Attribute Types: NOTATION

- NOTATION: Specify the format of non-XML data, and we use it to describe external entities
  - In DTD:
    ```
    <!NOTATION gif SYSTEM "image/gif">
    <!NOTATION jpg SYSTEM "image/jpeg">
    <!ATTLIST
        customer image NMTOKEN #IMPLIED
        image_type NOTATION (gif | jpg) #IMPLIED>
    ```
  - In XML:
    ```
    <customer image="image.gif"
        image_type="gif">
    ```

Attribute Types: ID

- ID: Its value is a proper XML name that is unique, not shared by other attributes of the ID type
  - In DTD:
    ```
    <!ATTLIST
        student student_id ID #REQUIRED>
    ```
  - In XML:
    ```
    <student student_id="S64228">
    ```
  - ID values must be proper XML names, thus they cannot start with a digit, such as “64228”
Attribute Types: IDREF

- IDREF: Holds the value of an ID attribute of some element, usually another element that the current element is related to
  - In DTD:
    ```xml
    <!ATTLIST employee employee_id ID #REQUIRED
    manager_id IDREF #IMPLIED>
    ```
  - In XML:
    ```xml
    <employee employee_id="e12"/>
    <employee employee_id="e13" manager_id="e12"/>
    ```

Attribute Types: IDREFS

- IDREFS: Multiple IDs of elements separated by whitespace
  - In DTD:
    ```xml
    <!ATTLIST employee
    employee_id ID #REQUIRED
    dept_id IDREFS #IMPLIED>
    ```
  - In XML:
    ```xml
    <employee employee_id="e05"
    dept_id="d01 d04">
    ```
Attribute Defaults (1/2)

- Attributes can have default values
- We include the default after the type or list of allowed values in the attribute list declaration
  - `<!ATTLIST shirt size (SMALL|MEDIUM|LARGE) “MEDIUM”>`
  - `<!ATTLIST shoes size NMTOKEN “12”>`

Attribute Defaults (2/2)

- Default values
  - The DTD author specifies the default value
- Implied attributes
  - The processor specifies the default value
- Required attributes
  - The XML author specifies the default value
- Fixed attributes
  - The attribute value is fixed and specified by the DTD author
Default Values

Include the default value after the type or list of allowed values in the attribute list declaration

- In DTD:
  ```xml
  <!ATTLIST shirt size (SMALL|MEDIUM|LARGE) "MEDIUM">
  ```

- In XML:
  ```xml
  <shirt><color>blue</color></shirt>
  <shirt size="LARGE">
    <color>red</color>
  </shirt>
  ```

Impliable Attributes

Allow the user to omit a value for a particular attribute without forcing a particular default

- In DTD
  ```xml
  <ATTLIST shirt size NMTOKEN #IMPLIED>
  ```

- In XML
  ```xml
  <shirt><color>blue</color></shirt>
  <shirt size="LARGE">
    <color>red</color>
  </shirt>
  ```
Required Attributes

- The XML author is required to specify the attribute values
- A value for an attribute is important and cannot reliably be default
  - In DTD
    ```xml
    <!ATTLIST course id ID #REQUIRED>
    ```
  - In XML
    ```xml
    <course id="c178375">XML and Web Services</course>
    ```

Fixed Attributes

- Attribute values cannot be overridden at all
- For the purpose of easy integration between documents
  - In DTD
    ```xml
    <!ATTLIST document language CDATA #FIXED "th">
    ```
  - In XML
    ```xml
    <document>
    <title>Phone Directory</title>
    </document>
    ```
Notation Declarations

- Notations referred to in various parts of an XML document, for describing the data content notation of different things.
- A data content notation is a definition of how the bits and bytes of class of object should be interpreted.
- `<!NOTATION GIF SYSTEM “gifmagic.exe”>`
- `<!NOTATION ISODATE SYSTEM “http://www.iso.ch/date_specification”>`

Entities

- XML allows flexible organization of document text by using entities.
- Entities allow a document to be broken up into multiple storage objects.
- Entities allow us to reuse and maintain text easily.
  - `<!ENTITY dtd “document type definition”>`
General & Parameter Entities

- There are two kinds of entities: general entities and parameter entities
- We declare an entity in a DTD and then refer to it in by reference in an XML document
- General entity references start with & and end with ;
- Parameter entity references start with % and end with ;

Internal & External Entities

- Entities can be internal and external
- An internal entity is defined completely inside the XML document that references it
  - The document itself is considered an entity
- An external entity derives their content from
  - an external source, such as a binary file, and
  - a reference to them usually includes a uniform resource identifier (URI) at which they can be found
Parsed & Unparsed Entities

- Entities can also be parsed or unparsed.
- The content of parsed entities is well-formed XML text.
- Unparsed entities hold data that we do not want to parse, such as binary data.

