

Data-centric XML

XML Syntax

What Is XML?

- Extensible Markup Language
- Derived from SGML (Standard Generalized Markup Language)
- Two goals:
 - large-scale electronic publishing
 - exchange of wide variety of data

XML 1.0

- Second edition of “W3C Recommendation”
 - <http://www.w3.org/TR/REC-xml>
- Development started 1996
- First published 1998, second edition 2000, third edition 2004
- XML 1.1 published on 04 February 2004
 - minor changes only, not widely implemented

XML 1.0 Design Goals

- straightforwardly usable over the Internet
- support a wide variety of applications
- compatible with SGML
- readily support writing XML-processing applications
- a minimum number of optional features (ideally none)
- documents should be human-legible and reasonably clear
- XML design should be prepared quickly
- XML spec shall be formal and precise
- terseness of markup is of minimal importance

Basic XML principles

- XML documents are made of storage units called **entities** (both parsed and unparsed data)
- Parsed data: sequence of characters
 - character data
 - markup
- **XML processor** vs. application

XML Terminology

- well-formedness constraint
- validity constraint
- “for compatibility”
 - e.g. “--” is disallowed in comments
- “for interoperability”
 - e.g. at most one attribute-list declaration per element type in a DTD

Documents

- well-formed:
 - matches production <document>
 - meets all well-formedness conditions
 - each parsed entity which is referenced meets well-formedness conditions
- valid: has associated document type declaration, and document complies with DTD constraints

document ::= prolog element Misc*

Characters

- Based on Unicode (ISO/IEC 10646-1993)
- any Unicode character, excluding the surrogate blocks, FFFE, and FFFF

```
Char ::= #x9 | #xA | #xD | [#x20-#xD7FF] |  
[#xE000-#xFFFF] | [#x10000-#x10FFFF]
```

XML Prolog

- [22] prolog ::= XMLDecl? Misc* (doctypeDecl Misc*)?
 - [23] XMLDecl ::= '<?xml' VersionInfo EncodingDecl?
 SDecl? S? '?>'
 - [24] VersionInfo ::= S 'version' Eq
 ("'" VersionNum "'" | '"' VersionNum '"')
 - [25] Eq ::= S? '=' S?
 - [26] VersionNum ::= ([a-zA-Z0-9_.:] | '-')+
 - [27] Misc ::= Comment | PI | S
-
- [3] S ::= (#x20 | #x9 | #xD | #xA)+

Prolog Parameters

[80] EncodingDecl ::= S 'encoding' Eq
 ("" EncName "" | """ EncName """)

[81] EncName ::= [A-Za-z] ([A-Za-z0-9._] | '-')*

- discussed in detail along with character sets presentation

[32] SDDecl ::= S 'standalone' Eq
 (("" ('yes' | 'no') "") | (" ('yes' | 'no') "))

- VC: must be “no” if external DTD subset
- discussed in detail along with DTDs

Elements

Primary means of storing information in XML documents

[39] element ::= EmptyElemTag | STag content ETag

- Well-formedness constraint: Name in start-tag and end-tag must match
- validity constraint: Element must be valid

[44] EmptyElemTag ::= '<' Name (S Attribute)* S? '/>'

[40] STag ::= '<' Name (S Attribute)* S? '>'

- Well-formedness constraints: Attributes must be unique

[42] ETag ::= '</' Name S? '>'

Names

- [4] NameChar ::= Letter | Digit | '.' | '-' | '_' | ':' | CombiningChar | Extender
- [5] Name ::= (Letter | '_' | ':') (NameChar)*
- [6] Names ::= Name (S Name)*
- [7] Nmtoken ::= (NameChar)+
- [8] Nmtokens ::= Nmtoken (S Nmtoken)*

- Names beginning with ‘xml’ or ((‘X’|‘x’) (‘M’|‘m’) (‘L’|‘l’)) are reserved
- Names are case-sensitive

Attributes

- Associate key/value pairs with an element

[41] Attribute ::= Name Eq AttValue

- Validity constraint: attribute must have been declared in DTD
- Well-formedness constraint: attributes must not contain external entity references (directly or indirectly)
- Well-formedness constraint: attributes must not contain "<"

