PB138 – Markup Languages

Tomáš Pitner

February 24, 2013

Obsah

Up-to-date Specifications of XML

- Original Specification (W3C Recommendation) XML 1.0 at W3C: `http://www.w3.org/XML/`
- 5th Edition (corrections, updates, no major changes At Extensible Markup Language (XML) 1.0 (Fifth Edition) (`http://www.w3.org/TR/REC-xml`)
- commented version at XML.COM (Annotated XML): `http://www.xml.com/pub/a/axml/axmlintro.html`
- XML 1.1 (Second Edition) (`http://www.w3.org/TR/xml11`) - changes induced by the introduction of *UNICODE 3* , easier *normalization* , the specification of handling procedure for "end of line" characters . XML 1.1 is not bound to specific version of UNICODE, but always on the latest version.

Which version to use?

Which version to use in new applications?
See W3C XML Core Working Group (`http://www.w3.org/XML/Core/#Publications`) for the answer:

- unless writing a parser or a XML-generating app. (editor), use XML 1.0 (backward-compatibility)
- new parsers should "know" XML 1.1

Validity of XML documents

- To repeat: every XML document must be WELL-FORMED.
- New: an XML doc can be VALID – which means a more strict requirements than WELL-FORMEDNESS.
  Usually, the conformance to a *DTD* (Document Type Definition) of the doc is meant by the validity, or
  more recently – conformance with an XML Schema or other schema (RelaxNG, Schematron).

Document Type Definition (DTD)

- Document Type Definition (usage/reference to this definition is then a **Document Type Declaration**).
- Specified in the (core) XML standard 1.0.
- Describes allowed **element content**, **attribute presence and content**, their default values, defines used **entities**.
- DTD might be either **internal** or **external** DTD (*internal and external subset*) or "mixed" – both.
- A document conformant with a DTD is denoted as *valid* ("platný" in Czech).
- DTD and languages for similar purpose are denoted as *modeling languages* – they model/define concrete markups.
- Syntax of DTD *IS NOT* XML (in constrast to XML Schema and many others modeling languages).

Motivation for DTD, comparison, pros and contras

Problems with DTD?

- Fundamental problem of DTD is its incompatibility with XML Namespaces and
- lack of modeling expressiveness – some constructs cannot be constrained by DTD.
- Direct, more powerful, but also more complex modeling language is W3C XML Schema (`http://www.w3.org/XML/Schema`).
- Powerful and simpler alternatives of XML Schema are e.g. RelaxNG (`http://relaxng.org`). (on Wikipedia:RELAX_NG (`http://en.wikipedia.org/wiki/RELAX_NG`))

Why use DTD?

Why use DTD at all?

- Simple. All parsers are fine with it.
- Sufficient for many markups.

DTD - tutorials

- Webreview: `http://www.webreview.com/2000/08_11/developers/08_11_00_2.shtml`
- ZVON: `http://www.zvon.org/xxl/DTDTutorial/General/contents.html`
- XML DTD Tutorial (101): `http://www.xml101.com/dtd/`
- W3Schools DTD Tutorial: http://www.w3schools.com (`http://www.w3school.com`)

DTD in more details / 1

DTD declaration is placed immediately before the root element!

- `<!DOCTYPE root-elt-name External-ID [ internal part of DTD ]>`

Internal orexternal part (*internal or external subset*) might or might not be present, or both can be present.

DTD in more details / 2

External identifier can be either

- `PUBLIC "PUBLIC ID" "URI"` (suitable for "public", generally recognized DTDs) or
- `SYSTEM "URI"` - for private- or other not-that-well established DTDs ("URI" neednot be just real URL on network, may also be a file on (local) filesystem, resolution according to system where it is resolved)

The significancy of internal a external parts is the same (they must not be in conflict - eg. two defeinitions of the same element). DTD contains a list of definitions for individual *elements, list of attributes of them, entities, notations*

DTD - conditional sections

For "commenting out" portions of DTDs e.g. for experimenting.

- `<![IGNORE[ this will be ignored ]]>`
- `<![INCLUDE[ this will be included into DTD (i.e. not ignored)]]>`

DTD - element type definition / 1

Describes allowed content of the element, in form of `<!ELEMENT element-name ... >`, where `...` can be

- `EMPTY` - for empty element which may be represented as `<element/>` or `<element></element>` - the same logical meaning
- `ANY` - any element content allowed, i.e. text nodes, child elements, ...
- may contain **child elements** - `<!ELEMENT element-name (specification of child elements)>`
- may be **mixed** - containing both text and child elements given by enumeration `<!ELEMENT element-name (#PCDATA | specification of child elements)*>`.
- for MIXED: the order or cardinality of concrete child elements cannot be specified.
- The star (*) is required - any cardinality is always allowed.

