ADOPTING STANDARDS BASED XML FILE FORMATS IN OPEN SOURCE LOCALISATION

AN HONOURS THESIS SUBMITTED TO
THE SCHOOL OF SOFTWARE ENGINEERING AND DATA COMMUNICATIONS,
QUEENSLAND UNIVERSITY OF TECHNOLOGY,
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF
BACHELOR OF INFORMATION TECHNOLOGY (HONOURS)

ASGEIR FRIMANNSSON
asgeirf@gmail.com
Declaration

I declare that the work presented in this Honours thesis is, to the best of my knowledge and belief, original and my own work, except as acknowledged in the text, and that this material has not been submitted, either in whole or in part, for a degree at this or any other university.

Asgeir Frimannsson 17 June 2005
Acknowledgements

I would first of all like to give special thanks to Dr James Michael Hogan for supervising my honours project. Thank you Jim for introducing me to localisation, XLIFF and other localisation standards, and for all help and advice you have given me.

I would also like to thank Red Hat for supporting me financially this past year through the QUT Red Hat Honours Scholarship in Software Internationalization. More importantly, I would like to thank the team at Red Hat for great support, and for including me as part of their team. Especially, I would like to thank Paul Gampe, Leon Ho, Michael Hideo Smith and Lawrence Lim. Your help, encouragement and support have meant a lot to me.

Next, I would like to thank everyone who has been part of the XLIFF Tools project at freedesktop.org through participation on the mailing list: Josep Condal, Bart Cornelis, Fredrik Corneliusson, Karl Eichwalder, Tim Foster, David Fraser, Bruno Haible, Rodolfo M. Raya and Yves Savourel. Thank you all for willingly sharing ideas, opinions and knowledge. From looking at the difference between the first and the final draft of the XLIFF 1.1 Representation Guide for PO, it is easy to see how your input has made tons of difference!

I would also like to thank people like Dwayne Bailey from translate.org.za, and Javier Solá from the Khmer Software Initiative. As I believe localised open source software is a contributing factor to decreasing the digital divide, your work and passion for localisation have been a great source of inspiration for me in completing this research.

Finally, I would like to thank my parents and my dearest Jessie for your love and support throughout my honours year.
Abstract

In recent years, major localisation vendors, backed by the Localisation Industry Standards Association (LISA) and The Organization for the Advancement of Structured Information Standards (OASIS), have agreed on open XML based standards for storage and exchange of data in the localisation process: The Translation Memory eXchange (TMX) file format for exchanging translation memory data, the TermBase eXchange (TBX) format for terminology exchange, and the XML Localisation Interchange File Format (XLIFF) for extracting and storing localisable data in a common file format. Up until recently, there have been very few open source tools available supporting these formats, and hence very few open source projects have adopted these formats in the localisation process. A majority of open source applications continue to use GNU Gettext and the Portable Object (PO) file format. This research evaluates the case for adopting XLIFF in localisation processes based on the GNU Gettext toolkit, and identifies needs for adopting other standards such as TMX, TBX and Translation Web Services in these processes.

Keywords: XML based localisation, open source software, translation, localisation, internationalisation
Table of Contents

<table>
<thead>
<tr>
<th>ABSTRACT</th>
<th>INTRODUCTION</th>
<th>FOUNDATIONS</th>
<th>RESEARCH APPROACH</th>
<th>ADOPTING XLIFF IN OPEN SOURCE LOCALISATION</th>
<th>LOCALISATION PROCESS IMPROVEMENTS THROUGH OPEN STANDARDS</th>
<th>CONCLUSIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1 INTRODUCTION</td>
<td>2 FOUNDATIONS</td>
<td>3 RESEARCH APPROACH</td>
<td>4 ADOPTING XLIFF IN OPEN SOURCE LOCALISATION</td>
<td>5 LOCALISATION PROCESS IMPROVEMENTS THROUGH OPEN STANDARDS</td>
<td>6 CONCLUSIONS</td>
</tr>
<tr>
<td></td>
<td>1.1 OBJECTIVES</td>
<td>2.1 OPEN SOURCE LOCALISATION</td>
<td>3.1 THE XLIFF TOOLS PROJECT</td>
<td>4.1 XLIFF REPRESENTATION OF THE PO FORMAT</td>
<td>5.1 SHARED TRANSLATION MEMORY REPOSITORIES</td>
<td>6.0</td>
</tr>
<tr>
<td></td>
<td>1.2 LIMITATIONS OF STUDY</td>
<td>2.2 LOCALISATION AND OPEN STANDARDS</td>
<td>3.2 TASKS</td>
<td>4.2 GETTEXT AND XLIFF IN THE DEVELOPMENT PROCESS</td>
<td>5.2 TERMINOLOGY MANAGEMENT</td>
<td>6.1</td>
</tr>
<tr>
<td></td>
<td>1.3 SIGNIFICANCE OF STUDY</td>
<td>2.3 OPEN STANDARDS IN OPEN SOURCE LOCALISATION</td>
<td>3.3 VERIFICATION AND TESTING OF RESULTS</td>
<td>4.3 TOOL SUPPORT</td>
<td>5.3 SERVICE BASED LOCALISATION WORKFLOWS</td>
<td>6.2</td>
</tr>
<tr>
<td></td>
<td>1.4 OUTLINE</td>
<td>2.4 SUMMARY</td>
<td>3.4 ETHICAL ISSUES</td>
<td>4.4</td>
<td>5.4</td>
<td>6.3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>6.4</td>
</tr>
</tbody>
</table>

