XML Content Localization and Unicode

By Ultan Ó Broin

Ultan Ó Broin is globalization analyst with Oracle Corporation's Applications Technology Group, based in Redwood Shores in the United States. He can be reached at ultan.obroin@oracle.com.

Abstract

Although the eXtensible Markup Language (XML) supports any defined character set, the basis for XML data globalization comes from the support of Unicode. In fact, UTF-8 is the default encoding for XML.

In this paper, we will examine the techniques for the internationalization and localization of XML-based content. To provide a basis for eventual localization, we will first consider typical internationalization requirements, the role of the Universal Character Set (which we refer to as Unicode), rendering and presentation mechanisms and outline language related features for Ruby text, vertical text, combined text, bi-directional text, the preservation of white space and more. Then we will examine how localization processes are best served by XML and the emerging XML Localisation Interchange File Format (XLIFF).

Our conclusion is that the use of Unicode and XML/XLIFF is the best solution for an open, scalable approach to the development of multilingual content for global deployment.

What we discuss in this paper is applicable to any XML-based data (software user interface strings, user assistance material text, marketing collateral and so on). Let's begin by looking at the building blocks of XML internationalization – character set encoding.

Character Support in XML

Although XML can support any defined character set, the basis for XML globalization comes from its support of the multilingual Unicode standard. This greatly simplifies internationalization and localization. At a minimum, XML processors must support the eight-bit UTF-8 and sixteen-bit UTF-16 encodings. Since UTF-8 is the default XML encoding and English language XML elements and attributes are frequently the basis for content development, the choice is usually in favor of UTF-8 as it allows for a smaller lower storage by allocating English characters one byte.

XML supports the range of Unicode character values for parsing shown here:

```
Char : : = #x9 | #xA | #xD | [#x20-#xD7FF] | [#xE000-#xFFFD] | [#x10000-#x10FFFF]
```

XML Unicode character support excludes the surrogate characters and developers should not use characters such as those for bi-directional embedding (`U+202A` to `U+202E`), line and paragraph separators (`U+2028` and `U+2029`) and some other defined limitations in their content. You can find more about these by referring to the XML specification (http://www.w3.org/XML/).

Here we see the XML encoding syntax (in this case for UTF-8). The encoding attributes should be one of the names identified by the Internet Assigned Numbers Authority (IANA):

```
<?xml version="1.0" encoding="utf-8"?>
```

If UTF-16 is chosen, then the encoded files must start with either of the Byte Order Mark (or BOM) values of `0xFEFF` or `0xFFFE`, which identifies the XML encoding as UTF-16 and tells the processor whether the leading or trailing value of each multi-byte character is the most significant value. Without the BOM, the file will be parsed as UTF-8 by XML processors.

The best solution for globalized content storage is to use normalized UTF-8 characters. If, for some reason, you choose an encoding other then Unicode, then you may find that a character cannot be represented directly by that encoding[1]. In this case, developers can use a numeric character reference (NCR). The NCR is the decimal or hexadecimal value for the character. For example, `&#xE;` is the hexadecimal value for "é". NCRs are a powerful way of providing for multilingual data migration since they can be converted to and from UCS values directly - without the need for predefinition.

Character entities are another option for representing characters. Developers can safely use the five character entities predefined in the XML specification (`<`, `>`, `&`, `&apos;` and `&quot;`). However, using further character entities can be problematic unless they are predefined in XML. Otherwise, it is unwise to assume that every text parsing tools and browsers will properly interpret the character entity.

Language Identification

The language of the XML content is identified with the `<xml:lang>` element. You must always use this element at the top of the XML file or to indicate a change of language *within* a document.

The `<xml:lang>` element supports the use of letter codes for the language *and* region. For example, here we see how Swiss documents are coded for French, German and Italian languages respectively:

```
<p xml:lang="fr-CH">Swiss French content</p>
<p xml:lang="de-CH">Swiss German content</p>
<p xml:lang="it-CH">Swiss Italian content</p>
```

However, developers should be very careful about requiring different languages in the same document. Although the deployment of multilingual XML content is a very efficient way of delivering content, unless each language segment can be individually parsed it cannot be localized easily because translation memories are generally built on a single source-target language basis.

Bear in mind that XML processors themselves make no assumptions about what the attributes of the `<xml:lang>` element mean in terms of character set or presentation of the associated data. How data is rendered, for example, needs to be handled by the eXtensible Style Sheet Language Formatting Objects capability (XSL-FO) or a Cascading Style Sheet.

