ITB295/ITN295 XML
Lecture Notes on Document Type Definition

1 Introduction

1.1 Topics

In these notes, we focus on the *structure* of XML documents. In particular, we discuss how to:

- Define the overall structure of an XML document (DTDs).
- Describe the allowable content of an element.
- Make use of existing DTDs by reference.
- Define different kinds of attributes.
- Identify individual elements.
- Cross reference elements.
- Make use of entities to build a document.

Reference:

✓ `http://www.w3.org/TR/REC-xml`

2 Defining the phonebook structure

2.1 The QUT phonebook

Consider, again, the following extract from the QUT phone book:

```
Edgar   Miss Pam   Optometry           KG B501 35695
Edmond  Dr   David Information Systems GP S842 32240
Edmonds Dr   Ian   Physical Sciences   GP M206 32584
```

Here is one way of encoding this document in XML:

```
<Phonebook>
  <Entry>
    <LastName Title="Miss">Edgar</LastName>
    <FirstName>Pam</FirstName>
    <School>Optometry</School>
    <Campus>GP</Campus>
    <Room>B501</Room>
    <Extension>35695</Extension>
  </Entry>

  <!-- other <Entry/> elements here -->

</Phonebook>
```

2.2 How would we describe it?

If we had to communicate the *nature* of this document another person – maybe to someone at the other end of a phone line – then what might we say?

1. "Suppose we call the whole thing a `Phonebook`."
2. "A `Phonebook` consists of a collection of `Entrys`."
3. "An `Entry` consists of a `LastName` followed by a `FirstName`, `School`, `Campus`, `Room` and `Extension`."
4. "A `LastName` has a `Title` decoration (attribute)."
5. "A `Title` is either `Mrs` or `Miss` or `Ms` or `Mr` or `Dr` or `Prof`."
6. "A `Campus` is either `GP` or `KG` or `CA`."
7. "An `Extension` is a 5-digit number."
8. "A `Room` is a single upper-case letter followed by three digits."

Example 1: Phonebook DTD

Here is the phonebook with an accompanying definition:

```
<?xml version="1.0"?>
<!DOCTYPE Phonebook [
  <!ELEMENT Phonebook (Entry)+ >
  <!ELEMENT Entry (LastName, FirstName, School,
                               Campus, Room, Extension)>
  <!ELEMENT LastName  (#PCDATA)>
  <!ELEMENT FirstName (#PCDATA)>
  <!ELEMENT School    (#PCDATA)>
  <!ELEMENT Campus    (#PCDATA)>
  <!ELEMENT Room      (#PCDATA)>
  <!ELEMENT Extension (#PCDATA)>
  <!ATTLIST LastName
   Title (Miss | Ms | Mrs | Mr | Dr | Prof) #REQUIRED>
]>
<Phonebook>
   <Entry>
      <LastName Title="Miss">Edgar</LastName>
      <FirstName>Pam</FirstName>
      <School>Optometry</School>
      <Campus>GP</Campus>
      <Room>B501</Room>
      <Extension>35695</Extension>
   </Entry>

  <!-- other <Entry/> elements here -->

</Phonebook>
```

2.3 Success?

Which of the requirements that we communicated down the phone line have been expressed in this DTD?

1. Have we given the whole thing a name?
2. Have we said that a `Phonebook` consists of a collection of `Entry`s?
3. Have we said that an `Entry` consists of a `LastName` followed by a `FirstName`, . . . ?
4. Have we mentioned that the `LastName` has a `Title` attribute?
5. Have we enumerated the allowable titles?
6. Have we said that a `Campus` is either `GP` or `KG` or `CA`?
7. Have we said that an `Extension` is a five digit number?
8. Have we said that a `Room` is a single upper-case letter followed by three digits?

Example 2: External DTDs

Here we refer to an externally defined DTD:

```
<?xml version="1.0"?>
<!DOCTYPE Phonebook SYSTEM "Phonebook.dtd">
<Phonebook>
  <Entry>
    <LastName Title="Miss">Edgar</LastName>
:
:
```

- The XML parser should use the file following the word `SYSTEM`.
- Rather than a local file, a URL is more likely.

.

2.4 The DOCTYPE

The relevant production rule is:

```
[28] doctypedecl ::= '<!DOCTYPE' S Name
                     (S ExternalID)? S?
                     ('[' intSubset ']' S?)?
                     '>'
```

- The Name in the document type declaration *must* match the element type of the root element.
- The ExternalId is a reference to an externally stored DTD.
- The intSubset is an internal DTD – one that appears within the XML document itself.
- Can you have both an external and an internal DTD in the same document?