APPENDIX A: XLIFF REPRESENTATION GUIDE FOR GETTEXT PO
APPENDIX B: OVERVIEW OF XLIFF TOOL
REFERENCES
# List of Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVS</td>
<td>Concurrent Versions Systems (version control system)</td>
</tr>
<tr>
<td>GNU</td>
<td>GNU’s Not Unix (recursive acronym)</td>
</tr>
<tr>
<td>HTML</td>
<td>Hypertext Markup Language</td>
</tr>
<tr>
<td>I18N</td>
<td>Internationalisation</td>
</tr>
<tr>
<td>KDE</td>
<td>The K Desktop Environment</td>
</tr>
<tr>
<td>L10N</td>
<td>Localisation</td>
</tr>
<tr>
<td>LISA</td>
<td>Localisation Industry Standards Association</td>
</tr>
<tr>
<td>MO</td>
<td>Machine Object or Message Object (file format)</td>
</tr>
<tr>
<td>OASIS</td>
<td>The Organization for the Advancement of Structured Information Standards</td>
</tr>
<tr>
<td>PO</td>
<td>Portable Object (file format)</td>
</tr>
<tr>
<td>POT</td>
<td>PO Template</td>
</tr>
<tr>
<td>SOAP</td>
<td>Simple Object Access Protocol</td>
</tr>
<tr>
<td>SRX</td>
<td>Segmentation Rules eXchange (file format)</td>
</tr>
<tr>
<td>SVN</td>
<td>Subversion (version control system)</td>
</tr>
<tr>
<td>TBX</td>
<td>TermBase eXchange (file format)</td>
</tr>
<tr>
<td>TM</td>
<td>Translation Memory</td>
</tr>
<tr>
<td>TMX</td>
<td>Translation Memory eXchange (file format)</td>
</tr>
<tr>
<td>TRANS-WS</td>
<td>Translation Web Services</td>
</tr>
<tr>
<td>XLIFF</td>
<td>XML Localisation Interchange File Format</td>
</tr>
<tr>
<td>XML</td>
<td>eXtensible Markup Language</td>
</tr>
<tr>
<td>XML-RPC</td>
<td>XML Remote Procedure Call</td>
</tr>
</tbody>
</table>
List of Tables

Table 1: Research Tasks ................................................................................................... 23
Table 2: Most commonly used tools in the Gettext package ........................................... 30
# List of Figures