Presentation in Different Languages

The rendering and presentation of XML content is external from the data, but still meets the typical internationalization challenges such as fonts, reading directions and other language specific conventions.

For rendering and presentation of global content, XML requires the use of a set of predefined presentation styles, then applied to the data using XML's eXtensible Style Sheet Language Formatting Objects capability (XSL-FO) or through a Cascading Style Sheet (CSS).

XSL-FO is an XML-based language used to define the formatting information in style sheets. It is designed for the application of style information through an XML parser, and supports incremental style sheet derivations, so it is ideal for supporting different locale and language rules with the same style sheet.

[1] For example, if there is a requirement that all content is delivered in US 7 ASCII only.

To take into account different language requirements, developers should split the same style sheet into different parts – the first with the general presentation rules and the latter with the language-specific rules. Style sheet entries also correctly render language conventions such as different quotation marks, list numbering style (decimal, Greek, Hebrew, Latin, Japanese Katakana, *etc*.) and so on.

Bi-directional languages like Arabic and Hebrew are also supported by data rendering through XSL-FO or CSS. In this case, the style sheet `direction` and `unicode-bidi` properties are used to specify the bi-directional text rendering in the XML document:

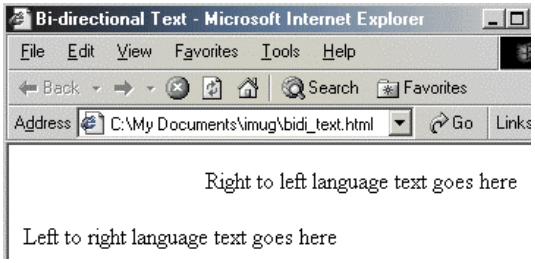
```
<rtl>
<p xml:lang="ar">Right to left language text goes here</p>
</rtl>
<ltr>
<p xml:lang="en">Left to right direction language text goes here</p>
</ltr>
```

XML document with different language directionality elements

```
rtl, rtlquote {
Direction: rtl; unicode-bidi: embed;
}
ltr, {
Direction: ltr; unicode-bidi: embed;
}
```

Style sheet `direction` and `unicode-bidi: embed` properties that render the language elements

Here we see the above bi-di code data rendered in a browser:



Rendering of Bi-di text in Internet Explorer 5.5

The unicode-bidi embedded properties provide for directional formatting codes that influence how the display of bi-di text. These are not specific to XML, but they provide for the correct semantics for characters that are stored logically, depending on when the text within a segment is left-to-right or right-to-left text.

XML has other powerful language-related features. This includes the support for Ruby text (an annotation text used to assist in Japanese and Chinese pronunciation). This is rendered using the `<ruby>`, `<rb>` and `<rt>` elements:

```
<h3>Example Ruby text (albeit in English)</h3>
  <p>
    <ruby>
          <rb>This is the Base Language Text Position</rb>
          <rt>This is the Ruby Language Text Position</rt>
    </ruby>
  </p>
```

Ruby and base text elements

Here we see the above code sample rendered in a browser:

Browser rendering of Ruby and base text elements

Vertical writing is used in Asian languages (typically in Chinese and Japanese). XML provides for the support of this required with the `writing-mode` properties:

```
<p style="writing-mode: tb;">Example of vertical text</p>
```

Combined text is a Japanese layout method, where characters are required to be grouped according to rules known as *kumimoji* or *warichu*. XML supports this requirement through XSL and CSS with the `text-combine` properties:

```
span.kumimoji { text-combine: letters; }
span.warichu  { text-combine: lines; }
```

The ability to preserve or override the white space delimiters in English language content is another important language-related feature. Spaces, tabs, carriage returns and line feeds can be handled by the `xml:space` element that can either have a `default` or `preserve` attribute, as required.

Unicode users must be careful not to use the UCS code points (for example the `ARABIC LETTER AIN` represented by `U+0369` and `ZERO WIDTH SPACE` represented by `U+200B`) to represent different forms of characters for context shaping, since this is dependent on the rendering for the correct context and not on the character's storage code point

Different languages use different styles to convey the concept of emphasis. In Asian languages such as Chinese and Japanese, this takes the form of additional marks above or below the character in question. The `font-emphasis-style` and `font-emphasis-position` properties in XSL and CSS allow these emphasis characters to be displayed in XML.

Additional support can be implemented through XSL-FO and CSS for the display of language-related data under different browsers and operating systems, including some workarounds. You can read more about these using the references at the end of this paper.

