XML Localisation Interchange File Format as an intermediate file format

Rodolfo M. Raya (rmraya@maxprograms.com)
Chief Technical Officer, Maxprograms
First published by IBM developerWorks, Oct 2004

The first article in this series briefly explained the most relevant XML standards used in the localisation industry. This second part focuses on XML Localisation Interchange File Format (XLIFF) and explains with practical examples how to use it for translating different kinds of documents. This article presents a step-by-step guide to translating multilingual documents using XLIFF as an intermediary file format, and provides useful resources for localizing Java applications.

XLIFF is a format that's used to exchange localisation data between participants in a translation project. This special format enables translators to concentrate on the text to be translated, without worrying about text layout. The XLIFF standard is supported by a large group of localisation service providers and localisation tools providers.

The most important reason for you to use XLIFF when translating documents is that you can use a single file format when translating different kinds of documents.

In my previous article, I described the steps to follow in a localisation project:

1. Extract all translatable text from the original documents.
2. Store the extracted strings in a special XML document.
3. Send out the XML document for translation.
4. Extract the translated strings from the XML document and reinsert them into the original documents.

The special XML document mentioned in step 2 is an XLIFF document. This article will focus on steps 1, 2, and 4.

Converting documents to XLIFF format

I'll now show you a translation process that uses XLIFF files as an intermediary format. Figure 1 illustrates the steps that you will follow.

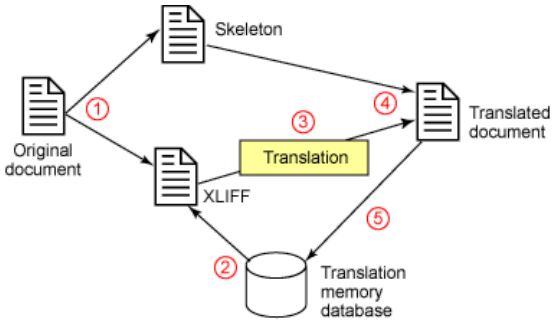

Figure 1. Translation process

The process can now be reformulated with more detail as follows:

1. **Text extraction:** Separation of translatable text from layout data.
2. **Pre-translation:** Addition of existing translation to the XLIFF file generated in the previous step.
3. **Translation:** Performed by a professional translator.
4. **Reverse conversion:** Generation of a translated document from the translated XLIFF file.
5. **Translation memory improvement:** Storage of new translations in a translation memory (TM) database for later reuse.

Text extraction

To aid the translator, translatable text is separated from text layout information. Listing 1 shows the code of a simple HTML page. This page contains two translatable sentences and several tags that are irrelevant for a translator.

```
<html>
   <head>
      <title>A Title</title>
   </head>
   <body>
      <p>One paragraph</p>
   </body>
 </html>
```

Listing 1. Simple HTML page

CONTENTS

Special programs called **filters** separate text and layout. Computer Aided Translation (CAT) tools usually have filters for the most common formats: HTML, RTF, XML, and plain text.

Filters store the non-translatable portions in special files called **skeletons**. All translatable sentences are replaced by special marks in the skeleton. Listing 2 shows a sample skeleton for the text in Listing 1.

```
<html>
   <head>
      <title>%%%1%%%</title>
   </head>
   <body>
      <p>%%%2%%%</p>
   </body>
</html>
```

Listing 2. Skeleton file

Each text fragment is stored in a translation unit element (<trans-unit>) in an XLIFF file. The mark used in the skeleton can be used as an id attribute for the translation unit to simplify the mapping between the skeleton and the XLIFF file.

The XLIFF format definition as stated in the formal specification is concise, clear, and practical:

> XLIFF is an XML application, as such it begins with an XML declaration. After the XML declaration comes the XLIFF document itself, enclosed within the <xliff> element. An XLIFF document is composed of one or more sections, each enclosed within a <file> element. The <file> element consists of a <header> element, which contains metadata about the <file>, and a <body> element, which contains the extracted translatable data from the <file>. The translatable data within <trans-unit> elements is organized into <source> and <target> paired elements. These <trans-unit> elements can be grouped recursively in <group> elements.

Listing 3 contains an XLIFF file for the HTML document shown in Listing 1.

```
<? xml version="1.0" ?>
<xliff version="1.0">
  <file original="sample.html"
        source-language="en"
        datatype="HTML Page">
    <header>
      <skl>
        <external-file href="sample.skl"/>
      </skl>
    </header>
    <body>
      <trans-unit id="%%%1%%%">
        <source xml:lang="en">A Title</source>
      </trans-unit>
      <trans-unit id="%%%2%%%">
        <source xml:lang="en">One paragraph</source>
      </trans-unit>
    </body>
  </file>
</xliff>
```

Listing 3. Sample XLIFF file

The complexity of a filter program depends on the format that has to be parsed. HTML and XML are well-documented formats. Filter programs use certain specifications as reference to distinguish between translatable text and formatting information. Word processors usually have proprietary file formats. Fortunately, most word processors can export and import documents in RTF, another well-documented format.

