XML and Internationalization

Sander van Zoest <sander@vanzoest.com>
ApacheCon 2002, Las Vegas, NV
$Revision: 1.7 $ -- Draft
<http://sander.vanzoest.com/talks/2002/xml_and_i18n/>

Table of Contents

1. Introduction

In the last few years, both the Extensible Markup Language (XML) and Internationalization (I18N) are gaining prominence on the Internet. Since the rise of the Internet and the development of electronic access to global information, it is becoming easier to stay in touch with, share with and learn from people all around the world. Because of this, the importance of being able to provide data and methods that allow for global consumption of the product is increasing. Localization is the process of tailoring products and content to a specific locale, i.e. to the language, cultural context, conventions and market requirements of that specific target market. This process normally has been a huge undertaking because of the lack of a superset one could develop their product in. With the rise of XML and Unicode the tasks required to properly localize a product is becoming easier.

With the turn to Web Services and Web accessible gateways; being able to determine the contents' country of origin, language and character set (the collection of elements used to represent textual information) is increasingly becoming more important.

By the means of XML, one can abstract the design, layout and business logic from the content. This process conveniently provides a layer to dynamically transform or to provide localized versions of the content. Specifying the language of the content provides the ability to distinguish and extract the content to provide automated or systematic translations of the elements in the native tongue when the default language does not match the one of the consumer. Combining the language with the country provides the ability for an even more localized version of the text in the particular countries dialect of the desired language. Without the standardized abstraction provided by XML most software developers would have to develop their own abstraction layers or potentially alter the source code itself to generate the content for each locale.

On the level of character sets, the Unicode Consortium (in cooperation with the International Organization for Standardization ISO/IEC/JTC1) has provided us with character encoding standards that define the identity of each character, its numeric value and how this value is represented in bits. The Consortium has defined three encoding forms (mappings from a character set definition to the actual code units used to represent the data) that allow the data to be transmitted in 8, 16 and 32-bits. These three forms, formally known as UTF-8, UTF-16 and UTF-32, provide developers with three ways to use Unicode. The decision to which encoding form should be used can be determined by weighing the importance between the ease of access to the characters and the memory footprint needed to represent the most commonly used characters. The ASCII character set provides the base for the byte values in UTF-8 and therefore UTF-8 tends to be the most popular encoding form for software developed in the ASCII character set. Using UTF-8 minimizes the need for software rewrites, while still providing a way to conform to the Unicode Standard. Although, most of what is discussed in this paper will apply to all three Unicode encoding forms, we will mostly be concentrating on UTF-8.

In this paper we will explore how XML and Internationalization work together to provide a workable solution for localization of content. We will explore the ways to translate from other character sets to UTF-8

2. Specifying document encoding

An encoding of an XML document is specified in the XML declaration using the encoding attribute. The values for the encoding declarations are defined by the IANA (See the References section for more information).

Example

```
<?xml version="1.0" encoding="iso-8859-1" ?>
```

If the XML document is transferred over an application protocol, such as HTTP, the document encoding may be specified by the application protocol. For instance, in HTTP the encoding can be defined using the charset parameter in the **`Content-Type`** header.

If the encoding is not specified using the above two methods, the XML document is considered to be of Unicode encoding. The type of unicode is specified by the Byte-Order-Mark (BOM). The BOM is a Unicode special marker placed at the top of the file that indicates its encoding. The BOM is optional for UTF-8.

BOM	Encoding
EF BB BF	UTF-8
FE FF	UTF-16 (big-endian)
FF FE	UTF-16 (little-endian)
00 00 FE FF	UTF-32 (big-endian)
FF FE 00 00	UTF-32 (little-endian)

Byte order only matters for encodings using more than 8-bits (i.e. UTF-16, UTF-32, etc.). It is irrelevant to UTF-8, but it is allowed to specify a BOM for UTF-8 for those who feel the need to be explicit.

3. Language Tags

The **`xml:lang`** attribute is provided to specify the language and an optional country for a particular element. The language is specified using the ISO Language code [ISO639-2] which is optionally followed by a dash '-' (U+002D) and the ISO Country code [ISO3166]. English would, therefore, be expressed as:

```
<element xml:lang="en">This is English</element>
```

To specify multiple languages within a single document, simply re-specify the **`xml:lang`** attribute for the portion that is in a different language.

