LIS900G

Webmastering III: XML

Thomas Krichel

http://openlib.org/home/krichel

Reading

Castro, Elizabeth (2001) "XML for the World Wide Web", Peachpit Press, 2001

Kay, Michael (2001) "XSLT", Wrox Press

Duckett, Jon et alii (2001) "Professional XML Schemas", Wrox Press

Rusty Harold, Eliotte and W. Scott Moss (2001) "XML in a Nutshell", O'Reilley

Bray, Tim, (1998) "The Annotated XML Specification", http://www.xml.com/axml/testaxml.htm

Call me Thomas

Born in Vöklingen, (Saarland) in 1965, I studied Economics and Social Sciences at the universities of Toulouse, Paris, Exeter and Leicester. Between February 1993 and April 2001 I lectured in the Department of Economics at the University of Surrey. In 1993 I founded NetEc, a consortium of In 1997, I founded the RePEc dataset to document Economics. Between October and December 2000, I held a visiting professorship at Hitotsubashi University.

Homepage http://openlib.org/home/krichel

Email krichel@openlib.org

traditional approach

- resource awareness
- resource usage
- resource evaluation

non-traditional approach

- resource creation
- resource description
- non-resource description (Arms' critique)

Webmastering trilogy: I

The static web site

- verry basic unix
- Character sets
- URIs
- http
- html
- basic configuration of apache
- principles of information architecture.

Webmastering trilogy: II

The dynamic web site

- verry basic unix
- PHP (pretty complete)
- introduction to SQL

Webmastering trilogy: III

XML. In fact, XXX rated

- XML (day 1)
- XSL (day 2 and 3)
- XML Schema (day 4 and 5)

gee. . .

assesment

1. quiz every day 50%
2. xml project
   - build a tiny collection of XML data
   - render with XSLT
   - validate with Schema.

hand in 3 files.

discussion of student background

- What background do you have?
- What are your long-run goal?
- How does this course fit in?

today's course structure

- Stuff that XML builds on
  - Unicode
  - URIs
- XML itself
- Stuff that builds on XML
  - XML namespaces
  - other stuff worth mentioning that is not covered elsewhere

from numbers to bytes.

in fact it is all a little bit more complicated, we have to distinguish three levels

- Abstract Character Repertoire
  the set of characters to be encoded, e.g., some alphabet or symbol set
- Coded Character Set
  a mapping from an abstract character repertoire to a set of non-negative integers
- Character Encoding Scheme
  a mapping from a character set to a serialized sequence of bytes

Coded character sets

Computers don't understand text, they only understand numbers. For computers to be able to treat text, there must be a correspondence between numbers and text characters.

Such a correspondence is called a coded character set.

Important examples are

- 7-bit ASCII
- ISO 8859–1
- cp1252

ISO 10646 and UCS character sets

Universal Character Set, is a coded character set first published in 1993. UCS give a character a number, written U+*hex number* and an official name.

UCS contains the characters required to represent practically all known languages, even the likes of Gurmukhi, Oriya, Telugu, Bopomofo, Runic, Tengwar...

ISO 10646 defines formally a 31-bit character set. The most commonly used characters, have been placed in one of the first 65534 positions (0x0000 to 0xFFFD). This is the basic multilingual plane, aka plane 0. Content fixed since first publication. The UCS characters U+0000 to U+007F are identical to those in US-ASCII (ISO 646 IRV) and the range U+0000 to U+00FF is identical to ISO 8859–1 (Latin–1).

A second part ISO 10646–2 was added in 2001 and defines characters encoded outside the BMP.

Unicode

The Unicode Standard published by the Unicode Consortium corresponds to ISO 10646 at implementation level 3. All characters are at the same positions and have the same names in both standards.

The Unicode Standard defines in addition much more semantics associated with some of the characters and is in general a better reference for implementors of high-quality typographic publishing systems.

