XLIFF 1.2 Representation Guide for HTML

Committee Draft 02

16 October 2006

Specification URIs:

This Version:
http://docs.oasis-open.org/xliff/v1.2/xliff-profile-html/xliff-profile-html-1.2-cd02.html
http://docs.oasis-open.org/xliff/v1.2/xliff-profile-html/xliff-profile-html-1.2-cd02.pdf

Latest Version:
http://docs.oasis-open.org/xliff/v1.2/xliff-profile-html/xliff-profile-html-1.2.html
http://docs.oasis-open.org/xliff/v1.2/xliff-profile-html/xliff-profile-html-1.2.pdf

Technical Committee:
OASIS XML Localisation Interchange File Format (XLIFF) TC

Chair(s):
Bryan Schnabel
Tony Jewtushenko

Editor(s):
Yves Savourel
Bryan Schnabel
Tony Jewtushenko
Doug Domeny

Abstract:

This document describes how HTML (in its different flavors), should be coded when extracted to an XLIFF document.

Status:

This document was last revised or approved by the XLIFF TC on the above date. The level of approval is also listed above. Check the "Latest Version" or "Latest Approved Version" location noted above for possible later revisions of this document.

Technical Committee members should send comments on this specification to the Technical Committee's email list. Others should send comments to the Technical Committee by using the "Send A Comment" button on the Technical Committee's web page at http://www.oasis-open.org/committees/xliff/.

For information on whether any patents have been disclosed that may be essential to implementing this specification, and any offers of patent licensing terms, please refer to the Intellectual Property Rights section of the Technical Committee web page http://www.oasis-open.org/committees/xliff/ipr.php.

The non-normative errata page for this specification is located at http://www.oasis-open.org/committees/xliff/.

Notices

Table of Contents

Appendices

1. Introduction

As different tools may provide different filters to extract the content of HTML documents it is important for interoperability that they represent the extracted data in identical manner in the XLIFF document.

1.1. Purpose

The intent of this document is to provide a set of guidelines to represent HTML data in XLIFF. It offers a collection of recommended mapping of the HTML elements and attributes developers of XLIFF filters can implement, and users of XLIFF utilities can rely on to insure a better interoperability between tools.

1.2 Transitional and Strict

XLIFF is specified in two "flavors". Indicate which of these variants you are using by selecting the appropriate schema. The schema may be specified in the XLIFF document itself or in an OASIS catalog. The namespace is the same for both variants. Thus, if you want to validate the document, the tool used knows which variant you are using. Each variant has its own schema that defines which elements and attributes are allowed in certain circumstances.

As newer versions of XLIFF are approved, sometimes changes are made that render some elements, attributes or constructs in older versions obsolete. Obsolete items are deprecated and should not be used even though they are allowed. The XLIFF specification details which items are deprecated and what new constructs to use.

- Transitional - Applications that produce older versions of XLIFF may still use deprecated items. Use this variant to validate XLIFF documents that you read. Deprecated elements and attributes are allowed.

```
xsi:schemaLocation='urn:oasis:names:tc:xliff:document:1.2 xliff-core-1.2-transitional.xsd'
```

- Strict - All deprecated elements and attributes are not allowed. Obsolete items from previous versions of XLIFF are deprecated and should not be used when writing new XLIFF documents. Use this to validate XLIFF documents that you create.

```
xsi:schemaLocation='urn:oasis:names:tc:xliff:document:1.2 xliff-core-1.2-strict.xsd'
```

2. General Considerations

This section discusses the general considerations to take in account when extracting HTML data.

2.1. HTML Flavors

There are many flavors of HTML that are used. HTML 4.01, XHTML, etc. There are also, probably in even greater quantity, many pages that are considered HTML but are not valid HTML. This document tries to address all these different flavors. Therefore, some of the examples may contain HTML elements that have been deprecated in the latest version of HTML.

In this document the term "HTML" is used generically, to designate any of the flavors. If the text refers to a specific flavor, it uses the more complete name for that flavor: for example "HTML 4.01", "XHTML", "XHTML 1.0", etc.

2.2. Server-Side Files

Many HTML documents are generated dynamically, in some cases using server-side script files which are often made of a mixture of HTML constructs and server-side instructions written in one of the server-side languages such as PHP, JSP, ASP, or many others.

While such source documents are generally outside of the scope of this document, an effort is made to try to address some of the issues you may run into when extracting such source documents.

2.3. Extraction Techniques

There are many ways to process a source HTML document and create its corresponding XLIFF output.

2.3.1. Using XSLT

One interesting method is to make use of XML standards, such as XSLT, XPath, or XSL-FO. Of these, XSLT is a particularly good tool for transforming HTML to XLIFF, and XLIFF back to HTML. See the Appendix "Example of XSLT Use to Process HTML" for a concrete example of how go back and forth between HTML and XLIFF.

XSLT works on any well-formed XML documents. So the input HTML document has to be a valid XHTML document or a well-formed HTML document.

If the input file is not at that stage, it can be pre-processed first, using tools such as Perl [Perl], HTML Tidy [HTMLTidy], or other utilities. They all can provide a good way to streamline the pre-processing task. See the Appendix "Pre-Processing HTML Files" for an example of how to use Perl to pre-process an HTML document not well-formed into a well-formed XML document.

2.3.2. Using Filters

Another method to extract HTML files to XLIFF is to use custom filter applications. Such tools can be written in a variety of programming and scripting languages such as Perl, Python, C, C++, C#, Java, and so forth.

This document makes no assumption on the type of language used to process the HTML input documents. It also makes no assumptions whether or not the tool creates a Skeleton file along with the XLIFF document generated, or if it creates one, how data are represented in the Skeleton.

2.4. Including Escaped Markup

The XLIFF specification allows for marking "beginning tags" and "ending tags" (<bpt, <ept) in a way that the markup may be escaped and preserved. This is generally seen as a way for non-XSLT based tools to abstract the markup. However, XSLT does not parse escaped code efficiently. Since there are efficient alternate ways to preserve the HTML code, it is not recommended to use the <bpt and <ept tags.

While the following HTML could be expressed using the <bpt and <ept tags, along with escaped HTML code:

```
  <i>picabo, big-air</i>,
 and <i> yard-sale</i>
```

like this:

```
  <bpt id='1-2'>&lt;i&gt;</bpt>picabo,
 big-air<ept id='1-2'>&lt;/i&gt;</ept>, and
 <bpt id='1-3'>&lt;i&gt;</bpt> yard-sale<ept id='1-3'>&lt;/i&gt;</ept>
```

It is recommended to use the cleaner, more XSLT-friendly approach, like this:

```
  <g id='n1' ctype='italic'>picabo, big-air</g>,
 and <g id='n2' ctype='italic'> yard-sale</g>
```

Although most of the XLIFF inline tags are represented in the TMX standard, the 'g' and 'x' tags are not. TMX is a standard to exchange Translation Memory (TM) data created by Computer Aided Translation (CAT) and localization tools. If you plan to store or deliver XLIFF text content using TMX, you may wish to use the 'bpt'/'ept' approach or you'll need to represent 'g' and 'x' tags in some alternate way in TMX.

2.5. Scope of Extraction

Representing an HTML document in XLIFF could be done in sundry ways. Those different representations however, can be grouped in two main categories one could simplistically label the "*maximalist*" and the "*minimalist*" approaches.

In the maximalist approach the XLIFF document tries to include as much original information as possible. In fact, an ideal XLIFF representation in this view is to be able to re-create the original HTML file completely from the XLIFF document. Such approach could be useful, for example, for tools that want to display the final HTML from the XLIFF data, using only XSL transformations.

In the minimalist approach, the XLIFF document contains strictly the parts needed for the translation, while most of the non-localizable parts are stored in the Skeleton file.

It is recommended to use a maximalist approach as often as possible, to insure a better interoperability between XLIFF documents generated with different filters. This document provides a framework for both approaches.

As an example, consider the following HTML file:

```
<html>
 <body>
  <h1 class="title">Report</h1>
  <table border="1" width="100%">
   <tr>
    <td valign="top">Text in cell r1-c1</td>
    <td valign="top">Text in cell r1-c2</td>
   </tr>
   <tr>
    <td bgcolor="#C0C0C0">Text in cell r2-c1</td>
    <td>Text in cell r2-c2</td>
   </tr>
  </table>
  <p>All rights reserved (c) Gandalf Inc.</p>
 </body>
</html>
```

This file can be completely mapped to XLIFF if so desired. In the following XLIFF document, the parts that need to have a "maximal" representation are in blue. Those would be omitted for a "minimal" representation, as shown in the next XLIFF output. In this latter case the parts not in the XLIFF document must be preserved in the Skeleton file, or inferred at merge time, to re-create a translated file with all the information in the original HTML file.

