



Getting ready for when the Internet becomes world-wide...

Shedding a Little Light on XML

When it gets too hot, go on vacation. This has always been my credo and by the time you are reading this article, I'll have long since returned from bliss and relaxation.

As every American traveler knows, if you can't speak other languages, learn a few important common expressions (hello, thank you, how much is bail?). Visiting other countries got me thinking that there are too few Web sites that support multiple languages. What is a bit unnerving is that according to a billboard in Europe, the prevalent language on the Web in 2007 will be Chinese. I guess the writing is on the wall, in a very literal way. This month's Q&A focuses on issues involving localization of XML documents. XML is definitely geared for this functionality, but it isn't always apparent. Sometimes you just need someone to shed a little light on the subject....

Customizing XSLT

Q: How can I create a multilingual version of my Web site?

A: This question can be answered in a few ways. I cover four in this article, along with source code to help demonstrate each of the techniques.

In each explanation a simple data entry form will be created to display basic information about an employee (name, address, hourly rate, department). As with many companies, it's necessary to display common data for users of disparate languages found in the same large companies. The goal is to display the data entry form in the language that's appropriate for the local user, regardless of the language the developer speaks. This process is known as *localization*.

Method 1: A separate XSLT stylesheet for each language

This approach uses a different stylesheet per language. Though a rather crude way of accomplishing localization, it's relatively easy to implement. In fact, given the languages you'll want to support, the difficulty may not be in creating the initial stylesheets, but rather in maintaining them. Remember, if you have the same page rendered in *x* number of languages, that will be *x* times the number of changes that will need to be done whenever a change to that page is requested. This risks turning into a maintenance headache. Listing 1 demonstrates the code behind using this approach. Additional demos can be found at www.xmlpitstop.com/default.asp?DataType=XMLEXAMPLES.

Method 2: Using lang() with a single XML translation document

Before going into detail about this method, here are a few quick XSLT functions and definitions that might help:

- **lang():** An XSLT function used to test whether the language specified matches the language of the XML text. It can also be used to filter XPath queries by elements that match a specific language.
- **xml:lang:** An XML attribute used to identify the language used for the XML text (see Listing 2). For example:
–English: <headings xml:lang="en">
–Spanish: <headings xml:lang="es">

In general, `xml:lang` is used to specify the language and `lang()` is used to test for a specified language or to filter elements by language as well as by a combination of language and country (see Listing 3). For example `fr-CA` (French in Canada) is different than `fr-FR` (French

in France), or `en-CA` (English in Canada) is different than `en-US` (English in the U.S.). `lang('EN')` will test true for both `xml:lang="en"` and `xml:lang="en-US"`.

Note that the `lang()` function will climb the hierarchy until it finds the closest ancestor with an `xml:lang`-attribute node. That node value is then checked against the argument. If the argument is just a language family, the language is checked against the language family of the value (which may be a country-qualified language; that's okay). If the argument is a country-qualified language, it's checked against the country-qualified value, not just the language value.

Tip: For localized applications, it's important to use a character set that will display all characters correctly. If you use UTF-8 (the default for XML), all languages will be supported in a single coded character set.

Now for the Details...

This method involves storing the localized data in a separate XML file and performing an XSLT "lookup" to convert the English terms into terms for the local language (see Listing 4). The "lookup" involves using an XPath query to retrieve the translated field heading for the given element name. In this example, the term *firstname* may be translated to "First Name," "El Primero Nombre," or "Le Premier Nom," depending on the local language of the user. The technique used to bridge the element name to the translation name is done in the following steps.

1. Store the translation language into an `xml:param`.

```
<xml:param name="isolang"
select="'fr'"/>
```

AUTHOR BIO

David Silverlight is chief XML evangelist for Infoteria Corporation (www.infoteria.com), an XML software development company based in Tokyo, Japan, and Beverly, Massachusetts. He also maintains www.XMLPitstop.com, a resource for XML examples and everything else XML.

2. Use the `document()` function to store the translation document into a variable.

```
<xsl:variable name="translations"
select="document('allheadings.xml')"/> 
```

3. Specify the encoding character set for the HTML page. This is a key point since omitting this will cause the page to display your HTML according to the character set defined in the browser settings, which may not be what is intended. A common symptom of omitting this is when the international characters are displayed as “garbage” characters in the browser even though the correct characters are displayed when the HTML source is viewed.

```
<META http-equiv="Content-Type"
content="text/html; charset=UTF-8"/>
```

4. Iterate through all the employee elements.

5. For each iteration the following occurs:

- The `name()` function is used to get the name of the element.