Character encoding

The two most obvious encodings store Unicode text as sequences of either 2 or 4 bytes sequences. The official terms for these encodings are UCS–2 and UCS–4 respectively. Of course, UCS–2 can only do plane 0.

Unless otherwise specified, the most significant byte comes first in these (Bigendian convention).

An ASCII or Latin–1 file can be transformed into a UCS-2 file by simply inserting a 0x00 byte in front of every ASCII byte. If we want to have a UCS–4 file, we have to insert three 0x00 bytes instead before every ASCII byte.

Fixed-length Character encoding

The two most obvious encodings for UCS use either 2 or 4 bytes sequences. The official terms for these encodings are UCS–2 and UCS–4 respectively. Of course UCS–2 is limited to plane 0.

An ASCII or Latin–1 file can be transformed into a UCS–2 file by simply inserting a 0x00 byte in front of the original byte. If we want to have a UCS–4 file, we have to insert three 0x00 bytes instead before every ASCII byte.

bigendian and littlendian

Unless otherwise specified , the most significant byte comes first in these (Bigendian convention). The opposite is littleendian.

In order to allow the automatic detection of the byte order, Microsoft want to start every Unicode file with the character U+FEFF (ZERO WIDTH NO-BREAK SPACE), also known as the Byte-Order Mark (BOM). Its byte-swapped equivalent U+FFFE is not a valid Unicode character, therefore it helps to unambiguously distinguish the Bigendian and Littleendian variants. This causes a lot of trouble on other systems.

UTF–8

This is variable-length encoding of UCS. It is the one used by default in XML. It has a number of desirable properties.

ASCII compatility: UCS characters U+0000 to U+007F (ASCII) are encoded simply as bytes 0x00 to 0x7F (ASCII compatibility). This means that files and strings which contain only 7-bit ASCII characters have the same encoding under both ASCII and UTF–8.

All UCS characters higher than U+007F are encoded as a sequence of several bytes, each of which has the most significant bit set. Therefore, no ASCII byte (0x00-0x7F) can appear as part of any other character.

Unicode Transformation Format UTF–8

The first byte of a multibyte sequence that represents a non-ASCII character is always in the range 0xC0 to 0xFD and it indicates how many bytes follow for this character. All further bytes in a multibyte sequence are in the range 0x80 to 0xBF. This allows easy resynchronization and makes the encoding stateless and robust against missing bytes.

UTF–8 encoded characters may theoretically be up to six bytes long, however 16-bit BMP characters are only up to three bytes long.

The sorting order of Bigendian UCS–4 byte strings is preserved. The bytes 0xFE and 0xFF are never used in the UTF–8 encoding.

UTF–8

```
U+00000000 to U+0000007F 0xxxxxxx
U+00000080 to U+000007FF 110xxxxx 10xxxxxx
U+00000800 to U+0000FFFF 1110xxxx 10xxxxxx 10xxxxxx
U+00010000 to U+001FFFFF 11110xxx 10xxxxxx 10xxxxxx
                         10xxxxxx
U+00200000 to U+03FFFFFF 111110xx 10xxxxxx 10xxxxxx
                         10xxxxxx 10xxxxxx
U+04000000 to U+7FFFFFFF 1111110x 10xxxxxx 10xxxxxx
                         10xxxxxx 10xxxxxx 10xxxxxx
```

All 2^{31} UCS chars can be encoded!

UTF–8 examples

The Unicode character U+00A9 = 1010 1001 (copyright sign) is encoded in UTF–8 as

11000010 10101001 = 0xC2 0xA9

The Unicode character U+2260 = 0010 0010 0110 0000 (not equal to) is encoded as:

11100010 10001001 10100000 = 0xE2 0x89 0xA0

This completes the discussion on character encodings.

URI Definition

A Uniform Resource Identifier (URI) is a compact string of characters for identifying an abstract or physical resource. They provide a simple and extensible means for identifying a resource.

There is a set of operations that can be applied to them.