Figure 1: English and Hebrew editions of Konqueror - an open source web browser ..... 2  
Figure 2: Typical Gettext based localisation process ................................................. 7  
Figure 3: GNU Gettext based workflow - merging translations .................................. 7  
Figure 4: DocBook files converted to PO for translation ............................................ 8  
Figure 5: PO file format structure ............................................................................ 9  
Figure 6: PO Plural Forms ..................................................................................... 9  
Figure 7: Germanic Plural Forms header ................................................................. 9  
Figure 8: Polish Plural Forms header .................................................................... 9  
Figure 9: Java .properties use of logical ids .............................................................. 10  
Figure 10: The Rosetta translation portal ................................................................. 11  
Figure 11: Shared web based glossary for the Norwegian translator teams ............. 11  
Figure 12: XLIFF localisation workflow ................................................................. 13  
Figure 13: Common XLIFF 'shell' ............................................................................ 14  
Figure 14: Minimal XLIFF document ..................................................................... 14  
Figure 15: Abstraction of inline codes ................................................................... 15  
Figure 16: Adding Context Information .................................................................. 15  
Figure 17: Workflow tracking through `<phase>` elements ..................................... 16  
Figure 18: Translation Web Services ...................................................................... 18  
Figure 19: The Action Research process .................................................................. 21  
Figure 20: XLIFF representation of PO - General Structure .................................. 25  
Figure 21: PO Header ............................................................................................. 25  
Figure 22: PO header as a `<trans-unit>` element .................................................. 26  
Figure 23: Plural Translation Unit in XLIFF ............................................................ 27  
Figure 24: Handling of PO comments in XLIFF ..................................................... 27  
Figure 25: XLIFF native support for references ...................................................... 28  
Figure 26: Possible XLIFF representation of PO references .................................... 28  
Figure 27: Abstraction of inline codes when representing PO in XLIFF .................. 28  
Figure 28: PO entry with escape sequences ............................................................. 29  
Figure 29: XLIFF translation unit with converted escape sequences ....................... 29  
Figure 30: Current PO based workflow .................................................................. 30  
Figure 31: XLIFF based translation within a PO based workflow ............................. 31  
Figure 32: Fully XLIFF based workflow .................................................................. 32  
Figure 33: XLIFF Gettext integration ..................................................................... 33  
Figure 34: XLIFF workflow with TMX based Translation Memory ....................... 36  
Figure 35: Translation Unit without identified terms ............................................... 37  
Figure 36: Translation Unit with terms identified .................................................... 37  
Figure 37: Marking terms in XLIFF documents ....................................................... 37  
Figure 38: RC-WINTRANS use of XLIFF project files .......................................... 38  
Figure 39: Localisation Infrastructure based on standards based XML formats ....... 39  
Figure 40: Composing Localisation Projects .......................................................... 40
1 Introduction

For a software product to succeed in today’s global marketplace, it cannot simply be exported in its original form, but needs to be adapted to local markets. The process of customising a product for a specific culture is called localisation (L10N\textsuperscript{1}), and involves cultural, content and technical issues as well as the linguistic process of translation. To ease this process, applications are internationalized: Internationalisation (I18N\textsuperscript{1}) is the process of changing or designing an application to be language-generic and culture neutral, minimizing the effort needed in the localisation process. This is done by externalizing culture sensitive resources such as software messages and graphics, and also covers handling of input and formatting of data such as dates and numbers.

![Figure 1: English and Hebrew editions of Konqueror - an open source web browser.](image)

In recent years, major localisation vendors, backed by the Localisation Industry Standards Association (LISA) and The Organization for the Advancement of Structured Information Standards (OASIS), have agreed on open XML based standards for use in the localisation process, including:

- The Translation Memory eXchange (TMX) file format for exchanging translation memory data;
- The TermBase eXchange (TBX) file format for exchanging terminology data; and
- The XML Localisation Interchange File Format (XLIFF) for storing locale-sensitive resources in a common file format.

Localisation of software messages in open source is, in the vast majority of projects, handled by GNU Gettext. This toolkit contains tools for extracting and merging messages from source code for localisation, as well as libraries for loading the translated messages from resource files at runtime. Gettext uses its own file format, the Portable Object (PO) format, for storing resources in the localisation process. With projects such as GNOME

\textsuperscript{1} The terms Internationalisation, Localisation and Globalisation are often abbreviated using the first letter of the word, followed by the number of letters in between the first and the last letter of the word, followed by the last letter of the word: I18N, L10N and G11N.
and KDE, today having translation teams for over 80 different languages, it is evident that Linux and open source is reaching out to a global market, not limited to English speaking cultures and communities. As industry and governments around the world continue the increasing adoption and embrace of open source software, localisation has, as with commercial software, become a critical success factor.

This research proposes a bridge between current localisation practices in open source, and localisation industry standards. As open source localisation is based around having PO as a common resource format, this thesis will have a strong focus on the XLIFF standard, evaluating the file format as a possible replacement for PO as the common resource format in the localisation process. Building on this foundation, we will look at how other localisation standards such as TMX and TBX can be incorporated in open source localisation processes, and discuss the need for a service-based architecture, evaluating emerging standards such as Translation Web Services in open source localisation.

### 1.1 Objectives

This research aims to investigate how Gettext based localisation processes can be supported in XLIFF. This will be achieved through studying current localisation practice in open source projects, as well as investigating standards involved in the localisation processes. The XLIFF standard was designed to address needs in commercial localisation processes and has mainly targeted localisation of Microsoft Windows based resource formats. This research aims to find out if there are aspects of GNU Gettext and other PO based localisation processes that XLIFF lacks support for. The heart of this study will be to define, in collaboration with contributors from the open source communities, a mapping between the Gettext PO format and XLIFF. The results will be submitted to the OASIS XLIFF Technical Committee, contributing to the committee’s work in defining guides for representing common file formats in XLIFF. To verify the representation guide, software filters for converting between PO and XLIFF will be developed, along with tools to support XLIFF in current open source localisation practices.

