XML

'Specification ' of XML
- Extensible Markup Language (XML) 1.0 -

Youichi Shibata    Synergy Incubate Inc.

XML documents

- Legal XML documents are said to be well-formed having a each logical tree with a single root
- The XML documents with an optional set of constraints (a DTD), they are said to be valid
Well-formed Tag/Attribute Syntax

- Element types and attribute names are case-sensitive
- Empty elements must be marked by closing />
- Attribute values must be quoted (' or ")

Element of XML Documents

- XML Documents are composed of:
  1. prolog
  2. element
  3. Misc

[1] document ::= prolog element Misc*
[27] Misc ::= Comment | PI | S

Synergy Incubate Inc.
> XML Declaration <

1. Specify of Version Number
2. Encoding Declaration
3. Standalone Document Declaration

example: `<?xml version="1.0" encoding="UTF-8" standalone="no" ?>`


Synergy Incubate Inc. √

> Encoding Declaration <

1. UTF-8
2. UTF-16
3. ISO-10646-UCS-2
4. ISO-10646-UCS-4
5. ISO-8859-1
6. ISO-8859-2
7. ...
8. ...
9. ISO-8859-9
10. Shift-JIS
11. EUC-JP

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

Synergy Incubate Inc. √
Standalone Document Declaration

- The value "yes" indicates that there are no markup declarations external to the document entity
- The value "no" indicates that there are or may be such external markup declarations

* Load reduction for XML processors

```plaintext
[32] SDDcl ::= S 'standalone' Eq ("" | ('yes' | 'no') '' | ('yes' | 'no') '"")
```

Synergy Incubate Inc. 🏡

Document Type Declaration

- The XML document type declaration contains or points to markup declarations

```plaintext
[28] doctypedecl ::= '<!DOCTYPE' S Name
(S ExternalID)? S? ('[' (markupdecl|PEReference|S)* ']' S?)? '>'

[75] ExternalID ::= 'SYSTEM' S SystemLiteral
| 'PUBLIC' S PubidLiteral S SystemLiteral

| ('''[^']*'''*'''

[12] PubidLiteral ::= ''' PubidChar* '''
| ''' (PubidChar - '''* '''

[13] PubidChar ::= #x20 | #xD | #xA | [a-zA-Z0-9]
| [\-()+\./:=?;!*#$%]
```

Synergy Incubate Inc. 🏡
› **Markup Declaration**

1. Markup declaration provide a grammar of documents
2. DTD (document type definition) is a grammar of documents
3. Markup declaration declares;
   1. element type declaration
   2. attribute-list declaration
   3. entity declaration
   4. notation declaration
   5. Processing Instruction
   6. Comments

[29] `markupdecl ::= elementdecl | AttlistDecl
     | EntityDecl | NotationDecl | PI | Comment`

Synergy Incubate Inc. 🌐

› **Element**

1. Two types Elements
   1. Empty-Element Tags
   2. Start-Tags, content, End-Tags

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

Synergy Incubate Inc. 🌐
Empty-Element Tags

Empty-element tags may be used for any element which has no content, whether or not it is declared using the keyword EMPTY. For interoperability, the empty-element tag must be used, and can only be used, for elements which are declared EMPTY.

example:
<IMG src="http://www.w3.org/Icons/WWW/w3c_home" />

Start-Tags, Content, End-Tags

The beginning of every non-empty XML element is marked by a start-tag. The Name in the start- and end-tags gives the element's type. The Name-AttValue pairs are referred to as the attribute specifications of the element, with the Name in each pair referred to as the attribute name and the content of the AttValue (the text between the ' or " delimiters) as the attribute value.

example:
<aff>Synergy Incubate Inc.</aff>

Synergy Incubate Inc.
Character data

- All text that is not markup constitutes the character data of the document
- The ampersand character (&) and the left angle bracket (<) may appear in their literal form only when used as markup delimiters, or within a comment, a processing instruction, or a CDATA section
- The right angle bracket (>) may be represented using the string "&gt;", and must, for compatibility, be escaped using "&gt;" or a character reference when it appears in the string "]]>" in content, when that string is not marking the end of a CDATA section

\[14\] \text{CharData} := \{^<\&\}^* - (\{^<\&\}^* \text{'}\text{'}\text{'} [^<\&\}^*)