Example of a "maximalist" representation (all the original HTML data are mapped):

```
<xliff xmlns='urn:oasis:names:tc:xliff:document:1.2'
       xmlns:html='http://www.w3.org/1999/xhtml'
       xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
       xsi:schemaLocation='urn:oasis:names:tc:xliff:document:1.2 xliff-core-1.2-transitional.xsd'
       version='1.2'>
 <file original='sample.htm' source-language='en' datatype='html'>
  <body>
  <group restype='x-html-html'>
   <group restype='x-html-body'>
    <trans-unit id='1' restype='x-html-h1' html:class="title">
     <source xml:lang='en'>Report</source>
    </trans-unit>
    <group restype='table' html:border="1" html:width="100%">
     <group restype='row'>
      <trans-unit id='2' restype='cell' html:valign="top">
       <source xml:lang='en'>Text in cell r1-c1</source>
      </trans-unit>
      <trans-unit id='3' restype='cell' html:valign="top">
       <source xml:lang='en'>Text in cell r1-c2</source>
      </trans-unit>
     </group>
     <group restype='row' html:bgcolor="#C0C0C0">
      <trans-unit id='4' restype='cell'>
       <source xml:lang='en'>Text in cell r2-c1</source>
      </trans-unit>
      <trans-unit id='5' restype='cell'>
       <source xml:lang='en'>Text in cell r2-c2</source>
      </trans-unit>
     </group>
    </group>
    <trans-unit id='6' restype='x-html-p'>
     <source xml:lang='en'>All rights reserved (c) Gandalf Inc.</source>
    </trans-unit>
   </group>
  </group>
  </body>
 </file>
</xliff>
```

Example of a "minimalist" representation (un-mapped parts are stored in the Skeleton file):

```
<xliff xmlns='urn:oasis:names:tc:xliff:document:1.2'
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance'
  xsi:schemaLocation='urn:oasis:names:tc:xliff:document:1.2 xliff-core-1.2-transitional.xsd'
  version='1.2'>
 <file original='sample.htm' source-language='en' datatype='html'>
  <header><skl><external-file href='the_skeleton_file.skl' /></skl></header>
  <body>
  <trans-unit id='1' restype='x-html-h1'>
   <source xml:lang='en'>Report</source>
  </trans-unit>
  <trans-unit id='2' restype='cell'>
   <source xml:lang='en'>Text in cell r1-c1</source>
  </trans-unit>
  <trans-unit id='3' restype='cell'>
   <source xml:lang='en'>Text in cell r1-c2</source>
  </trans-unit>
  <trans-unit id='4' restype='cell'>
   <source xml:lang='en'>Text in cell r2-c1</source>
  </trans-unit>
  <trans-unit id='5' restype='cell'>
   <source xml:lang='en'>Text in cell r2-c2</source>
  </trans-unit>
  <trans-unit id='6' restype='x-html-p'>
   <source xml:lang='en'>All rights reserved (c) Gandalf Inc.</source>
  </trans-unit>
  </body>
 </file>
</xliff>
```

Any representation between these two extremes would be also acceptable. For instance, one could choose to have `<group>` elements for the table but not for each row.

2.6. Order of Extraction

The flow of the extracted data in the XLIFF document should be in the same order as the flow of data in the original HTML document, regardless of any layout placement. In other words, how the text is stored in the source document and how it is processed (most of the time displayed) by the user agent are two different things. The extraction order should reflect the order of the data in the source document, and the author is responsible to group logical parts of the text together as much as possible.

2.7. Identifiers

The identifier used for matching, leveraging, and other ID-related functions is stored in the `resname` attribute. That means the ID-related attributes of an HTML document, such `name` or `id`, when appropriate, should be stored in the XLIFF attribute `resname`.

The required attribute `id` of the XLIFF `<trans-unit>` and `<bin-unit>` elements is an identifier allowing extraction tools to merge back the data. Its value is completely determined by the filter. It may or may not correspond to an HTML identifier; it may or may not be unique within the document, but it must be unique within the enclosing `<file>` element.

2.8. Preserving Attribute Values

The XLIFF standard, in some cases, provides specific placeholders for semantic HTML attributes. In some cases it does not. A very efficient way to capture these attributes is to make use of the extension points the XLIFF standard provides. Consider this example:

```
<p>Questions will appear in <font color="#339966">Green
 face</font>, while answers will appear in <font color="#333399">Indigo
 face</font>.</p>
```

Since XLIFF does not provide an attribute specifically for color, one way of preserving it is by making use of the extension point for attributes, in the <g element, like this:

```
<trans-unit id="c0" resname="p">
 <source>Questions will appear in <g id="c1" ctype="x-font" xhtml:color="#339966">Green
 face</g>, while answers will appear in <g id="c2" ctype="x-font"
 xhtml:color="#333399">Indigo face</g>.</source>
 </trans-unit>
```

More information about HTML attributes, pertaining to translatable vs. non translatable attribute values, along with recommendations about declaring the XHTML namespace, is given in the next section.

2.9. Non-HTML Content

The content of some elements and attributes may be something else than normal HTML text. Special care needs to be used in order to map such content in XLIFF.

2.9.1. Styles

XLIFF provides an attribute `css-style` that allows you carry directly any CSS style information applied to a specific item through the use of the HTML `style` attribute.

When processing for XLIFF, style content should be parsed with a filter appropriate to CSS, and the translatable parts should be extracted.

2.9.2. Scripts

Script content can be found in different places in an HTML file: the `<script>` elements and the various event-related attributes such as `onclick`, `onfocus`, etc.

When processing for XLIFF, script content should be parsed with a filter appropriate to the scripting language used (e.g. JavaScript, VBScript, Perl, TCL, etc,) and the translatable parts should be extracted.

2.10. CDATA Sections

The CDATA section permits the special characters such as '&', '<', '>', etc. to be included in the text without being escaped. This can be useful when a paragraph contains a lot of such characters.

The use of CDATA blocks is not recommended from the localization viewpoint:

- Numeric character references (NCRs) cannot be used in CDATA sections. This can potentially lead to the loss of data if the document is converted from one encoding to another (and the CDATA is to be preserved), an operation not uncommon during the localization process.
- Keeping track of CDATA markers in a translation memory segment implies the use of inline codes that have no use beyond allowing the un-escaped syntax, and possibly causing loss of matches.

From XML's point of view, CDATA is processed as if it were text. For example:

```
<p>This is an example <![CDATA[of <sgml> markup that is not <painful> to write with < and such]]>.</p>
```

is exactly the same as:

```
<p>This is an example of &lt;sgml&gt; markup that is not &lt;painful&gt; to write with &lt; and such.</p>
```

When extracting a CDATA section into XLIFF it is recommended to not use the CDATA notation in the XLIFF file. For example, the following fragment of HTML code:

```
<p>The characters<![CDATA[ '<', '&', and '>' are special]]>.</p>
```

should be represented as follow in XLIFF:

```
<source>The characters '&lt;', '&amp;' and '&gt;' are special.</source>
```

2.11. Multilingual Documents

There are two kinds of multilingual files:

- The ones that are written in a main language with block of content in other languages (citation for example). Such files are meant to be processed with all languages.
- The ones that have the same content in two or more languages and use some kind of filtering mechanism at runtime to display the content in one given language.

Multilingual HTML documents belong to the first category: There is a main language and the parts in other languages belong to the same content flow. Therefore, an XLIFF filter should extract all the text of an original HTML document, while, if possible, keeping track of the language switches when they occur.

The `xml:lang` attribute is available for that purpose. Here is an example of a HTML paragraph with

```
<P>The words <Q lang="fr">Je me souviens</Q> are the motto of Québec.</P>
```

This could be represented in XLIFF as:

```
<source xml:lang="en">The words <g ctype="x-html-Q" xml:lang="fr" id="b1">Je me souviens</g>
are the motto of Québec.</source>
```

2.12. Entity References

HTML uses several types of entity references.