3 Defining elements

3.1 Allowable element content

The relevant production rules are:

```
[45] elementdecl ::= '<!ELEMENT' S Name S contentspec S? '>'
[46] contentspec ::= 'EMPTY' | 'ANY' | Mixed | children
```

'EMPTY'	When no content is to be allowed.
'ANY'	Allows any kind of content – as long as all child elements are used as defined.
Mixed	When the element may contain text and possibly other elements.
children	When the element consists *entirely* of other (child) elements.

3.2 ANY and EMPTY

Example 3: ANY

```
<!ELEMENT myHomePage ANY>
```

Example 4: EMPTY

```
<!ELEMENT NewLine EMPTY>

<NewLine/>
```

3.3 Mixed

The production rule for mixed content is:

```
[51] Mixed ::= '(' S? '#PCDATA' (S? '|' S? Name)* S? ')*'
             | '(' S? '#PCDATA' S? ')'
```

Example 5:

```
<!ELEMENT publisher (#PCDATA)>
<!ELEMENT year (#PCDATA)>
```

```
<publisher>Penguin</publisher>
<year>2000</year>
```

Example 6:

```
<!ELEMENT message (#PCDATA | bold | italic)*>
<!ELEMENT bold    (#PCDATA)>
<!ELEMENT italic  (#PCDATA)>
```

```
<message>
You really <bold>must</bold> try this
delicious <bold>new</bold> recipe
 for <italic>sticky date pudding</italic>.
</message>
```

3.4 children

The production rules for elements with only children are:

```
[47] children ::= (choice | seq) ('?' | '*' | '+')?
[48] cp       ::= (Name | choice | seq) ('?' | '*' | '+')?
[49] choice   ::= '(' S? cp ( S? '|' S? cp )+ S? ')'
[50] seq      ::= '(' S? cp ( S? ',' S? cp )* S? ')'
```

The non-terminal symbol `cp` represents any "content particle".

Example 7:

```
<!ELEMENT book (author+, publisher)>
```

A `book` element consists of one or more `author` children followed by a `publisher` element.

```
<book>
<author>Delia Smith</author>
<author>Tom Jones</author>
<publisher>Penguin</publisher>
</book>
```

Example 8:

```
<!ELEMENT goldMedalist (firstName, lastName, country)>
```

```
<goldMedalist>
  <firstName>Jodie</firstName>
  <lastName>Henry</lastName>
  <country>Australia</country>
</goldMedalist>
```

Example 9:

```
<!ELEMENT shopping (item)*>
```

```
<shopping>
<item>Apple juice</item>
<item>Sliced bread</item>
</shopping>
```

Example 10:

```
<!ELEMENT RSVP (yes | no )>
<!ELEMENT yes EMPTY>
<!ELEMENT no EMPTY>
```

This could appear in a document as:

```
<RSVP>
<yes/>
</RSVP>
```

It *cannot* appear in a document as:

```
<RSVP>yes</RSVP>
```

4 Defining attributes

4.1 Use of attribute declarations

The relevant productions are:

```
[52] AttlistDecl ::= '<!ATTLIST' S Name AttDef* S? '>'
[53] AttDef      ::= S Name S AttType S DefaultDecl
```

- In production 52, the `Name` is the name of the element whose attributes are being defined.
- In production 53, the `Name` is the name of an attribute being defined.

4.2 Type constraints

The kinds of values an attributes may take are:

- `CDATA`
- `ID`, `IDREF` and `IDREFS`
- `ENTITY` and `ENTITIES`
- `NMTOKEN` and `NMTOKENS`
- enumeration

4.3 Default values

To provide a value for an attribute, if appropriate and if the attribute is optional.

```
[60] DefaultDecl ::= '#REQUIRED'
                   | '#IMPLIED'
                   | (('#FIXED' S)? AttValue)
```

There are four options:

`#REQUIRED`	On every occasion, the associated element must have this attribute.
`#IMPLIED`	The attribute is optional and no default value is supplied.
`AttValue`	This is the default value for the attribute.
`#FIXED AttValue`	The attribute, if present, must have the associated value.

