XLIFF 1.2

A white paper on version 1.2 of the XML Localisation Interchange File Format (XLIFF)

Revision: 1.0
Issue Date: October 17, 2007

Table of Content

1.0 Introduction

Industry, technology and government standards are the basis of modern, civilized life. They are used for abstracting and negotiating all aspects of interpersonal relations. They are key to manufacturing, scientific research and development, commerce, telecommunications, and computer and information technology. Standards create new markets where none existed before, and promote the maturity of industry practices. Ultimately, they benefit consumers by reducing the cost and time required to deliver high quality goods, technologies and services.

When XLIFF was initially envisioned many years ago, it was in response to the growing complexity of the software localization process. At the time, the Internet “revolution” was forcing software publishers to converge upon new technologies based on software standards such as HTML, XML and Java. Although these standards were designed with the international market in mind and were intended to simplify the development of globalized applications, they had the opposite effect on localization. The proliferation of so many disparate software resource formats meant that the process of localising Internet based applications was complex, expensive and opaque. Software publishers seeking to localize these products had to choose between two equally expensive options: develop complex localization tools internally, or outsource the entire localization process as a "black box”.

In addition to responding to the complexity of the software localization process, XLIFF also began to be considered as a solution to the equally complex documentation and technical communication localization process. Challenges in translating proprietary word processor and desktop publishing files, and even challenges having to do with the age-old task of localizing text strings in graphics showed proved to be a potential good fit for XLIFF.

XLIFF reduces this complexity of localising software by providing a standard, XML- based, end-to-end, tool neutral resource container. Software and documentation publishers can extract their localizable content into XLIFF and localize them using shrink-wrapped tools solutions, customized tools or automated enterprise workflow systems. Additional process efficiency is achieved by XLIFF’s built-in support for Computer Aided Translation technologies such as translation memory and machine translation.

Since the official release of XLIFF 1.0 in April 2002, its adoption throughout the software localization industry has been steadily growing. Several large multinational software publishers have deployed the standard for use in their internal localization processes. A number of tools providers have implemented XLIFF support in their shrink-wrapped tools offerings and localization services providers have embraced the standard and have been encouraging their customers to do the same. XLIFF is also playing a prominent role in emerging XML based Web Services technologies. XLIFF continues to be prominently discussed and presented at software development and localization seminars, industry trade shows, and academic events.

As XLIFF 1.0 began taking root in the software localization industry, the XLIFF Technical Committee set to work on further enhancing the emerging standard in order to provide more loss-less data interchange between tools, simplify its extensibility, and to refine its definition. On October 31, 2003, the XLIFF 1.1 Specification and XML Schema were released. Work on XLIFF 1.2 commenced in 2004 and the 1.2 Committee Specification was approved in July 2007. The XLIFF 1.2 committee specification will be submitted to the OASIS Standards Review board for consideration as a full OASIS standard.

This white paper is provided as a high level guide to anyone who seeks to better understand XLIFF in general terms, with particular emphasis on XLIFF 1.2’s features.

Note: All references to specific tools, services or companies are for illustrative purposes only – the XLIFF Technical Committee makes no endorsements or recommendations.

2.0 Overview of XLIFF 1.2

2.1 XLIFF TC Charter

At the start of the XLIFF group's life, our first challenge was to define our goals in a single, succinct statement. Writing and agreeing to the mission statement proved to be a significant challenge in and of itself. After XLIFF 1.0 was released and before work on XLIFF 1.2 commenced, the mission statement was further refined to emphasize our reliance on XML technology to achieve our goals.

The purpose of the OASIS XLIFF TC is to define, through extensible XML vocabularies, and promote the adoption of, a specification for the interchange of localizable software and document based objects and related metadata. To date, the committee has published two specifications - XLIFF 1.0 and XLIFF 1.1 - that define how to mark up and capture localizable data that will interoperate with different processes or phases without loss of information. The specifications are tool-neutral, support the entire localization process, and support common software and document data formats and mark-up languages. The specifications provide an extensibility mechanism to allow the development of tools compatible with an implementer's data formats and workflow requirements. The extensibility mechanism provides controlled inclusion of information not defined in the specification.

The state of software and documentation localization before XLIFF was that a software or documentation provider delivered their localizable resources to a localization service provider in a number of disparate file formats. Once software providers and technical communicators commenced implementing XLIFF, the task of interchanging localization data was simplified. Using proprietary and/or non-standard resource formats force either the source provider or the localization service provider to implement costly and inefficient pre-processing of localizable content. For publishers with many proprietary or non-standard formats, this requirement becomes a major hurdle when attempting to localize their software. For software developers and technical communicators employing enterprise localization tools and processes, XLIFF defines a standard but extensible vocabulary that captures relevant metadata for any point in the lifecycle which can be exchanged between a variety of commercial and open-source tools.

The first phase, completed 31 October 2003, created a 1.1 version committee specification that concentrated on software UI resource file localizable data requirements. The next phase consists of promoting the adoption of XLIFF throughout the industry through additional collateral and specifications, continuing to advance the committee specification towards an official OASIS standard, and revising the XLIFF spec to 1.2 version to support document based content segmentation and alignment requirements. To encourage adoption of XLIFF, the TC will define and publish implementation guides for some of the most commonly used resource formats (HTML, Java Resource Bundles, and Gettext PO Files).

XLIFF TC work and deliverables adhere to OASIS IPR policy.

The Charter captures the key aspects of our design objectives:

- The file format serves as a container for externalized data to be interchanged between software publishers, documentation writers (including, but not limited to documents written in DITA, Docbook, HTML, and other XML document formats), localization tools, and software services providers in order to facilitate all the phases of the localization process. To achieve this objective, the standard must be tool neutral.

- The file format must be extensible in order to support new and proprietary data formats. However, it must be structured and well defined so that tools that support the format would be reliable and consistent.

- End to end localization lifecycle support – information relevant to all project phases could be stored within the same file.

2.2 Why XLIFF is Needed

The typical localization project process poses many technical and logistical challenges. Chief among them are:

- **Many different formats:** The typical localization project is comprised of data stored within many unique file formats. For example, at last count, Oracle Corporation products were comprised of 17 actively localized, unique file formats. Some of these formats are proprietary (Oracle Forms, Seedata, Reports), but most are common across the industry (Java properties, Windows resources, HTML files, etc.)

- **Version Management:** Localization projects often run concurrently with core development, which means that source files will be made of up of multiple versions (or drops). Version management at the segment level is a very useful feature.

- **Workflow Metadata:** During the localization process, data passes through many hands. Data to be localized is typically externalized by software publishers, handed off to a localization service provider, who in turn may hand it over to translation subcontractors. At each stage of the process, the types of data required are unique to the particular phase (source/target text, Translation Memory, Machine Translation, Term base, etc).

- **Lack of Standards:** Previous localization standards limited support to very specific types of localization related data. For example, translation memory has TMX (LISA's Translation Memory eXchange), terminology glossaries can be represented in TBX (LISA's TermBase eXchange), and basic localization data could be extracted to Opentag. However, some of these standards are implementation specific, so a file created using one tool may not be used by another tool, even though both tools claim to support the same standard. Access to localization project data should be transparent to the localization tool regardless of the original native format.

- **Potentially missed translation memory opportunities:** Localization projects benefit from translating content once, and reusing the translations over and over. Risk exists that segmentation can cause content that should be a 100% match to appear not to match. This risk is amplified when the source of the content comes from different proprietary software formats. XLIFF 1.2 enables specific mechanisms for managing segmentation.

XLIFF was specifically designed to address each of these challenges.

2.3 Advantages of XLIFF

2.3.1. Localization Customer

XLIFF provides a number of benefits to the localization customer. It is an industry-standard single file format that completely encapsulates a customer's localizable data.

- **Less dependency on vendors that are able to work with proprietary file formats:** As an industry-standard format, XLIFF gives the localization customer "vendor independence." Customers with multiple proprietary file formats no longer need to concern themselves with providing information and/or tools for those files. The only file format the vendor needs to support is XLIFF.

- **Ease of file handling:** A localization customer may bundle all localizable data from multiple files into a single *project* XLIFF file for delivery to a vendor. Additionally, reference material and related binary objects, such as icons and bitmaps, can be packaged in the single XLIFF file.

- **Tighter control on what goes to localization:** In moving localizable data to XLIFF, the data can be filtered as to what should or should not go to the localization vendor. Additionally, even data that is sent to provide context for other translations can be marked as non-translatable within the XLIFF file.

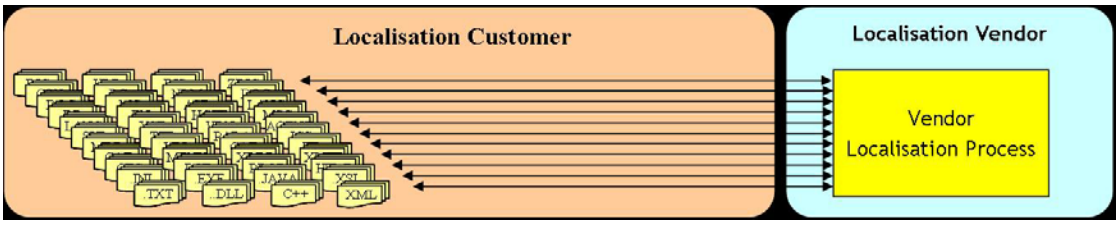
- **Controlled information flow:** Information about the data (author/developer notes, item properties, etc.) can be included with each translation unit to assist in the localization and provide context.

- **Ease of round-trip processing:** XLIFF can include data from other namespaces making it possible for a localization customer with an XML vocabulary of another namespace to include specific information necessary for easily creating language versions of the original document or other post-localization processing.