- An XPath query translates the element name to the local language. Since most of the work is done here and the XPath query might seem lengthy, breaking it down a bit (as shown below) might help.

6. Output the translated name and an HTML textbox for the element.

The XPath statement used to perform the translation is:

```
<xsl:variable name="fieldheader"
select="$translations/allheadings/
headings[lang($isolang)]/
heading[@category=$fieldname]"/>
```

The breakdown of the above XPath statement is:

- **`$translations/allheadings`**: This part of the XPath statement puts us in the context of all of the heading entries of the translation document (see Figure 4).
- **`headings[lang($isolang)]`**: `$isolang` is a variable that stores the abbreviation of the target translation language. For example, by following an element with the predicate `[lang("es")]`, the elements will be filtered to include only those for which `xml:lang` is defined as “es” (for Spanish text).
- **`heading[@category=$fieldname]`**: Last, but not least, this portion of the XPath statement will match the element in the XML translation document to the current element name. The translated value is produced using the matching heading element.

The end result of the above XPath statement is a node stored in the variable named “fieldheader” whose contents will then be displayed in the browser just to the left of the textbox that displays the contents.

Method 3: Using `xsl:lang` with an XML translation document for each language

This approach involves the same type of lookup as demonstrated in Method 2. However, instead of keeping all the translations in a single file, each translation file is kept in a directory for that specific language. To accomplish this, use the ISO language abbreviation to help build the path to the XML translation document. Each directory will house a file with the correct translations. Here are some rough examples:

- **English**: `C:\xmlwidgets\translations\en\headers.xml`
- **Spanish**: `C:\xmlwidgets\translations\es\headers.xml`
- **Italian**: `C:\xmlwidgets\translations\it\headers.xml`



to structure the
? of the ?
in ? a ? that I ?
? it
as the ? for ?



To accomplish this a few different steps are involved.

1. For starters, instead of using a parameter, the language is extracted from the XML file and stored into a variable named “`$isolang`”. *Note*: This decision is purely a matter of architectural style. A parameter would also have worked as nicely here as it did in Method 2. This method, however, demonstrates how the `xml:lang` attributes of the XML document text can be extracted.

```
<xsl:variable name="isolang">
  <xsl:value-of
select="/employees/@xml:lang"/>
</xsl:variable>
```

2. Next, the language variable in conjunction with the XSLT `concat()` function is used to build the path to the translation file.

```
<xsl:variable name="headingpath"
select="concat('translations/', $isolang,
'/headings.xml')"/>
```

3. Finally, now that the path to the XML translation file is defined, we can do a similar “lookup” translation as in Method 2.

The XPath statement used to perform the translation is:

```
<xsl:variable name="fieldheader"
select="document($headingpath)/
headings/heading[@category=$field-
name]"/>
```

The breakdown of the above XPath statement is:

- **`document($headingpath)`**: We’re referencing the document stored in the path. Note that once the path is defined, the document function allows access to it the same way as in Method 2.
- **`/headings/heading[@category=$fieldname]`**: Last, but not least, this final portion of the XPath statement will match the element in the XML translation document with the current element name. The translated value is produced using the matching element.

Caution: This technique has issues if the author of the XML correctly uses a country-qualified language such as “en-CA” and you want it to properly use “/en/” if there’s no “/en-CA/” subdirectory. What’s more, the comparison between the argument to `lang()` and the value of `xml:lang` is case insensitive, but country/language-group sensitive. It’s important to make sure all country/language directories are defined.

Method 4: Using translations when needed

Method 4 simply demonstrates the use of the `lang()` function to sparingly translate XML data when it’s needed. In simple cases of a few translations, it may be faster and easier to implement than some of the previous approaches. Also, this technique correctly accommodates country-qualified languages. A note of warning, however, is that it could risk leading to cluttered code if used too frequently.

```
<xsl:choose>
  <xsl:when
test="lang('en')">Hello</xsl:when>
  <xsl:when
```

```
test="lang('fr')">Bonjour</xsl:when>
<xsl:when
test="lang('es')">Hola</xsl:when>
<xsl:when
test="lang('he')">Shalom</xsl:when>
<xsl:when
test="lang('ru')">Daspedana</xsl:when>
<xsl:otherwise>Hello</xsl:otherwise>
</xsl:choose>
```

Elements of Design

While we are on the topic of localization, it seems fitting to list some informative localization resources:

1. Savourel, Y. (2001). *XML Internationalization and Localization*. Sams. The title of this book says it all.
2. *The Localization Institute*: www.localizationinstitute.com. "Quality training for localization and Internationalization." This site includes seminars (online versions as well), roundtables,

and training on localization by some of the leaders in the industry.