2.12.1. Character Entity References

As a general rule, when extracted to XLIFF, the filter should make all effort to resolve any character entity references to their corresponding Unicode characters. As a last resort, and only if there is no way for the filter to convert the character entity reference, the construct should be treated as an inline code. For example, the following original element:

```
<p>&aacute;=a-acute, &mychar;=W-circumflex</p>
```

Should be represented this way:

```
<source>á=a-acute, ŵ=W-circonflex</source>
```

Or, at the last resort (this is not the preferred solution) unresolved character entity references could be represented this way:

```
<source>á=a-acute, <ph id='1'>&amp;mychar;</ph>=W-circonflex</source>
```

2.12.2. Other Entity References

HTML documents can also contain various other entity references, which do not represent a single character. Such entity references should be treated like variables and must be preserved as inline codes, using the <ph> or <x/> element.

For example, the following HTML document:

```
<h1>Online Help for &ProductName;</h1>
```

should be represented in XLIFF as:

```
<source>Online Help for <ph id='1'>&amp;ProductName;</ph>.</source>
```

2.13. Numeric Character References

Also known as NCR, the numeric character references are the decimal or hexadecimal notation of characters. When extracted to XLIFF, numeric character references should never be set as inline codes. Obviously, if the XLIFF document is in an encoding that does not support the character, the character can be represented as an NCR.

For example, the following paragraph:

```
<p>&#x00e0;=a-grave</p>
```

Should be represented as a normal character, that is as in one of the following notations:

```
<source>à=a-grave</source>
<source>&#x00e0;=a-grave</source>
<source>&#224;=a-grave</source>
```

But never as:

```
<source><ph id='1'>&lt;#x00e0;</ph>=a-grave</source>
```

Note that because XLIFF files may—for various reasons—have to be handled by non-XML tools or non-XML editors, it is always better to use normal raw characters than NCRs.

2.14. Comments

As a general rule from the localization viewpoint it is recommended to not have HTML comments inside text content (for example within a <p> element). They create potential problems for translation memory matching. Comments outside text content are not an issue since they usually do not affect the markup of any translatable entries.

If an XLIFF filter tool finds comments inside a run of text, the comment should be preserved by being treated as inline code. For example, the following HTML paragraph:

```
<p>The team members had colorful nicknames, like
<!-- use volume 2 here -->
<i>picabo, big-air</i>, and <i>yard-sale </i>
<!-- back to volume 1 here -->
which the media had difficulty understanding.
</p>
```

should be mapped to a `<trans-unit>` where the comments are preserved inside a <ph> or an `<x/>` element. The XLIFF output would look something like this:

```
<trans-unit id='1' restype='x-html-p'>
  <source xml:lang='en'>The team members had colorful nicknames, like
  <ph id='1-1' ctype="x-html-comment"> use volume 2 here </ph>
  <g id="a" ctype='italic'>picabo, big-air</g>,
  and <g id="b" ctype='italic'> yard-sale</g>
  <ph id='1-4' ctype="x-html-comment"> back to volume 1 here </ph>
  which the media had difficulty understanding.
  </source>
 </trans-unit>
```

2.15. Processing Instructions

As a general rule from the localization viewpoint it is recommended to `not` have processing instruction inside text content (for example a <p> element). They create potential problems for translation memory matching. Processing instructions outside text content are not an issue since they do not affect the markup of any translatable segments.

If an XLIFF filter tool finds a processing instruction inside a run of text, it must be preserved by being treated as inline code. For example, the following HTML paragraph:

```
<p>The team members had colorful nicknames, like
<?Trans-instruct use volume 2 here ?>
<i>picabo, big-air</i>, and <i>yard-sale </i>
<?Trans-instruct back to volume 1 here ?>
which the media had difficulty understanding.
</p>
```

should be mapped to a `<trans-unit>` where the processing instructions must be preserved inside `<ph>` or `<x/>` elements. The XLIFF output would look something like this:

```
<trans-unit id='1' restype='x-html-p'>
 <source xml:lang='en'>The team members had colorful nicknames, like
  <ph id="b" ctype="x-html-pi">Trans-instruct use volume 2 here </ph>
  <g id='1-2' ctype="italic">picabo, big-air</g>, and <g id='1-3' ctype="italic">yard-sale </g>
  <ph id="a" ctype="x-html-pi">Trans-instruct back to volume 1 here </ph>
which the media had difficulty understanding.</source>
</trans-unit>
```

Note that some processing instructions may be specific to a server-side scripting language (e.g. PHP). In this case they would need to be processed with an appropriate filter and their translatable content handled the same way any inline code with translatable content would be handled.

For example the following PHP instructions:

```
<p>Your account has
<?php if ($Quotas&gt;100) echo "exceeded"; else echo "not exceeded"; ?>
its allocated quotas.</p>
```

would be mapped to XLIFF as follow:

```
<trans-unit id='1' restype='x-html-p'>
 <source xml:lang='en'>Your account has
```

<ph id='1-1' ctype='x-html-pi'>&lt;?php if ($Quotas>100) echo "<sub>exceeded</sub>"; else echo "<sub>not exceeded</sub>"; ?&gt;</ph>
its allocated quotas.</source>
</trans-unit>

2.16 White Spaces

In HTML the following characters are considered white spaces:

- ASCII Space (U+0020)
- ASCII Tab (U+0009)
- Zero-width space (U+200B)
- Carriage-return (U+000D)
- Line-feed (U+000A)
- Thin space (U+2009)
- en-space (U+2002)
- em-space (U+2003)
- ideographic space (U+3000)

More information about white space characters is available at the Unicode site

When extracting to XLIFF, whitespace characters must be preserved for the content of the `<pre>` and `<textarea>` elements.

In XHTML the content of any element for which the attribute `xml:space` is set to `preserve` must also be extracted with the white spaces preserved. Another way, in HTML, of indicating the white spaces should be preserved is to use a CSS style. If possible, that information should be carried into the extracted file as well.

3. General Structure

While there are some elements and attributes in HTML that will require unique consideration, many can be mapped to XLIFF according to their characteristics. Elements that are wrappers, for example that contain no text, should generally be mapped differently than elements that contain just text. Elements that have mixed content should generally be mapped differently than inline elements. Attributes that contain no translatable text should be mapped differently than attributes with text that needs to be translated.

3.1. Mapping HTML Elements

In general, HTML elements can be classified in a few distinct ways:

- Wrappers with no text (like `<html>`, `<head>`, `<table>`, `<body>`, and so on)
- Elements that can contain mixed content (like `<p>`, `<h1>`, `<h2>`, `<font>`, and so on)
- Elements that contain just text, and no child elements (like `<title>`)
- Elements specifically meant to be inline (like `<b>`, `<i>`, `<sub>`, `<sup>`, `<a>`, and so on)
- Elements that are empty, with no text, no child elements, but can contain attributes (like `<col>`, `<br>`, `<hr>`, `<img>`, and so on) some of these elements may be also inline elements (like `<br>` or `<img>`).

Filter tools should create a new `<trans-unit>` element each time a new non-inline element is found and some meaningful text (i.e. meaningful white-spaces or non-white-space characters) was processed before. This is the only *segmentation* discussed here. The segmentation of sentences is not in the scope of this document.

3.1.1. Wrappers

Wrapper elements should generally be mapped as group elements. The value of their `restype` attribute should be one of the pre-defined values. If no pre-defined value corresponding exists, it should be the concatenation of `'x-html-'` and the name of the element (in lowercase). For example: `<body>` would be mapped to `<group restype='x-html-body'>`.

3.1.2. Mixed Content

In some (but not all) circumstances, mixed content can go into `<trans-unit>`:

```
<p>In Portland, Oregon one may <i>ski</i> on the mountain, <b>wind surf</b> in the gorge, and <i>surf</
```

```
i> in the ocean, all on the same day.</p>
```

should be mapped to:

```
<trans-unit id='1' restype='x-html-p'>
 <source xml:lang='en'>In Portland, Oregon one may <g id='c' ctype='italic'>ski</g> on the mountain, <g
id='d' ctype='bold'>wind surf</g> in the gorge, and <g id='e' ctype='italic'>surf</g> in the ocean, all on
the same day.</source>
</trans-unit>
```

3.1.3. Simple Text

In cases where the element is the child of a wrapper element, and its contents consist only of text, a simple `<trans-unit>` usually works best:

```
<td>silver</td>
```

should be mapped to:

```
<trans-unit id='1' restype='cell'>
 <source xml:lang='en'>silver</source>
</trans-unit>
```

3.1.4. Inline Elements

In most cases, inline elements are very well suited to be mapped to <g>. The value of their `ctype` attribute should be a concatenation of '`x-html-`' and the name of the element (in lowercase). For example: <b> would be mapped to `<g ctype='bold'>`.