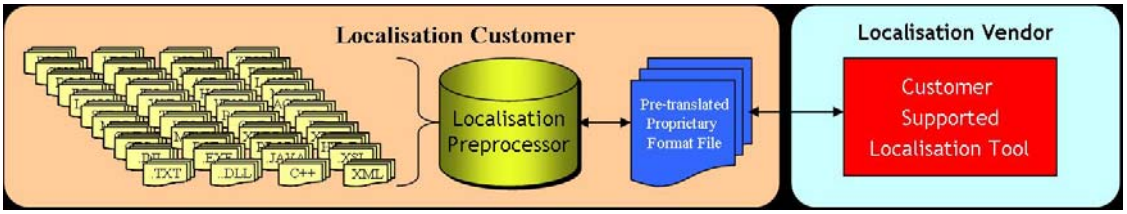
- **Completeness of the data:** Because XLIFF was designed to capture as much information as is commonly found in today's user interface file formats, the localizable data can completely and accurately be described in the XLIFF file. Thus, leaving out the guess work and multiple file hand-offs in attempting to get the UI right.

- **All advantages of XML-based processing:** Because XLIFF is an XML vocabulary, all the internationalization and processing advantages of XML are available to the localization customer. Additionally, often open-source utilities can be obtained for easily generating the XLIFF files from proprietary file formats.

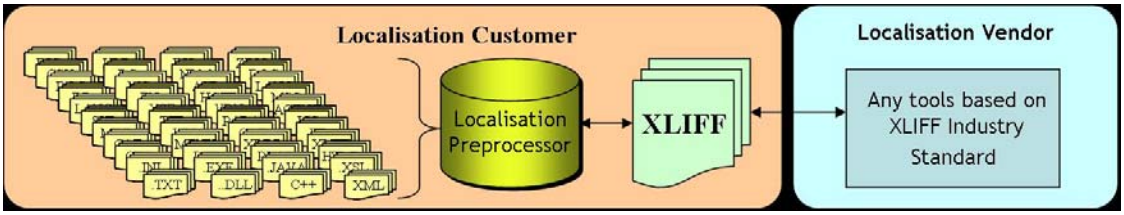
- **Streamlines localization file exchange:** For localization customers with many file formats, some of which are proprietary, delivery of files for localization involved sending all files to a localization vendor. The localization vendor then had to contend with the file format diversity. Often the customer had to support the vendor in handling proprietary formats. The vendor would have to become familiar with each customer's file formats.



Some localization customers created their own solution by converting their diverse file formats into a single file format. Since no single interchange format was available, the interchange file would either be a proprietary format created by the customer or a standard format that could not express all desired data. For proprietary data formats, the customer had to develop and maintain the tools and train the localization vendor on the use of the tools. The vendor would have to learn each customer's toolset.



XLIFF allows these customers to convert their proprietary and diverse file formats into a single industry-standard file format. As such, the localization vendor can select which tool to use from the many tools that support the standard. This tool can be used for any customer that uses XLIFF.



2.3.2.Tool Vendor

- **Focus on development of core functionality:** The adoption of the XLIFF file format allows tools vendors to focus on the development of core features for the localization process, such as those influencing cost and quality, rather than continually having to concentrate on writing accurate parsing technology for the latest file format that customers are using. Since XLIFF clearly identifies segments for localization, the tools vendor can concentrate on handling those segments accurately, such as improving leveraging, or validation algorithms rather than consuming resources locating those segments in the source file format. This greater focus on functionality benefits the industry as a whole.

- **XLIFF increases the footprint of the tool vendor's products:** Providing support for XLIFF automatically means that the tool vendor provides support for all those formats that can be converted into XLIFF. As XLIFF increases in popularity, more and more converters will be written to convert text content files into XLIFF. The tools vendor benefits from all these converters that are written.

- **Increased market share potential:** Along with the increasing product footprint, the tools vendor also gains from an increasing market based on these newly supported file formats.

- **Industry standard format:** Being an independent standard, XLIFF outlines a single mechanism for defining localization attributes such as memos, locks, source and target segments etc. Since this is a format defined by the industry, it becomes the industry standard way of defining these items. When customers are looking for advice on how best to store such information, the tools vendor can now point to an independent format that customers should follow and be sure that this information can be handled immediately.

- **Complete container for localization data:** XLIFF is a strong, well thought out format. Being defined by the various contingents in the localization industry, all points of view are covered. So, where tools vendors provide features spanning the localization industry, XLIFF provides a mechanism to do so.

- **All advantages of XML-based processing:** The XLIFF file format also provides all advantages of XML-based processing, such as support for different encodings, platform independence, existing parsers and browsers, support for associated x-path and xml transformations among many more.

2.3.3.Service provider

From the perspective of a company offering its services as a localization vendor there are a number of important advantages offered by XLIFF including:

- **Standard file format:** A major issue facing localization vendors is that many of their customers have proprietary file formats often requiring specific tools or certain processes. XLIFF removes this by having a standard interchange file format and, if used widely, will remove the need to become expert in localising particular proprietary file formats.

- **Tool independent:** XLIFF allows the creation of better quality tools as the tool providers are concentrating their work on improving the tool rather than on providing filters for different formats. It also has the advantage of allowing localizers to be expert in a small number of tools they know very well rather than being less expert with a wide range of tools.

- **Incorporation with workflow:** As an XML format XLIFF allows itself to being incorporated within a vendor's workflow.

- **Advantages of XML:** As an XML format XLIFF offers the advantages of XML such as the wide range of tools and parsers available and the use of XSLT stylesheets.

- **Open standards:** XLIFF is a standard designed by a committee involving people within the localization industry including those working for publishers, vendors and tool providers. As such it offers a solution that has been rigorously designed to meet the needs of the whole industry.

2.3.4.Benefits of XML

XLIFF as a vocabulary of XML provides many benefits to the users and implementers:

- **Powerful rendering options**:

  - XSL-FO
  - CSS
  - XSL can be used to perform many tasks on XLIFF documents, for example:
    - Display translatable content in Web browser
    - Generate statistics (e.g. number of localizable objects)

- **Low Cost and Ease of Development:**
  - Access to existing and often open-source XML implementations (lower costs)
  - Availability of many XML engines makes developing XLIFF applications inexpensive and easy
  - Web browsers can perform content-related checks (e.g. that certain characters appear as textual contents)
- **Use of XML internationalization features**
- **Better interoperability and cross-platform support**
- **Powerful transformation options (XSLT)**
- **Greater integration with Web services**

2.4 Origins of XLIFF

XLIFF got its start as an informal group of like-minded localization and globalization professionals based in Dublin, Ireland. The group was founded following informal discussions between Ian Dunlop of Novell, Paul Quigley formerly of Oracle, and Liz Tierney of Sun.

These IT executives recognized that the lack of established data standards in the localization industry resulted in universal additional localization costs and additional sometimes unreliable processes to support all the various native file formats to be localized.

Companies present at kick-off meeting: Novell, Oracle, Sun, and Berlitz. The original group expanded to include Alchemy Software Development, Lotus/IBM, Moravia IT, RWS Group, and Lionbridge.

The original group used the Yahoo! eGroup “DataDefinition” as its repository for program management, discussions, and documentation. Access to Yahoo! group was liberally administered in order to permit anyone interested to view progress but participation in meetings was limited to invited individuals in order to maximize productivity. The XLIFF 1.0 development schedule was quite aggressive and the 1.0 specification was delivered on time.

Meetings were conducted fortnightly, entirely via teleconference. Two face-to-face meetings took place - both hosted by Novell at both ends of the project lifespan. Supplemental deliverables were a DTD, sample files, and a White Paper describing features and functionality of XLIFF.

Once the core deliverables were completed, the group began preparing for public distribution of the specification throughout the industry for comment. In the process of preparing for the press announcement for the specification, it was discovered that our respective corporate legal departments had not approved the Intellectual Property Rights status of XLIFF. The ensuing legal process to define the IPR regime for XLIFF concluded in November 2001.

Although XLIFF had not suffered from any IPR conflicts and therefore defining an IPR policy was fairly straight forward, the rigorous and lengthy process made it abundantly clear to our group that XLIFF could no longer continue as an informal, ad hoc, inter-company consortium. We lacked the organisational infrastructure and procedural expertise that was needed to support the advancement of our work. Therefore, in the autumn of 2001, the XLIFF group decided to investigate and select a new home within a formal standards body.

In December of 2001, the XLIFF group reviewed a number of consortia as potential homes for our work. Specifically, we looked at the processes and benefits of working within OASIS, W3C, and LISA. We decided to make OASIS our home because of its XML focus and because it provided us with a turnkey infrastructure to operate within. Another important factor in choosing OASIS was that many of our member companies were already members of the consortium.

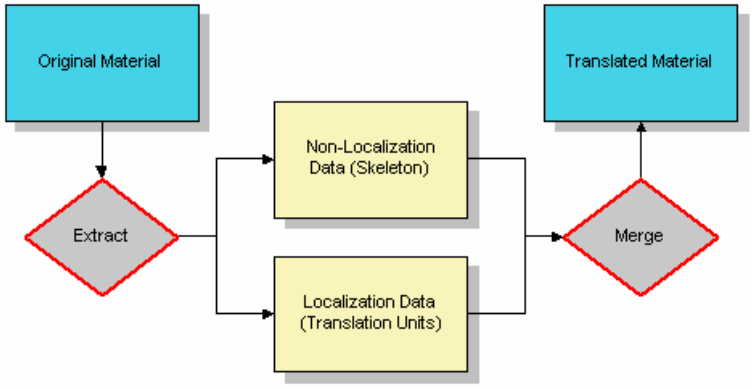
The OASIS XLIFF Technical Committee (TC) held its first meeting in January 2002. The XLIFF 1.0 Specification was submitted to the TC as the starting point for its work. The contributed work was reviewed and approved by the TC on 15 April 2002.

In October of 2003, the XLIFF 1.1 Specification (Committee Specification) and XML Schema were approved. Work on XLIFF 1.2 began shortly after. XLIFF 1.2 was approved as Committee Specification in May 2007. The XLIFF 1.2 Committee Specification is located at http://docs.oasis-open.org/xliff/xliff-core/xliff-core.html.

3.0 Architecture Overview

XLIFF is based on the concept of extracting the source localization-related data from the original format, and merging it back in place after the localization has been done.