\text{Synergy Incubate Inc.}

Reference

1. Entity Reference
   - An XML document may consist of one or many storage units. These are called entities
   - An entity reference refers to the content of a named entity. References to parsed general entities use ampersand (&) and semicolon (;) as delimiters
2. Character Reference
   - A character reference refers to a specific character in the ISO/IEC 10646 character set, for example one not directly accessible from available input devices

\[67\] \text{Reference} := \text{EntityRef} | \text{CharRef}
\[68\] \text{EntityRef} := \text{'}\text{'} \text{Name} \text{'}\text{'}
\[66\] \text{CharRef} := \text{'}0-9\text{'}|\text{'\'}|\text{'&\#x'}0-9a-fA-F\text{'}\text{'}\text{'}\text{'}

\text{Synergy Incubate Inc.}
CDATA Sections

- CDATA sections may occur anywhere character data may occur; they are used to escape blocks of text containing characters which would otherwise be recognized as markup. CDATA sections begin with the string "<![CDATA[" and end with the string "]>")

element:
<![CDATA[<greeting>Hello, world!</greeting>]]>

[18] CDsect ::= CDStart CData CEnd
[19] CDStart ::= '<![CDATA[
[20] CData ::= (Char* - (Char* ']')]' Char*)
[21] CEnd ::= ']]>'

Synergy Incubate Inc.

(PI) Processing Instructions

- Processing instructions (PIs) allow documents to contain instructions for applications

element:
<?metaviewer color green ?>

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

Synergy Incubate Inc.
Comments

- Comments may appear anywhere in a document outside other markup; in addition, they may appear within the document type declaration at places allowed by the grammar. They are not part of the document's character data.

equation:
```xml
<!-- declarations for & -->
```

[15], Comment ::= '"!' (Char | \)'

Synergy Incubate Inc.
Element Type Declarations

- The element structure of an XML document may, for validation purposes, be constrained using element type and attribute-list declarations. An element type declaration constrains the element's content.
- Element type declarations often constrain which element types can appear as children of the element.

Element Content

- An element type has element content when elements of that type must contain only child elements (no character data).
- An element is valid if there is a declaration matching elementDecl where the Name matches the element type, and one of the following holds:
  1. The declaration matches EMPTY and the element has no content.
  2. The declaration matches children and the sequence of child elements belongs to the language generated by the regular expression in the content model, with optional white space (characters matching the nonterminal S) between each pair of child elements.
  3. The declaration matches Mixed and the content consists of character data and child elements whose types match names in the content model.
  4. The declaration matches ANY, and the types of any child elements have been declared.

\[45\] \text{elementDecl ::= '<!ELEMENT' Name S contentspec S? ' />'}

\[46\] \text{contentspec ::= 'EMPTY'|'ANY'|Mixed|children}
Attribute-List Declarations

- Attribute-list declarations may be used
  - To define the set of attributes pertaining to a given element type
  - To establish type constraints for these attributes
  - To provide default values for attributes

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

[53] AttDef ::= S Name S AttType S DefaultDecl

Attribute Types

- XML attribute types are of three kinds: a string type, a set of tokenized types, and enumerated types

[54] AttType ::= StringType | TokenizedType
    | EnumeratedType
[55] StringType ::= 'CDATA'
[56] TokenizedType ::= 'ID' | 'IDREF' | 'IDREFS' | 'ENTITY'
     | 'ENTITIES' | 'NMTOKEN' | 'NMTOKENS'
[57] EnumeratedType ::= NotationType | Enumeration
[58] NotationType ::= 'NOTATION' S
     '(' S? Name (S? ')') S? Name)* S? ')
[59] Enumeration ::= '(' S? Nmtoken
    (S? ')' S? Nmtoken)* S? ')'

Synergy Incubate Inc.
Attribute Defaults

An attribute declaration provides information on whether the attribute's presence is required, and if not, how an XML processor should react if a declared attribute is absent in a document.

[60] DefaultDecl ::= '#REQUIRED' | '#IMPLIED' |
| {('#FIXED' $)? AttValue}

Physical Structures

An XML document may consist of one or many storage units. These are called entities; they all have content and are all identified by name. Each XML document has one entity called the document entity:
1. General/Parameter Entities
2. Internal/External Entities
3. Parsed/Unparsed Entities
Entity Declarations

[70] EntityDecl ::= GEDecl | PEDecl
    /* GEDecl: General Entities */
    /* PEDecl: Parameter Entities */

[71] GEDecl ::= '<ENTITY' S Name S EntityDef S? '>

[72] PEDecl ::= '<ENTITY' S '%%' S
              Name S PEdef S? '>

[73] EntityDef ::= EntityValue
                | (ExternalID NDataDecl?)