Example:

```
<p>In Portland, Oregon one may <i>ski</i> on the mountain, <b>wind surf</b> in the gorge, and <i>surf</
i> in the ocean, all on the same day.</p>
```

should be mapped to:

```
<trans-unit id='1' restype='x-html-p'>
 <source xml:lang='en'>In Portland, Oregon one may <g id='i1' ctype='italic'>ski</g> on the mountain,
<g id='i1' ctype='bold'>wind surf</g> in the gorge, and <g id='i1' ctype='italic'>surf</g> in the ocean, all on
the same day.</source>
</trans-unit>
```

3.1.5. Empty Elements

Empty elements have no text in the element content, and therefore do not need to be translated. In most cases they have attributes. It is important to include references to the empty elements in the XLIFF file for purposes of preserving them during the reconstruction of the translated document. Cases for handling attributes that need to be translated vs. attributes that do not need translation are documented below.

Empty elements are mapped to `<x/>`. For example:

```
<p>This is Mount Hood: <img src='mthood.jpg'></p>
```

should be mapped to:

```
<trans-unit id='1' restype='x-html-p'>
 <source xml:lang='en'>This is Mount Hood: <x id='1-1' ctype="image" xhtml:src="mthood.jpg" /
></source>
</trans-unit>
```

3.1.6. Introducing New Elements

Some non-valid, but well-formed HTML files make use of non HTML elements. If a new element is introduced, the same rules that apply to HTML elements are useful for the new element. Determine if the new element is a wrapper, text, inline, empty, or mixed content model, and apply the rules outlined earlier in this section.

Example:

```
<p>In Portland, <index primary="yes">Oregon</index> one may <i>ski</i> on the mountain, <b>wind
surf</b> in the gorge, and <i>surf</i> in the ocean, all on the same day.</p>
```

should be mapped to:

```
<trans-unit id='1' restype='x-html-p'>
 <source xml:lang='en'>In Portland, <g id='i0' ctype='x-index'>Oregon</g> one may <g
id='i1' ctype='italic'>ski</g> on the mountain, <g id='i1' ctype='bold'>wind surf</g> in the gorge, and <g
id='i2' ctype='italic'>surf</g> in the ocean, all on the same day.</source>
</trans-unit>
```

3.2. Mapping HTML Attributes

There are generally two scenarios for HTML attributes:

- In most cases they are not to be translated, but must be preserved for the reconstruction of the translated file.
- In less frequent cases, they might contain text that needs to be translated.

3.2.1. Non-translatable Attributes

Each XLIFF element recommended for mapping has points of extensibility for attributes. This provides a very good way to preserve attributes that do not need to be translated, but do need to be preserved for the reconstruction of the translated document, or for other purposes (like rendering) during the translation phase.

Since XHTML has a namespace, it is recommended that this be used. For example it can be declared at the beginning of the XLIFF document:

```
<xliff xmlns="urn:oasis:names:tc:xliff:document:1.2"
       xmlns:xhtml="http://www.w3.org/1999/xhtml"
       version="1.2">
...
```

So the following HTML entry:

```
<h2 class="article-title">Life and Habitat of the Marmot</h2>
```

can be mapped to:

```
<trans-unit id='1' restype='x-html-h2' xhtml:class='article-title'>
 <source xml:lang='en'>Life and Habitat of the Marmot</source>
</trans-unit>
```

You can use a namespace even if the original input HTML is not XHTML. In this case, you may want to us your own namespace to avoid any conflict with the real XHTML namespace.

For instance, in the following HTML entry, two attributes `bordercolorlight` and `bordercolordark` are not part of HTML 4.01 or XHTML 1.0:

```
<td valign="top"
 bordercolorlight="#008000" bordercolordark="#800000">Text in cell</td>
```

They can be represented using your own namespace, as follow:

```
<xliff xmlns="urn:oasis:names:tc:xliff:document:1.2"
      xmlns:htm="urn:myHTMLInfo"
      version="1.2">
...
 <trans-unit id='1' restype='cell' htm:valign="top"
  htm:bordercolorlight="#008000" htm:bordercolordark="#800000">
  <source xml:lang='en'>Text in cell</source>
 </trans-unit>
...
</xliff>
```

3.2.2. Translatable Attributes

Depending on to what type of HTML element a translatable attribute belongs to, it can be mapped either as a separate `<trans-unit>` or as the content of a `<sub>` element inside an XLIFF inline code.

The `restype` value corresponding to the extracted attribute value should be one of the pre-defined values. In the cases where a corresponding pre-defined value does not exist it should be the concatenation of '`x-html-`', the name of the element (in lowercase), '-', and the name of the attribute (in lowercase). For example: `<isindex prompt="Enter your search phrase: ">` would correspond to: `restype='x-html-isindex-prompt'`.

Note that <ph> is required instead of <g> when <sub> elements are used.

Here are some examples. The following HTML code:

```
<p title='Information about Mount Hood'>This is Mount Hood: <img src="mthood.jpg" alt="Mount
Hood with its snow-covered top"></p>
```

should appear like this:

```
 <trans-unit resname="x-html-p" id="a_1">
  <source xml:lang="EN">
  <ph id="a_2"><sub ctype="x-html-p-title">Information about Mount Hood</sub>
  </ph>This is Mount Hood:
  <ph id="a_3" ctype="x-html-img" xhtml:src="mthood.jpg">
  <sub ctype="x-html-img-alt">Mount Hood with its snow-covered top
  </sub>
  </ph>
  </source>
 </trans-unit>
```

3.2.3. Irregular Attributes

Some non-valid, but well-formed HTML files make use of non HTML attributes. If a new attribute is introduced, the same rules that apply to HTML attributes are useful for the new attribute. Determine if the new attribute is translatable, or non-translatable, and apply the rules outlined earlier in this section.

4. Details by Element and Attribute

The following table list all the elements used in the various flavors of HTML and their properties.

Legend --

- Element: name of the element;
  note: element names in XHTML are all lower case
- Inline: indicates if the element can be extracted as inline code;
- Empty: indicates if the element is empty;
- PCDATA: indicates if the element can contain only PCDATA;
- Mixed: indicates if the element can contain text and other elements;
- Wrapper: indicates a group;
- Status: Indicates the current status of the element;

- `Type value`: value to use for `ctype` or `restype`.

Element	**Inline?**	**Empty?**	**PCDATA?**	**Mixed?**	**Wrapper?**	**Status**	**Type Value (`ctype` if inline, `restype` otherwise)**
A	inline				wrapper		`x-html-a`
ABBR	inline			mixed			`x-html-abbr`
ACRONYM	inline			mixed			`x-html-acronym`
ADDRESS				mixed			`x-html-address`
APPLET	inline			mixed	wrapper	deprecated	`x-html-applet`
AREA		empty					`x-html-area`
B	inline			mixed			`bold`
BASE		empty					`x-html-base`
BASEFONT		empty				deprecated	`x-html-basefont`
BDO	inline			mixed			`x-html-bdo`
BGSOUND		empty				not 4.01	`x-html-bgsound`
BIG	inline			mixed			`x-html-big`
BLOCKQUOTE					wrapper		`x-html-blockquote`
BLINK	inline			mixed		not 4.01	`x-html-blink`
BODY					wrapper		`x-html-body`
BR	inline	empty					`lb`
BUTTON	inline			mixed			`x-html-button`
CAPTION				mixed			`caption`
CENTER						deprecated	`x-html-center`
CITE	inline			mixed			`x-html-cite`
CODE	inline			mixed			`x-html-code`
COL		empty					`x-html-col`
COLGROUP					wrapper		`x-html-colgroup`

DD				mixed			`x-html-dd`
DEL	inline			mixed			`x-html-del`
DFN	inline			mixed			`x-html-dfn`
DIR					wrapper	deprecated	`x-html-dir`
DIV				mixed			`x-html-div`
DL					wrapper		`x-html-dl`
DT				mixed			`x-html-dt`
EM	inline			mixed			`x-html-em`
EMBED	inline					not 4.01	`x-html-embed`
FACE	inline			mixed		RobotHelp	`x-html-face`
FIELDSET				mixed	wrapper		`groupbox`
FONT	inline			mixed		deprecated	`x-html-font`
FORM					wrapper		`dialog`
FRAME		empty					`frame`
FRAMESET							`x-html-frameset`
H1				mixed			`x-html-h1`
H2				mixed			`x-html-h2`
H3				mixed			`x-html-h3`
H4				mixed			`x-html-h4`
H5				mixed			`x-html-h5`
H6				mixed			`x-html-h6`
HEAD					wrapper		`header`
HR		empty					`x-html-hr`
HTML					wrapper		`x-html-html`
I	inline			mixed			`italic`
IA						not 4.01	`x-html-ia`