Based on the foundation of having XLIFF as a common resource format in the localisation process, the objective of this thesis is further to examine how other localisation standards can be used to improve processes in open source localisation. This evaluation include handling of terminology, translation memories and localisation workflows, areas that have that have up until now only been addressed on an ad-hoc basis in open source localisation.

### 1.2 Limitations of Study

In evaluating open source localisation, we mainly focus on processes based around GNU Gettext and the PO file format. We are fully aware that many large open source projects, for example OpenOffice.org and Mozilla, do not use PO, but rely on other file formats in the localisation process. In evaluating XLIFF in open source, due to the limited time frame for this research, we focus solely on PO as used for localising software messages – but our aim is to argue the case for adopting XLIFF in all open source localisation processes, not only processes based on GNU Gettext and the PO format.

In evaluating localisation standards in open source, we thoroughly cover the adoption of XLIFF, but only discuss high-level solutions for adopting other standards such as TMX, TBX and Translation Web Services. We limited ourselves to XLIFF
because that is the only format in which there is an “equivalent”, the PO format, in present open source localization processes. Use of Translation Memory technologies, structured Terminology Databases and service based architectures, are very limited in open source localization, as open source tools for these technologies do not exist. When we present models incorporating these aspects, we therefore build on industry best practices, and not current practices in open source localization.

1.3 Significance of Study

This research is simply a first step into looking at how open source can benefit from standards based localization formats, and hence, much of the focus of the thesis is to identify areas needing further research. This thesis builds a solid foundation for further research, as it provides a solution for representing the longstanding PO format in XLIFF, and argues the case for using XLIFF as a common resource format in all open source localization processes.

As open source software continues to gain acceptance from industry and governments, it becomes increasingly important that localization processes also become aligned with industry standards. By supporting XLIFF, GNU Gettext based localization processes will become aligned with industry standards in the field. This can lead to increased commercial software development for the Linux platform, allowing vendors to apply their existing localization processes when developing for Linux. Localizable data can then be exchanged and any XLIFF-enabled tool can be used for the localization process.

This research further ensures standards alignment in regards to the representation of the PO file format in XLIFF. As different localization tool vendors are starting to create custom XLIFF filters for the GNU Gettext PO format, it becomes increasingly important to promote a reference representation for this file format in XLIFF, ensuring interoperability between XLIFF-enabled localization tools.

The results of this thesis provide a significant contribution to the open source community, by improvements of the localization processes: Replacing PO with XLIFF as the common resource format enables more meta-data to be stored in the localization process. XLIFF includes support for translator comments and suggestions of alternative translations, facilities desperately needed in collaborative localization processes. Further, the XLIFF standard includes support for workflows such as translation phases, which can be utilized in projects such as Fedora to define localization phases such as translation, proofreading and approval.

Adopting standards based formats in open source localization also provides unique opportunities for verifying the validity of these standards in truly open environments. Up until now, most implementations taking advantage of these standards are proprietary solutions, only tested by a single vendor. Successful adoption of these standards in open source will in turn provide a playing field in which these standards can evolve and mature independently of commercial interests.

1.4 Outline

This thesis is divided into 6 sections. In the following section, background drawn from literature is presented, thoroughly surveying the foundations of localization standards and open source localization. In Section 3 the research approach is discussed, before the results are presented in the following two sections: Section 4 discusses the adoption of XLIFF in open source localization, and in combination with Appendix A
(XLIFF 1.1 Representation Guide for Gettext PO) presents a complete mapping between PO and XLIFF. In this section incorporation of XLIFF in existing build systems and development processes is also discussed. Section 5 continues building on the foundation of having XLIFF as a common resource format – discussing the need for adopting other standards based file formats in open source localisation. Finally, in Section 6, we conclude and summarize the findings of this thesis, and discuss the road ahead for localisation standards and open source localisation.
2 Foundations

In this section we survey the background and analyse research done on localisation based on standard file formats, and analyse current open source localisation processes and workflows. Specifically, by analysing previous research and literature, this section will:

1. Highlight the strengths and weaknesses of open source localisation, specifically focusing on processes based around GNU Gettext and the PO format.
2. Discuss the background of open standards based XML formats, specifically XLIFF.
3. Discuss the need for standards based XML formats such as XLIFF and TMX in open source localisation.

By highlighting these areas, this literature review will provide a platform of knowledge on which to build a strong case for adapting XLIFF and other standards based localisation formats in localisation processes traditionally associated with GNU Gettext and the PO format.

