XML: the refresher

Lou Burnard
July 2000

Topics

- A smidgeon of theory
- The rules of the game
- Are you well formed?
- Making the rules
- Are you valid?
- What use is a DTD?

Electronic Texts in Humanities Research

- the works we study are more than simply sequences of glyphs
- they have **structure** and **context**
- and they also have multiple readings
- **encoding** or **markup** provides a means of making such readings explicit
- only that which is explicit can be digitally processed

Are the needs of the sciences really that different?

XML: what it is and why you should care

- XML is **structured data** represented as strings of text
- XML looks like HTML, except that:-
  - XML is **extensible**
  - XML must be **well-formed**
  - XML can be **validated**
- XML is application-, platform-, and vendor-independent
- XML empowers the **content provider** and facilitates data integration

A smidgeon of theory

materia appetit formam, ut virum foemina

materia bytes, chars, strings, numbers, dates, part-numbers...

forma records, tables, trees

Two XML principles:

- content and structure are separated from processing
- content is self-describing

XML terminology

An XML document contains:-

- elements, possibly bearing attributes
- processing instructions
- entity references
- CDATA sections

An XML document must be *well-formed* and may be *valid*

The rules of the XML Game

- An XML document represents a (kind of) *tree*
- It has a single *root* and many nodes
- Each node can be
  - a subtree
  - a single *element* (possibly bearing some *attributes*)
  - a string of *character data*
- Each element has a type or *generic identifier*
- Attribute names are predefined for a given element; values can also be constrained

Representing an XML tree

- An XML document is encoded as a linear string of characters
- It begins with a special *processing instruction*
- Element occurrences are marked by *start-* and *end-tags*
- The characters < and & are Magic and must always be "escaped"
- *Comments* are delimited by <!– and –>
- *CDATA sections* are delimited by <![CDATA[and]]>
- Attribute name/value pairs are supplied on the start-tag and may be given in any order
- Entity references are delimited by & and ;

An example XML document

```
<?xml version="1.0" encoding="utf-8" ?>
<cookBook>

  <recipe n="1">
   <head>Nail Soup</head>
   <ingredientList>  .... </ingredientList>
   <procedure>  ....   </procedure>
  </recipe>

  <recipe n="2">
  <!- contents of second recipe here ->
  </recipe>

<!- hic desunt multa ->

</cookBook>
```

XML syntax: the small print

What does it mean to be *well-formed*?

1. there is a single root node containing the whole of an XML document
2. each subtree is properly nested within the root node
3. names are always case sensitive
4. start-tags and end-tags are always mandatory (except that a combined start-and-end tag may be used for empty nodes)
5. attribute values are always quoted

Splot the mistake

```
<greeting>Hello world!</greeting>
<greeting>Hello world!</Greeting>

<greeting><grunt>Ho</grunt> world!</greeting>
<grunt>Ho <greeting>world!</greeting></grunt>
<greeting><grunt>Ho world!</greeting></grunt>

<grunt type=loud>Ho</grunt>
<grunt type="loud"></grunt>

<grunt type= "loud">
<grunt type ="loud"/>
```

Defining the rules

A **valid** XML document will reference a *document type declaration* (DTD) :

```
<!DOCTYPE cookBook SYSTEM "cookbook.dtd">
```

A DTD specifies:

- names for all your elements
- names and default values for their attributes
- rules about how elements can nest
- names for re-usable pieces of data (entities)
- and a few other things

n.b. A DTD does*not* specify anything about what elements "mean"

Defining an element

An element declaration takes the form

```
<!ELEMENT name contentModel >
```

name is the name of the element

contentModel defines valid content for the element

The *content* of an element can be:

- #PCDATA
- EMPTY
- other elements
- *mixed* content combines PCDATA and other elements

Content models

Within a content model:

- *sequence* is indicated by comma
- *alternation* is indicated by |
- *grouping* is indicated by parentheses

Occurrence indicators:

[nothing]	once	?	optionally once	
+	one or more times	*	zero or more times	If

#PCDATA appears in a content model...

- it can only appear once
- it must appear **first**
- if in an alternation, only the * occurrence indicator is allowed

For example...

```
<!ELEMENT a (b+) >
<!ELEMENT b EMPTY>
<!ELEMENT c (#PCDATA)>
<!ELEMENT a (b,c) >
<!ELEMENT a (b|c)* >
<!ELEMENT a (#PCDATA|b|c)* >
<!ELEMENT a (b,  (c|d)*) >
<!ELEMENT a (b?, (c|d)+) >
<!ELEMENT a (b?, (c+|d+)) >
```