The ISO Language codes come in either two or three letter flavors, where the two-letter version is required in the case where there exist both a two and three letter version. Besides the ISO Language Code, an IANA Language Code (prefixed by **`i-`**) and a user defined code (prefixed by **`x-`**) [LANGTAGS] are possible alternatives. The use of Unicode Language Tags is discouraged in XML, but are an alternative when dealing with plain text protocols.

Examples

Language Tag	Description
en-US	American English
i-klingon	IANA registered Klingon
x-piglatin	User defined Pig Latin

XPath is a language for addressing parts of an XML document, designed to be used by both XSLT and XPointer. XPath provides the ability to match the **`xml:lang`** attribute using the **`lang()`** function. This function returns **`true`** or **`false`** matching the current **`xml:lang`** value. **`lang()`** is case-insensitive and returns true when matching for

the ISO Language code even when the optional country code is also specified. When invoked with a supplied country code, it will only match when the language and country code exist.

4. Numeric Character References

Numeric Character Reference (NCR) is a term often used to describe a character in hexadecimal or decimal format in XML. The hexadecimal notation is of the form **`&#xHHHH;`** where HHHH is the hexadecimal value of the given Unicode character. Since the unicode characters are usually expressed in hexadecimal this form is the preferred form since it is easier to refer to the Unicode value. The decimal format is of the form **`&#DDDD;`** where DDDD is the decimal value of the given Unicode character.

The value of an NCR is **always** the value of the Unicode code-point, regardless of the encoding attribute of the XML file. NCRs are only needed when a character is not included in the encoding currently in use. NCRs (and character entity references) cannot be used in element and attributes names, in CDATA sections, in processing instructions and in comments.

Examples

Character	Hexadecimal	Decimal	Description
©	&#x00A9;	&#0169;	COPYRIGHT CIRCLE C (U+00A9)
	&#x20AC;	&#8364;	EURO SIGN (U+20AC)
’	&#x2019;	&#8217;	RIGHT SINGLE QUOTATION MARK (U+2019)

5. Character Entity References

As in HTML, it is possible to use and define Character Entity References, such as **`&euro;`** to provide a more legible representation of the NCR. It is recommended, however, to use NCRs instead of Character Entities, since the Unicode hexadecimal value is more versatile and easier to process by XML processors.

The entity references **`&`**, **`<`**, **`&apos;`**, **`&quot;`** and **`>`** used for escaping and differentiating between XML markup and escaping strings containing both single and double quotes must be provided by XML processors. For clarity, it might be good idea to explicitly declare them regardless. Since they would be defined as the same constant as set by the XML processor, there is no real harm in doing so, assuming your XML tools support it.

Character Entities are defined as follows:

```
<!ENTITY euro "&#x20AC;">
```

The character entity definition can be included in the DTD, or inline with the **`DOCTYPE`** defined by the XML Document.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
                      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd" [
  <!ENTITY  amp "&#x0026;">
  <!ENTITY   lt "&#x003C;">
  <!ENTITY   gt "&#x003E;">
  <!ENTITY quot "&#x0022;">
  <!ENTITY apos "&#x0027;">
]>
```

As of this writing, a few browsers such as Omniweb v4.1 and Mozilla 1.0 do not seem to handle this correctly. For interoperability reasons, it is better to define the entities in your DTD. This is more widely supported then inline entity references.

6. Characterset transcoding

One of the age old issues with internationalization is the abundance of character sets. Since most of the content is not in Unicode and not all software supports it yet. There is a good chance that conversion between charactersets, transcoding, will still be a common requirement when localizing XML files. Although, XML

processors internally are required to use either UTF-8 or UTF-16, most existing programs expect data in other character sets such as ASCII, the japanese Shift_JIS or ISO-8859-1 (Latin1).

Most XML processors also support additional character sets to allow for easy transcoding of XML. Depending on the parser it might inherit these capabilities from the language it was written in such as Java or compile against C/C++ libraries as GNU libiconv and IBM ICU to be able to handle foreign character sets.

XML Transcoding using XSLT

The easiest way to do the transcoding is to use an XSL/T processor with a similar stylesheet as the one provided below. It converts the XML to the Chinese Big5 character set.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:output encoding="Big5"/>
  <xsl:template match="node()|@*">
    <xsl:copy>
       <xsl:apply-templates select="node()|@*"/>
    <xsl:copy>
  <xsl:template>
<xsl:stylesheet>
```

