

Document markup

XML markup

- Elements
- Attributes
- Entities
- Processing instructions

Motivation

- First we present the ideas behind markup languages in document processing, then
- this lecture will elaborate on the basic XML syntax
- Parts are not essential to all applications, but the standard is small enough to know in its entirety, so that
- you can use documents produced by others
- In addition, we look at what *kinds* of document markup can exist

Elements

- A structural (logical) part of the document
- Separating tags: start and end tags
- Element name
- Element contents:
 - other elements or (and)
 - text

Lecture contents

- Markup language basics
- More detail on XML markup — elements and attributes
- Tree-structure, documents and XML
- Entities and processing instructions
- Different kinds of markup

Example elements

```
<capital>Helsinki</capital>
```

```
<country>
  <cname>Finland</cname>
  <capital>Helsinki</capital>
</country>
```

Document markup

“Traditional markup”

- Text file structure (e.g. CSV files)
- Typesetting systems
 - markup (tags) and application
 - e.g. procedural
 - *Ita This *Rom message is *Bold important
 - *This* message is **important**
 - WYSIWYG
 - macro definitions, a markup tag contains several tags, e.g. TITLE=BOLD, ITALIC, Helvetica 14

Granularity

```
<states>
  <state>
    Washington
    Olympia
  </state>
  <state>
    Washington D.C.
    Washington
  </state>
</states>
```

Granularity

```
<states>
  <state>
    <state_name>Washington</state_name>
    <capital>Olympia</capital>
  </state>
  <state>
    <state_name>Washington D.C.</state_name>
    <capital>Washington</capital>
  </state>
</states>
```

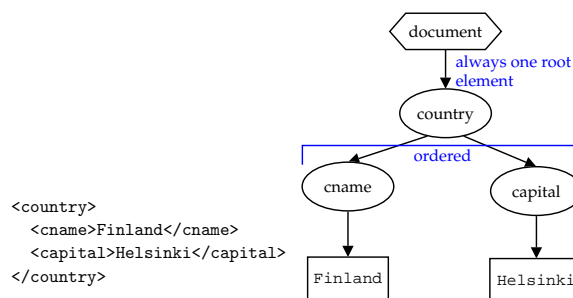
XML and trees

- XML documents represent trees. It is *essential* that we understand what kind of trees they are.
- We'll first see simple documents with simple trees, and add different constructs and look at the resulting trees later.
- The formal definition of the information content of an XML document is given in the XML Information Set recommendation.

Granularity

- The same information can be presented with more or less labeling
- The finer the granularity the more tags
- Fine — coarse
- The finer the granularity the more specific the information
- This affects
 - searching (more information)
 - formatting (more information, possibly more work)
 - authoring (often more work)

Document tree



Element nesting

- Both start and end tags must be present
- Elements must be fully contained within other elements (no overlapping elements)
- Element hierarchy
 - only one root (document element)
 - tree form
- These are the basic requirements of *well-formedness*

Tree terminology

- root, nodes, leaves
- branches, subtrees
- parent — children
- siblings
- ancestors — descendants (grandparents — grandchildren)
- names

Element nesting

```
<small_example>
  <first>this is</first>
  <second>so right</second>
</small_example>

<small_example>
</small_example>
```

Element contents

- An element may contain
 - other elements,
 - text (including CDATA sections and entity references)
 - or both (mixed content)
- As well as
 - comments
 - processing instructions
- An element may also be empty

Empty elements

- The element has no contents
 - `<nothing></nothing>`
 - `<nothing/>`
- Why?
 - the element presence may convey information
 - `<number><one /><zero /></number>`
 - the contents may be linked
 - `<image file="picture.jpg" />`

Element syntax (XML 1.0)

- [3] S ::= (#x20 | #x9 | #xD | #xA)+
- [4] NameChar ::= Letter | Digit | '.' | '-' | '_' | ':' | CombiningChar | Extender
- [5] Name ::= (Letter | '_' | ':') (NameChar)*
- [39] element ::= EmptyElemTag | STag content ETag
- [40] STag ::= '<' Name (S Attribute)* S? '>'
- [42] ETag ::= '</' Name S? '>'

Empty elements

Can be used to represent overlapping trees
(To be avoided? Unless you really have to do this ...)