Example 11:

A rectangle must have height and width attributes, and may also have a title.

```
<!ELEMENT Rectangle EMPTY>
<!ATTLIST Rectangle
          Height CDATA #REQUIRED
          Width  CDATA #REQUIRED
          Title  CDATA #IMPLIED>
```

Example 12:

The default postcode in an address is to be 4001.

```
<!ATTLIST address
          postcode CDATA #FIXED "4001">
```

This is OK:

```
<address>2 George St</address>
```

This is OK:

```
<address postcode="4001">2 George St</address>
```

This will be rejected:

```
<address postcode="4000">2 George St</address>
```

Example 13:

A last name must have an associated title, which must be one of six enumerated values.

```
<!ATTLIST LastName
          Title (Miss | Ms | Mrs | Mr | Dr | Prof)
          #REQUIRED>
```

4.4 ID, IDREF and IDREFS

Consider the following document:

```
<msg>
<text>
Hello <friends names="cheeky danny susie jackie"/>,

Yesterday I saw <friend name="jimbo"/> at the Napoleon.
He said that last week he met <friends names="macca danny"/>
at the footy, and later on, bumped into his old mate
<friend name="robbie"/> at the cricket.

</text>
<matelist>
<mate nick="cheeky">Sean Smith</mate>
<mate nick="danny">Daniel Mackay</mate>
<mate nick="jackie">Jacqueline Wong</mate>
<mate nick="jimbo">James Mason</mate>
<mate nick="macca">Ian McDonald</mate>
<mate nick="robbie">Rob Wood</mate>
<mate nick="susie">Susan Wood</mate>
</matelist>
</msg>
```

Example 14:

We can write the following DTD:

```
<!DOCTYPE msg [
 <!ELEMENT msg (text, matelist)>
 <!ELEMENT text (#PCDATA|friend|friends)*>
 <!ELEMENT friend EMPTY>
 <!ELEMENT friends EMPTY>
 <!ATTLIST friend name IDREF #REQUIRED>
 <!ATTLIST friends names IDREFS #REQUIRED>
 <!ELEMENT matelist (mate)*>
 <!ELEMENT mate (#PCDATA)>
 <!ATTLIST mate nick ID #REQUIRED>
]>
```

5 Exercises

Example 15:

We expressed an SQL query in XML form as:

```
<sql>
  <select order="Cost">
  <col>CarNr</col>
  <col>Make</col>
  <col>Cost</col>
  </select>
 <from>
  <table>Cars</table>
 </from>
</sql>
```

Complete the following DTD:

```
<!DOCTYPE sql [
<!ELEMENT sql (.............., ..............)>
<!ELEMENT select (col+)>
<!ATTLIST ............................ CDATA #IMPLIED>
<!ELEMENT col (#PCDATA)>
<!ELEMENT from (table+)>
<!ELEMENT table (#PCDATA)>]>
```

5.1 Hot Gossip

```
The Hot Gossip Report
...............................
Name   Contact

Ann    22 Strand Bvd, Copenhagen
Bill   3391 1615
Sue    8223 2555
Doug   3 Via Appia, Rome
...............................
```

Example 16:

Suppose that the document encoding begins:

```
<HotGossip>
 <Friend>Ann</Friend>
  <Contact>
   <Address>22 Strand Bvd, Copenhagen</Address>
  </Contact>
 <Friend>Bill</Friend>
  <Contact>
   <PhoneNr>3391 1615</PhoneNr>
  </Contact>
```

Complete the DTD for the Hot Gossip report:

```
<!DOCTYPE .............. [
  <!ELEMENT HotGossip (Friend, Contact)*>
  <!ELEMENT Friend (#PCDATA)>
  <!ELEMENT Contact (PhoneNr..............Address)>
  <!ELEMENT PhoneNr (#PCDATA)>
  <!ELEMENT Address (#PCDATA)>
]>
```

6 Entities

6.1 Entities

- Internal entities are defined within the document; external entities are defined in a separate file.
- General entities contain fragments of XML data; parameter entities contain fragments of DTDs.
- Parsed entities are processed by the parser; unparsed entities are not.