Sorting International Content

The linguistic sorting of localized content is a challenge for any technology. The `<xsl:sort/>` element provides for ascending and descending linguistic sorting sequences for global content. Content is sorted at run-time using an XSL transformation which, when combined with a language attribute, ensures that the order is correct for each language[2].

The `<xsl:number/>` element in XSL allows the output of a formatted integer from XML content. This element is mostly used in the creation of numbered information. The `grouping-separator` and `grouping-size` attributes specify the character for numeric separators and the number of digits in each grouping.

Date and Time Settings

Date and time information should also be stored in a manner that allows for locale independence. The best way to do this is to use the XML schema specifications settings, which follow the values defined in the ISO 8601 standard.

These date and time values enables data to be exchanged easily between globally distributed XML processors and tools. For example, XML supports the date time value of `CCYY-MM-DDThh:mm:ss` (`CC` stands for the century, `YY` for the year, `MM` for the month and `DD` for the day). This date time information is coded in ASCII characters and can be immediately followed by a `Z` to indicate Universal Time Coordinated (UTC) or the time zone difference between local time and UTC.

Adopting a universal date and time values enables data to be exchanged easily between globally distributed XML processors and localization tools.

XML Localization

Now let's examine the development team requirements for the localization of XML. XML data is independent of presentation, so it is ideal for the authoring of large amounts of information that is stored in a database and presented to the user at run-time. In general, for successful localization of content, XML DTDs and schema must provide for the following features:

- First, the localizable content must be easily identifiable by an element in the XML file. In addition, this element can be used to mark any content that should *not* be localized.
- Second, localizable strings must be associated with a unique and persistent identifier of one or more keys generated automatically by the content development. These identifiers are important as they provide the means for an automatic and secure reuse of content by localization tools and utilities.
- Third, context information must be provided to allow for a more accurate localization. This information typically consists of an element showing a string type attribute which indicates whether the target string is a message, title, prompt and so on.
- Fourth, any maximum length restrictions that are required for a localizable string must be communicated to the localizer. Usually this restriction is defined as a number of bytes in the XML content, and the localization tool enforces the restriction during localization.
- Fifth, a note or description element must be provided, containing information for the localizer on how the content should be localized.

[2] This feature needs careful verification before deployment, and at the time of writing support for it in the major browsers is poor. One option is to utilise the interntional sort features of your database instead.

For example:

```
<OraTranslatability>
  <XlatElement Name="BusinessArea">
   <XlatID>
    <Key>Developer Key</Key>
   </XlatID>
    <XlatAttribute Name="Name" MaxLen="100" Expansion="70" Note="Mandatory">
    </XlatAttribute>
    <XlatAttribute Name="Description" MaxLen="240" Expansion="Max">
    </XlatAttribute>
</XlatElement>
```

Example general DTD showing key localization features

Developers should avoid the use of localizable element names or variables in XML content. Keep the element names in the same language as the source language so the localizers can understand their intention. Also, do not rely on localizable variables that are substituted at run-time (for example, book titles or product names) or used to build dynamic content (for example in a document's table of contents). These kinds of entries cannot be localized properly.

```
<tableofcontents-entry id=23 field=”heading:sub-heading”>
```

Incorrect use of localizable variables in elements

```
<tableofcontents-entry id=23>
       <heading>Heading text for localization</heading>
       <subheading>Subheading text for localization</subheading>
<tableofcontents-entry>
```

Correct solution

Finally, the DTD or schema should be provided to the localization management group or vendor so that the localization tools can be configured to parse the data properly.

XML Localisation Interchange File Format

Although it is possible to localize XML directly, or to develop your own DTD for that purpose, the best way to provide for localization of XML is to use the XML Localisation Interchange File Format (or XLIFF) DTD. XLIFF is an XML-based file format for the exchange of localization data, based on OpenTag 1.2 and including features of TMX. It was developed by a group of localization partners including Oracle, Novell, IBM/Lotus, Sun Microsystems, Alchemy, Berlitz, LionBridge, Moravia-IT, and the RWS Group. XLIFF is now maintained under the aegis of the Organization for the Advancement of Structured Information Standards (OASIS), located at http://www.oasis-open.org/.

XLIFF defines a specification for an extensible format that caters specifically for localization requirements for any file parsable file format (RTF, HTML, RC, PSD, *etc*.). It allows any software publisher to produce a single interchange format understandable by any localization service provider. It requires that the format should be tool independent, standardized, and support the whole localization process. In this case, our concern is how XLIFF relates to XML.