`<another_small_example>`

`<paragraph>This is <revised_start>`

`a text that has been revised.</paragraph>`

`<paragraph>And the revisions<revised_end>`

`spans several paragraphs.</paragraph>`

`</another_small_example>`

Element syntax (XML 1.0)

- [43] content ::= CharData?
((element | Reference |
CDSect | PI | Comment)
CharData?)*
- [44] EmptyElemTag ::= '<' Name (S Attribute)* S? '/>'

XML syntax

- XML allows whitespace (spaces, tabs, linebreaks) in most places within markup (= tags, declarations, etc.)
- There are a couple of places where whitespace isn't allowed, and you can always deduce them from one fact:
- SGML (whose syntax XML inherits) was designed for parsers with *one character lookahead*: the parser when seeing a character gets to look at most at the next character before having to decide what kind of a token is coming

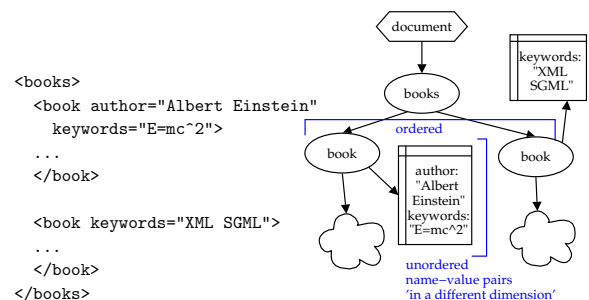
Attributes

- Refines or modifies the meaning of the element or its contents
- Attached to the start tag, each attribute consisting of
 - a name and
 - a value
- The same attribute may appear only once in one element
- Value may contain any literal characters except '<' and '&', these must be represented by character references

XML syntax

- For example, whitespace is not allowed between the '<' and the beginning of the element name. If it were, the parser would not be able to distinguish between opening and closing tags.

Attributes in the tree



Reserved attributes

- Any attribute beginning with 'xml' (in any case) is reserved by the standard
- 'xml:lang' is defined to represent the language of an element
 - ISO 639 language code
 - additional ISO 3166 country code
 - registered IANA language code
 - user-defined

Attribute syntax

- [41] Attribute ::= Name Eq AttValue
- [25] Eq ::= S? '=' S?
- [10] AttValue ::= "' ([^&" | Reference)* "'
| '"' ([^&'] | Reference)* '''
- [67] Reference ::= EntityRef | CharRef

Reserved attributes — xml:lang

```
<product>
  <paragraph xml:lang="en">
    ...
  </paragraph>
  <paragraph xml:lang="fi">
    ...
  </paragraph>
</product>
```

Markup declaration

Contain instructions to a (validating) XML-processor

- Begins with '<!''
- Ends with '>'
- Structure of the document type, element declarations, etc.
- More on these in the next part

Reserved attributes — xml:lang

```
<p xml:lang="en">The quick brown fox jumps
over the lazy dog.</p>

<p xml:lang="en-GB">What colour is it?</p>
<p xml:lang="en-US">What color is it?</p>
<sp who="Faust" desc='leise' xml:lang="de">
  <l>Habe nun, ach! Philosophie,</l>
  <l>Juristerei, und Medizin</l>
  <l>und leider auch Theologie</l>
  <l>durchaus studiert mit heissem Bemühn.</l>
</sp>
```

Comments

Comments are primarily a means for embedding information available in manual editing. Processors *may* provide information about them to the application.