Java String Locale Converter

The below code provided by Sun Microsystems [JAVAI18N] provides us a way to convert from Java's UTF-16 character encoding format to the unix friendly UTF-8.

```
<http://java.sun.com/docs/books/tutorial/i18n/text/example-1dot1/StringConverter.java>
/*
 * Copyright (c) 1995-1998 Sun Microsystems, Inc. All Rights Reserved.
 *
 * Permission to use, copy, modify, and distribute this software
 * and its documentation for NON-COMMERCIAL purposes and without
 * fee is hereby granted provided that this copyright notice
 * appears in all copies. Please refer to the file "copyright.html"
 * for further important copyright and licensing information.
 *
 * SUN MAKES NO REPRESENTATIONS OR WARRANTIES ABOUT THE SUITABILITY OF
 * THE SOFTWARE, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED
 * TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE, OR NON-INFRINGEMENT. SUN SHALL NOT BE LIABLE FOR
 * ANY DAMAGES SUFFERED BY LICENSEE AS A RESULT OF USING, MODIFYING OR
 * DISTRIBUTING THIS SOFTWARE OR ITS DERIVATIVES.
 */

import java.io.*;
import java.util.*;

public class StringConverter {

    public static void printBytes(byte[] array, String name) {
       for (int k = 0; k < array.length; k++) {
          System.out.println(name + "[" + k + "] = " + "0x" +
              UnicodeFormatter.byteToHex(array[k]));
       }
    }

    public static void main(String[] args) {

        System.out.println(System.getProperty("file.encoding"));
       String original = new String("A" + "\u00ea" + "\u00f1"
                                    + "\u00fc" + "C");

       System.out.println("original = " + original);
       System.out.println();

      try {
           byte[] utf8Bytes = original.getBytes("UTF8");
           byte[] defaultBytes = original.getBytes();

           String roundTrip = new String(utf8Bytes, "UTF8");
           System.out.println("roundTrip = " + roundTrip);

          System.out.println();
           printBytes(utf8Bytes, "utf8Bytes");
```

```
            System.out.println();
            printBytes(defaultBytes, "defaultBytes");
        } catch (UnsupportedEncodingException e) {
            e.printStackTrace();
        }

    } // main

}
```

Stream based character set conversion is done by providing an encoding argument to either the `InputStreamReader` and `OutputStreamWriter` classes.

Perl Locale Converter

Perl has had Unicode support since version 5.6. Although, it recommended to use perl 5.8.0 or later, since this release perl has achieved a much more intuitive support for unicode in comparision to earlier revisions [PERLI18N].

Perl uses the Encode module to provide support convert perl strings to different encodings.

```
#!/usr/local/bin/perl -wT

use 5.8.0;
use strict;
use Encode 'from_to';

our $data = "This is a single quote: \x{2019}";
from_to($data, 'utf-8', 'iso-8859-1'); # transcode UTF-8 to Latin1
```

To transcode a file to a different encoding simply open the file with the appropriate `:encoding(...)` pragma.

```
open(my $nihongo, '<:encoding(iso2022-jp)', 'text.jis');
open(my $unicode, '>:utf8',                 'text.utf8');
while (<$nihongo>) { print $unicode }
```

The Java and Perl examples are not XML specific, but can be used to transcode data before inserting them into XML or your DOM tree. Most java classes and perl modules to manipulate content will work with UTF-8, but because of the recent changes in perl, there is always a chance that a particular piece of code does not fully support the new perl 5.8.0 yet.

7. Text Presentation

With a universal character encoding, it is now possible to theoretically convert any text into any language. It is often forgotten, however, that not only are characters important, but so are punctuation, text direction , formatting and accents. Just as character composition, these elements vary wildly. In Japanese for example, there is no such as bold or italics, these formatting elements are applied differently. Instead tag the elements with their purpose and reasoning behind applying the particular text formatting.

Similarily, it becomes more important to take more care in using the appropriate element tags in document types, such as HTML and DocBook. When creating a list, use the list elements, rather then `<br />` elements. Avoid putting formatting elements in your text as much as possible and provide the formatting using style-sheets instead.

Other areas to take extra care with are date, timestamps, and numbers in general. Functions in XSL such as format-number() can be used to put a period (U+002E) between thousands for most european languages, where in the United States, people seperate thousands with commas (U+002C).

8. References and Further Reading

[I18NGURUS]
Thiery Sourbier, Website: <http://www.i18ngurus.com/>.
[ORAXML]
O'Reilly and Associates, Website: <http://www.xml.com/>.