3. *Microsoft Localization*: www.microsoft.com/globaldev/default.asp. A good resource for the Microsoft junkies out there, including articles, references, a real-world application, and "Ask Dr. Internationalization."
4. *Unicode, Inc.*: www.unicode.org. A good site for anybody involved in globalization of software in general (Internet or not). It also lists upcoming Unicode conference dates.
5. Young, G. (1999). "DXML" Goes Global: *Localization and the XML/DHTML Menus*. <http://msdn.microsoft.com/library/default.asp?URL=/library/enus/dncodecorn/html/corner080999.asp>. An oldie but goodie.
6. *Free online language translation utility*: <http://ets.freetranslation.com:5081/>. A cool, quick online translation utility for

translating words or expressions between languages.

7. *XMLPitstop.com*: www.xmlpitstop.com. Last but not least, yours truly has compiled a good list of resources in the localization section of xmlpitstop.com

• • •

This monthly column is an opportunity for me to respond to the questions that tug at our brain cells and beg to be answered. So send them in my direction at dsilverlight@infoteria.com. I'll select those that fall into one or more of the following categories:

- **FAQ:** What you always wanted to know
- **Black Belt:** The toughest of the tough
- **Customizing XSLT:** My personal favorite and by far the most popular
- **Going Wireless:** Bring the world of wireless to your eyes

DSILVERLIGHT @ INFOTERIA.COM

LISTING 1 Method 1 – Transforming XML and XSL documents based on a language variable

```
set objXML = Server.CreateObject("Microsoft.XMLDOM")
set objXSL = Server.CreateObject("Microsoft.XMLDOM")
'Load the XML object with the XML from the GetXMLData function
objXML.async = false
objXML.loadXML
strDataXML

' Load the XSL from the XSL Filename
Select Case Request("LANG")
Case "FR"
strXSLName = "NWDDetails_FR.xml"
Case "ES"
strXSLName = "NWDDetails_ES.xml"
Case else
strXSLName = "NWDDetails.xml"
end select
objXSL.async = false
objXSL.load Server.MapPath(strXSLName)
```

LISTING 2 Method 2 – Using xml:lang to specify the local language

```
<employees xml:lang="en">
<employee>
<firstname size="30">John</firstname>
<lastname size="30">Smith</lastname>
<hourlyrate size="12">25</hourlyrate>
<department size="20">Internet</department>
</employee>
</employees>
```

LISTING 3 Method 2 – Using lang() with a single XML translation document

```
<?xml version='1.0' encoding='utf-8' ?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="html" />

<xsl:template match="/">
<xsl:variable name="isolang" select="'fr'"/>

<xsl:variable name="translations"
select="document('allheadings.xml') />
<META http-equiv="Content-Type"
content="text/html; charset=UTF-8"/>

<xsl:for-each select="employees/employee">

<xsl:for-each select="*">
<xsl:variable name="fieldname" select="name(.)"/>
```

```
<!-- Look up header using the current element name -->
<xsl:variable name="fieldheader"
select="document('allheadings.xml')/
allheadings/headings[lang($isolang)]/
heading[@category=$fieldname]"/>
```

```
<xsl:value-of select="$fieldheader"/>
<xsl:element name="input">
```

```
<!-- Set the Size of the textbox -->
<xsl:attribute name="size">
<xsl:value-of select="@size"/>
</xsl:attribute>
```

```
<!-- Set the value of the text box -->
<xsl:attribute name="value">
<xsl:value-of select="text()"/>
</xsl:attribute>
<br/>
</xsl:element>
```

```
</xsl:for-each>
</xsl:for-each>
</xsl:template>
```

```
</xsl:stylesheet>
```

LISTING 4 Method 2 – The single translation XML document

```
<?xml version="1.0" encoding="UTF-8"?>
<allheadings>
<headings xml:lang="en">
<heading category="firstname">First Name</heading>
<heading category="lastname">Last Name</heading>
<heading category="hourlyrate">Hourly Rate</heading>
<heading category="department">Department</heading>
</headings>
<headings xml:lang="fr">
<heading category="firstname">Prénom</heading>
<heading category="lastname">Deuxième Nom</heading>
<heading category="hourlyrate">Taux heurée</heading>
<heading category="department">Service</heading>
</headings>
<headings xml:lang="es">
<heading category="firstname">Primer nombre</heading>
<heading category="lastname">Segundo Nombre</heading>
<heading category="hourlyrate">Por hora Tasa</heading>
<heading category="department">El departamento</heading>
</headings>
</allheadings>
```

DOWNLOAD THE CODE @ www.xml-journal.com