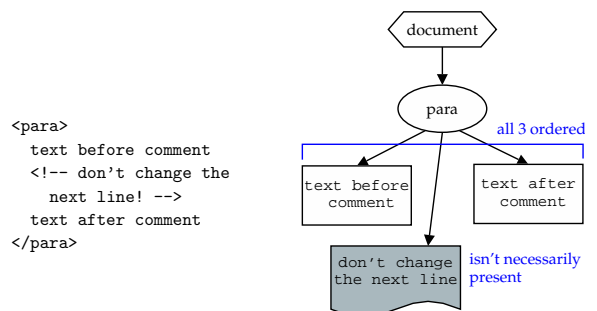
[15] Comment ::= '<!--' ((Char - '-') | ('-' (Char - '-')))* '-->'

<!-- Example comment -->

Reserved attributes

- 'xml:space' is used to signal the intended white-space handling of an element
 - default, or
 - preserve
- XML Namespaces attaches special meaning to names (including attribute names) that contain ':', so ':' shouldn't be used unless using Namespaces

Comments in the tree



Entities

- An entity is a named (physical) object that is used to store information
- Each document has at least one entity, the main document
- E.g. for storing content used multiple times, or
- Dividing a larger document into parts
- Internal entities are defined within the document (or DTD) as named replacement text
- Always specified in the DTD, so we'll first look at the use and later at specification

Entity processing

An XML processor must handle the entities correctly:

- Replace the text (a non-validating parser doesn't always have to expand entities)
- Inform the application about unparsed entities (not included into the document)
- Continue the parsing in the way defined by the standard

Entities

- External entities are stored in a separate physical file (or other locator)
 - Parsed (text) entities are replacement XML text
 - Unparsed (possibly binary) entities may contain any type of (non-XML) data, such as images, sound or video
- *Parameter* entities are used within DTDs, *general* entities within the document instance

Entity reference syntax

[68] EntityRef ::= '&' Name ';' ;

- Somewhat stricter than HTML (SGML), which allowed e.g. '&' if it was followed by whitespace, which was possible with SGML rules

Entity use

If for example the entity named 'univ' contains the string
University of Helsinki

we may write

```
I study at the &univ;.
```

and this will be processed into

```
I study at the University of Helsinki.
```

Syntactic constraints in text

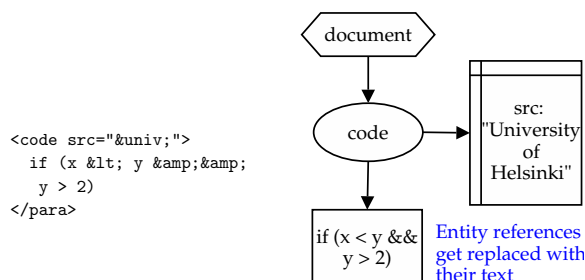
- You cannot use '<', '&' or ']]>' other than to signal markup
- Pre-defined entities '<';', '&';' and '>';', which have to be used instead
- If you are not editing manually, you normally don't have to care about this:
 - user writes '<' or '&'
 - authoring tool transforms to '<';' and '&';'
- This is actually the same as for HTML; historically people have escaped '>' and '"' as well due to buggy implementations

Pre-defined (character) entities

Entity name	corresponding text	from
>	>	greater than
<	<	less than
"	"	quote
'	'	apostrophe
&	&	ampersand

Additionally '&#{number}';' is a character entity reference that is replaced by the Unicode character corresponding to *number*

Entity references in the tree



CDATA sections

- [18] CDsect ::= CDstart CDdata CDEnd
- [19] CDstart ::= '<![CDATA['
- [20] CDdata ::= '(Char* - (Char* '])>' Char*)
- [21] CDEnd ::= '])>'

XML declaration

- Example: `<?xml version="1.0" encoding="ISO-8859-1" standalone='yes'?'>`
- XML version 1.0
- character encoding
- no external entities needed for processing
- If no character encoding given, processor assumes Unicode. Can automatically detect whether UTF-8 or UTF-16 (and endianness of UTF-16).