Figure 1 – Extraction/Merge principle:



Depending on the extract/merge method, the parts that are not related to localization can be preserved temporarily into the *Skeleton*. Or, usually when the source is already XML, the non localizable parts can be preserved within the XLIFF hierarchy using <group> elements to preserve the hierarchy.

There are no rules to date on how to represent the data in the Skeleton itself, this is left to the discretion of the filters. XLIFF 1.2 focuses on how to store and organize the extracted parts. Skeletons can be either embedded directly in the XLIFF document with the `<internal-file>` element or simply referred to with the `<external-file>` element.

3.1 Extracted Data

The text extracted from the original source material is stored in *translation units*. Each `<trans-unit>` element contains a `<source>` element where the original text is copied. The translation goes into a corresponding `<target>` element. The content of the `<target>` element depends on the stage at which the document is. Often tools set the initial translation text to the source text.

Example of extracted text, a sentence in Middle-English (and its contemporary English transcription):

```
<trans-unit id='2'>
 <source xml:lang='enm'>A lovely lady of leere &#x2022; in lynnen yclothed,
  Cam doun fom the castel &#x2022; and called me faire.</source>
 <target xml:lang='en'>A lady, lovely of looks &#x2022; in linen clothed,
  Came down from a castle &#x2022; and called me fairly.</target>
</trans-unit>
```

3.2 Abstracted Inline Codes

Inline codes (e.g. markers for bold or italics, links information, or image references) can be represented using either an encapsulation mechanism or a placeholder method. Those are derived respectively from TMX (LISA's Translation Memory Exchange Standard), and OpenTag, a localization data container (www.opentag.com).

The encapsulation mechanism consists of bracketing the inline codes between special elements. XLIFF has syntax very close to TMX: <bpt> (begin paired-tag), <ept> (end paired-tag), <it> (isolated tag), and <ph> (placeholder tag). If some text exists within a span of encapsulated code, it can be delimited by a <sub> element, if the tool does not extract it as a separate segment, which would be the best solution.

The placeholder method consists of extracting the inline codes out to the skeleton file, and replacing them with placeholders to indicate their locations. The <g> element is the placeholder equivalent of <bpt> and its ending tag </g> the equivalent of <ept>. The elements <bx/> and <ex/> allows for the handling of overlapping inline codes; and the <x/> element is the equivalent of <ph>.

Example of inline tagging (with code portions underlined):

```
Click <bpt id='1'>&lt;a href="start.htm"></bpt>here<ept id='1'>&lt;/a></ept> to
start.
```

In addition to the elements representing inline codes, XLIFF also provides a general-purpose element to delimit span of content inside the text. The <mrk> element can be used to demarcate various properties necessary to the tools during translation.

When it is important that the XLIFF file can be processed efficiently with XML standards, like XSLT, it is better to not use the encapsulation mechanism. For example, 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>
```

3.3 Binary Data

Non-textual components such as images, cursor, icons, etc. can also be included in an XLIFF document using a <bin-unit> element. Like the Skeleton, they can be either set as an external reference, or embedded within the document itself.

Reference to binary data, here a Windows cursor:

```
<bin-unit id='1' resname='IDC_POINTER_COPY'
 mime-type='image/cursor' restype='cursor'>
 <bin-source>
```

```
  <external-file href='arrowcop.cur'/>
 </bin-source>
</bin-unit>
```

Same example as above, but this time embedded in the document, as Base-64 text:

```
<bin-unit id='1' resname='IDC_POINTER_COPY'
    mime-type='image/cursor' restype='cursor'>
 <bin-source>
  <internal-file form='base64'>
AAACAAEAICAAAAEAAQAwAQAAFgAAACgAAAAgAAAAQAAAAAEAAQAAAAAAgAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAA////AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAMAAAADAAAABgAAAAYAAAAMAAAAjAAAANgAAAD4AAAA/wAAA
P4AAAD8AAAA+CAAAPBQAADg2AAAwQQAAIDYAAAAUAAAACAAA////////////////////////
////////////////////////////////////8////+H////h////w///+8P///
mH///4h///+A////gA///4Af//+AP///gH///4D3//+B4///g8H//4eA//+Pwf//n+P////3
//8=</internal-file>
 </bin-source>
</bin-unit>
```

If any text is associated to the binary object it can be stored and translated using one or more optional <trans-unit> in each <bin-unit> element.

3.4 Pre-Translation

An important feature of XLIFF comes from the requirement of being able to pre-translate entries with one or more propositions. The <alt-trans> element allows great flexibility for this. An unlimited number of <alt-trans> elements can be associated with a given translation unit. This provides not only a best match, but as many matches as desired. If desired, you could associate translation memory matches with each source text entry and send the document for translation without a companion TM.

Example of a pre-translated translation unit:

```
<trans-unit id='1'>
 <source xml:lang='en'>The text to translate.</source>
 <alt-trans origin='MT system' match-quality='high'>
  <target xml:lang='fr'>Le text &#xe0; traduire.</target>
 </alt-trans>
 <alt-trans origin='Excalibur Project' match-quality='80%'>
  <source xml:lang='en'>The sentence to translate.</source>
  <target xml:lang='fr'>La phrase &#xe0; traduire.</target>
 </alt-trans>
 <alt-trans origin='Project-Duncton' match-quality='75%'>
  <source xml:lang='en'>The text to change:</source>
  <target xml:lang='fr'>Le text &#xe0; changer:</target>
 </alt-trans>
</trans-unit>
```

If a <target> element is available under the <trans-unit> element, it must contain the latest translation. If no <source> element is present in the <alt-trans> element, it is assumed the proposed translation is for the same text as the main <source> element.

3.5 Keeping Track of Things

Still using the <alt-trans> element, XLIFF provides ways to keep track of the changes done to the data during their journey across the successive stages of the localization process. Each <target> element in an alternate translation unit can be flagged with a phase-name attribute. This attribute refers to the phase of the process defined in the header of the document, where you can find all the details of that corresponding phase.

Example of change log for a translation unit:

```
<header>
 <phase-group>
  <phase phase-name='trans' process-name='translation' tool-id="BabelEditor"
   contact-email='marie-charlotte@trad_boutique.com'
   date='2002-10-01T23:32:23Z'></phase>
  <phase phase-name='edit' process-name='edit' tool-id='Borneo'
   contact-email='roland@roncevaux-traduction.com'
   date='2002-10-02T14:20:03Z'></phase>
 </phase-group>
 <tool tool-id="BabelEditor" tool-name="BabelEditor" />
 <tool tool-id="Borneo" tool-name="Borneo" />
</header>
<body>
 ...
 <trans-unit id='1'>
  <source xml:lang='en'>.</source>
  <target xml:lang='fr'>Le texte &#xe0; traduire.</target>
  <alt-trans>
   <target xml:lang='fr' phase-name='trans'>Le texte a traduire</target>
  </alt-trans>
  <alt-trans>
   <target xml:lang='fr' phase-name='edit'>Le texte &#xe0; traduire.</target>
  </alt-trans>
 </trans-unit>
</body>
```

This allows users to see what changes have been done, by whom, when, using which tool, and so forth. Such a mechanism can prove very useful during edit, proof and review stages. In all cases, the main <target> element of the <trans-unit> should contain the last version of the localized data.

3.6 Metadata

Another aspect of using a standard format to carry data from different original formats is to consolidate all the information of the same nature under a common set of attributes. XLIFF offers a wide range of such metadata. For example, the attributes maxwidth and minwidth indicate the maximum and minimum number units allowed for the length of the content. By default, the unit is the pixel, but you can use the size-unit attribute to change it to another unit such as byte, or character.

Example of fixed-size strings:

```
<trans-unit xml:space='preserve' id='1045'  maxwidth='10' minwidth='10'
 size-unit='char'>
 <source xml:lang='en'>   pages: </source>
 <target xml:lang='pl'>  strony: </target>
</trans-unit>
```

Other types of metadata are also available.

4.0 Using XLIFF

The best use of XLIFF assumes that the standard is implemented at all the different stages of the localization process. By understanding the constructs within XLIFF and applying them in authoring applications, users can contribute to a more seamless and cost effective localization process.

While XLIFF contains features for all the different stages of the localization process and provides benefits at all stages, it is more likely that adoption by an organization be piecemeal. Small sections of the process can be converted over to the new format, and a greater understanding of its mechanisms achieved before converting more significant sections of the process to XLIFF.

This piecemeal approach could involve strands of the localizable content going through the entire process in an XLIFF format, while other strands remain in a traditional format. This is effectively a parallel approach. Alternatively, in a series fashion, it could involve sections of information being converted to XLIFF for a particular stage of the process. E.g. files may be converted to XLIFF for linguistic review.

It should be understood that XLIFF places considerable importance on the organic nature of information gathered throughout the localization process and so offers many benefits when used to transfer information between different processes where historical information is important. E.g. XLIFF can store information about where a particular translation came from, or why an alternative translation was rejected as well as dialogue between players in the localization process.

Since XLIFF is a standard interchange format for localization, it offers benefits over native file formats when moving between tools. Tools that interpret the XLIFF format understand not only the content, but also the meta data, e.g. status flags, memo information, source and target text, etc. This allows for seamless transfer of information between tools and achieves the desire in the mission statement that XLIFF be tool neutral.

There are now commercial tools on the market supporting the XLIFF standard. These fall into two categories. There are those that support the standard itself and interpret the XLIFF statements in a document. Since XLIFF is based on xml, there are also some tools that support XLIFF via a standard xml layer.

4.1 Native XLIFF-Enabled Translation Tools

Tools in this category understand the XLIFF schema and provide features in the tool that read and interpret the actual XLIFF elements and attributes.

Eg.