Paul Quigley (now an independent I18N consultant) was one of the prime movers behind XLIFF while Tools Director of Oracle Corporation's Worldwide Product Translation Group. He says “the XLIFF data format successfully meets the goal of the separation of localization data and process, providing a focus on automation, stopping the proliferation of internal XML formats, and turning localization into a commodity for all players. Software publishers are freed to focus on producing international products and vendors are freed to focus on translating without managing multiple translation tools or file formats”.

XLIFF Structure

XLIFF must be a valid XML document, declared using `<!DOCTYPE xliff PUBLIC "-//XLIFF//DTD XLIFF//EN">`. The format uses a hierarchical structure of primary elements to represent the localization data:

```
<header>
      <body>
            <group>
            <trans-unit>
            <source>
            <target>
            <alt-trans>
```

The header element is specially designed to support the localization process by providing metadata used by processors and tools to identify parts of the localization process (translation, edit, linguistic QA, updating and so on) using the `phase-name` and `process-name` properties. For example:

```
<header>
      <phase-group>
            <phase
            phase-name="translationedit"
            process-name="translation"
            date="2002-01-12T 12:11:21Z"
            />
      </phase-group>
</header>
```

XLIFF defines the source and target text using the `<trans-unit>`, `<source>` and `<target>` elements. One important feature allows XLIFF to carry *more* than one translation for a source string; this is done using the `<alt-trans>` element. In addition, XLIFF provides information for the translator in order to explain the context or intended usage of a string. This is done with the `<note>` element:

```
<trans-unit id=”bigirishcolumn_145” restype=”title” maxwidth=”90” size-unit=”byte”>
       <source xml:lang=”EN”>Database manager</source>
            <target xml:lang=”GA”>Feighlí feasa</target>
            <alt-trans>
            <target xml:lang=”GA”>Gocamán na ngiotán</target>
            </alt-trans>
                  <note>Manager means administration tool - not a person</note>
</trans-unit>
```

XLIFF addresses the challenge of dialog resizing through the `<trans-unit>` element's `coord` attributes, which specifies the x, y, cx and cy coordinates of the text for a given element. In practice, for resizing, tools would require the use of the original source reference file (through the `<skl>` or skeleton element - an idea borrowed from OpenTag). However, resizing capability is a function of the translation tool or whether the localizer can access the equivalent of a runtime development environment. The XLIFF technical committee is discussing the adoption of standard APIs for rendering to resolve this issue.

As the XLIFF DTD is an open standard, it offers a synergy of a single format exchanged between parties. For example, XLIFF can be used not only by content publishers but also by customers who can then modify XLIFF content using commercially available localization tools such as SDL*x*, Trados Tag Editor, Star Transit, ForeignDesk, Alchemy Catalyst or any localization tool that allows the user to define which XML element to localize[3].

XML content can be transformed to XLIFF using an XSL transformation or a tool-based conversion method. For further information about XML internationalization and XLIFF, free XSL templates and localization utilities for the support of XLIFF, the XLIFF settings files for the localization of XLIFF documents with SDL*x* or Trados TagEditor, go to http://www.opentag.com/xliff.htm.

Coverage of the commercially available localization tools support for the various internationalization and language related features of XML is beyond the scope of this paper. However, for an overview of the supported features as well as a comprehensive reference of XML internationalization and localization techniques you should refer to *XML Internationalization and Localization* by Yves Savourel (ISBN:0-672-32096-7, Jul-2001).

Summary

We have outlined some of the basic techniques for the internationalization and localization of XML-based content, and seen how Unicode provides a very powerful basis for the storage of multilingual content, which when combined with XML's excellent internationalization support for storage and rendering opens a whole new range of possibility for the processing of multilingual content.

The merging XLIFF standard offers a very attractive internationalized and scalable proposition for the deployment and localization of global content. Although a relatively new combination, the use of Unicode and XML/XLIFF will become the optimal solution for an open, scalable approach to the development of multilingual content for global deployment. If you are seeking a solution, the area merits serious attention.

References

- XML Specifications: http://www.w3.org/XML/
- XLIFF: http://www.oasis-open.org/
- Internet Assigned Numbers Authority (IANA): http://www.iana.org/
- XLIFF, tools, templates and more: http://www.opentag.com
- *XML Internationalization and Localization* by Yves Savourel (ISBN:0-672-32096-7, Jul-2001)
- Localization Institute Seminars on XML i18n and l10n: http://www.localizationinstitute.com

[3] Localization tools need to be able to parse the `<source>` element, however in some cases, utilizing the `<alt-trans>` element may be a challenge. Check with the tool developer.

Acknowledgements

Thank you to Yves Savourel, Tony Jewtushenko and Paul Quigley for their thoughts and information.