The accompanying source code contains a complete Java-language implementation of a filter for Java properties files (see Resources).

Dealing with formatting information

Table 1 shows two versions of the same sentence, one in HTML format and the other in RTF.

HTML	RTF
This is ****bold**** text	This is **\b** bold**\b0** text

Table 1. HTML and RTF formatting

Both versions contain the same text -- only the formatting codes are different. Markup code should not be translated. A translator who uses an HTML editor and a word processor has to translate twice -- once for each tool. The translator needs to work with a single tool that can show the relevant text, hide markup information, and allow for reuse of translations despite differences in markup.

Text layout information is stored in special elements called **inline elements** that the XLIFF standard defines for that purpose. Listing 4 shows the markup of the examples in Table 1 enclosed in the appropriate inline elements.

Notice that the elements used to wrap markup are identical; only their content varies. This allows a CAT tool to search for translations in its TM database using the information provided in the attributes. For example, if the database contains a translation for the HTML version, it can be used with the RTF version by automatically adjusting the content of the <bpt> and <ept> elements.

Translators should not see the inline elements as XML when translating with a CAT tool. They must be presented in a friendly way, either as an image or as a text mark that can be inserted easily by clicking a button or using shortcut keys.

Table 2 contains the complete list of inline elements and explanations of their usage.

Element	Description
<g>	**Generic group placeholder:** The <g> element is used to replace any inline code in the original document that has a beginning and an end, does not overlap other paired inline codes, and can be moved within its parent structural element.
<x/>	**Generic placeholder:** The <x/> element is used to replace any code in the original document.
<bx/>	**Begin paired placeholder:** The <bx/> element is used to replace a beginning paired code in the original document. It should be used for paired codes that do not follow XML well-formedness rules (meaning, no overlapping elements).
<ex/>	**End paired placeholder:** The <ex/> element is used to replace an ending paired code in the original document. It should be used for paired codes that do not follow XML well-formedness rules (meaning, no overlapping elements).
<bpt>	**Begin paired tag:** The <bpt> element is used to delimit the beginning of a paired sequence of native codes. Each <bpt> has a corresponding <ept> element within the translation unit.
<ept>	**End paired tag:** The <ept> element is used to delimit the end of a paired sequence of native codes. Each <ept> has a corresponding <bpt> element within the translation unit.
<ph>	**Placeholder:** The <ph> element is used to delimit a sequence of native standalone codes in the translation unit.
<it>	**Isolated tag:** The <it> element is used to delimit a beginning/ending sequence of native codes that does not have a corresponding ending/beginning within the translation unit.
<mrk>	**Marker:** The <mrk> element delimits a section of text that has special meaning, such as a terminological unit, a proper name, or an item that should not be modified

Table 2. Inline elements

```
1: <source>This is
     <bpt id="1" ctype="bold">&lt;b&gt;</bpt>
     bold
     <ept id="1">&lt;/b&gt;</ept>
     text
   </source>

2: <source>
     This is
     <bpt id="1" ctype="bold">\b</bpt>
     bold
     <ept id="1">\b0</ept>
     text
   </source>
```

Listing 4. Markup of Table 1 examples with inline elements

Segmentation

Once the extracted text is examined and all markup is wrapped with the correct inline elements, a new process called **segmentation** can be applied to the translation unit. Segmentation is the breaking down of a block of text into smaller translatable segments of text, such as a sentence, a paragraph, or a phrase. It is important to keep translation units as small as possible to maximize the chances of finding usable translations in the TM database.

Usually, a segmenter (the tool that performs segmentation) breaks up text according to these basic rules:

- Break when a period or full stop is followed by a white space; this is assumed to be the end of a sentence or paragraph.
- Break when a semicolon is followed by a white space; this is assumed to be the end of a conjunction or phrase.
- Break after a colon followed by a white space; this is assumed to be the start of a list.

Special care is required when breaking after a period because the period could be part of an abbreviation.

The segmentation rules listed above are not applicable for Asian languages such as Chinese, Japanese, and Korean.

Translating documents

Actual translation must be performed by a human translator. A machine can't substitute for a human here; it doesn't matter how good a translation program is, the translation must be checked by a real person to ensure quality.

It is possible to assist the translator by providing translations of similar, perhaps identical, texts made previously. A professional translator can decide if a given translation is good enough to reuse.

After the original document has been converted to XLIFF, the tool used to perform conversion can iterate over all translation units and search for matching translations in a TM database. Whenever a suitable match is found, an <alt-trans> element is added to the translation unit. This process is usually called **pre-translation**.

