PB138 — Fundamental XML Standards

Outline

- XML Specification
- DTD
- XML Base

Specifications and validity of XML

- Original Specification *W3C Recommendation XML 1.0* 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 specification*): http://www.xml.com/axml/testaxml.htm
- *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 XML 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 but do not use it unless needed otherwise

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 contrast to *XML Schema* and many others modeling languages).

Motivation for DTD, comparison

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, http://en.wikipedia.org/wiki/RELAX_NG)

Why use DTD?

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

DTD tutorials

- http://www.zvon.org/xxl/DTDTutorial/General/contents.html (including a Czech version)
- http://edutechwiki.unige.ch/en/DTD_tutorial (not just DTD but much more)
- http://www.w3schools.com

DTDeclaration

DTDeclaration is placed *immediately* before the root element!

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

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

Identifiers in DTDeclaration

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” need not 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

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>`` with 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)>`

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 and any number of occurencies is always allowed.

DTD - element type definition - child elements

For specifying the child elements, we use:

- sequence operator (sekvence, follow with) `,`
- choice operator (výběr, select, choice) `|`
- parenthesis `()` have usual meaning
- Diferrent operators CANNOT be combined within a group!
- The child elements cardinality (occurence) can be specified/limited using the Kleeny operators `?` (0..1), `*` (0..n), `+` (1..n).
- 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. (value1|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:
  - then the attribut is optional, but
  - if not present, then the *implicit value* is used instead.

Physical Structure (Entities)

We distinguish:

- entity **declaration**
- entity **reference** (ie. use) of a (declared) entity, eg. `&gt;`

General entities

- **parsed** files with a (well formed) markup,
- **not-parsed** eg. binary files,
- **character** entities 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

Example from XML Base specification http://www.w3.org/TR/xmlbase/

```xml
<?xml version="1.0"?>
<e1 xml:base="http://example.org/wine/">
   <e2 xml:base="rosé"/>
</e1>
```

In the example below, the base URI of element e2 should be returned as "http://example.org/wine/rosé". [1]

[1] Note the use of the reserved prefix xml