

Chapter 3: Miscellaneous Tagging

In This Chapter:

- Language Settings
- Space Handling
- Date and Time Representation

Several text properties are common to all types of XML documents, regardless of their final purpose. These characteristics occur so frequently that the various XML standards often include special provisions for them.

These properties cover the following challenges:

- **How to identify the language of the content**—For example, how to distinguish a paragraph in German from one in Czech.
- **How to handle white spaces**—Basically, how to note the difference between spaces that are meant to make the structure of the document more readable (spaces such as indentations and empty lines), and the ones that are meaningful.
- **How to represent date and time information**—In our case especially, how to display this information both from locale-specific and from locale-neutral viewpoints.

Language Settings

To address language identification, XML provides a default attribute to specify the language of any element: `xml:lang`. In many respects this attribute can also be seen as a *locale*.

A locale is, roughly, the combination of a language and a region. The classic example is the difference between French, the language, and the two locales: French for France, and Canadian French (Québécois). Many other examples exist: the various flavors of Spanish, Brazilian versus Portuguese, and so forth.

In addition to linguistic differences, the locale also often indicates possible variations on how to process data: Currency, numbers, date/time formatting, sorting, and character uppercasing and lowercasing are some of the locale-specific areas. Sometimes the locale even goes beyond and points to deeper differences such as the type of writing system (for example, Classical Mongolian versus Cyrillic Mongolian, or Azerbaijani Arabic versus Azerbaijani Cyrillic).

NOTE

A good example of a language where differences are clear between locales is Spanish.

Spanish is spoken in many countries and therefore comes in many different varieties. When

localizing for a specific market you must decide which flavor you need.

For example, Spaniards use "*utilidades*" for "*utility programs*," Argentines use "*utilitarios*," and Mexicans use "*utilerías*." Another example is the term "*computer*." Spaniards use the word "*ordenador*" but all Latin Americans use "*computadora*" instead. Such discrepancies cause a few dilemmas when you want to have only one Spanish translation for all markets.

To reduce costs, companies often try to use a "neutral" or "international" Spanish. This is an artificial creation, as is "Latin American Spanish."

Finally, to avoid confusion, you might want to refer to the Spanish spoken in Spain as "Iberian Spanish" rather than "Castilian Spanish," the term "*Castellano*" being often used in South America to refer to the Spanish spoken there.

When defining your own XML vocabulary, you should use `xml:lang` as your attribute to specify the locale information, rather than come up with your own attribute. There are a couple of good reasons for this.

First, `xml:lang` will be understood immediately by any XML user. And second, it will allow you to take advantage of interoperability among the various XML-related technologies such as XSL or CSS.

If you use a DTD to specify your format, `xml:lang` must still be declared, just as with any other attribute. For example:

```
<!ATTLIST p
  xml:lang NMTOKEN #IMPLIED >
```

Language Codes

The values of the `xml:lang` attribute should conform to the language tags defined in the XML specifications, as shown in Listing 3.1.

Listing 3.1 Definition of the Value for the `xml:lang` Attribute

```
LangValue ::= Langcode ('-' Subcode)*
Langcode  ::= ISO639Code | IanaCode | UserCode
ISO639Code ::= ([a-z] | [A-Z]) ([a-z] | [A-Z])
IanaCode   ::= ('i' | 'I') '-' ([a-z] | [A-Z])+
UserCode   ::= ('x' | 'X') '-' ([a-z] | [A-Z])+
Subcode    ::= ([a-z] | [A-Z])+
```

In addition, according to RFC1766, the part on the right of the '-' can be up to 8 characters long.

Currently the language codes use ISO 639 2-letter codes, but as of January 2001, RFC1766 has been superseded by RFC3066, which introduces the use of ISO 639 3-letter codes.

According to this last RFC, if a language can be identified with both types of code, the 2-letter code must be used. The 3-letter codes should be used only for representing languages that do not have 2-letter

codes. For example, the code for Korean must always be `ko` and never `kor`.