```
<trans-unit translate=înoî>
    <source>Hello World</source>
</trans-unit>
```

In the above example, a native XLIFF enabled tool identifies the segment, and its source text, however, it understands that the ‘translate’ attribute in the <trans-unit> element relates to the text ‘Hello World’ and so that string will be locked in the editor. Tools that do not natively understand XLIFF will simply present

all <source> items for translation. Some tools may actually present the value of translate, i.e. ‘no’ for translation.

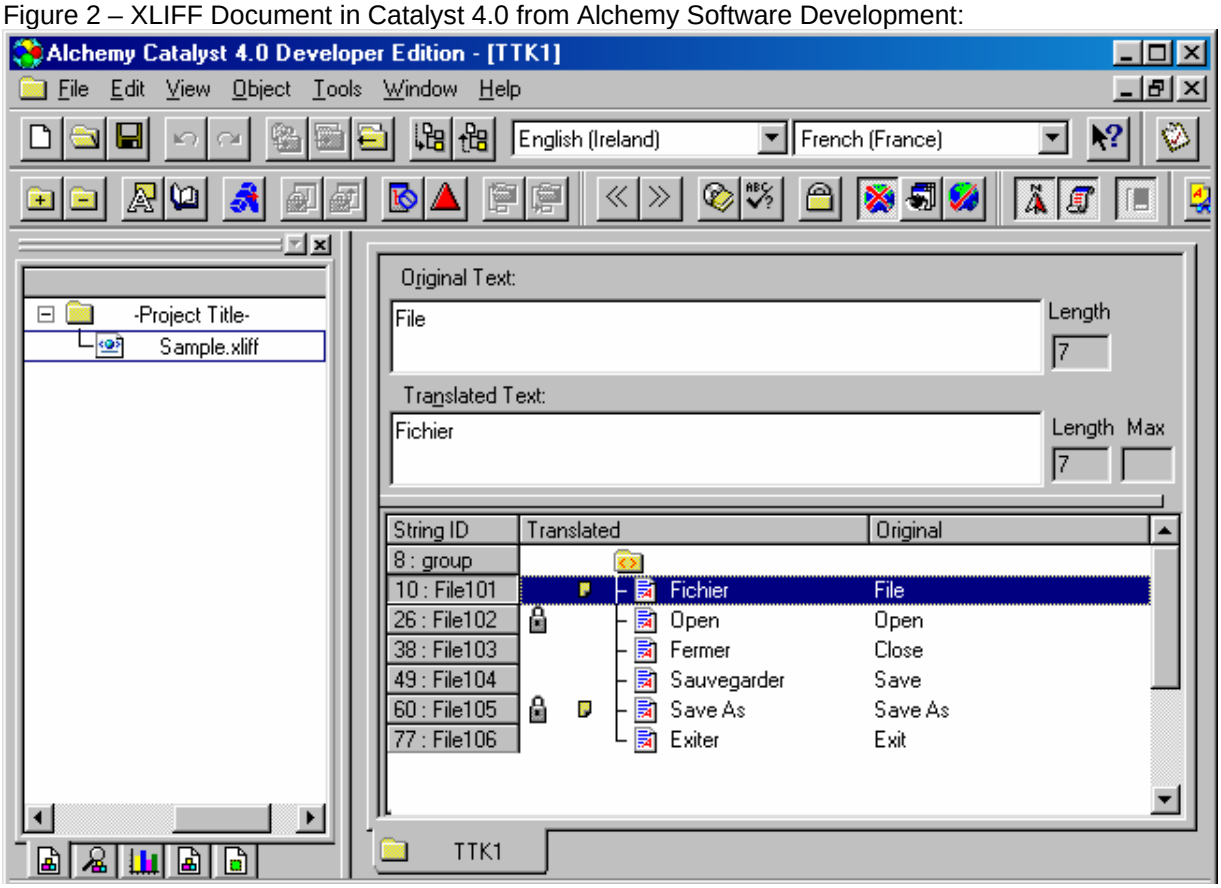
Below is a simple XLIFF document that contains the menu text from an application. Things to note here are…

- the source and target text elements
- the translate = “no” instruction on the second item
- the application resource identifier contained in the resname attribute
- the note instruction from the engineer to the translator

```
<xliff>
    <file source-language="EN" datatype="plaintext" original="User">
      <body>
         <group>
            <trans-unit id="101" resname="File101">
               <source>File</source>
               <target>Fichier</target>
               <note from="Joe" priority="6">Special attention needed</note>
            </trans-unit>
            <trans-unit id="102" resname="File102" translate="no">
               <source>Open</source>
               <target>Open</target>
            </trans-unit>
            <trans-unit id="103" resname="File103">
               <source>Close</source>
               <target>Fermer</target>
            </trans-unit>
            <trans-unit id="104" resname="File104">
               <source>Save</source>
               <target>Sauvegarder</target>
            </trans-unit>
            <trans-unit id="105" resname="File105" translate="no">
               <source>Save As</source>
               <target>Save As</target>
               <note from="Joe" priority="1">Needed for documentation </note>
            </trans-unit>
            <trans-unit id="106" resname="File106">
               <source>Exit</source>
               <target>Exiter</target>
            </trans-unit>
         </group>
      </body>
    </file>
</xliff>
```

Below can be seen an image of how this file looks when inserted into Alchemy Catalyst. When this file is inserted these items are understood and displayed as they are intended because Catalyst interprets the meaning intended in these XLIFF constructs. Catalyst displays source and target text appropriately, it also locks the strings that should not be translated and adds a memo to the items that contain <notes> in the XLIFF document.

Figure 2 – XLIFF Document in Catalyst 4.0 from Alchemy Software Development:



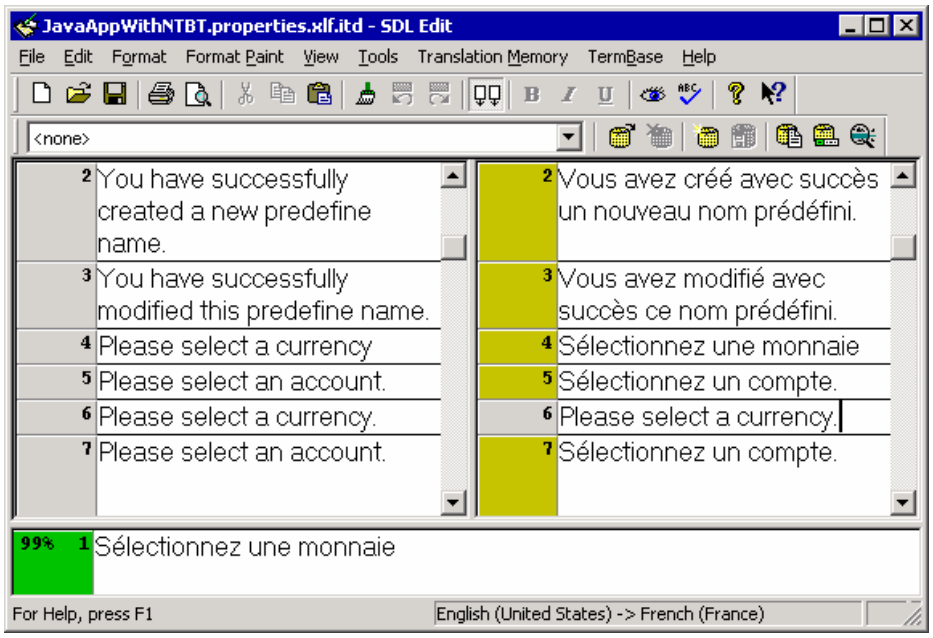
4.2 XML-Enabled Translation Tools

There is a group of tools that provide support for XLIFF via a standard xml parser. Some of the core XLIFF statements such as translate = yes | no and the <note> element may not be interpreted, but simple text editing is available.

Let's assume you want to use an XML-enabled translation tool that does not have specific support for XLIFF. You can still use the application by treating the XLIFF input as just another XML format. You will not benefit from all of the features an XLIFF file could offer—an important limitation—but simple translation tasks using a TM will be available. The only requirement is to make sure the translatable text is initially duplicated in the `<target>` element of each translation unit. This is already something many XLIFF filters do by default.

Once an XLIFF document is ready, it can be used with any of the XML-enabled translation tools available. All of them require the one-time creation of a profile where you describe what elements and attributes are to be translated for a given document type. For XLIFF only the `<target>` element is to be translatable. With that done, you can simply translate the XLIFF document as shown in Figure 2.

Figure 3 – XLIFF Document in SDLX from SDL International:



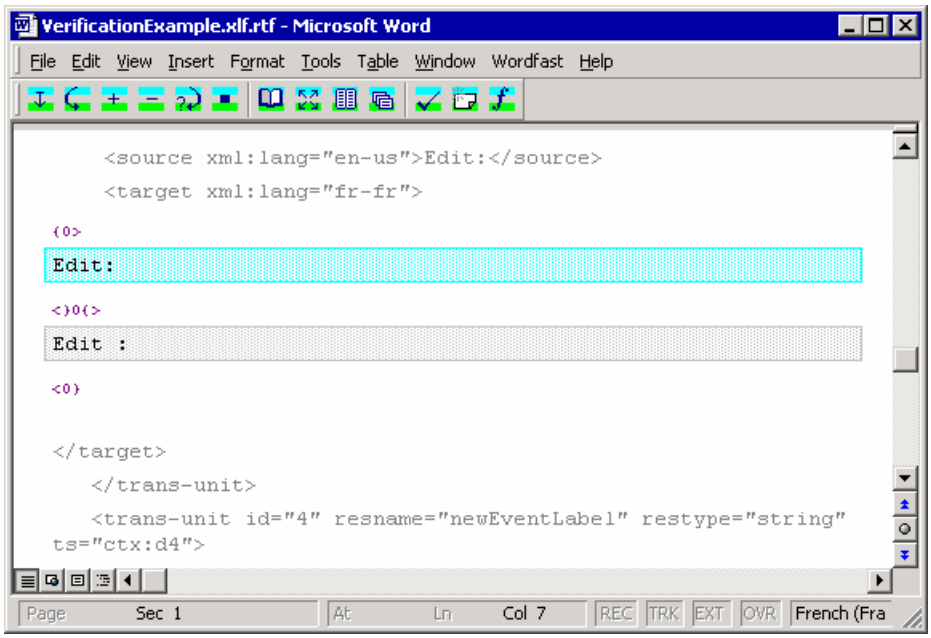
It is very important to keep in mind that translating an XLIFF document using a non-XLIFF-enabled tool is **not** the long-term solution. There is a huge difference between being able to simply translate the content of a set of `<target>` elements, and being able to use all the powerful features XLIFF offers. A tool that truly supports XLIFF should be able to:

- Open any valid XLIFF document (without any workaround being necessary)
- Interpret many of the XLIFF statements and maintain any it does not understand
- Keep track of the changes made between the start and end of a given task (translation, edit, proof, etc.)

4.3 Non-XML Translation Tools

It is also not very difficult to prepare XLIFF documents for tools that are not XML-aware. A traditional way of handling such case is to add a layer of colour-coded RTF codes on top of the XLIFF document, transforming it into an RTF file that most translation utilities can deal with. You simply need to remove the RTF layer after translation to get back the translated document in its plain XLIFF form. Obviously, here as well, the file needs to have the original text in the `<target>` elements. The Figure 3 shows prepared in such a way.

Figure 4 – XLIFF Document with RTF Layer used with Wordfast:



4.4 XML and XLIFF

More and more formats are now XML applications: XHTML, SVG, Docbook, DITA, ebXML and XLIFF are examples of this trend. XML is a highly structured format and having your original source format in XML makes the localization process much easier. In some cases it is not necessary to extract the data out of the original format to perform the translation. However, in some cases, having the data in an XLIFF format can provide additional advantages. This leads to the need of converting data from a proprietary XML format to and from XLIFF. Being xml based makes XLIFF very easy to work with. The following shows in few lines of an xml transformation how to convert from one xml format to XLIFF.

Consider the following sample source file in xml containing 4 text strings.