To understand if a given URI instance is valid, we have to study the operations applied to URIs.

uniformity

Uniformity provides several benefits:

- it allows different types of resource identifiers to be used in the same context, even when the mechanisms used to access those resources may differ
- it allows uniform semantic interpretation of common syntactic conventions across different types of resource identifiers
- allows introduction of new types of resource identifiers without interfering with the way that existing identifiers are used;
- it allows the identifiers to be reused in many different contexts, thus permitting new applications or protocols to leverage a pre-existing, large, and widely-used set of resource identifiers.

resource identifier

A resource can be anything that has identity. Not all resources are network "retrievable". The resource is the conceptual mapping to an entity or set of entities, not necessarily the entity which corresponds to that mapping at any particular instance in time.

An identifier is an object that can act as a reference to something that has identity. In the case of URI, the object is a sequence of characters with a restricted syntax.

URI, URL, URN

A URI can be further classified as a locator, a name, or both. The term "Uniform Resource Locator" (URL) refers to the subset of URI that identify resources via a representation of their primary access mechanism (e.g., their network "location"), rather than identifying the resource by name or by some other attribute(s) of that resource. The term "Uniform Resource Name" (URN) refers to the subset of URI that are required to remain globally unique and persistent even when the resource ceases to exist or becomes unavailable.

URN

A URN differs from a URL in that it's primary purpose is persistent labeling of a resource with an identifier. That identifier is drawn from one of a set of defined namespaces, each of which has its own set name structure and assignment procedures. The "urn" scheme has been reserved to establish the requirements for a standardized URN namespace, as defined in "URN Syntax" RFC2141 and its related specifications.

transcribability

The URI syntax was designed with global transcribability as one of its main concerns. A URI may be represented in a variety of ways, e.g. on the back of a napkin. Therefore:

- A URI is a sequence of characters
- A URI may be transcribed from a non-network source, and thus should consist of characters that are most likely to be able to be typed into a computer
- A URI often needs to be remembered by people, and it is easier for people to remember a URI when it consists of meaningful components.
- Therefore it has a restricted set of characters, only US–ASCII.

HTML

```
<HTML><head><TITLE>life and times of
Thomas Krichel</TITLE></head><p><table><tr><td><h2>
The life and times of Thomas Krichel</h2><p> Born in
<a href=http://www.voelklingen.de>V&ouml;klingen</a>,
(<a href=http://www.saarland.de>Saarland</a>)

...

<td><table><tr><td><img src="tk.gif">
</td></tr><tr><td align=center><small><i>I can't cook,
but who cares?</i></small></td></tr></table></td>
</table></tr><p></BODY></HTML>
```

XML

Is a W3C recommendation. It is a new (1998) markup language that will transport a lot of contents over the Internet in the future.

As its level of complexity goes it sits in between HTML and SGML.

HTML, XML, SGML

XML is more flexible than HTML. It allows to define tags.

XML is less flexible than SGML. It does not allow to implement seldomly used features of SGML.

XML has not yet replaced HTML, but it may do so one day.

Design Goals

1. XML shall be straightforwardly usable over the Internet.
2. XML shall support a wide variety of applications.
3. XML shall be compatible with SGML.
4. It shall be easy to write programs which process XML documents.
5. The number of optional features in XML is to be kept to the absolute minimum, ideally zero.
6. XML documents should be human-legible and reasonably clear.
7. The XML design should be prepared quickly.
8. The design of XML shall be formal and concise.
9. XML documents shall be easy to create.
10. Terseness in XML markup is of minimal importance.

XML document structure

prolog

root element

other stuff optional

XML document example

```
<?xml version="1.0"?>

<-- a file to contain sofix data-->

<sofix>

</sofix>
```

well-formed XML documents

- the document must have exactly one top level (root or document) element
- elements must be nested
- each element must have a start and an end tag
- element names are case-sensitive

another XML document example

```
<?xml version="1.0"?>
<!-- break every rule in the book -->}
<sofix> <work> <contributor>
 <name> <role> </name>
 </role> </contrIbutor>
 </work> </work>
</sofix><sofix></sofix>
```

Element names

Element names must begin with a letter or underscore. They may be followed by zero or more letters, digits, periods, hyphens or underscores.