IFRAME	inline			mixed			`x-html-iframe`
IMG	inline	empty					`image`
INPUT	inline	empty					`x-html-input`
INS	inline			mixed			`x-html-ins`
ISINDEX		empty				deprecated	`x-html-isindex`
KBD	inline			mixed			`x-html-kbd`
LABEL	inline			mixed			`x-html-label`
LEGEND				mixed			`x-html-legend`
LI				mixed			`listitem`
LINK		empty					`x-html-link`
LISTING				mixed		not 4.01	`x-html-listing`
MAP	inline						`x-html-map`
MARQUEE				mixed		not 4.01	`x-html-marquee`
MENU					wrapper	deprecated	`menu`
META		empty					`x-html-meta`
NOBR	inline	empty				not 4.01	`x-html-nobr`
NOEMBED					wrapper	not 4.01	`x-html-noembed`
NOFRAMES					wrapper		`x-html-noframes`
NOSCRIPT					wrapper		`x-html-noscript`
OBJECT	inline			mixed	wrapper		`x-html-object`
OL					wrapper		`x-html-ol`
OPTGROUP					wrapper		`x-html-optgroup`
OPTION			PCDATA				`x-html-option`
P				mixed			`x-html-p`
PARAM	inline	empty					`x-html-param`
PLAINTEXT			PCDATA			obsolete	`x-html-plaintext`
PRE				mixed			`x-html-pre`

Q	inline			mixed			x-html-q
RB	inline			mixed		ruby	x-html-rb
RBC	inline			mixed		ruby	x-html-rbc
RP	inline			mixed		ruby	x-html-rp
RT	inline			mixed		ruby	x-html-rt
RTC	inline			mixed		ruby	x-html-rtc
RUBY	inline			mixed		ruby	x-html-ruby
S	inline			mixed		deprecated	x-html-s
SAMP	inline			mixed			x-html-samp
SCRIPT			PCDATA				x-html-script
SELECT	inline				wrapper		x-html-select
SMALL	inline			mixed			x-html-small
SPAN	inline			mixed			x-html-span
SPACER	inline	empty				not 4.01	x-html-spacer
STRIKE	inline			mixed		deprecated	x-html-strike
STRONG	inline			mixed			x-html-strong
STYLE			PCDATA				x-html-style
SUB	inline			mixed			x-html-sub
SUP	inline			mixed			x-html-sup
SYMBOL	inline			mixed		RobotHelp	x-html-symbol
TABLE					wrapper		table
TBODY					wrapper		x-html-tbody
TD				mixed			cell
TEXTAREA	inline		PCDATA				x-html-textarea
TFOOT					wrapper		footer
TH				mixed			x-html-th
THEAD					wrapper		x-html-thead
TITLE			PCDATA				x-html-title

TR					wrapper		`row`
TT	inline			mixed			`x-html-tt`
U	inline			mixed		deprecated	`underlined`
UL					wrapper		`x-html-ul`
VAR	inline			mixed			`x-html-var`
WBR	inline	empty				not 4.01	`x-html-wbr`
XML				mixed	wrapper	not 4.01	`x-html-xml`
XMP			PCDATA			not 4.01	`x-html-xmp`
<any other>	not inline			possibly mixed			`x-html-<any other>`

The following table lists the attributes used in the various flavors of HTML and their properties. If an attribute is not listed in this table, it is not translatable.

Attribute	Element(s) where it is used	Type Value	Notes
`abbr`	`<td>` and `<th>`	`x-html-<elem>-abbr`	
`accesskey`	`<a>`, `<area>`, `<button>`, `<input>`, `<label>`, `<legend>`, and `<textarea>`	`x-html-<elem>-accesskey`	`size-unit='char' maxwidth='1'`
`alt`	`<applet>`, `<area>`, `<img>`, and `<input>`	`x-html-<elem>-alt`	
`content`	`<meta>`	`x-html-<elem>-content`	
`label`	`<option>` and `<optgroup>`	`label`	
`prompt`	`<isindex>`	`x-html-isindex-prompt`	
`standby`	`<object>`	`x-html-<elem>-standby`	
`summary`	`<table>`	`x-html-<elem>-summary`	
`title`	All elements except `<base>`, `<basefont>`, `<head>`, `<html>`, `<meta>`, `<param>`, `<script>`, and `<title>`	`x-html-<elem>-title`	
`value`	`<input>`	`x-html-<elem>-value`	
`value`	`<option>`	NONE	Not translatable
`value`	`<param>`	`x-html-<elem>-value`	The `value` attribute in a `<param>` element is translatable only in some cases depending on what the parameter is.
`value`	`<button>`	`x-html-button-value`	
`value`	`<li>`	NONE	Not translatable

4.1. Inline Elements

Inline elements are HTML elements that should be treated as codes embedded within a run of text, for example <b>, <em>, etc.

The following HTML elements should be treated as inline codes:

`<a>`, `<abbr>`, `<acronym>`, `<applet>`, `<b>`, `<bdo>`, `<big>`, `<blink>`, `<br>`, `<button>`, `<cite>`, `<code>`, `<del>`, `<dfn>`, `<em>`, `<embed>`, `<face>`, `<font>`, `<i>`, `<iframe>`, `<img>`, `<input>`, `<ins>`, `<kbd>`, `<label>`, `<map>`,

<nobr>, <object>, <param>, <q>, <rb>, <rbc>, <rp>, <rt>, <rtc>, <ruby>, <s>, <samp>, <select>, <small>, <span>, <spacer>, <strike>, <strong>, <sub>, <sup>, <symbol>, <textarea>, <tt>, <u>, <var>, and <wbr>.

4.2. <meta> Elements

The <meta> element can be used to carry many types of information. During localization some need to remain untouched, some need to be translated.

Often, the value of the attribute value for name or http-equiv determines whether the value of the content attribute needs translation. The following table lists some of the more common occurrences:

Value of name	**Value of http-equiv**	**Value of content**
	keywords	To extract
	content-language	Not to extract, but the filter should modify it if necessary
	content-type	Not to extract, but the filter should modify it if necessary
	<other value>	Not to extract
generator		Not to extract
author		Not to extract
progid		Not to extract
date		Not to extract
<other values>		To extract

For example:

```
<meta name="GENERATOR" content="Microsoft FrontPage 4.0">
<meta name="ProgId" content="FrontPage.Editor.Document">
<meta http-equiv="Content-Language" content="en-us">
<meta http-equiv="Content-Type" content="text/html; charset=windows-1252">
<meta http-equiv="keywords" content="localization tool, translation tool">
<meta name="test" content="meta data for test">
```

The HTML fragment above should be represented as:

```
<trans-unit id="1" restype="x-meta-content">
 <source xml:lang="en">localization tool, translation tool</source>
</trans-unit>
<trans-unit id="2" restype="x-meta-content">
 <source xml:lang="en">meta data for test</source>
</trans-unit>
```

4.3. <img> Elements

The <img> element is used to hold a reference to an image. The image itself is not stored in the XLIFF document, but its metadata, and translatable text should be. Here's an example. The following paragraph contains an image with attributes that describe the image source, and an alternate text. In this case, the source should not be translated but the alternate text might (many browsers display the alternate text on a mouse-over. A case could be made that this is eligible for translation). Consider this example:

```
<p>My picture,
<img src="mthood.jpg" alt="This is a shot of Mount Hood" />
and there you have it.</p>
```

A good way to handle this is to start a new <ph> element for the <img> element, put the non-translatable attributes in namespace attributes, and put the alternate text in a <sub> element, like this:

```
<trans-unit resname="p" id="d0e1">
<source xml:lang="EN">My picture,
<ph id="d0e3" ctype="image" xhtml:src="mthood.jpg"><sub ctype="x-img-alt">This is a shot of Mount
Hood</sub></ph>
and there you have it.</source>
```

This is a good approach because the XLIFF schema allows for extensible attributes for the <ph> element.