```
<?xml version="1.0"?>
<file>
    <text>File</text>
    <text>Open</text>
    <text>Close</text>
    <text>Exit</text>
</file>
```

These next few lines are the transformation.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
        <xliff>
            <body>
                <xsl:for-each select="file">
                    <xsl:for-each select="text">
                        <trans-unit>
                               <source>
                                <xsl:apply-templates />
                               </source>
```

```
                                <target>
                                <xsl:apply-templates />
                                </target>
                            </trans-unit>
                        </xsl:for-each>
                    </xsl:for-each>
                </body>
            </xliff>
        </xsl:template>
    </xsl:stylesheet>
```

If these lines are saved as ToXliff.xslt, and the following line is added to the top of the original xml file

```
<?xml-stylesheet type="text/xsl" href=îToXliff.xslt"?>
```

then an XLIFF file is the output of the transformation.

```
<?xml version="1.0" encoding="UTF-16" ?>
- <xliff>
  - <body>
    - <trans-unit>
        <source>File</source>
        <target>File</target>
      </trans-unit>
    - <trans-unit>
        <source>Open</source>
        <target>Open</target>
      </trans-unit>
    - <trans-unit>
        <source>Close</source>
        <target>Close</target>
      </trans-unit>
    - <trans-unit>
        <source>Exit</source>
        <target>Exit</target>
      </trans-unit>
  </body>
</xliff>
```

The resulting XLIFF file, as shown by Internet Explorer is shown in this image.

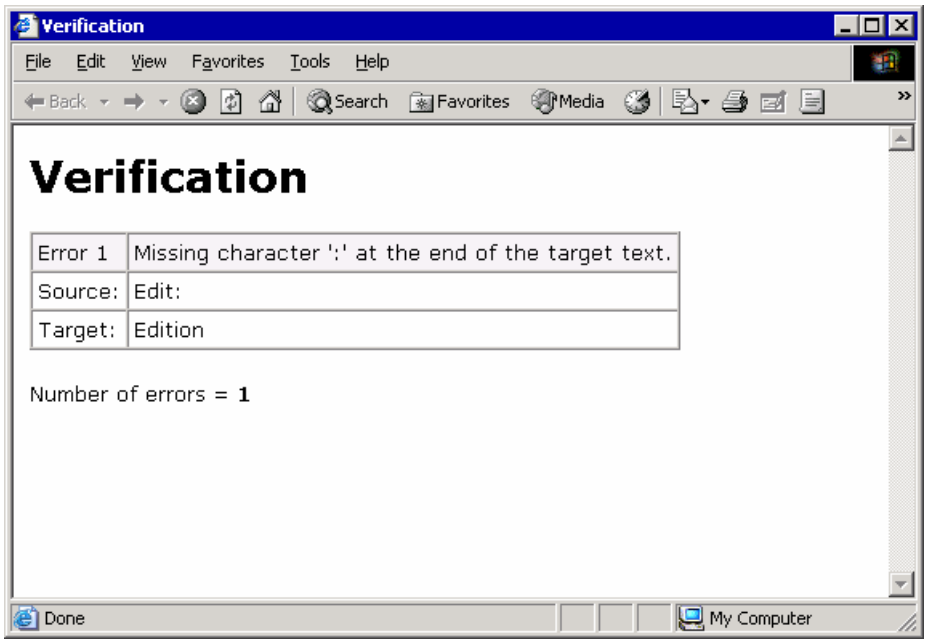
While this is a simple example, it clearly shows how powerful the combination of xml features with XLIFF can be.

Still using XML technologies and XLIFF, you can also easily develop style-sheets to execute various generic verification tasks on any XLIFF documents and display the results when the document is open on a browser. The following example illustrates such usage: an XSLT template compares the source and target ending characters and flags any translated text that has no colon when the source has one. An example of the result is shown in Figure 4, when applied to the same file as in Figure 3. The principle can be extended, for each language, to any other punctuation checking, leading and trailing mandatory spaces, and so forth.

Simple example of an XSLT template for verification:

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<xsl:stylesheet
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0"
 xmlns:msxsl="urn:schemas-microsoft-com:xslt"
 xmlns:hog="http://www.hogwart-translations.com/xml-verification">
 <msxsl:script language="JScript" implements-prefix="hog">
  <![CDATA[
  var g_nCount = 0;
  function ErrorCount () {
     return(g_nCount);
  }
  function CheckLastChar(p_Src, p_Trg, p_Char) {
     var sSrc = new String(p_Src.nextNode().text);
     if ( sSrc.length<1 ) return("");
     var sTrg = new String(p_Trg.nextNode().text);
     if ( sTrg.length<1 ) return("No target text.");
     var cTmp = sSrc.charAt(sSrc.length-1);
     if ( cTmp == p_Char ) {
        if ( cTmp != sTrg.charAt(sTrg.length-1) ) {
           g_nCount++;
           return("Missing character '" + p_Char +
                  "' at the end of the target text.");
        }
     }
     return("");
  }
  ]]>
 </msxsl:script>
 <xsl:template match="text()"/>
 <xsl:template match="comment()"/>
 <xsl:template match="//alt-trans"/>
 <xsl:template match="/xliff">
  <html>
   <head>
    <title>Verification</title>
   </head>
   <body>
    <h1>Verification</h1>
    <table border="1" cellspacing="0" cellpadding="3">
     <xsl:apply-templates/>
    </table>
    <p>Number of errors = <b><xsl:value-of select="hog:ErrorCount()"/></b></p>
   </body>
  </html>
 </xsl:template>
 <xsl:template match="//source">
  <xsl:variable name="R1" select="hog:CheckLastChar(.,../target,':')"/>
  <xsl:if test="$R1!=''">
   <tr>
    <xsl:attribute name="style">background:#F0F0F0</xsl:attribute>
    <td>Error <xsl:value-of select="hog:ErrorCount()"/></td>
    <td><xsl:value-of select="$R1"/></td>
   </tr>
   <tr>
    <td>Source:</td>
    <td><xsl:value-of select="."/></td>
   </tr>
   <tr>
    <td>Target:</td>
    <td><xsl:value-of select="../target"/></td>
   </tr>
  </xsl:if>
 </xsl:template>
</xsl:stylesheet>
```

Figure 5 – Result of the previous XSLT template view in Internet Explorer:



As these examples show, there are many ways to take advantage of the XML technologies. Using XLIFF as the common format makes your utilities easier to create and maintain.

4.5 Interoperability

One of the primary aspects of XLIFF is to allow data to progress from one step to another using different tools—commercial or in-house. Interoperability was one of the key requirements of XLIFF and it continues to be an important aspect of the 1.2 specification.

Having a common set of attributes and elements does not necessarily bring interoperability. They also must be used in a consistent manner. Some formats can have quite complex representations once extracted into XLIFF. With the release of XLIFF 1.2, Representation Guides were offered for HTML, Java Bundles, and Gettext PO. Subsequent to the XLIFF 1.2 specification release the TC will define profiles of XLIFF representations for other widespread source formats.

4.6 Extensibility

Along with interoperability, extensibility is also very important. The XLIFF 1.2 specification comes with an XML Schema representation of the format. This move opens the door for using the XML Namespace mechanism to its full potential. The new version prescribes specific *points of extension* where users can insert their own elements and attributes, defined in separate namespaces.

XLIFF 1.2 introduced the concept of Strict vs. Transitional XML Schemas.

XML Schemas may be written in ways to enable users to use content models from other XML Schemas, in specified places, and enforce three different degrees of validation (http://www.w3.org/TR/2001/REC-xmlschema-1-20010502/#process_contents). The validation may be skipped, optional, or required.

Validation of Included Content

XML Schema allows the following rules regarding validation for included content:

If the *process contents* attribute is set to *skip*, no validation of the included content is done. If the *process contents* attribute is set to *lax*, validation is done if the schema for the included content is available. If the *process contents* attribute is set to *strict*, it is required that the schema for the included content be available, and the validation must take place.

XLIFF 1.2 sets the *process contents* attribute to *skip* for included content in the Transitional schema, and to *strict* in the Strict schema.

Points of Extension in XLIFF 1.2

XLIFF 1.2 has many elements with extension points.

XLIFF provides several extension points in the following elements: `<alt-trans>`, `<bin-unit>`, `<group>`, `<header>`, `<tool>`, `<trans-unit>`, and `<xliff>`.

The following elements allow non-XLIFF attributes: `<alt-trans>`, `<bin-source>`, `<bin-target>`, `<bin-unit>`, `<bpt>`, `<bx/>`, `<ept>` , `<ex/>`, `<file>`, `<g>` , `<group>`, `<it>`, `<mrk>` , `<ph>`, `<seg-source>`, `<source>`, `<target>`, `<tool>`, `<trans-unit>`, `<x/>`, and **`<xliff>`**.

Examples of Extensibility with XLIFF 1.2

The first step is to declare the namespace for the content that will be added:

```
<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-strict.xsd
       urn:tektronix2 all_tek_elements.xsd"
       version="1.2"
       xmlns:tek="urn:tektronix2"
```

This allows content to be added at a prescribed extension point. An example of an element in XLIFF 1.2 with legal extension points is in the trans-unit element. Here's a sample:

```
   <trans-unit id='Z123' tek:prelimsource="c2-18-87j2-27-90p1-13-60">
    <source xml:lang='en'>one two three</source>
    <target xml:lang='de'>eine zwei drei</target>
    <tek:table>
     <tek:title>Model Specifications</tek:title>
     <tek:tgroup cols="2">
      <tek:colspec colname="col1"/>
      <tek:colspec colname="col2"/>
      <tek:tbody>
       <tek:row>
        <tek:entry>
         <tek:Para>Bandwidth (-3 dB)</tek:Para>
        </tek:entry>
        <tek:entry>
         <tek:Para>DC to 500 MHz (in a host instrument with bandwidth
          &gt;1.5 GHz)</tek:Para>
        </tek:entry>
       </tek:row>
```

```
    <tek:row>
     <tek:entry>
      <tek:Para>Probe Tip Bandwidth
       (with P6139A at -3 dB)</tek:Para>
     </tek:entry>
     <tek:entry>
      <tek:Para>DC to 500 MHz (host instrument bandwidth
       &gt;1.5 GHz)</tek:Para>
     </tek:entry>
    </tek:row>
   </tek:tbody>
  </tek:tgroup>
 </tek:table>
</trans-unit>
```

Obviously, you only want to use your own extensions if XLIFF does not already support your needs.

4.7 Beyond XLIFF Documents

Using namespaces, there are opportunities to allow other XML formats to use XLIFF constructs directly. This “embedded XLIFF” could simplify immensely the creation of localization-aware proprietary formats.

As long as the schema for the document that embedded XLIFF is to go in has its own extension point, the XLIFF elements could be processed in the context of that document. For example, if a table model included an extension point like this:

```
<xsd:element name="entry">
  <xsd:complexType>
   <xsd:choice minOccurs="0"
               maxOccurs="unbounded">
    <xsd:any namespace="##other"
             processContents="strict"
             minOccurs="0"
             maxOccurs="unbounded"/>
    <xsd:element ref="Para" />
    <xsd:element ref="TableSub" />
   </xsd:choice>
    <xsd:attribute name="namest" type="xsd:string" />
    <xsd:attribute name="nameend" type="xsd:string" />
 ...
  </xsd:complexType>
 </xsd:element>
```

An XLIFF trans-unit could be embedded, like this:

```
<table>
 <title>Windows Memory Requirements for Oscilloscope or PC
  When Using TDSDVD</title>
 <tgroup cols="3">
  <colspec colname="col1"/>
  <colspec colname="col2"/>
  <colspec colname="col3"/>
  <tbody>
   <row>
    <entry namest="col1" nameend="col3">
     <xlf:trans-unit id='Z123'>
      <xlf:source xml:lang='en'>one two three</xlf:source>
      <xlf:target xml:lang='de'>ein zwei drei</xlf:target>
     </xlf:trans-unit>
    </entry>
```