Defining an attribute list

An attribute list declaration takes the form

```
<!ATTLIST name attributelist >
```

name is the name of the element bearing these attributes

attributeList is a list of attribute specifications, each containing

- an attribute name
- a declared value
- a default value

For example:

```
<!ATTLIST recipe serves CDATA #REQUIRED
                 id     ID    #IMPLIED
                 tested (yes|no|maybe) "maybe">
```

Defining an attribute list (2)

The range of possibilities is actually rather limited:

declared value can be

- an explicit list e.g. (fish|fowl|herring)
- CDATA
- ID, IDREF, or IDREFS

default value can be

- an explicit value e.g. "fish"
- #IMPLIED
- #REQUIRED
- FIXED

An example DTD

```
<!ELEMENT cookBook (recipe+)>
<!ELEMENT recipe (head?, (ingredi-
entList|procedure|para)*) >
<!ATTLIST recipe serves CDATA #IMPLIED>
<!ELEMENT head (#PCDATA) >
<!ELEMENT ingredientList (ingredient+)>
<!ELEMENT ingredient (#PCDATA|food|quantity)* >
<!ELEMENT procedure (step+) >
<!ELEMENT food (#PCDATA)>
<!ATTLIST food
   type  (veg|prot|fat|sugar|flavour|unspec) "unspec"
   calories (high|medium|low|none|unknown) "unknown" >
<!ELEMENT quantity EMPTY >
<!ATTLIST quantity value CDATA #REQUIRED
                  units CDATA #IMPLIED
                  exact (Y|N) "N">
<!ELEMENT para (#PCDATA|food)*>
<!ELEMENT step (#PCDATA|food)*>
```

Entities

An *entity* is a named sequence of characters, predefined in a DTD for convenience.
Typical uses include:

- to represent characters which cannot reliably be typed in
- as a short cut for boiler plate text
- containers for external (non-XML) data such as graphics
- as a means of abbreviating parts of a DTD (parameter entities)

The Unicode standard includes entity names and encodings for most of the world's writing systems

Entities: some examples

```
<!ENTITY mdash "&#x2014;">
<!ENTITY hcu "Humanities Computing Unit">
<!ENTITY fig1 SYSTEM "fig1.bmp" NDATA BMP>
<!ENTITY % foodTypes
      "(veg|prot|fat|sugar|flavour|unspec)">
```

A parameter entity is one way of changing the range of values permitted for attribute values.

```
<!ATTLIST food type %foodTypes; #IMPLIED>
```

Entity definitions (of whatever kind) in the DTD may be over-ridden in the *DTD subset*:

```
<!DOCTYPE cookBook SYSTEM "cookbook.dtd" [
<!ENTITY % foodTypes "(good|bad|indifferent)">
]>
```

What use is a DTD?

- A DTD is very useful at data preparation time (e.g. to enforce consistency), but redundant at other times
- If a document is well-formed, its DTD can be (almost) entirely recreated from it.
- DTDs don't allow you to specify much by the way of content validation
- Unlike other parts of the XML family, DTDs are not expressed in XML

The XML Schema Language addresses these issues, and may eventually replace the DTD entirely... maybe.

XML: a licence for ill?

XML allows you to make up your own tags, and doesn't require a DTD... isn't that rather dangerous?

- XML allows you to name elements freely
- one man's <p> is another's <para> (or is it?)
- the appearance of interchangeability may be worse than its absence

Namespaces provide a partial solution (but are incompatible with the use of a DTD)

Namespaces

A name space associates a *namespace prefix* with some unique identifier (looks like a URL but isn't)
It is usually defined on the root element of a document (but need not be)

```
<root xmlns:mutt="mutt.co.uk"
            xmlns:jeff="www.jeff.org">
```

The namespace prefix can then be used to distinguish for example

```
<mutt:table> .... </mutt:table>
<jeff:table> .... </jeff:table>
```

An XML processor can be told to process elements from different namespaces differently

Defaulting namespaces

- If no namespace prefix appears in a tagname, it is said to belong to the *default namespace*

```
<jeff:table><!- a jeff type table -></jeff:table>
<table>Some other kind of table</table>
```

- The default namespace may be defined on the root element of the document

```
<root xmlns="mutt.co.uk">
```

DTD : what does it really mean?

- To get the best out of XML, you need two kinds of DTD:
  - document type **declaration**: elements, attributes, entities, notations (syntactic constraints)
  - document type **definition**: usage and meaning constraints on the foregoing
- Published specifications (if you can find them) for XML DTDs usually combine the two, hence they lack modularity

Some typical scenarios

1. Make up your own DTD
   - ... starting from scratch
   - ... by combining components from one or more pre-existing conceptual frameworks (aka **architecture** or **namespace**)
2. Customize a pre-existing DTD
   - **definitions** should be meaningful within a given user community
   - **declarations** should be appropriate to a given set of applications

The TEI is a good candidate for the second approach