Example 17: Internal entities

Sometimes the same piece of text appears in a number of different places in a document:

```
<!DOCTYPE book [
  <!ELEMENT book (title, author+)>
  <!ATTLIST book copyright CDATA #REQUIRED>
  <!ELEMENT title (#PCDATA)>
  <!ELEMENT author (#PCDATA)>
  <!ENTITY marky "Mark Jackson">
]>
<book copyright="&marky;">
 <title>&marky;: my words</title>
 <author>&marky;</author>
 <author>Sue Hacker</author>
</book>
```

The entity is used in an attribute and in two elements, and the resulting document looks like this:

```
<book copyright="Mark Jackson">
 <title>Mark Jackson: my words</title>
 <author>Mark Jackson</author>
 <author>Sue Hacker</author>
</book>
```

6.2 Predefined entities

XML offers five predefined internal entities:

Entity References	Character
`&lt;`	<
`&amp;`	&
`&gt;`	>
`&quot;`	"
`&apos;`	'

Example 18: Referencing an entity

In the XML source:

```
<concert>U2 &amp; Friends </concert>
```

The result:

```
<concert>U2 & Friends </concert>
```

Example 19: General external entities

Sometimes it might be desirable to construct a document from several (other) files:

```
<!DOCTYPE sql [
<!ELEMENT sql (select, from)>
<!ELEMENT select (col+)>
<!ATTLIST select order CDATA #REQUIRED>
<!ELEMENT col (#PCDATA)>
<!ELEMENT from (table+)>
<!ELEMENT table (#PCDATA)>
<!ENTITY select SYSTEM "select.xml">
<!ENTITY from   SYSTEM "from.xml">]>
<sql>&select;&from;</sql>
```

Where the file `select.xml` contains the following:

```
<select order="cost">
<col>CarNr</col>
<col>Make</col>
<col>Cost</col>
</select>
```

And the file `from.xml` contains:

```
<from>
<table>Cars</table>
</from>
```

Example 20: Parameter entities

It might be a good idea to fragment the DTD in the same way that the document content is partitioned:

```
<!DOCTYPE sql [
<!ELEMENT sql (select, from)>
<!ENTITY % seldef SYSTEM "select.dtd">
%seldef;
<!ENTITY % fromdef SYSTEM "from.dtd">
%fromdef;
<!ENTITY select SYSTEM "select.xml">
<!ENTITY from   SYSTEM "from.xml">]>
<sql>&select;&from;</sql>
```

Where the select syntax is defined in a file `select.dtd` as:

```
<!ELEMENT select (col+)>
<!ATTLIST select order CDATA #REQUIRED>
<!ELEMENT col (#PCDATA)>
```

and the from syntax is defined in `from.dtd` as:

```
<!ELEMENT from (table+)>
<!ELEMENT table (#PCDATA)>
```

Example 21: Unparsed entities

Non-xml data can be incorporated into a document by means of “unparsed” entities:

```
<!-- Example of two unparsed entities -->
<!DOCTYPE gallery [
  <!NOTATION jpeg SYSTEM "jpg">
  <!ENTITY straddiePic SYSTEM "straddie.jpg" NDATA jpeg>
  <!ENTITY cooktownPic SYSTEM "cooktown.jpg" NDATA jpeg>
  <!ELEMENT gallery (title, place, date)+>
  <!ELEMENT title (#PCDATA)>
  <!ATTLIST title entityref ENTITY #REQUIRED>
  <!ELEMENT place (#PCDATA)>
  <!ELEMENT date (#PCDATA)>
]>
<gallery>
<title entityref="straddiePic">A bit of heaven.</title>
<place>North Stradbroke Island: Deadman’s Beach</place>
<date>Jan 2003</date>
<title entityref="cooktownPic">A sign of the times.</title>
<place>North Queensland: Cooktown</place>
<date>Nov 2002</date>
</gallery>
```

Any application using this document is expected to be able to deal with these entities.

7 Conclusions

7.1 Shortcomings of DTDs

- No data typing
- No reuse
- Not in XML format
- No allowance for namespaces

7.2 This week’s topics

This week we looked at a notation for defining:

- The structure of elements.
- The specification of an element’s attributes.
- The identification and cross-referencing of elements.

We also looked at the physical make up of a document:

- General and parameter entities
- Unparsed entities

7.3 Next week: XML Schema

To balance our discussion of the DTD notation, consider this snippet of XML Schema:

```
<xsd:attribute name="Title" use="required">
    <xsd:simpleType>
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="Miss"/>
            <xsd:enumeration value="Ms"/>
            <xsd:enumeration value="Mrs"/>
            <xsd:enumeration value="Mr"/>
            <xsd:enumeration value="Dr"/>
            <xsd:enumeration value="Prof"/>
        </xsd:restriction>
    </xsd:simpleType>
</xsd:attribute>
```

Now compare it to the DTD version:

```
  <!ATTLIST LastName
   Title (Miss | Ms | Mrs | Mr | Dr | Prof) #REQUIRED>
```