```
    </row>
    <row>
     <entry morerows="1">
      <Para>Oscilloscope Record Length</Para>
     </entry>
     <entry namest="col2" nameend="col3">
      <Para>Windows RAM on Oscilloscope or PC where TDSDVD is Running</Para>
     </entry>
    </row>
   </tbody>
  </tgroup>
 </table>
```

4.8 Filters that allow transforming different formats to XLIFF and back

There are now many commercial and open source filters that allow transforming different formats to XLIFF and back. The following document types are some of the formats that can be directly processed using XLIFF:

Software Localisation:

- Windows C/C++ Resources (RC)
- Windows .NET Resources (ResX)
- Java .Properties
- Portable Objects (PO/POT)
- Binary resources (.exe and .dll)
- JavaScript/EcmaScript
- International Components of Unicode (ICU) Resource Bundle
- Oracle Business Applications

Documentation:

- HTML
- FrameMaker (MIF)
- Adobe InDesign (INX)
- Microsoft Office 2007
- OpenDocument (OpenOffice/StarOffice)
- Rich Text Format (RTF)
- Generic XML

Specialised XML Filters:

- DocBook
- DITA
- XHTML
- xml:tm
- Microsoft Word 2003 ML

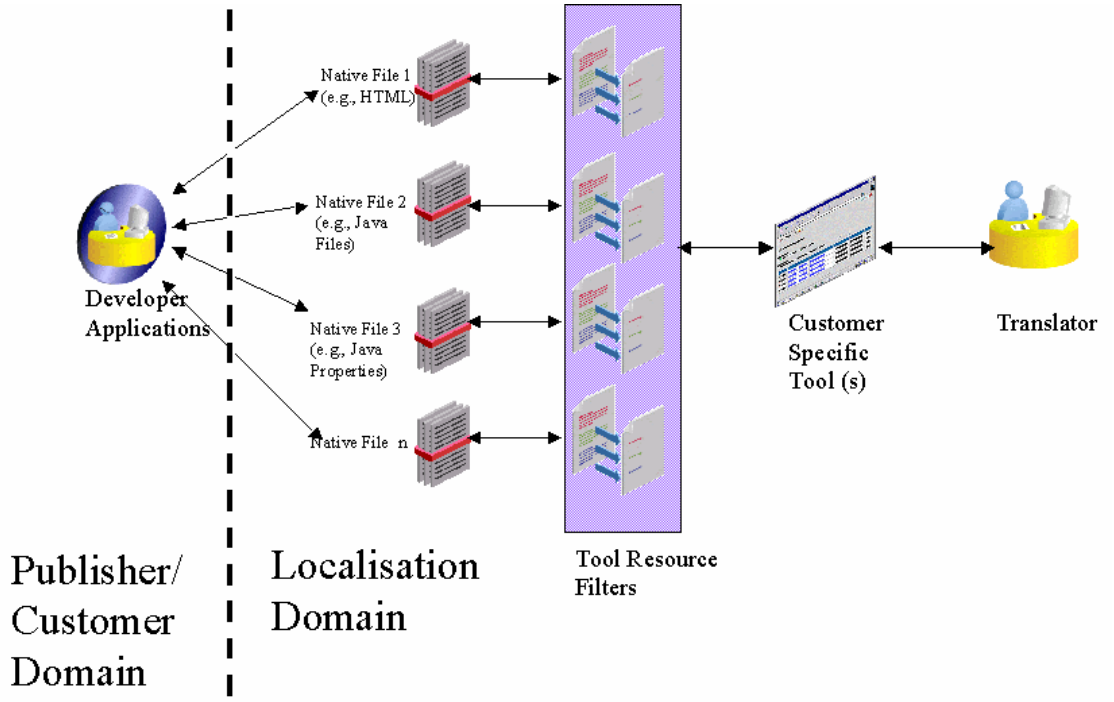
5.0 Use Cases

Business use cases are useful for visualizing workflow processes. The following set of business use cases illustrates how XLIFF can be used to streamline and improve the localization process.

5.1 Without XLIFF

This use case describes a very primitive localization process. In this example, a developer writes code, and hands it off to a localization engineer. All of the process complexity exists in the localization domain. The localization engineer receives all of the resources in their original native format. In order for the native files to be localized, tool filters must be available that interpret the localizable resources in the native file, or possibly complicated multi-tool solutions are required in order to translate all the native files.

Figure 6 – Workflow without XLIFF:

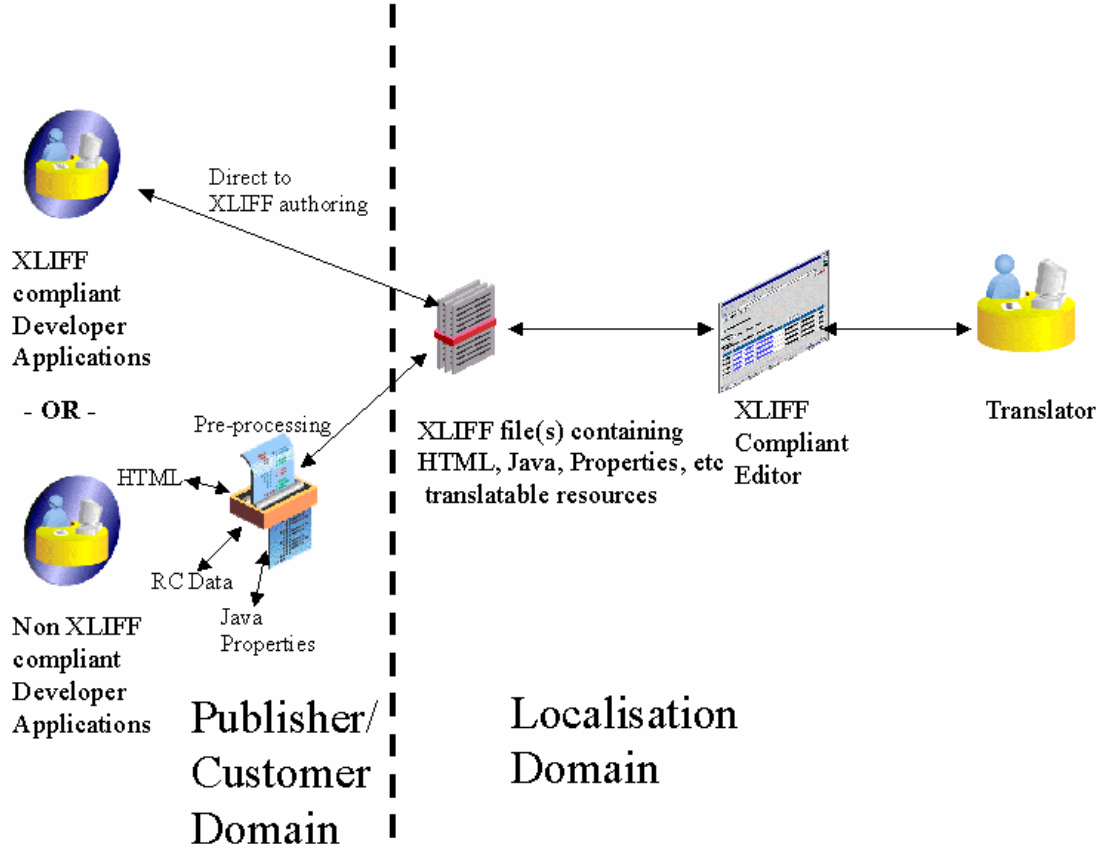


Each time a new native format is introduced or when an existing one is changed, localization tools engineers who may not be experts in the native format must revise the tool and/or filter. And since new or changed resource types are generally discovered when the tools fail during the midst of a project, supporting internal localization tools is a firefight. This model is highly reactive, and will inevitably result in project delays and costs due to frequent retooling. It is also more likely introduce potential poor quality of translated work due to misinterpreting data when converting between native format and the localization tool's internal data representation.

5.2 With XLIFF

This use case illustrates a simple workflow where XLIFF is introduced into a process very similar to the previous one. Note that only XLIFF based data is handed off from the developer/publisher to the localization specialists and back again.

Figure 7 – Workflow with XLIFF:

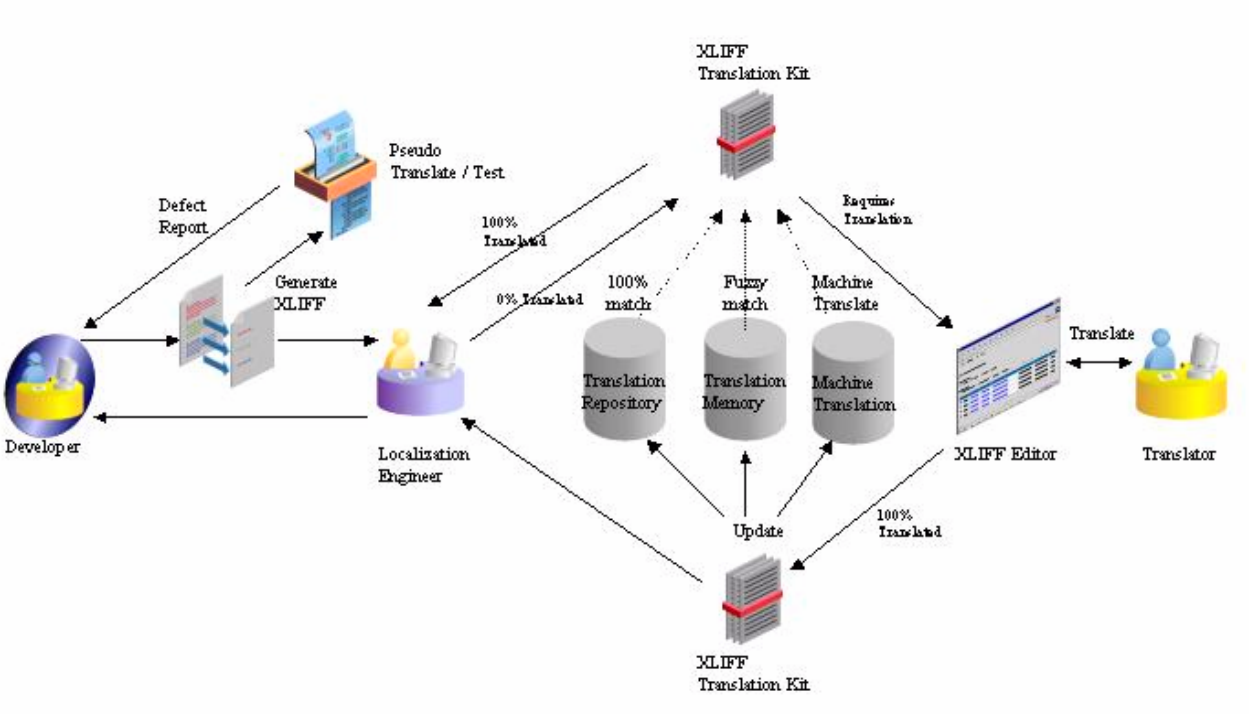


In this model, an XLIFF compliant tool outputs directly to XLIFF and this file is handed off to the localization engineers. Another scenario may be that developers output their work to native files as before, but before the files are handed off for localization a pre-processor converts the data into XLIFF. In each of these use cases, when new formats are introduced into the development process or existing ones are changed, developer/publishers are responsible for handing off the data as XLIFF

This proactive model simplifies the formats that localization tools must support, and removes process complexity in the localization engineering domain. It also places the responsibility for converting the native data to XLIFF with those who are most knowledgeable about the native format.

5.3 Simple Automated Localization

Figure 8 – Automated Workflow with XLIFF

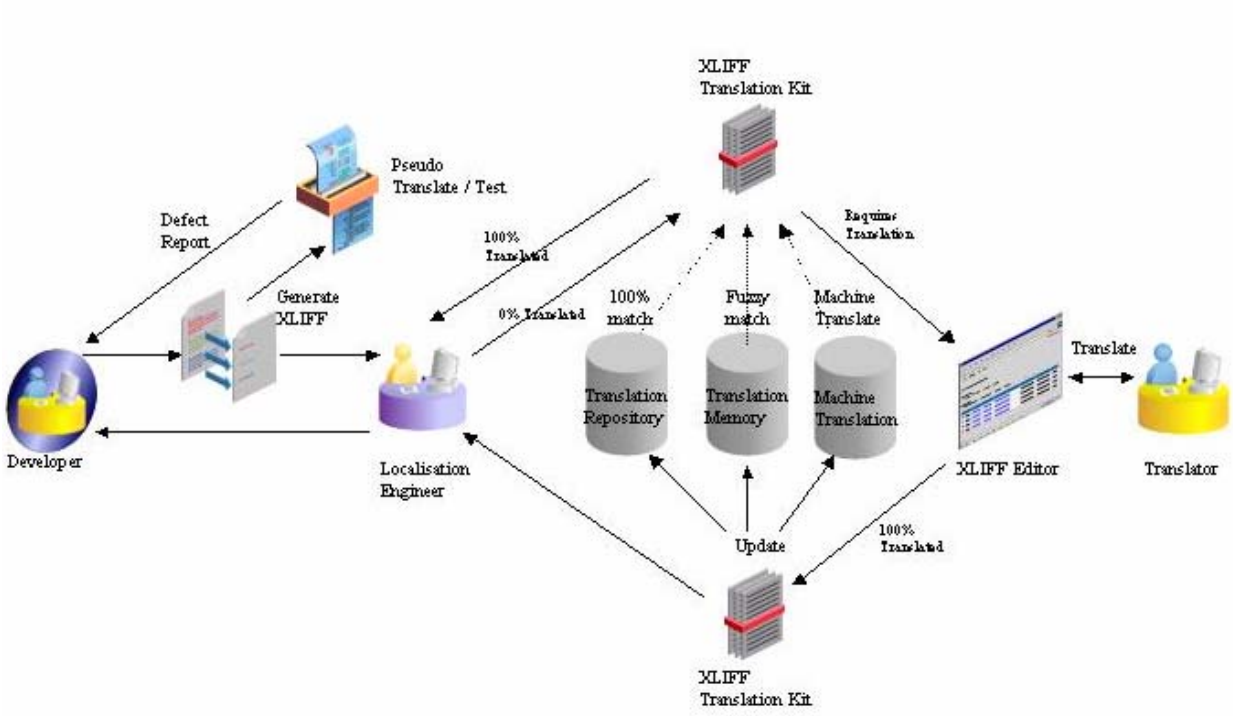


This use case builds upon the previous use case. Here, a developer extracts his localizable resources to an XLIFF file, which may be optionally automatically pseudo translated and tested. Pseudo translation helps to reduce the amount of rework done by localization engineers or translators due to design problems that result in localization or internationalization bugs. After testing, the XLIFF file is handed off to a localization engineer, who submits the work to an automated process that recycles previous translations, and applies them to exactly matching XLIFF content (100% matching).

If all the XLIFF contents are leveraged from the repository, the translated file is returned to the localization engineer, but if un-translated content remains the file is forwarded to the translator via automation. The translator completes their translation, and hands off the XLIFF file back into the automated process, which updates the repository with the new translations, and returns to the Localization Engineer, and back onward to the developer.

5.4 Automated Localization with Computer Aided Translation

Figure 9 – Automated Workflow with XLIFF & CAT Tools:



This use case further extends the workflow to include CAT, or Computer Aided Translation tools. In this scenario, the XLIFF files are moved through the workflow as before, but additionally translation memory fuzzy matches may be added to the XLIFF file as `<alt-trans>`, and additionally machine translations may also be added. XLIFF tools that support `<alt-trans>` may present to the translator these "alternative translations" to enhance their productivity.

Additionally, reference to related glossary data can be stored in the XLIFF file and handed off to the translator.

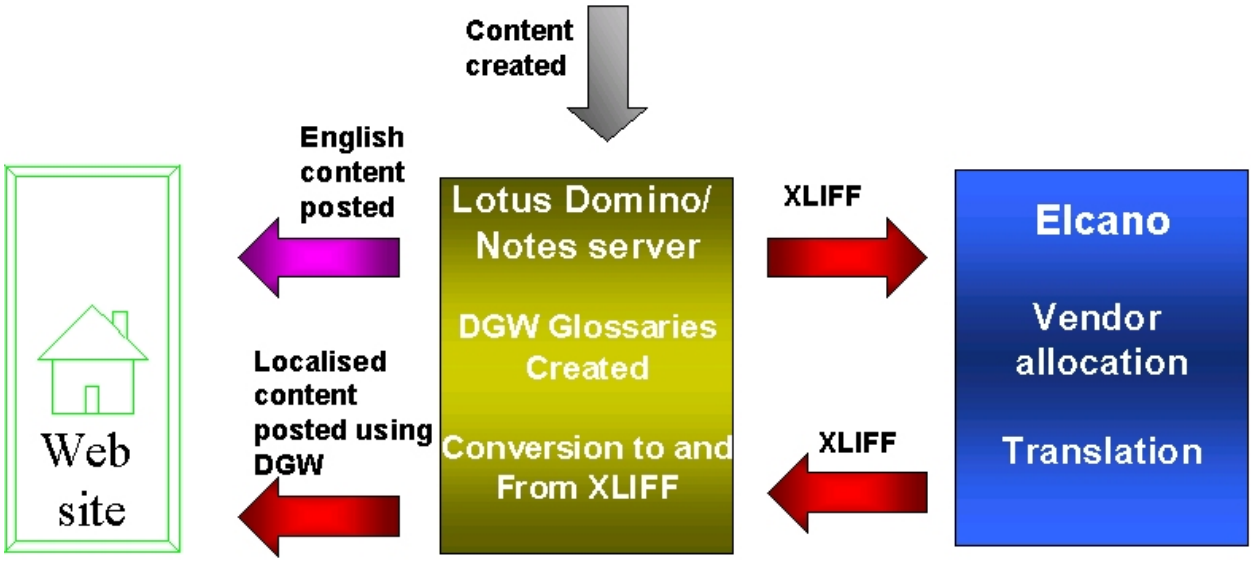
6.0 Case Study

Since version 1.0 was published as a committee specification in early 2002 the standard has been used by a number of companies to aid and improve the process they use for localization. The following case study exhibits how two companies have already collaborated to use version 1.0 of XLIFF but there are a number of companies who are now working to update to XLIFF version 1.2.

XLIFF is now becoming more widely used within the localization industry and there is unlikely to be any conference within this industry that does not mention XLIFF within a case study or presentation of a tool. The emerging standard has been discussed and presented at a Localisation Institute Summit, numerous LISA conferences and workshops, a UNICODE conference and several Localization Research Centre (LRC) Summer School sessions and conferences.

6.1 Lotus Domino Global WorkBench™ & Elcano™

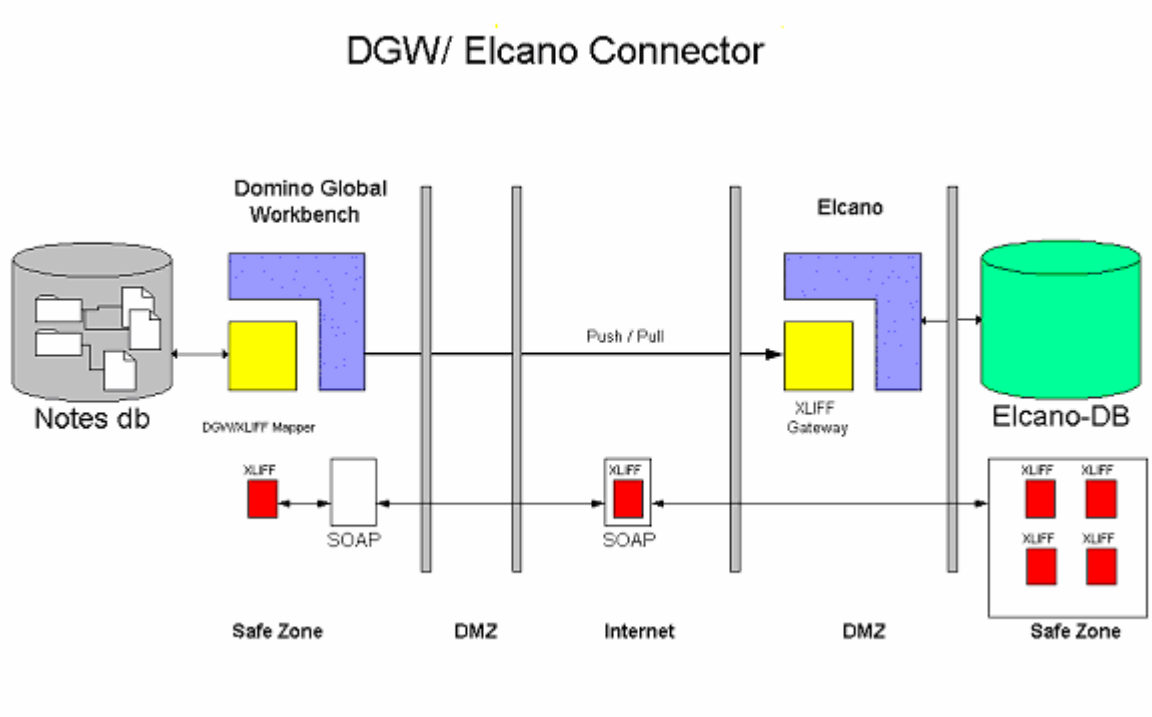
IBM and Bowne Global Solutions have come together to offer a technological solution for translating Lotus Notes/Domino databases which uses Web Service technology. IBM has developed a technology called Lotus Domino Global WorkBench (DGW)™ which has been enhanced to use the Elcano online translation service from Bowne Global Solutions.



Users of Lotus Domino from IBM will be already aware of the functions offered by DGW. This is a tool which is used to manage the translation of Notes/Domino databases and websites. The Elcano

integration links this technology with a pipeline to professional human translators, who offer reliable high-quality translation. When invoked DGW glossary content is converted to XLIFF. The XLIFF file is sent to the Elcano WSDL document using SOAP technology. Here, the sender and other information included in the XLIFF file is also automatically identified along with the translatable content. The XLIFF file is then translated using the Elcano process and returned to DGW using Web Services technology. On receiving the translated XLIFF file from Elcano, DGW updates the glossary with the new translations.

Figure 10 – DGW/ Elcano integration



The XLIFF below provides an example of the file sent from DGW to Elcano.