4.4. SVG Images

Historically, images in HTML have been one of two raster formats, JPEG and GIF. This meant that the task of localizing text within images could prove complex. This was done:

- By modifying the source file used to produced the image (e.g. the Photoshop file).
- By re-creating the image from scratch.
- By modifying the raster image in a raster graphic editor.

All of those solutions usually meant additional overhead and the need for graphic artists to work on the images in addition to translators.

It is now possible to display Scalable Vector Graphics (.svg) images in browsers via HTML. SVG files are XML documents and have editable (therefore translatable) text. They can be used through an <object> element, or with XHTML, directly embedded in HTML using the XML namespace mechanism.

Such SVG data could be processed with via XML standards such as XSLT. The representation of extracted SVG in XLIFF is outside of the scope of this document, but a short example can show how easily SVG simplify the translation tasks.

For instance, in this SVG image (you need to have an SVG rendering engine to see the image properly):

Note: Sometimes browser security settings prohibit SVG rendering engines to display the SVG image. In Internet Explorer, for example, you may see a note that says: "To help protect your security, Internet Explorer has restricted this file from showing active content that could access your computer. Click here for options..." Consult the browser support page for more information.

The translatable strings are represented like this one:

```
<text id="XMLID_1_" transform="matrix(1 0 0 1 0 32.0503)">
 <tspan x="0" y="0" fill-rule="evenodd"
clip-rule="evenodd" font-family="'TimesNewRomanPSMT'"
font-size="9.25">TDR on indicator (80E04)</tspan></text>
```

They are easily mapped to `<trans-units>` elements like this:

```
<trans-unit id="A001-d0e499" resname="tspan">
<source>TDR on indicator (80E04)</source>
<target state="needs-translation">TDR on indicator (80E04)</target>
</trans-unit>
```

After the `<target>` is localized, it can be transformed back into SVG:

```
<text id="A001-XMLID_1_" transform="matrix(1 0 0 1 0 32.0503)">
<tspan id="A001-d0e499" x="0" y="0" fill-rule="evenodd"
clip-rule="evenodd" font-family="Arial Unicode MS"
font-size="9.25">表示器(80E04 は) のTDR</tspan></text>
```

This gives you the following image (you need to have an SVG rendering engine to see the image properly):

4.5. <object> and <param> Elements

Sometimes the `value` attribute needs to be translated. Sometimes it must not be translated. This is often indicated by the content of the `name` attribute. Consider this example regarding the HTML code used to generate the Index tab of an HTMLHelp System:

```
<UL>
  <LI>
  <OBJECT type="text/sitemap">
  <param name="Name" value="Product Identification">
  <param name="Local" value="ProductIdentification.htm">
  </OBJECT>
  </LI>
  <LI>
  <OBJECT type="text/sitemap">
  <param name="Name" value="Features">
  <param name="Local" value="Features.htm">
  </OBJECT>
  </LI>
```

In this example, each `<param` has a `name` attribute. In cases where the value of `name` is set to "Name," the `value` attribute is to be translated. In cases where the `name` attribute is set to "Local," the `value` attribute is not to be translated (in this example because the `<param` element refers to an external file).

4.6. HTML Forms

An HTML form can be assimilated to a dialog box. However, there is generally no coordinates associated with the controls. When extracted to XLIFF, each form should be mapped to a `<group>` element with its `restype` attribute set to `dialog`.

The `<fieldset>` element allows the HTML controls to be grouped together, using the content of the `<legend>` element to store the caption for the given group.

The `<input>` element can be used for different types of controls depending on its `type` attribute: `text`, `password`, `checkbox`, `radio`, `submit`, `reset`, `file`, `hidden`, `image`, and `button`. Translatable text can be found in the `alt` attribute. The label corresponding to the input control is in the same flow that contains the control.

The `<button>` element acts almost like the `<input>` element, but the caption text is the content of the element.

The `<select>` element represents a scrolling list, which can be rendered as a listbox, a drop-down menu, etc. It contains `<optgroup>` and `<option>` elements. In XLIFF, `<select>` is represented by a `<group>`, `<optgroup>` is also mapped to a `<group>`, and each `<option>` element has its corresponding `<trans-unit>`.

The `<textarea>` element allows the user to input a block of text. A default text can be set in the content of the element.

The element `<isindex>` is deprecated. XLIFF filter should treat it as inline code. the `prompt` attribute of `<isindex>` is translatable.

The `<label>` element is used to associate a label with a control that does not have an implicit label. The attribute `title` is translatable. The attribute `accesskey` should also be localized.

Summary of the Form-related elements and attributes:

HTML Elements	**XLIFF Mapping**
`<form>`	`<group restype='dialog'>`
`<fieldset>`	`<group restype='groupbox'>`
`<legend>`	`<trans-unit restype='caption'>`
`<input type="text">`	`<ph>` or `<x/>`
`<input type="password">`	`<ph>` or `<x/>`
`<input type="checkbox">`	`<ph>` or `<x/>`
`<input type="radio">`	`<ph>` or `<x/>`
`<input type="submit">`	`<ph>` or `<x/>`
`<input type="reset">`	`<ph>` or `<x/>`
`<input type="file">`	`<ph>` or `<x/>`
`<input type="hidden">`	`<ph>` or `<x/>`
`<input type="image">`	`<ph ctype='image'>` or `<x ctype='image'/>`
`<input type="button">`	`<ph>` or `<x/>`
`<button>`	`<ph>` or `<x/>`
`<select>`	`<group restype='listbox'>` or `<g ctype='x-'>`
`<optgroup>`	`<group restype='heading'>`
`<option>`	`<trans-unit restype='listitem'>`
`<textarea cols='20' rows='2'>`	`<trans-unit restype='textbox' xml:space='preserve' html:cols='20' html:rows='2'>`
`<isindex>`	`<ph>` or `<x/>`
`<label>`	`<trans-unit restype='label'>`
HTML Attributes	**XLIFF Mapping**
`prompt`	`<trans-unit>` or `<sub>` or use namespaced attribute (`xhtml:prompt`)
`label`	`<trans-unit>` or `<sub>` or use namespaced attribute (`xhtml:label`)
`title`	`<trans-unit>` or `<sub>` or use namespaced attribute (`xhtml:title`)
`accesskey`	`<trans-unit>` or `<sub>` or use namespaced attribute (`xhtml:accesskey`)
`standby`	`<trans-unit>` or `<sub>` or use namespaced attribute (`xhtml:standby`)
`value` for `<option>`	Not translatable
`value` for `<input type='text'>`	`<trans-unit>` or `<sub>` or use namespaced attribute (`xhtml:type`)
`value` for `<input type='password'>`	Not translatable
`value` for `<input type='checkbox'>`	Not translatable
`value` for `<input type='radio'>`	Not translatable
`value` for `<input type='submit'>`	`<trans-unit>` or `<sub>` or use namespaced attribute (`xhtml:type`)
`value` for `<input type='reset'>`	`<trans-unit>` or `<sub>` or use namespaced attribute (`xhtml:type`)
`value` for `<input type='file'>`	Not translatable
`value` for `<input type='hidden'>`	`<trans-unit>` or `<sub>` (In this case, `value` may or may not be translatable, depending on the intended use of `<input>`)

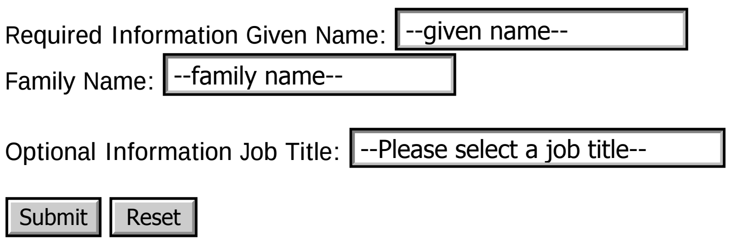
`value` for `<input type='image'>`	Not translatable
`value` for `<input type='button'>`	`<trans-unit>` or `<sub>`
`size` (`<input>`)	in pixel except when type=text or password then in chars
`maxlength` (`<input>`)	in pixel except when type=text or password then in chars

Here are some examples of forms and their representation in XLIFF. In the HTML code the extractable content is underlined, and its translatable parts are in black and bold. The translatable attributes are in blue and bold.