In addition, there are 2 types of 3-letter codes: Terminology and Bibliography. Currently none of the languages that should be using a 3-letter code have a discrepancy between the Terminology form and the Bibliography form. If such a conflict occurs in the future, the Terminology code should be used.

Finally, if a language has both an ISO code and an IANA code, the ISO code must be used.

Table 3.1 Use of ISO Codes

Languages	639-1	639-2/T	639-2/B	xml:lang
French	fr	fra	fre	fr
German	de	deu	ger	de
Manipuri		mni	mni	mni
Navajo	nv	nav	nav	nv

NOTE

Normally, attribute values in XML are case sensitive. However, for simplification purposes and to match the RFC3066 standard, the values of the `xml:lang` attribute are not case sensitive. For example, the four values "`pt-BR`", "`PT-BR`", "`pt-br`", and "`PT-br`" (Brazilian Portuguese) are considered identical.

Usually the language code is represented in lowercase and the country code in uppercase, but this is not a rule.

User-Defined Codes

In some cases the list of variant codes you can build from the predefined language and region codes is not enough.

For instance, as we have seen already, you might have to localize a document in two types of Spanish: one for the audience in Spain (Iberian Spanish) and the other for the Latin American market. The first should be coded "`es-ES`", or simply "`es`" because Spain is the default country for Spanish. For the second, however, no country code corresponds to "Latin America." To solve this you can create your own locale codes as defined by `UserCode` in Listing 3.1. For example, you could use something such as "`x-es-LatAm`" for your Latin-American Spanish document.

A special kind of user-defined code exists: the one registered to the IANA. Most of them start with the prefix `i-`. The list of these language tags is updated regularly and you can find it at <http://www.iana.org/assignments/language-tags>.

NOTE

Be aware that some localization tools might be programmed to handle only 4-letter codes, and might not be able to process IANA or user-defined codes correctly.

For a detailed list of language codes, see Appendix D.

Multilingual Documents

As you saw in Chapter 2, "Character Representation," one characteristic of XML is its capability to handle content in different languages when necessary.

For example, as shown in Listing 3.2, a SOAP data file could store description of an item in several languages.

Listing 3.2 soap1.xml—SOAP Envelope with Multilingual Entries

```
<!-- SOAP excerpt -->
<Envelope
  xmlns="http://schemas.xmlsoap.org/soap/envelope/"
  encodingStyle="http://schemas.xmlsoap.org/soap/envelope/">
  <Body>
    <d:GetItem xmlns:d="uri:myData" xml:lang="en">
      <d:PartNum>NCD-67543</d:PartNum>
      <d:InStock>5</d:InStock>
      <d:Desc>Manual water pump</d:Desc>
      <d:Desc xml:lang="fr">Pompe à eau manuelle</d:Desc>
      <d:Desc xml:lang="ja">3F3F3F3F3F3F3FB73F3F3F</d:Desc>
    </d:GetItem>
  </Body>
</Envelope>
```

The default language from the `<d:GetItem>` element level is set to `en` (English). The child elements inherit the property, so the first `<d:Desc>` element does not need to repeat the attribute. However, because the second one contains the description in French, you need to override the default `xml:lang` attribute.

Always keep in mind that XML element and attribute names can have non-ASCII characters as well. In such occurrences, the language specifications work the same. Listing 3.3 shows the same SOAP envelope, but this time with the user data marked up with a Russian vocabulary. The data are identical and the `xml:lang` mechanism is expected to behave the same: It applies to the content, not to the tags.