DTD - element type definition / 2

For specifying the child elements, we use:

- **sequence** operator (sekvence, *follow with*) **,**
- **choice** operator (výběru, *select*, *choice*) **|**
- parenthesis **()** have usual meaning
- various operators CANNOT be combined within a group **, |**
- the child elements cardinality (occurence) can be specified/limited by "star", "question mark", "plus" having usual meaning.
- No specifier means just one occurence allowed.

DTD - attribute definition

Describes (data) type and/or implicit attribute values for the respective element.

```
<!ATTLIST element-name attribute-name
attribute-value-type implicit-value>
```

DTD - definition of attribute value type

Allowed value types are as follows:

- `CDATA`
- `NMTOKEN`
- `NMTOKENS`
- `ID`
- `IDREF`
- `IDREFS`
- `ENTITY`
- `ENTITIES`
- enumeration - eg. `(hodnota1|hodnota2|hodnota3)`
- enumeration of notations - eg. `NOTATION (notace1|notace2|notace3)`

DTD - cardinality of attributes

Attributes may have obligatory presence:

- `#REQUIRED` - attribute is required
- `#IMPLIED` - attribute is optional
- `#FIXED "fixed-value"` - is required and must have the value `fixed-value`

DTD - implicit attribute value

Attribute (incl. optional one) might have an implicit value:

- `"implicit value"` - attribut is optional, but if not present, then the `implicit value` is used instead.

Entity - declaration and usage

We distinguish:

- declaration
- reference (ie. use) of a (declared) entity.

General entities may be

- *parsed* - files with a (well formed) markup,
- *not-parsed* - eg. binary files,
- *character* entities - characters, eg. `>` refers to a char entity.

Parametric entities

- only inside of DTD, somehow similar to "macros" in pg. languages
- suitable eg. for declations of *attribute lists* (if long and multiply used)
- see DTD for HTML 4.01 - `http://www.w3.org/TR/html4/sgml/dtd.html`
- definition of a parametric entity is eg. `<!ENTITY % heading "H1|H2|H3|H4|H5|H6">`

XML Base

- XML Base (second edition), W3C Recommendation 28 Jan 2009: `http://www.w3.org/TR/xmlbase/`
- Standard for evaluation of relative URLs in links to/from XML docs. Facility similar to that of HTML BASE, for defining base URIs for parts of XML documents.
- Defines how to use a reserved attribute `xml:base` denoting the base URI for relative URIs.
- It complements with the *XLink* spec.
- It works based on "overriding" of XML base from parent (ancestor) elements.

XML Base - example

```
  <!--  Slides RelaxNG locations   -->
- <group xml:base="schema/relaxng/" id="slides-relaxng"
    prefer="public">
    <uri name="slides.rng" uri="slides.rng" />
    <uri name="slides-full.rng" uri="slides-full.rng" />
  </group>
```

Note the use of the reserved prefix `xml:`

XML Namespaces (jmenné prostory)

- XML Namespaces (W3C Recommendation, currently *Namespaces in XML 1.0 (Third Edition) W3C Recommendation 8 Dec 2009*): `http://www.w3.org/TR/REC-xml-names`
- to new XML, there exists *Namespaces in XML 1.1 W3C Recommendation* (*http://www.w3.org/TR/xml-names11/*) (Second Edition) 16 August 2006. Andrew Layman, Richard Tobin, Tim Bray, Dave Hollander
- They define logical spaces for names of elements, attributes in XML document.
- They give the elements and attributes the "third dimension".
- To each NS in XML, there is exactly one ("globally") unique identifier, given by URI (URIs is a superset of URLs).
- NS corresponding to an URI does not anyhow relate to content that would potentially be available under the URL

Prefixes and Equivalence of NSs /1

- Instead of URIs for denoting a namespace in document, one uses *prefixes* for these NS mapped to the respective URI using `xmlns:prefix="URI"`.
  Element- or attribute-name containing colon (:) *is denoted as Qualified Name, QName*.
- Two NS are equal iff their URIs are one-to-one-character the same (in UNICODE).
- NS do not apply to text nodes.