The use of the name string “xml” at the beginning of an element name, in any capitalization is reserved for future standards.

Do not use the colon in element names.

Element contents

Character data in elements can contain any data except `<` (use `&#60;`), `&` (use `&#38;`), and `]]>` (don't use it).

An element may also contain

- Entity references
- Character references
- CDATA sections
- Processing instructions
- Comments

CDATA section

`[CDATA[` opens a CDATA section.

`]]>` closes a CDATA section.

In the CDATA section you can have any type of character data. For example, you can include a HTML page. Can be placed anywhere where comments can be placed.

CDATA sections do not nest.

Processing instructions

Provide instructions to XML parsers. General form

<? *target instruction* ?>

Example

```
<?xml-stylesheet type="text/css" href="sofix.css"?>
```

Can be placed anywhere where comments can be placed.

Comments

Start with <!-- and end with -->

Example:

```
<!-- I can put < and & in a comment -->
```

Comments can be placed any where except inside the markup, i.e. the stuff that is between < and >.

Element attributes

General form

<*element_name attribute_name*="*attribute_value*">
...
</*element_name*>
or

<*element_name attribute_name*='*attribute_value*'>
... </*element_name*>

Attribute names are subject to the same restrictions as element names.

Value may not contain its delimiter or <. It may contain & only if this is part of an entity reference (to be discussed in another lecture).

Empty Elements

Elements may be empty, of the form

<*element_name*/>

Empty elements may have attributes

<*element_name attribute_name*="*attribute_value*"/>

or

<*element_name attribute_name*='*attribute_value*'>

Attribute names are subject to the same restrictions as element names.

Value may not contain its delimiter or <. It may contain & only if this is part of an entity reference (to be discussed in another lecture).

The XML declaration

This has to be at the beginning of the file

```
<?xml version="1.0"?>
```

The version number is always 1.0.

Introduction to sofix

This is a format for CD cataloging using XML.

Format called "sofix".

Thomas and students have made it all up.

Conceptual framework

Three key concepts

- Item: an individual CD or a collection of CDs kept physically together (i.e. sold together)
- Work: a piece of music as recorded on a CD. For simplicity, we do not distinguish between composition and recording of that composition.
- Track: semantics associated with physical separation on disk

Generalities

- All titles in English.
- If no English title provided, use a translation if it is obvious.
- If the translation is not obvious, use original language.
- All personal names as *Lastname*, *Initials*.

XML implies a nested structure

```
<item>
 <work>
   <track>
   </track>
   ...
 </work>
 ...
</item>
```

attributes of item

<labelname>*name of label* </labelname>

<number>*number of the CD* </number>

attibutes of work

<work>

<compositionyear>*take last year if there are several, including several possible years* </compositionyear>

<recordingyear>*take last year if there are several* </recordingyear>

<title>*take full title including opus number and key* </title>

<contributor role="*role*"> *name of person or group with a role in work/performance* </contributor>

controlled vocabulary of roles

composer, conductor

chamber_orchestra, orchestra, piano_trio, string orchestra, string quartet

alto, baritone, bass, soprano, speaker choir

alto sax, bassoon, cello, clarinet, flute, french_horn, horn, oboe, organ piano, prepared_piano, recorder, viola, violin, xylophone

The complete listing of all possible values is held at

http://wotan.liu.edu/home/krichel/xml/sofix“˙roles

attributes of a track

<track>

<title>*full title as given on CD* </title>

<time>*minutes*:*seconds* </time>

</track>

</work>

</item>

Well-formed and valid documents

All XML documents must obey to certain constraints. An XML that does obey the constraints is called well-formed.

An XML document is called valid, if, in addition to being well-formed, it obeys some further constraints. These constraints are being called validity constraint.