[10] AttValue ::= "" ([^<&"] | Reference)* "" |
 "" ([^<&"] | Reference)* ""

Element Content

```
[43] content ::= CharData? ((element | Reference  
| CDSEct | PI | Comment) CharData?)*
```

- Using elements inside content allows to **nest** elements, forming a **tree**
 - elements thus have a **parent-child** relationship
 - the outer-most element is called the **root** element
- CharData are not further interpreted in XML (contrast XML Schema)
- using only elements in content gives **element** content
- combining both markup and character data in content gives **mixed** content
 - often avoided in data-oriented XML to simplify processing
- No content: empty element
 - can be represented as EmptyElemTag as well

Character Data

[14] CharData ::= [^<&]* - ([^<&]* ']>' [^<&]*)

- ‘&’, ‘<’ reserved exclusively for markup
 - usage allowed inside comments, processing instructions, or CDATA sections
 - escape with & or <
 - alternatively escape with #38; or #60
 - alternatively escape with #x26; or #x3c;
- ‘>’ can be escaped with >
 - for compatibility, must be escaped when appearing as part of the string ‘]]>’

CDATA sections

- used to represent “literal” text, mostly in document-oriented processing

[18] CDSEct ::= CDStart CData CDEnd

[19] CDStart ::= '<![CDATA['

[20] CData ::= (Char* - (Char* ']'])>' Char*))

[21] CDEnd ::= '']]>'

- only CDEnd is markup
- CDATA section cannot nest

References

[67] Reference ::= EntityRef | CharRef

[66] CharRef ::= '&#' [0-9]+ ';' | '&#x' [0-9a-fA-F]+ ';'</p></div><div data-bbox="47 361 967 408" data-label="List-Group">• WFC: only valid characters (matching Char) can be declared</div><div data-bbox="73 424 583 471" data-label="Text"><p>[68] EntityRef ::= '&' Name ';</p></div><div data-bbox="47 482 957 788" data-label="List-Group">• WFC: entity must have been declared (or be predefined) for stand-alone documents– predefined are amp, lt, gt, apos, quot• VC: entity must be declared• WFC: entity name must be a parsed entity• WFC: entity definition must not be recursive</div><div data-bbox="3 746 24 988" data-label="Page-Footer"><p>© 2006 Martin v. Löwis</p></div><div data-bbox="396 967 597 993" data-label="Page-Footer"><p>Datenorientiertes XML</p></div><div data-bbox="964 975 988 996" data-label="Page-Footer"><p>17</p></div>

Comments

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

- for compatibility, -- cannot occur inside a comment
- no markup is recognized except for -->
- allowed nearly anywhere, outside other markup
 - between elements
 - before and after the document element
 - can occur in, but are not part of, character data
- XML processors may, but need not, make comment text available to application

Processing Instructions

[16] PI ::= '<?' PITarget (S
 (Char* - (Char* '?>' Char*)))? '?>'

[17] PITarget ::= Name - (('X' | 'x') ('M' | 'm') ('L' | 'l'))

- allows document producer to pass instructions for document consumer
- not part of the character data, but must be passed to application
- example:
`<?xmlstylesheet href="mystyle.css" type="text/css"?>`
- NOTATIONs can be used to define PITargets formally

White Space Handling

- “significant” and “insignificant” white space
- processor must report all white space that is not markup
- validating processor must also report whether white space is element content or not
- attribute `xml:space` can be used
 - two possible values: `default`, `preserve`
 - unless otherwise specified: root element assumes no intentions wrt. white space handling
- white-space normalization in attributes, based on DTD

End-of-Line Handling

- Multiple line break characters:
 - #xD (carriage return)
 - #xA (line feed)
 - #xD#xA
- XML processor performs normalization
 - transforms #xD#xA into #xA

Language Identification

- attribute `xml:lang` defines the language of an element and all contents within
- nested elements can override language
- language names should follow RFC 1766
 - two-letter language code from ISO 639
 - two-letter country code from ISO 3166
 - additional IANA-registered or user-defined codes