Listing 3.3 soap2.xml—SOAP Envelope with Multilingual Entries and Some Non-ASCII Elements

```
<!-- SOAP excerpt -->
<Envelope
  xmlns="http://schemas.xmlsoap.org/soap/envelope/"
  encodingStyle="http://schemas.xmlsoap.org/soap/envelope/">
  <Body>
    <E4:Äîñðàðüîáúâêð
      □xmlns:E4="uri:%D0%9C%D0%BE%D0%B8%D0%94%D0%B0%D0%BD%D0%BD%D1%8B%D0%B5"
```

```

<?xml:lang="en">
<E4:íîîáðîíáúáêðà>NCD-67543</E4:íîîáðîíáúáêðà>
<E4:Áíàëè+èè>5</ä:Áíàëè+èè>
<E4:îîèñàíèà>Manual water pump</E4:îîèñàíèà>
<E4:îîèñàíèà xml:lang="fr">Pompe à eau manuelle</E4:îîèñàíèà>
<E4:îîèñàíèà xml:lang="ja">3F3F3F3F3F3F3F3F73F3F3F</E4:îîèñàíèà>
</E4:Áíñðàððüíáúáêðð>
</Body>
</Envelope>

```

Note the value of the `xmlns` attribute: The namespace prefix `ä` is associated to a URI reference (`íîîèÄàííúá`), but here the URI has already been coded into its UTF-8/escaped form as described in Chapter 2.

The lang Attribute in XHTML

For historical reasons, in addition to `xml:lang`, XHTML also allows the attribute `lang` to specify language switch. Both have exactly the same significance.

In case the same element has both `xml:lang` and `lang` with two different values, `xml:lang` takes precedence over `lang`.

NOTE

Using `xml:lang` or `lang` has no direct impact on the way the text is rendered. For example, specifying a paragraph as Arabic does not trigger right-to-left display. You must use the style sheets and the various internationalization elements and attributes such as `<bdo>`, `<dir>`, and `<ruby>` for XHTML to indicate to the user-agent how the text should be displayed.

However, take into account that language is important in some cases: for example, to select an appropriate font. If a document is encoded in UTF-8 or UTF-16, there is no easy way to distinguish Chinese from Japanese, because most ideographs have been unified.

The lang() Function in XPath

XPath is the language used in various XML applications to specify a "path notation" that allows you to navigate through the hierarchical structure of any XML document. It is also used to test whether the node of a document instance matches a given pattern. XPath is used, for example, in conjunction with XPointer and XSLT.

XPath designers have wisely provided a function to match languages: `lang()`.

The function uses the `xml:lang` attribute to match a given parameter. This is very useful because, following the XML specifications, the function is not case sensitive and allows you to match a language value very simply.

When you specify only a language code rather than a locale code (for example, `en` versus `en-GB`), the function returns true for any attributes where the first part of its value matches the argument. The separator between both parts of the value is '-'. Consider the following XSL statement:

```
<xsl:for-each select="lang('es')">
```

When this command is used on the XML document shown in Listing 3.4, it will return true for all the following elements:

```
<p xml:lang="es">Spanish text</p>
<p xml:lang="ES">Spanish text</p>
<p xml:lang="es-ES">Iberian Spanish text</p>
<p xml:lang='es-mx'>Mexican Spanish text</p>
```

Listing 3.4 `spanish.xml`— Multilingual Document with Different Spanish Flavors

```
<?xml version="1.0" ?>
<document>
  <p xml:lang="es">Spanish text</p>
  <p xml:lang="fr">French text</p>
  <p xml:lang="ES">Spanish text</p>
  <p xml:lang="CA-es">Catalan text</p>
  <p xml:lang="es-ES">Iberian Spanish text</p>
  <p xml:lang="es-mx">Mexican Spanish text</p>
</document>
```

Keep in mind that not all XSL processors support all XSL features yet. The `lang()` function is not supported in all browsers, for example.

Space Handling

Although internationalization is often about separating the presentation information from the content, a few instances exist where the presentation parameters must be known by the tools for a more efficient translation. One of them is the information about how to handle white spaces.

White spaces are defined as spaces, tabs, carriage returns, and line-feeds:

```
WS ::= (#x20 | #x9 | #xD | #xA)+
```

You will notice that other "space"-like characters such as `NO-BREAK SPACE (U+00A0)`, `IDEOGRAPHIC SPACE (U+3000)`, `EM SPACE (U+2003)`, `THIN SPACE (U+2009)`, `EN QUAD (U+2000)`, and so forth are not included in the white space list. They are treated just like regular characters as far as XML processors are concerned.