The text found in the database does not need to be identical to the text that needs translation. The example in Listing 5 shows how translations from HTML and RTF documents can be mixed in a translation unit. Notice that the second <alt-trans> element has a match quality of 96% due to the differences in markup format, but the text is good enough to be accepted.

```
<trans-unit approved="no" id="5" resname="res5" xml:space="default">
  <source xml:lang="en">
    This is <bpt id="1" ctype="bold"><b></bpt>bold<ept id="1"></b></ept> text
  </source>
  <target xml:lang="es"/>
  <alt-trans match-quality="100" origin="web" tool="TM Search">
    <source xml:lang="en">
      This is <bpt id="1" ctype="bold"><b></bpt>bold<ept id="1"></b></ept> text
    </source>
    <target xml:lang="es">
```

```
      Este es texto <bpt id="1" ctype="bold"><b></bpt>enfatizado<ept id="1"></b></ept>
    </target>
  </alt-trans>
  <alt-trans match-quality="96" origin="rtf" tool="TM Search">
    <source xml:lang="en">
      This is <bpt id="1" ctype="bold">\b</bpt>bold<ept id="1">\b0</ept> text
    </source>
    <target xml:lang="es">
      Este es texto <bpt id="1" ctype="bold">\b</bpt>enfatizado<ept id="1">\b0</ept>
    </target>
  </alt-trans>
</trans-unit>
```

Listing 5.

Converting translated documents back to original format

When the XLIFF file is finally ready, it is sent to a professional translator. The translator then uses an XLIFF-enabled CAT tool to add all the missing translations and to verify the ones provided at the pre-translation stage.

The translated XLIFF must now be merged with the skeleton file to produce a translated document in the desired output format.

A filter is used to read the skeleton and process all special marks sequentially. For each mark found in the skeleton, a corresponding translation unit is revised in the XLIFF file; if the translation unit has been marked as approved by the translator, the text in the <target> element replaces the mark in the skeleton. If there is no translation for the segment, or if the included translation is not approved, the text in the <source> element is used instead. After all marks have been replaced with the corresponding text from the XLIFF file, the skeleton becomes a translated document and should be saved under a new name.

Most document formats require fixes in the layout of the translated document; the XML, HTML, and RTF formats generally require the fewest post-translation adjustments.

One final task remains: extraction of <source>/<target> pairs from the approved <trans-unit> elements of the XLIFF file. Store these pairs in the TM database for later reuse. These pairs are usually stored in a special XML format called **Translation Memory eXchange (TMX)**, which all important translation tools support.

Sample application

The accompanying Java source code (see Resources) contains two filters:

- A program that converts Java properties files to XLIFF format
- A program that performs the reverse conversion, from XLIFF to Java properties

Properties files are used in Java programs to store the text displayed by the application GUI. These are text files that can have multiple lines, where each line is either a comment, a blank line, or a <key>=<value> entry.

Segmentation and pre-translation routines are not included in the sample code as they add complexity beyond the scope of this article. Nevertheless, the code contains placeholders where the reader can implement those processes, if desired.

Summary

This article has shown how to translate a document using XLIFF as an intermediate file format, explaining all the stages of the process with simple examples. The included source code demonstrates in a practical way how to implement an XLIFF-based solution for Java localisation projects, a solution that can be extended to support other formats.

The next article in the series will explain how a TM database works, emphasizing the relevance of the TMX format for translation exchange.

Resources

- Download the file x-localis2_filters.zip, which contains Java implementations of filters that convert Java Properties files to XLIFF and back.
- Read the previous article in this series, "XML in localisation: A practical analysis" (developerWorks, August 2004). It explains the role of XLIFF in the localisation process and its interaction with other related XML standards.
- Read the XLIFF 1.2 Specification, which defines the XML Localization Interchange File Format (XLIFF). The purpose of this vocabulary is to store localizable data and carry it from one step of the localization process to the other, while allowing interoperability between tools.
- Download a free trial of Swordfish Translation Editor from Swordfish's download site and use it to translate your own XLIFF files generated with the sample code.
- Peruse this white paper on version 1.1 of XLIFF (PDF file, XLIFF Technical Committee, July 2003). It contains interesting and useful information about the history, architecture, and usage of the XLIFF standard.
- Read the developerWorks article "Application Framework for e-business: Globalization" (July 2001), which provides resourceful reading for developers working on Web-oriented applications aimed at the global market.
- Find more XML resources on the developerWorks XML zone.
- Finally, find out how you can become an IBM Certified Developer in XML and related technologies.

About the author

Rodolfo Raya is Maxprograms' CTO (Chief Technical Officer), where he develops multi-platform translation/localisation and content publishing tools using XML and Java technology. He can be reached at rmraya@maxprograms.com.

Copyright © 1999-2015 Maxprograms. All rights reserved.