Sadly, validity constraints are different from well-formedness constraints because they do not not apply to all XML documents but only to some.

DTD and Schema

A DTD is the classic way in which constraints on SGML documents can be defined. XML can be derived out of full SGML with the use of a DTD. XML can be further constraint through the use of DTDs.

XML Schemas are new ways in which constraints can be applied to XML documents. These constraints are much more refined. For example, they allow to constrain element contents, for example to say that the time of a track is of the form *1_or_2_digits*:*2_digits*.

There are different schema languages that are proposed on the web. These are experimental.

other XML related standards

- Cascading Style Sheets
- XHTML
- RDF
- XPath & XPointer
- XLinks
- DOM & SAX

XHTML

is HTML redefined so that it become XML compliant.

For example, instead of ** **, you have to write ** **.

Pain without gain.

RDF

A framework to encode meaning to make it computer processable.

Uses the approach of a directed graph.

Generalizes a

object / property / value

approach, where the value may be another object. Objects are URI identifiable. A paper on RDF available at http://openlib.org/home/krichel/papers/anhalter.letter.pdf

RDF XML syntax is defined but currently being reworked.

Cascading Style Sheets CSS

A non-XML way of writing stylesheets that can be applied to both XML and HTML. Widely supported by browsers.

Written as a sequence of rules. Example

```
compositionyear, recordingyear  {
               color: red;
               font-family: sans-serif }
```

XPath and XPointer

are non-XML syntaxes referring to parts of an XML document, specific ranges, points, and sets of XML document.

There are used in other XML related standards, in particular, in XSL will be covered as part of XSL.

XLinks

is an XML syntax to link XML documents.

They go way beyond the conventional linking capabilities of HTML, but there is no obvious way for the browser to represent them.

DOM

is the Document Object Model, "a platform- and language-neutral interface that will allow programs and scripts to dynamically access and update the content, structure and style of documents. The Document Object Model provides a standard set of objects for representing HTML and XML documents, a standard model of how these objects can be combined, and a standard interface for accessing and manipulating them."

Now at "Level 3". Verry complicated. Works by building a tree out of a document.

SAX

is the Simple API (application programming interface) for XML, an event-based paring model. It reports parsing events (such as the start and end of elements) directly to the application through callbacks, and does not usually build an internal tree.

A lot less resource-intensive, in particular when the document is large and the task is simple.

XML information sets.

best understood through an example. Consider two XML snippets. Snippet 1

```
<person sex="female"> Margarete Krichel</person>
```

Snippet 2

```
<person sex='female'>Margarete Krichel </person>
```

Are they the same?

XML Namespaces

Allow to make XML element names and attribute name globally unique by associating them with a particular URI, usually a URL. The globally unique name is called the qualified name or qname, for short. The name without the namespace URI called the local name.

This is done through a namespaces declaration, and a prefix. The namespace declaration associates a short string, called a prefix with the namespace. The fully qualified name can then be written as *prefix*:*localname*

General syntax

<*element* xmlns[:*prefix*]= *URI*>

...

</*element*>

where [ ] indicate that it is optional. If the prefix is missing it means that all elements that have no namespace prefix belong, by default to the declared namespace.

Example

```
<academicdata
 xmlns="http://amf.openlib.org"
 xmlns:liu="http://www.liu.edu">
<person>
 <name>Thomas Krichel</name>
 <liu:teachesclass>lis900g</liu:teachesclass>
</person>
</academicdata>
```

Here the default namespace is http://amf.openlib.org. academicdata is in no namespace.

Avoiding cerebral indigestion related to namespaces

Expect nothing if you retrieve the namespace URI, when it is a URL.

Prefixes can be any short string. Some prefixes are customary, like xsi for http://www.w3.org/2001/XMLSchema-instance.

Default attributes only apply to elements not attributes. Attributes belong to the namespace of their elements, unless it has an explicit prefix.