Prefixes and Equivalence of NSs /2

- Element/attribute *need not be in a namespace*.
- NS prefix declaration or declaration or the implicit NS recursively applies to all descendants (child elements, their children etc.), unless another declaration "remaps" the given prefix.
- One NS is co-called *implicit* (*default) NS*, declared by attribute `xmlns=`
- Default NSs are NOT applied to attributes!!!, thus attributes without an explicit prefix do not belong to any NS.

Default NS – example

```
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" la
   <body>
      <h1>Hurááááá</h1>
   </body>
</html>
```

Explicit (prefixed) NS – example

```
<xhtml:html xmlns:xhtml="http://www.w3.org/1999/xhtml" xml
   <xhtml:body>
      <xhtml:h1>Hurááááá</xhtml:h1>
   </xhtml:body>
</xhtml:html>
```

Issues related to NS

NS are NOT compatible with DTD.
DTD strictly differentiates between eg. `name xi:include` and `include` even if they belong to the same NS and should thus have the same interpretation/meaning for applications.

XML Information Set (XML Infoset) - goals

- *XML Infoset 2nd Edition W3C Recommendation* First published on 24 October 2001, revised 4 February 2004, John Cowan, Richard Tobin, `http://www.w3.org/TR/xml-infoset/`
- Infoset describes "what all info can we get from a node (element, document, attribute...)"
- In other words: an application should not rely on any other info, such as attribute order etc.
- Any well-formed XML document conformant to XML Namespaces has its Infoset.

XML Infoset - structure

- Infoset comprises of *Information items*
- Infoset relates to document with expanded (resolved) entities
- We distinguish among infoset *of document*, *element*, *attribut*, *character*, PI, *not-expanded entity*, *not-analysed entity*, *notation*.

Canonical Form of XML Document

- Canonical XML Version 1.0, W3C Recommendation 15 March 2001, `http://www.w3.org/TR/xml-c14n`
- The goal of CF is to describe criteria and algorithm how to define equivalence on XML documents that are "logically" the same and expose just differences in physical form (entities, attribute order, char encoding)
- Canonication "wipes-out" differences that are not significant for applications.
- Canonication in inevitable in some important applications , e.g. *electronic signature* of XML data (when calculating *digest*).

Canonical Form - principles /1

Main principles for constructing the canonical form of an XML document:

- encoding in UTF-8
- line breaks (CR, LF) normalized according to the algorithm mentioned in XML 1.0 Spec.
- attribute values normalized
- references to character and parsed entites replaced by their content
- CDATA section also replaced by their content
- prolog "xml" and DTD removed

Canonical Form - principles /2

- whitespaces outside of the root element normalized
- otherwise (except of line breaks), the whitespaces are preserved
- attribute values always in double quotes ”
- special chars in attr. values replaced by refs to character entities
- superflous NS declarations removed
- default attribute values added to all element where relevant
- attributes and NS declarations will be ordered lexikographically

Issues with Canonical Form

Certain information loss (mostly info from DTD):

- not-parsed entity (eg. binary ones) are not accessible anymore after canonicalization
- notations
- attribute types (incl. default values)

API Task

- offer simple standardized XML access
- connect application to the parser and applications together
- XML processing without knowledge of physical document structure (entities)
- effective XML processing.

XML APIs Fundamental Types

- Tree-based API
- Event-based API
- API based on pulling events/elements off the document (Pull API).

Map XML Document to Memory Based Tree Structure

- allows to traverse the entire DOM Tree
- best-known - *Document Object Model* (DOM from W3C, see http://www.w3.org/DOM (`http://www.w3.org/DOM/`))

Programming Language Specific Models

- Java: JDOM - `http://jdom.org`
- Java: `dom4j` - `http://dom4j.org`
- Java: XOM - `http://www.xom.nu`
- Python: 4Suite - `http://4suite.org`
- PHP: SimpleXML - `http://www.php.net/simplexml`

Generate Sequence of Events while parsing the Document

- technical realization - using callback methods
- application implements *handlers* (processing the generated events)
- event-based API:
  - works on lower-level than tree-based
  - application should do more processing
  - saves memory - does not create any persistent objects.

Event Examples

- start document, end document
- start element, end element - contains the attributes as well.
- processing instruction
- comment
- entity reference
- Best-known event-based API - SAX
  `http://www.saxproject.org`

SAX - Document Analysis Example

```
<?xml version="1.0"?>
<doc>
      <para>Hello, world!</para>
<!-- that's all folks -->
      <hr/>
</doc>
```

generates following events:

```
start document
start element: doc {list of attributes: empty}
      start element: para {list of attributes: empty} chara
      end element:
para comment: that's all folks
start element: hr
end element: hr
end element: doc
```

When to use event-based API?