```
<?xml version="1.0"?>
<xliff version="1.0">
<file original="C12568710034A542" source-language="en-us" target-language="es-MX"
datatype="Notes" tool="Domino Global WorkBench Glossary Application 6.0"
xml:space="default">
    <header>
          <phase-group>
                <phase phase-name="XLIFF Document Creation" process-name="DGW
                Export" date="2003-04-10T16:23:51Z" contact-
                name="mark_levins@ie.ibm.com"/>
                </phase-group>
                <note priority="1">This XLIFF document was generated by the DGW
                glossary application for the glossary:
                glsV5TemplateLocalization.nsf on Local and was created on
                Thursday, April 10, 2003</note>
                <count-group name="Glossary Word Count">
                      <count count-type="word">4</count>
                </count-group>
    </header>
    <body>
          <trans-unit id="0080C5CD86256D03" translate="yes" reformat="yes"
          xml:space="default">
                <source>Composed Date</source>
                <target>Composed Date</target>
                <context-group name="Term Context Information">
                      <context context-type="Database" match-
                      mandatory="no">Template Best Practices</context>
                      <context context-type="Element" match-
                      mandatory="no">Views</context>
                      <context context-type="ElementTitle" match-
                      mandatory="no">All Documents|($All)</context>
                      <context context-type="Record" match-mandatory="no">Column
                      Title</context>
                </context-group>
          </trans-unit>
          <trans-unit id="0080C5A886256D03" translate="yes" reformat="yes"
          xml:space="default">
                <source>Welcome to</source>
                <target>Welcome to</target>
                <context-group name="Term Context Information">
                      <context context-type="Database" match-
                      mandatory="no">Template Best Practices</context>
                      <context context-type="Element" match-
                      mandatory="no">Page</context>
                      <context context-type="ElementTitle" match-
                      mandatory="no">DbaseTitleDisplay |
                      DbaseTitleDisplay</context>
                      <context context-type="Record" match-mandatory="no">Static
                      Label</context>
                </context-group>
          </trans-unit>
    </body>
</file>
</xliff>
```

7.0 Resources

7.1 Tool support for XLIFF

Alchemy Software – Alchemy Catalyst™ 4.05, XLIFF 1.1 Editor, and Alchemy Catalyst™ 5.0, XLIFF 1.1 Visual Editor (http://www.alchemysoftware.ie).

Bowne Global Solutions – Elcano™, Online Translation Service has a Web service based connector for XLIFF files (http://elcano.bowneglobal.com).

Ektron CMS400.NET – Ektron's CMS400.NET web content management solution uses XLIFF to export and import content for translation (http://www.ektron.com).

Heartsome – XLIFF Editor (http://www.heartsome.net/).

IBM – Lotus Domino Global Workbench™ Version 6 (http://www6.software.ibm.com/devcon/devcon/docs/dwkbbet6.htm).

Okapi Template Collection – A set of open source XSLT templates for XLIFF and other localisation formats (http://sourceforge.net/project/showfiles.php?group_id=42949&release_id=67485).

PASS Engineering – Passolo Editor™ (http://www.passolo.com).

RWS Group – Extraction utility for XLIFF 1.1 (http://dotnet.goglobalnow.net/); and various utilities: (http://www.translate.com/shared/tools).

SDL International – SDLX™ support for XLIFF currently in development (http://www.sdlx.com).

Sun – Internal XLIFF Editor as described in this article: http://www.sun.com/developers/gadc/technicalpublications/articles/xliff.html.

Trados™ – No direct XLIFF support. Can edit XLIFF files using modified INI file.

xliffRoundTrip Tool – Any XML file can be filtered to XLIFF, then translated, then filtered back into as a translated file with its identical initial hierarchy (http://sourceforge.net/projects/xliffroundtrip/).

XML-Intl – XLIFF Editor (http://www.xml-intl.com).

8.0 Appendix

8.1 Contact Information

XLIFF Web site: http://www.xliff.org.

The current XLIFF Technical Committee is composed of the following officers:

Bryan Schnabel*, bryan.s.schnabel@tektronix.com, Chair
Tony Jewtushenko*, tony.jewtushenko@productinnovator.com, Chair
Mary McRae, mary.mcrae@oasis-open.org, OASIS Staff Contact

The XLIFF 1.2 Committee Specification is located at http://docs.oasis-open.org/xliff/xliff-core/xliff-core.html.