[CHARMOD]
Martin Dürst, François Yergeau, Richard Ishida, Misha Wolf, Asmus Freytag, Character Model for the World Wide Web, W3C Working Draft, April 2002, <http://www.w3.org/TR/charmod/>.
[UNICODE-XML]
Martin Dürst, Asmus Freytag, Unicode in XML and other Markup Languages, W3C, Unicode, Februrary 2002, <http://www.w3.org/TR/unicode-xml/>.
[RUBY]
Marcin Sawicki, Michel Suignard, 石川 雅康 (ISHIKAWA Masayasu), Martin Dürst, Tex Texin, Ruby Annotation, W3C Recommendation, May 2001, <http://www.w3.org/TR/ruby/>.
[RFC3066]
H. Alvestrand, Tags for the Identification of Languages, IETF, January 2001, <http://www.ietf.org/rfc/rfc3066.txt>.
[UTF-BOM]
Unicode Frequently Asked Questions: UTF and BOM <http://www.unicode.org/unicode/faq/utf_bom.html>.
[EURO-XML]
Rick Jelliffe, Euro-XML, XML.com, September 2002, <http://www.xml.com/pub/a/2002/09/18/euroxml.html>.
[PRIMER]
Tony Graham, Unicode: A Primer, M&T Books, ISBN 0-7645-4625-2; 1st Edition, April 2000 <http://www.menteith.com/unicode/primer/>.
[XMLI18N]
Yves Savourel, XML Internationalization and Localization, Sams Publishing, ISBN 0-672-32096-7; 1st Edition, June 2001 <http://www.opentag.com/xmli18n>
[XMLI18NFAQ]
Yves Savourel, XML Internationalization and Localization FAQ, <http://www.opentag.com/xmli18nfaq.htm>.
[ISO10636-1993]
ISO/IEC 10646 ISO (International Organization for Standardization). ISO/IEC 10646-1993 (E). Information technology -- Universal Multiple-Octet Coded Character Set (UCS) -- Part 1: Architecture and Basic Multilingual Plane. [Geneva]: International Organization for Standardization, 1993 (plus amendments AM 1 through AM 7) <http://www.iso.org/>.
[ISO10636-2000]
ISO/IEC 10646-2000 ISO (International Organization for Standardization). ISO/IEC 10646-1:2000. Information technology -- Universal Multiple-Octet Coded Character Set (UCS) -- Part 1: Architecture and Basic Multilingual Plane. [Geneva]: International Organization for Standardization, 2000 <http://www.iso.org/>.
[UNICODE3]
The Unicode Consortium, The Unicode Standard, Version 3.0, Addison-Wesley, ISBN 0-201-61633-5; 1st Edition, February 2000 <http://www.unicode.org/unicode/uni2book/u2.html>.
[OASIS]
Robin Cover, XML and Unicode, Technology Report, September 2002 <http://www.oasis-open.org/cover/unicode-xml.html>.
[XMLCHARSETS]
<http://www.w3.org/TR/REC-xml#charsets>.
[CHARSETS]
IANA (Internet Assigned Numbers Authority). Official Names for Character Sets, ed. Keld Simonsen et al. <http://www.iana.org/assignments/character-sets>.
[LANGTAGS]
IANA (Internet Assigned Number Authority). Official Names for Language Tags. <http://www.iana.org/assignments/language-tags>.
[RFC2279]
Yergeau, F., UTF-8, a transformation format of ISO 10646, RFC 2279, 1998 <http://www.ietf.org/rfc/rfc2279.txt>.
[RFC2781]
Hoffman, P. and F. Yergeau, UTF-16, an encoding of ISO 10646, RFC 2781, 2000. <http://www.ietf.org/rfc/rfc2781.txt>.
[PERLI18N]
James Briggs, Perl, Unicode and i18N FAQ, February 2002, <http://rf.net/~james/perli18n.html>.
[JAVAI18N]

Dale Green, The Java Tutorial: Internationalization, 2002 <http://java.sun.com/docs/books/tutorial/i18n/>.

[ISO639-2]
ISO/IEC 639-2 ISO (International Organization for Standardization). ISO/IEC 639-2:1998 Codes for the Representation of Names of Languages -- Part 2: Alpha-3 Code. [Geneva]: International Organization for Standardization, 1998 <http://lcweb.loc.gov/standards/iso639-2/langhome.html>.

[ISO3166]
ISO/IEC 3166 ISO (International Organization for Standardization). ISO 3166-2:1998 Codes for the representation of names of countries and their subdivisions -- Part 2: Country subdivision code. [Geneva]: International Organization for Standardization, 1998 <http://www.iso.org/iso/en/prods-services/iso3166ma/>.

$Date: 2002/10/11 05:34:07 $