Example:

```
<form method="post">
 <fieldset>
  <legend accesskey='U'>Required Information</legend>
  <label for='gname' accesskey='G'>Given Name: </label>
  <input type='text' id='gname' name='gname' value='--given name--'><br>
  <label for='fname' accesskey='F'>Family Name: </label>
  <input type='text' id='fname' name='fname' value='--family name--'>
 </fieldset>
 <fieldset>
  <legend>Optional Information</legend>
  Job Title: <select name="prof" >
   <option selected label="Profession" value="none">--Please select a job title--</option>
   <optgroup label="Engineering">
    <option label="1.1. Engineer" value="it_eng">Engineer</option>
    <option label="1.2. Technician" value="it_tech">Technician</option>
   </optgroup>
   <optgroup label="Translation">
    <option label="2.1. Translator" value="tr_translator">Translator</option>
    <option label="2.2. Editor" value="tr_editor">Editor</option>
    <option label="2.3. Proofer" value="tr_proofer">Proofer</option>
   </optgroup>
   <optgroup label="Management">
    <option label="3.1. Project Manager" value="mg_pm">Project Manager</option>
    <option label="3.2. Translation Coordinator" value="mg_coord">Translation Coordinator</option>
    <option label="3.3. Project Assistant" value="tr_pa">Project Assistant</option>
   </optgroup>
  </select>
 </fieldset>
 <p><input type="submit" value="Submit" name="B1"> <input type="reset" value="Reset" name="B2"></p>
</form>
```

Actual rendering of the form in your browser:



Required Information Given Name: --given name--

Family Name: --family name--

Optional Information Job Title: --Please select a job title--

Corresponding XLIFF representation:

```
   <group resname="dialog" xhtml:method="post">
   <group resname="dialog" id=" N10003" xhtml:method="post">
```

```
    <group resname="groupbox">
     <group resname="groupbox" id=" N10005">
      <trans-unit id="caption-x-mch1-N10006" resname="caption" xhtml:accesskey="U">
       <source>Required Information</source>
      </trans-unit>
      <trans-unit id="label-x-mch1-N10009" resname="label" xhtml:for="gname" xhtml:accesskey="G">
       <source>Given Name: </source>
      </trans-unit>
      <group id="slllp2">
      <trans-unit id="N1000D">
       <source>
        <x id="input-x-mch2-N1000D" xhtml:type="text" xhtml:id="gname" xhtml:name="gname" xhtml:
value="--given name--"/>
       </source>
      </trans-unit>
     </group>
      <group id="slllp2">
      <trans-unit id="N10011">
       <source>
        <x id="br-x-mch2-N10011"/>
       </source>
      </trans-unit>
      </group>
      <trans-unit id="label-x-mch1-N10012" resname="label" xhtml:for="fname" xhtml:accesskey="F">
       <source>Family Name: </source>
      </trans-unit>
      <group id="slllp2">
      <trans-unit id="N10016">
       <source>
        <x id="input-x-mch2-N10016" xhtml:type="text" xhtml:id="fname" xhtml:name="fname" xhtml:
value="--family name--"/>
       </source>
      </trans-unit>
      </group>
     </group>
    </group>
    <group resname="groupbox">
     <trans-unit id="groupbox-x-N1001A" resname="groupbox">
      <source>
      <g ctype="x-caption" id="caption-x-N1001B">Optional Information</g>
 Job Title: <g ctype="x-select" id=" N1001E" xhtml:name="prof">
       <g id="option-x-mch1-N10020" ctype="x-option" xhtml:label="Profession" xhtml:value="none">--
Please select a job title--</g>
       <g ctype="x-optgroup" id=" N10024" xhtml:label="Engineering">
        <g id="option-x-mch1-N10026" ctype="x-option" xhtml:label="1.1. Engineer"
xhtml:value="it_eng">Engineer</g>
        <g id="option-x-mch1-N1002A" ctype="x-option" xhtml:label="1.2. Technician"
xhtml:value="it_tech">Technician</g>
       </g>
       <g ctype="x-optgroup" id=" N1002E" xhtml:label="Translation">
        <g id="option-x-mch1-N10030" ctype="x-option" xhtml:label="2.1. Translator"
xhtml:value="tr_translator">Translator</g>
        <g id="option-x-mch1-N10034" ctype="x-option" xhtml:label="2.2. Editor" xhtml:
value="tr_editor">Editor</g>
        <g id="option-x-mch1-N10038" ctype="x-option" xhtml:label="2.3. Proofer"
xhtml:value="tr_proofer">Proofer</g>
       </g>
       <g ctype="x-optgroup" id=" N1003C" xhtml:label="Management">
        <g id="option-x-mch1-N1003E" ctype="x-option" xhtml:label="3.1. Project Manager"
xhtml:value="mg_pm">Project Manager</g>
        <g id="option-x-mch1-N10042" ctype="x-option" xhtml:label="3.2. Translation Coordinator"
xhtml:value="mg_coord">Translation Coordinator</g>
```

```
        <g id="option-x-mch1-N10046" ctype="x-option" xhtml:label="3.3. Project Assistant"
xhtml:value="tr_pa">Project Assistant</g>
       </g>
      </g>
      </source>
     </trans-unit>
    </group>
    <group resname="p">
     <group resname="p" id="N1004A">
      <group id="slllp2">
      <trans-unit id="N1004B">
       <source>
        <x id="input-x-mch2-N1004B" xhtml:type="submit" xhtml:value="Submit" xhtml:name="B1"/>
       </source>
      </trans-unit>
      </group>
      <group id="slllp2">
      <trans-unit id="N1004F">
       <source>
        <x id="input-x-mch2-N1004F" xhtml:type="reset" xhtml:value="Reset" xhtml:name="B2"/>
       </source>
      </trans-unit>
      </group>
     </group>
    </group>
   </group>
   </group>
```

4.7. XForms Forms

With the advent of XHTML the concept of forms has been extended and generalized through the creation of XForms [XForms].

XForms was designed to make the coding of HTML/XHTML forms easier and to be output independent. A well designed XForms is capable of being rendered in VoiceXML as well as HTML, XHTML or as a valid/well formed XML document. In addition validating constraints can be specified for input values — something which requires extensive scripting in HTML.

The goal of XForms is to provide the 20% of necessary functionality in order to eliminate 80% of the need for scripting. An XForms processor is needed to render an XForms form into an instance.

XForms offers several controls:

Element	**HTML Equivalent**	**Description**
`<input>`	`<input type="text">`	For entry of small amounts of text
`<textarea>`	`<textarea>`	For entry of large amounts of text
`<secret>`	`<input type="password">`	For entry of sensitive information
`<output>`	none	For inline display of any instance data
`<range>`	none	For smooth "volume control" selection of a value
`<upload>`	`<input type="file">`	For upload of file or device data
`<trigger>`	`<button>`	For activation of form events
`<submit>`	`<input type="submit">`	For submission of form data
`<select>`	`<select multiple="multiple">` or multiple `<input type="checkbox">`	For selection of zero, one, or many options
`<select1>`	`<select>` or multiple `<input type="radio">`	For selection of just one option among several

The content of the following XForms elements are translatable:

- The content of the `<label>` element.
- The content of the `<message>` element.
- The content of the `<help>` element.
- The content of the `<hint>` element.
- The content of the `<alert>` element.

And the following items may be translatable depending on the context:

- The value of the `value` attribute of the `<setValue>` element.