[74] PEDef ::= EntityValue | ExternalID

Internal/External Entities

Example of an internal entity declaration
<!ENTITY Pub-Status "This is the specification.">

If the entity is not internal, it is an external entity

[75] ExternalID ::= 'SYSTEM' S SystemLiteral
                | 'PUBLIC' S PubidLiteral S SystemLiteral

[76] NDataDecl ::= S 'NDATA' S Name

Examples of external entity declarations:
<!ENTITY open-hatch
    SYSTEM
    "http://www.synergy.co.jp.boilerplate/OpenHatch.xml">
<!ENTITY open-hatch
    PUBLIC
    "-//synergy//TEXT Standard open-hatch boilerplate//EN"
    "http://www.synergy.co.jp.boilerplate/OpenHatch.xml">
<!ENTITY hatch-pic
    SYSTEM "../grafiX/OpenHatch.gif"
    NDATA gif >
\textbf{Parsed/Unparsed Entities}

- Parsed Entities
  
  Text Data, parsed as the ingredients of XML Documents

- Unparsed Entities
  
  Entities that may not be parsed (Graphic and/or Text Data)

\textbf{Entities may be used without declaration}

- \&lt; : < 
- \&gt; : >
- \&amp; : &
- \&apos; : '
- \&quot; : "

Synergy Incubate Inc.
Entity Reference

An entity reference refers to the content of a named entity. References to parsed general entities use ampersand (&) and semicolon (;) as delimiters. Parameter-entity references use percent-sign (%) and semicolon (;) as delimiters.

[67] Reference ::= EntityRef | CharRef
[68] EntityRef ::= '&' Name ';'          {&examples}
[69] PEReference ::= '%' Name ';'       {&examples}

Notation Declarations

Notations identify by name the format of unparsed entities, the format of elements which bear a notation attribute, or the application to which a processing instruction is addressed. Notation declarations provide a name for the notation, for use in entity and attribute-list declarations and in attribute specifications, and an external identifier for the notation which may allow an XML processor or its client application to locate a helper application capable of processing data in the given notation.

[82] NotationDecl ::= '<!NOTATION' S Name S
                       (ExternalID | PublicID) S? '>
[83] PublicID ::= 'PUBLIC' S PubidLiteral

Synergy Incubate Inc.
White Space

S (white space) consists of one or more space (#x20) characters, carriage returns, line feeds, or tab

\[ S ::= (#x20 \mid #x9 \mid #xD \mid #xA) + \]

Language Identification

In document processing, it is often useful to identify the natural or formal language in which the content is written. A special attribute named xml:lang may be inserted in documents to specify the language used in the contents and attribute values of any element in an XML document

\[ \begin{aligned}
\text{LanguageID} & ::= \text{Langcode} ('-' \ Subcode)* \\
\text{Langcode} & ::= \text{ISO639Code} \mid \text{IanaCode} \mid \text{UserCode} \\
\text{ISO639Code} & ::= ([a-z] \mid [A-Z]) (\text{[a-z]} \mid \text{[A-Z]}) \\
\text{IanaCode} & ::= ('i' \mid 'I') '-' ([a-z] \mid [A-Z]) + \\
\text{UserCode} & ::= ('x' \mid 'X') '-' ([a-z] \mid [A-Z]) + \\
\text{Subcode} & ::= ([a-z] \mid [A-Z]) + 
\end{aligned} \]
Declaring Namespaces

. XML namespaces are based on the use of qualified names, which contain a single colon, separating the name into a namespace prefix and the local name. The prefix, which is mapped to a URI, selects a namespace. The combination of the universally-managed URI namespace and the local schema namespace produces names that are guaranteed universally unique.

[1] NamespacePI ::= '<?xml:namespace' (S (PrefixDef | NSDef | SrcDef))+ '?>'
[2] NSDef ::= 'ns' Eq SystemLiteral
[3] SrcDef ::= 'src' Eq SystemLiteral
[4] PrefixDef ::= 'prefix' Eq ('"" NCName "" | '" NCName "")
[5] NCName ::= (Letter | '_') (NCNameChar)*
[6] NCNameChar ::= Letter | Digit | '.' | '-' | '_' | CombiningChar | Extender

Synergy Incubate Inc.