As for the language, XML defines a special attribute to indicate how white spaces should be handled in a given element set: `xml:space`.

The attribute can have two values: `default` or `preserve`. The first one lets the XML processor behave as its default mechanism is set, whereas the second one indicates that all white spaces must be preserved and passed without transformation.

If you use a DTD to specify your format, `xml:space` must be declared, just as any other attribute:

```
<!ATTLIST SourceCode
  xml:space (default|preserve) 'preserve' >
```

or

```
<!ATTLIST pre
  xml:space (preserve) #FIXED 'preserve'>
```

Always keep in mind that `xml:space` is an indicator for the parser, not the rendering engine, although some rendering engines are taking it into consideration (such as Adobe's SVG viewer).

Localization tools should take into account the presence of `xml:space` when extracting content. It is the best indicator to specify whether the white spaces of a run of text should be left alone. This information should be carried during the translation.

Listing 3.5 shows an XML file where the element `<cmdline>` contains preformatted text, while the multiple spaces in the `<p>` element should be reduced to a single blank.

Listing 3.5 `spaces1.xml`—Usage of the `xml:space` Attribute

```
<?xml version="1.0" ?>
<doc>
  <cmdline id="1" xml:space="preserve">Command line:
-x      run the tool with option x
-f[name] specify [name] for font</cmdline>
  <p id="2">Text where
any set of white spaces is reduced to 1.</p>
</doc>
```

XHTML

The XHTML specifications add a few clauses to the handling of white spaces.

In addition to line-breaks, tabulations, and space, the characters FORM FEED (U+000C) and ZERO WIDTH SPACE (U+200B) must also be treated as white spaces.

Leading and trailing white spaces in block elements should be removed unless the `xml:space` attribute is set to `preserve`. In other words, the following XHTML fragments are identical to one another.

```
<p> This is an example </p>
<p>
This is an
example
</p>
<p>This is an example</p>
```

CSS

When rendering is involved, you can use the `white-space` property of CSS to specify how the preformatting should be handled. The values available are `normal`, `pre`, `nowrap`, and `inherit`.

If you take the document shown in Listing 3.5 and apply to it the style sheet displayed in Listing 3.6, you can see in Figure 3.1 that the rendering of the `<cmdline>` is done correctly.

Listing 3.6 `spaces3.css`—Style Sheet used to Display Figure 3.1

```
doc {
  display: block;
  margin-top: 10px; margin-left: 10px;
}

p {
  display: block;
  margin-bottom: 10px;
}

cmdline {
  display: block;
  margin-bottom: 10px;
  white-space: pre; font-family: "Courier New";
}
```

Figure 3.1

Rendering of white spaces with the `white-space` CSS attribute in Navigator 6.

Note that the same example would not work with Internet Explorer 5.5, which does not support the `white-space` property correctly yet (version 5.5.4522.1800, with SP1).

Date and Time Representation

As is true for any software application, XML documents should store date and time information in a locale-independent manner or with enough information about what locale was used to format it.

Time stamps will become more and more important as distributed applications across several time zones will increase the complexity of synchronizing tasks.

One challenge with XML is that documents can be used for a wide range of applications and can serve many different purposes. Ideally, storing date/time information as a number would be the best way to proceed: The correct formatting could be applied at rendering time. However, this is practical only if you process the documents with your own application, such as SOAP.

In many cases, you will have XML documents in which the user will interact directly with the file, without the benefit of a reformatting of the content. In such occurrences, the date/time must be directly readable by the user.

ISO Format

As always, the best way to implement locale-specific information is to use existing standards when possible. ISO offers a wide palette of standardized representations for date, time, duration, and intervals in the ISO 8601:1988 specifications.