Example of XForms entries:

```
<xforms:select1 ref="where">
  <xforms:label>Select Ski Resort:</xforms:label>
  <xforms:item>
    <xforms:label>Park City</xforms:label>
    <xforms:value>West</xforms:value>
  </xforms:item>
  <xforms:item>
    <xforms:label>Mont Tremblant</xforms:label>
    <xforms:value>East</xforms:value>
  </xforms:item>
</xforms:select1>
<xforms:input ref="season">
  <xforms:label>Fall/Winter/Spring:</xforms:label>
</xforms:input>
<xforms:input ref="duration">
  <xforms:label>Number of Days:</xforms:label>
</xforms:input>
<xforms:submit submission="submit">
  <xforms:label>Submit</xforms:label>
</xforms:submit>
```

Corresponding XLIFF extraction for example 1:

```
<body>
 <group resname="xforms:select1" xmrk:ref="where">
 <group resname="xforms:select1" id="N10007" xmrk:ref="where">
  <trans-unit id="xforms:label-x-mch1-N10009" resname="xforms:label">
   <source>Select Ski Resort:</source>
  </trans-unit>
  <group resname="xforms:item">
   <group resname="xforms:item" id="N1000B">
    <trans-unit id="xforms:label-x-mch1-N1000C" resname="xforms:label">
    <source>Park City</source>
    </trans-unit>
    <trans-unit id="xforms:value-x-mch1-N1000E" resname="xforms:value">
    <source>West</source>
    </trans-unit>
   </group>
  </group>
  <group resname="xforms:item">
   <group resname="xforms:item" id="N10010">
    <trans-unit id="xforms:label-x-mch1-N10011" resname="xforms:label">
    <source>Mont Tremblant</source>
    </trans-unit>
    <trans-unit id="xforms:value-x-mch1-N10013" resname="xforms:value">
    <source>East</source>
    </trans-unit>
   </group>
  </group>
```

```
   </group>
   </group>
   <group resname="xforms:input" xmrk:ref="season">
   <group resname="xforms:input" id="N10015" xmrk:ref="season">
    <trans-unit id="xforms:label-x-mch1-N10017" resname="xforms:label">
     <source>Fall/Winter/Spring:</source>
    </trans-unit>
   </group>
   </group>
   <group resname="xforms:input" xmrk:ref="duration">
   <group resname="xforms:input" id="N10019" xmrk:ref="duration">
    <trans-unit id="xforms:label-x-mch1-N1001B" resname="xforms:label">
     <source>Number of Days:</source>
    </trans-unit>
   </group>
   </group>
   <group resname="xforms:submit" xmrk:submission="submit">
   <group resname="xforms:submit" id="N1001D" xmrk:submission="submit">
    <trans-unit id="xforms:label-x-mch1-N1001F" resname="xforms:label">
     <source>Submit</source>
    </trans-unit>
   </group>
   </group>
  </body>
```

4.8. Bidirectional Markers

Some languages, such as Arabic and Hebrew may require the use of bidirectional ("bidi") markers to help user agents to display the text correctly. Unicode defines five markers for this:

RLE	Right-to-Left Embedding
LRE	Left-to-Right Embedding
RLO	Right-to-Left Override
LRO	Left-to-Right Override
PDF	Pop Display Formatting

While bidi functions can be done by using the Unicode special characters. However, when the text is stored in a marked up document, it is strongly recommended to use markup rather than characters.

HTML provides the `<bdo>` element and the `dir` attribute for these functions:

`dir="rtl"`	Right-To-Left
`dir="ltr"`	Left-To-Right
`<bdo dir="rtl">`	Right-to-Left Override
`<bdo dir="ltr">`	Left-to-Right Override

In order to carry the correct presentation information, XLIFF must provide a way to specify these bidi marks.

For example:

```
<p>The title says "<span dir="rtl">פעילות הבינאום, W3C</span>"
in Hebrew.</p>
```

The HTML fragment above could be represented as follow:

```
<trans-unit id="1">
 <source xml:lang="en">The title says "<g ctype='phrase' id='b1' html:dir='rtl'><bpt id="1">&lt;
span dir="rtl"></bpt>text...<ept id="1">&lt;/span></ept></g>" in Hebrew.</source>
</trans-unit>
```

4.9. <br> Element

The `<br>` element is used to mark a line break. It could be mapped to an `<ph>` or `<x/>` element with the `ctype` attribute set to "`lb`".

For example:

```
<p>First line<br>second line</p>
```

The HTML fragment above could be represented as follows:

```
<source>First line<x id="1" ctype="lb"/>second line.</source>
```

Note that `<br>` is not processed as an isolated tag (`<it>`, a beginning or end tag without its ending or beginning counterpart tag). The element `<br>` is defined as empty (`<br />` in XHTML).

4.10. Languages Switch

Element content can have runs of text in different languages. Each run of text can be marked up so authoring tools can process the text accordingly, for example, switching dictionaries when checking spelling.

For example:

<p>She added that "**<span lang='fr'>**je ne sais quoi**</span>**" that made her casserole absolutely delicious.
</p>

Corresponding XLIFF:

<source xml:lang='en'>She added that "**<g id='1' ctype='x-html-span' xml:lang='fr'>**je ne sais quoi**</
g>**" that made her casserole absolutely delicious.</source>

4.11. Pre-formatted Content

The `<pre>` element must be marked with the `xml:space="preserve"` attribute.

For example, the following HTML fragment:

```
<pre>First line
and second line</pre>
```

Should be represented as:

```
<trans-unit id="1" xml:space="preserve">
 <source xml:lang="en">First line
and second line</source>
</trans-unit>
```

4.12. Script Content

Script content can be found in the `<script>` element as well as in the following attributes: `onblur`, `onchange`, `onclick`, `ondblclick`, `onfocus`, `onkeydown`, `onkeypress`, `onkeyup`, `onload`, `onmousedown`, `onmousemove`, `onmouseout`, `onmouseover`, `onmouseup`, `onreset`, `onselect`, `onsubmit`, and `onunload`.

Such content should be processed with the relevant parser (JavaScript, PHP, Perl, etc.) and the translatable parts extracted. It is recommended to regroup all the strings of each `<script>` element in a `<group>`.

For example, the following HTML `<script>` element:

```
<SCRIPT type="text/javascript">
<!-- to hide script contents from old browsers
function square(i) { return i * i }
document.write("The function returned: ",square(5))
// end hiding contents from old browsers -->
</SCRIPT>
```

Could be represented as the following XLIFF code:

```
<group restype="x-html-script">
 <trans-unit id="1">
  <source xml:lang="en-US">The function returned: </source>
  <target xml:lang="fr-FR">The function returned: </target>
 </trans-unit>
</group>
```

4.13. Style Content

Style content can be found in the `<style>` element as well as in the `style` attribute. Such content should be processed with the relevant parser and translatable parts extracted.

A. Contributions

The following people have contributed to this document:

- Eiju Akahane, IBM
- Gérard Cattin des Bois, Microsoft (until Nov-21-04)
- Doug Domeny
- Paul Gampe
- Tony Jewtushenko
- Milan Karásek
- Christian Lieske, SAP
- Mat Lovatt, Oracle
- Magnus Martikainen, SDL
- Enda McDonnell
- David Pooley, SDL
- John Reid, Novell
- Peter Reynolds, Idiom Technologies, Inc
- Florian Sachse
- Yves Savourel
- Reinhard Schäler, LRC
- Bryan Schnabel
- Shigemichi Yazawa
- Andrzej Zydron

B. Example of XSLT Use to Process HTML

XSLT can be used to convert an HTML document into XLIFF and back.

1. HTML source document:
   ExampleXSLTUse_1_Source.html
2. XSL Transformation template to convert the HTML document into XLIFF:
   ExampleXSLTUse_2_xhtml2xliff.xsl
3. XML language configuration file (called by the XSLT file):
   tlang.xml
4. Result of the transformation: the XLIFF document before translation:
   ExampleXSLTUse_3_BeforeTrans.xlf
5. XLIFF document after the translation:
   ExampleXSLTUse_4_AfterTrans.xlf
6. XSL Transformation template to convert the XLIFF document back into HTML:
   ExampleXSLTUse_5_xliff2xhtml.xsl
7. Final result, the translated HTML document:

ExampleXSLTUse_6_Translated.html

C. Pre-Processing HTML Files

Perl is a powerful cross-platform programming language that can be used to convert an HTML document not well-formed.
For example by adding quotes to any unquoted attribute values, etc.

1. Original HTML document (invalid):
   ExamplePerlUse_1_Before.html
2. Perl script to fix up some of the issues:
   ExamplePerlUse_2_PerlFixer.html
3. Output file after being processed by the Perl script:
   ExamplePerlUse_3_After.html

D. References

[HTMLTidy]
HTML Tidy, HTML Clean-up Open source Utility
http://www.w3.org/People/Raggett/tidy/

[ITS]
W3C Internationalization Tag Set (ITS)
http://www.w3.org/International/its/

[ISO]
International Organization for Standardization Web site.

[OASIS]
Organization for the Advancement of Structured Information Standards Web site.

[Perl]
The Perl Programming Language
http://www.perl.org/

[RFC 3066]
RFC 3066 Tags for the Identification of Languages. IETF (Internet Engineering Task Force), Jan 2001.

[Unicode]
Unicode Consortium Web site.

[W3CQA-bidi]
Q&A: (X)HTML and bidi formatting codes versus mark-up.

[XForms]
XForms, the Next Generation of Web Forms
http://www.w3.org/MarkUp/Forms/