- Easier to parser programmer, more difficult to application programmer.
- No complete document available to application programmer. He must keep the state of analysis him-self.
- Suitable for tasks, that can be solved without the need of entire document.
- The fastest possible processing usually.
- Difficulties while writing applications can be solved using extensions like Streaming Transformations for XML (STX) (`http://stx.sourceforge.net`)

Optional SAX Parser Features

The SAX parser behavior can be controlled using so called *features* a *properties*.

- *For optional SAX parser's features see* `http://www.saxproject.org/?selected=get-set`
- For more details on properties and features see Use properties and features in SAX parsers (???) (IBM DeveloperWorks/XML).

SAX filters

The SAX filters (implementation of org.xml.sax.XMLFilter interface) can be programmed using the SAX API.
Such a class instance accepts input events, process them and sends them to the output.
For more information on event filtering see Change the events output by a SAX stream (`http://www.ibm.com/developerworks/xml/library/x-tipsaxfilter/`) (IBM DeveloperWorks/XML) for example.

Additional SAX References

- Primary source - `http://www.saxproject.org`
- SAX Tutorial on JAXP
  `http://java.sun.com/webservices/reference/tutorials/jaxp/html/sax.html`

Pull-based APIs

- Application does not process incoming events, but it pulls data from the processed file.
- Can be used when programmer knows the structure of an input data and he can pull them off the file.
- ... opposite to event-based API.
- Very comfortable to an application programmer, but implementations are usually slower the push event-based APIs.
- Java offers the *XML-PULL parser API* - see Common API for XML Pull Parsing (`http://www.xmlpull.org/`) and also
- newly develop API - Streaming API for XML (StAX) (`http://www.jcp.org/en/jsr/detail?id=173`) developed like a product of JCP (Java Community Process).

Streaming API for XML (StAX)

The API may become the part of the Java API for XML Processing (JAXP) in the future.
Offers two ways to pull-based processing:

- pulling the events using iterator - more comfortable
- low-level access using so called cursor - faster.

StAX - an Iterator Example

```
import java.io.FileNotFoundException;
import java.io.FileReader;
import javax.xml.namespace.QName;
import javax.xml.stream.XMLInputFactory;
import javax.xml.stream.XMLStreamConstants;
import javax.xml.stream.XMLStreamException;
import javax.xml.stream.XMLStreamReader;

public class ParseByIterator {

    public static void main(String[] args) throws FileNotFo
        // Use reference implementation
        System.setProperty("javax.xml.stream.XMLInputFactor
        XMLInputFactory xmlif = XMLInputFactory.newInstance
        XMLStreamReader xmlr = xmlif.createXMLStreamReader(
        while (xmlr.hasNext()) {
```

StAX - an Cursor Example

```
import java.io.FileNotFoundException;
import java.io.FileReader;
import javax.xml.namespace.QName;
import javax.xml.stream.XMLInputFactory;
import javax.xml.stream.XMLStreamConstants;
import javax.xml.stream.XMLStreamException;
import javax.xml.stream.XMLStreamReader;

public class ParseByIterator {

    public static void main(String[] args) throws FileNotFo
        // Use reference implementation
        System.setProperty("javax.xml.stream.XMLInputFactor
        XMLInputFactory xmlif = XMLInputFactory.newInstance
        XMLStreamReader xmlr = xmlif.createXMLStreamReader(
        while (xmlr.hasNext()) {
```

Basic Interface to Process and Access the Tree Representation of an XML Data

- Three versions of DOM: *DOM Level 1, 2, 3*
- DOM - does not depend on the XML Parsing.
- Described using IDL + API descriptions for particular programming languages (C++, Java, etc.)

HTML Documents Specific DOM

- The HTML Core DOM is more less consolidated with the XML DOM
- Designated to CSS
- Used for dynamic HTML programming (scripting using VB Script, JavaScript, etc)
- Contains the browser environment (windows, history, etc) besides the document model itself.