Each application can have different requirements and you should pick the format that best fits the type of information you need to represent.

As an example, you can look at the TMX format. This XML standard for translation memory exchange uses several attributes related to date/time information: `changedate`, `creationdate`, and `lastusedate`. All three of them use the same format:

```
YYYYMMDDThhmmssZ
```

For instance, the string `20000811T133402Z` represents August 11, 2000 at 1:34 p.m. 2 seconds, in UTC.

Listing 3.8 presents a short TMX document (just one entry: a software string in English and Esperanto) generated by a translation memory utility that demonstrates how the different attributes are used.

Listing 3.8 `Date_Time.tmx`—TMX Document with the Three Date/Time Attributes

```
<?xml version="1.0" ?>
<!DOCTYPE tmx SYSTEM "tmx12.dtd">
<tmx xmlns="http://www.lisa.org/tmx" version="1.2">
  <header creationtool="Rainbow"
    creationtoolversion="2.00-1"
    datatype="PlainText"
    segtype="sentence"
    adminlang="en-US"
    srclang="en-US"
    o-tmf="Rainbow"
    creationdate="20000101T163812Z"
    creationid="YvesS"
    changedate="20000314T023401Z"
    changeid="Amity"
    o-encoding="iso-8859-3">
  </header>
  <body>
    <tu tuid="0001"
      datatype="Text"
      usagecount="2"
      lastusedate="20000314T023401Z">
      <tuv lang="EN"
        creationdate="20000212T153400Z"
        <seg>Search for PATTERN in each FILE or standard input.</seg>
      </tuv>
      <tuv lang="EO"
        creationdate="20000309T021145Z"
        changedate="20000314T023401Z"
        <seg>Serci pri SABLONO en ciu DOSIERO au la normala enigo.</seg>
      </tuv>
    </tu>
  </body>
</tmx>
```

When developing a new schema or porting an XML DTD to a schema definition, you might want to take advantage of the predefined types for date and time. The XML Schema offers a collection of predefined types for date, time, duration, and intervals.

For example, the three date/time-related attributes we have seen in TMX would be of the type `timeInstant`, as in the excerpt below:

```
<attribute name='lastusedate' type='timeInstant' use='optional' />
<attribute name='creationdate' type='timeInstant' use='optional' />
```

```
<attribute name='changedate' type='timeInstant' use='optional' />
```

Summary

In this chapter we have seen three types of information often encountered in XML documents. Make sure you address them when developing your own XML schema or when writing localization tools.

First, use `xml:lang` to indicate the different locales within your documents.

Second, make sure preformatted blocks of text are clearly indicated to the XML parser, so that tools can carry on that information during translation and avoid unnecessary reformatting. The attribute `xml:space` is a good way to do this.

Third, for any date or time related attribute, use one of the forms of the ISO 8601:1988 standard, or a locale-independent raw numerical value if no human is meant to read the XML document.

References

The following is a list of some online reference documents. Make sure you get the latest information.

ISO 639-2, Language Codes (Library of Congress Network Development & MARC Standards Office): <http://lcweb.loc.gov/standards/iso639-2>

ISO 3166 country codes (ISO 3166 Maintenance Agency):
<http://www.din.de/gremien/nas/nabd/iso3166ma/index.html>

IANA Language Tags: <http://www.iana.org/assignments/language-tags>

RFC1766, Tags for the Identification of Languages: <http://www.ietf.org/rfc/rfc1766.txt>

RFC3066, Tags for the Identification of Languages: <http://www.ietf.org/rfc/rfc3066.txt>

White spaces in XML: <http://www.w3.org/TR/REC-xml#sec-white-space>

White spaces in XHTML: <http://www.w3.org/TR/xhtml1/#uaconf>

XML Schema Part 2: Datatypes: <http://www.w3.org/TR/xmlschema-2>

ISO 9801:1988 Date and Time Formats (International Organization for Standardization):
<http://www.iso.ch>

© Copyright Pearson Education. All rights reserved.