Internal General Entities

- Internal parsed general entities are the simplest type of entity.
- They are abbreviations defined in the DTD of the XML document.
- All internal general entities are parsed entities.
  - `<!ENTITY xml "Extensible Markup Language">`
  - `<course>&xml; for Enterprise Data Management</course>`
External General Entities (1/2)

- Entities can also be external, which means we should provide a URI directing the XML processor to the entity.
- External entities can be simple both a text file and a binary file.

External General Entities (2/2)

- When an entity refers to a text file, its content is inserted at the point of reference and parsed as part of the document.
- When an entity refers to a binary file, its content is not parsed. It may only be referenced in an attribute.
External General Parsed Entities

- The parsed entities are text files
- In DTD:
  
  ```xml
  <!ENTITY section1 SYSTEM "path/to/section1.xml">
  ```

- In XML:
  
  ```xml
  <?xml version="1.0" encoding="UTF-8"?>
  <document>
    &section1;
  </document>
  ```

External General Unparsed Entities

- The unparsed entities are non-XML files, such as a sound or image file, so you can refer to such files in your document
- When we want to declare external general unparsed entities, we use NOTATION declaration
Notation Declarations

- The general form of a notation can be either of these:
  - `<!NOTATION name PUBLIC std>`
  - `<!NOTATION name SYSTEM URL>`
- Where
  - name is the name you are giving to the notation
  - Std is the published name of a public notation
  - URL is the a reference to a program that can render a file

Connecting an Attribute to a Notation

1. Declare the notation
   ```xml
   <!NOTATION PICTURE PUBLIC "gif/jpeg/tiff/bmp">
   ```
2. Declare the entity
   ```xml
   <!ENTITY watch1 SYSTEM "watch.jpg" NDATA PICTURE>
   ```
3. Declare the attribute
   ```xml
   <!ATTLIST display object ENTITY #REQUIRED>
   ```
4. Use the attribute in an XML document
   ```xml
   <display object="watch1"></display>
   ```
Parameter Entities

- Parameter entities can be used only in the DTD.
- This entity is declared with a syntax similar to that of general entities, but it has a percent sign between the string <!ENTITY and the entity’s name.

DTD with Parameter Entities Example 1

- File “br.xml”

```xml
<?xml version="1.0"?>
<!DOCTYPE br [
  <!ENTITY % br "<!ELEMENT br EMPTY>" %br;]>
<br/>
```
DTD with Parameter Entities Example2

- File “students.dtd”
  ```
  <!ELEMENT นักศึกษา (ปี1|ปี2|ปี3|ปี4)*>
  <!ENTITY % ข้อมูล "ชื่อ,นามสกุล?,เกรดเฉลี่ย,ที่อยู่?, เบอร์โทร?, อีเมล?”>
  <!ELEMENT ปี1 (%ข้อมูล;)>  
  <!ELEMENT ปี2 (%ข้อมูล;)>  
  <!ELEMENT ปี3 (%ข้อมูล;)>  
  <!ELEMENT ปี4 (%ข้อมูล;)>  
  <!ELEMENT ชื่อ (#PCDATA)>  
  <!ELEMENT เกณฑ์เฉลี่ย (#PCDATA)>  
  ```

Internal DTD

- The content of DTD is part of the XML file
  ```
  <?xml version="1.0"?>
  <!DOCTYPE br [  
  <!ENTITY % br "<!ELEMENT br EMPTY>"%br;]>  
  <br/>
  ```
External DTD

- The actual DTD is stored in an external file (usually with the extension .dtd)

```xml
<?xml version="1.0" encoding="tis-620" ?>
<!DOCTYPE นักศึกษา SYSTEM "students.dtd">
<นักศึกษา>
  <ปี2><ชื่อ>ชัดเจน</ชื่อ><เกรดเฉลี่ย>3.90</เกรดเฉลี่ย></ปี2>
  <ปี1><ชื่อ>แปง</ชื่อ><เกรดเฉลี่ย>3.80</เกรดเฉลี่ย></ปี1>
</นักศึกษา>
```

Summary

- Document type definition is for creating a document type to specify the structure of an XML document

- Each document type must have
  - Document type declaration
    ```xml
    <!DOCTYPE ...>
    ```
  - Element type declaration
    ```xml
    <!ELEMENT ...>
    ```
  - Attribute-list declaration
    ```xml
    <!ATTLIST ...>
    ```
References

- Guide to the W3C XML Specification DTD, Version 2.1
- W3Schools DTD Tutorial
  [http://www.w3schools.com/dtd/default.asp](http://www.w3schools.com/dtd/default.asp)
- ZVON DTD Tutorial