CDATA sections

- CDATA sections are a way of writing text that contains '<'s and '&'s easier
- Within the CDATA sections all valid characters are accepted and given as-is to the application, except for the closing '])>' delimiter
- **CDATA is just an alternative way of inputting text, and doesn't modify the information content of the document**
- Note that there is no way of having '])>' in the contents, since entity references are not recognized

Processing instructions

- A processing instruction is a command or note that the XML processor presents to the application
- Delimited by '<?' and '?>'
- Used for example to link stylesheets to XML documents
- Considered mostly undesirable in XML except for standardized purposes (≈ don't make your own without a good reason; e.g., interfacing with legacy SGML data)

CDATA sections

- CDATA sections are basically important, when
- manually editing documents, or
- wanting text that contains '<'s and '&'s to be easier to read when looking at the raw document, e.g. when debugging

Syntax of processing instructions

- [16] PI ::= '<?' PITarget (S (Char* - (Char* '?>' Char*)))? '?>'
- [17] PITarget ::= Name - (('X' | 'x') ('M' | 'm') ('L' | 'l'))

- Note that the meaning is not really to syntactically disallow PI targets containing 'xml', but to reserve such names for future standards.
- The syntax allows any string except '?>' in the PI after the target (which has to be a valid XML name). It is totally up to the application how to parse the data.

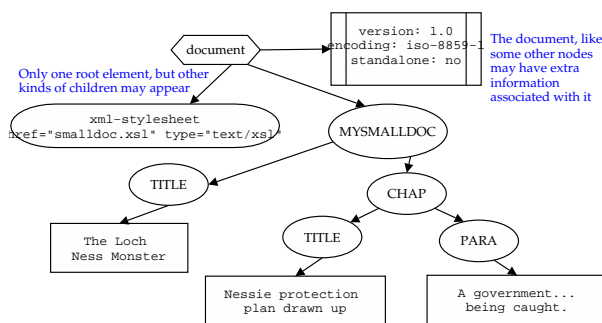
Example CDATA section

```
<webpage>
<![CDATA[
<html>
  <title>A bit of non-wf html
<body>
  <p>This is quite ok HTML, but
  cannot be included in XML as-is
</html>
]]>
</webpage>
```

XML document instance

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet href="smallldoc.xsl" type="text/xsl"?>
<MYSMALLDOC>
  <TITLE>The Loch Ness Monster</TITLE>
  <CHAP>
    <TITLE>Nessie protection plan drawn up</TITLE>
    <PARA>A government agency has drawn up a
      contingency plan to cope with the possibility
      of the Loch Ness Monster being caught.
    </PARA>
  </CHAP>
</MYSMALLDOC>
```

XML document instance as a tree



Content markup

- Meaning of the information in an abstract sense
- Real world objects
- Examples:
 - addresses, street names, zip codes
 - product names, quantities, prices
 - recipes, ingredients, temperatures, cooking times
- Provide information for several kinds of processing
- E.g. a domain-specific DTD, like a parts catalogue

Well-formedness

We have now seen most aspects of XML syntax. A document that follows the XML *syntactic* rules is called *well-formed*.

According to the rules the document

- Contains only one root element, and all elements begin and end within the same element (so that elements do not overlap)
- Follows the restrictions on well-formed documents set by the XML recommendation (many of which we have seen in this lecture)
- Every parsed entity used is well-formed (does not have to have a single root element)

Structural markup

- Represent structure of publishing/exposition/narrative
- Can tell about both content and presentation
- Examples:
 - paragraphs
 - lists and list items
 - chapters, sections, subsections, etc.
 - tables
- Less processing possibilities
- E.g. HTML

Kinds of markup

Presentation markup

- Tell about the intended presentation
- Examples:
 - font families and sizes
 - line and page breaks
 - indentation
 - bold, italic, colours
- Complete control over presentation
- E.g. XSL
- Not preferable as document markup according to the XML "philosophy"

What to mark up

- Content, structure, presentation
- Different ways of looking at the information in a document
- Tell different things by the document, often affect element names:
 - Content: <definition>...
 - Structure: <paragraph>...
 - Presentation: <bold><italics>...

Further literature

- XML recommendation
<http://www.w3.org/TR/REC-xml>
- XML Information Set recommendation
<http://www.w3.org/TR/xml-infoset/>

This lecture in literature

- Bradley: 3, 4
- **or**
- Erik T. Ray: "Markup and Core Concepts", ch 2 of Learning XML: Creating Self-Describing Data, <http://www.oreilly.com/catalog/learnxml/chapter/ch02.html>, upto 'Getting the Most out of Markup'