DOM references

- JAXP Tutorial, part dedicated to the DOM Part III: XML and the Document Object Model (DOM) (`http://java.sun.com/xml/jaxp/dist/1.1/docs/tutorial/dom/index.html`)
- Portal dedicated to the DOM `http://www.oasis-open.org/cover/dom.html`
- DOM 1 Interface visual overview `http://www.xml.com/pub/a/1999/07/dom/index.html`
- Tutorial "Understanding DOM (Level 2)" available athttp://ibm.com/developer/xmlhttp://ibm.com/developer/xml (`http://ibm.com/developer/xml`)

DOM Implementation

- Included in many parsers, the. Xerces (`http://xml.apache.org`) parser for example.
- Part of the JAXP (Java API for XML Processing) - `http://java.sun.com/xml/jaxp/index.html`
- Standalone implementations independent on parsers:
  - `dom4j` - `http://dom4j.org`
  - `EXML` (Electric XML) - `http://www.themindelectric.net`

What do we need?

Native DOM support in the new Java versions (JDK and JRE) - no need of additional library.
Applications need to import needed symbols (interfaces, classes, etc.) mostly from package `org.w3c.dom`.

What will we need often?

Most often used interfaces are:

Element corresponds to the element in a logical document structure. It allows us to access name of the element, names of attributes, child nodes (including textual ones). Useful methods:

- `Node getParentNode()` - returns the parent node
- `String getTextContent()` - returns textual content of the element.
- `NodeList getElementsByTagName(String name)` - returns the list of ancestors (child nodes and their ancestors) with the given name.

Node super interface of `Element`, corresponds to the general node in a logical document structure, may contain element, textual node, comment, etc.

NodeList a list of nodes (a result of calling

Example 1 - creating DOM tree from file

Example of method, reading a DOM tree from an XML file (see Home work 1):

```
import java.io.IOException;
import java.net.URL;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;
import org.w3c.dom.Document;
import org.xml.sax.SAXException;

public class Uloha1 {

    /**
     * Constructor creating new instance of Uloha1 class by
     * on the given URL.
     */
```

Example 2 - DOM tree modification

Example of a method manipulating a document DOM tree (see Homework 1):

```
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.NodeList;

public class Uloha1 {
    Document doc;
    /**
     * ************************************************
     * Method for a salary modification. If the person's sa
     * <code>minimum</code>, the salary will increased to
     * <code>minimum>.
     * No action is performed with the rest of persons.
     */
    public void adjustSalary(double minimum) {
```

Example 3 - storing a DOM tree into an XML file

Example of the method storing a DOM tree into a file (see Homework 1)
The procedure utilizes a *transformation* we do not know yet. Let use it as a black box.

```
import java.io.File;
import java.io.IOException;
import javax.xml.transform.Transformer;
import javax.xml.transform.TransformerConfigurationExceptio
import javax.xml.transform.TransformerException;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;
import org.w3c.dom.Document;

public class Uloha1 {
    Document doc;
```

XML Object Model (XOM)

- XOM (*XML Object Model*) created as an one man project (author Elliote Rusty Harold).
- It is an interface that strictly respect XML data logical model.
- For motivation and specification see the XOM home page (`http://cafeconleche.org/XOM/`).
- You can get there the open-sourceXOM implementation (`http://cafeconleche.org/XOM/xom-1.0d24.zip`) and
- the API documentation (`http://cafeconleche.org/XOM/apidocs/`) too.

Alternative parsers and tree models - NanoXML

- Very small (in the mean of a code size) tree-based interface and parser all in one
- available as open-source at `http://nanoxml.n3.net`
- adopted for mobile devices as well
- *not the best* in the mean of a run-time speed and memory efficiency.

DOM4J - practically good usable tree-based model

- comfortable, fast and memory efficient tree-oriented interface
- designed and optimized for Java
- available as open-source at `http://dom4j.org`
- perfect"cookbook" (`http://dom4j.org/cookbook/cookbook.html`) available
- dom4j is powerful, seetree-based models efficiency comparison (`http://www.ibm.com/developerworks/xml/library/x-injava/`)

Events → tree

- Allow us either to skip or to filter out the "uninteresting" document part using the event monitoring and then
- create memory-based tree from the "interesting" part of a document only and that part process.

Tree → events

- We create an entire document tree (and process it) and
- we go through the tree than and we generate events like while reading the XML file.
- It allows us easy integration of both processing types in a single application.

Virtual object models

- Document DOM model is not memory places, but is created on-demand while accessing particular nodes.
- combines event-based and tree-based processing advantages (speed and comfort)
- Implementation is the `Sablotron` processor for example (see `http://www.xml.com/pub/a/2002/03/13/sablotron.html` or `http://www.gingerall.org/charlie/ga/xml/p_sab.xml`)