2.1 Open Source Localisation

2.1.1 Software Process

The open source development process has been described as a user-driven, just-in-time approach, driven by a global developer community (Berglund and Priestly, 2001). Development is driven by demand for the product within the developer community, and new features are implemented as a result of requests from the user base. Raymond (2000) describes the process as a ‘bazaar’ model where software is released early and often, the software process is open and transparent and work is delegated as much as possible. In contrast, traditional software is developed by closed teams having long release cycles and is only released after being thoroughly tested.

Less than one in five open source participants are paid for their involvement, and participants are motivated by various internal and external factors (Hars and Ou, 2001). Such characteristics are typical of those localizing open source applications.

The characteristics described above imply that localisation of open source software is an ongoing process, driven primarily by voluntary efforts. Localisation tools have to be able to deal with constantly changing sources, and a translation can only be up to date for a limited period of time, depending on the activity of the project.

2.1.2 Gettext and the Localisation Process

Localisation of software messages is in most open source projects handled by GNU Gettext - as specified in the OpenI18N Globalisation Specification (Free Standards Group, 2003). The GNU Gettext library is based around two file formats:

- The Portable Object (PO) file format: a simple string table for storing translation units in the localisation process.
- The Machine Object (MO) file format: a binary representation of a string table – used by an application to retrieve translated strings at runtime.
By GNU Standards, software, including software messages, is to be written in American English (Free Software Foundation, 2004). The original software messages are not externalised to resource files, but stored in source code, providing a way for application to run in the original language (English) without needing any resource files. GNU Gettext works extracting strings marked for localisation from source code, as depicted in the following diagram:

**Figure 2: Typical Gettext based localisation process**

Gettext tools extract localisable strings from source code into PO string tables (1). Normally, each application only uses a single String table, easing the work for translators in the localisation process. Translators then translate (2) these files using PO based localisation tools. After translation, the PO files are converted to binary Machine Object (MO) resource files (3). Applications retrieve (4) translated strings from the language-specific binary MO files at runtime. When no translation is available for a given string, the original (English) string is used.

Open source development processes are typically characterized by short release cycles and rapid change. GNU Gettext is tailor-made for such processes, and provides mechanisms for updating and merging string tables:

**Figure 3: GNU Gettext based workflow - merging translations**

As also described above, Gettext tools first extract localisable strings from source code into a PO string table (1). Next, the newly extracted PO string table is then merged with an existing (translated) PO string table. In addition, new entries are matched against entries of a PO Compendium – a string table acting as a translation memory, holding translations combined from multiple PO Files. Translators then translate and review changes in the updated PO string table (3) before the PO file is converted to MO for use at runtime.

Projects such as Fedora, GNOME and KDE use the DocBook XML format for software documentation (Diehl et al., 2002, Gwynne and Harries, 2004, Wang, 2004). There is no standard for the translation of DocBook files; different projects use different
approaches. In KDE and Fedora, translatable segments are extracted to PO files by custom filters, as shown in the following diagram:

**Figure 4: DocBook files converted to PO for translation**

DocBook files are converted to PO for translation. After translation, files are converted back into language specific DocBook documents. By using this approach, translators can use the same tools for translating software messages as they do for translating documentation. Custom filters are also used for other file formats, including Glade User Interface Dialogs and Desktop Entry files, where translatable segments are stored in the applications PO file in the localisation process.

### 2.1.3 The PO file format

The PO file format is at the heart of Gettext, and has three different variants: PO Template files (POTs), Regular bi-lingual PO files, and PO Compendium files. Entries are extracted from source code, using the Gettext ‘xgettext’ tool, and stored in POT files. POT files are initialized (using the ‘msginit’ tool) or merged with existing translated PO files (using the ‘msgmerge’ tool) and stored in a language-specific PO file, which are then in turn used in the translation process. In addition, translation entries from across projects are stored in PO Compendiums, providing simple translation memory abilities.

Each PO file generally maps to a Gettext domain, where a domain is a collection of unique translation units stored in a single MO file. Most commonly, and highly recommended in projects such as GNOME and KDE, each application only use a single Gettext domain, and the name of that domain is generally the same as the name of the application.

The following example shows the basic structure of GNU Gettext PO files:
A PO file contains as a set of translation units, each having a source (msgid) and target (msgstr) field, where the first translation unit generally contains header meta-data. Each translation unit can, in addition to an original and translated string, include translator comments, extracted comments from source code, source code references (source filename and line-number) and Gettext specific flags.

### Plural Forms

PO translation units support parameter-determined plural form translation units. These translation units contain the singular English form in the msgid field, and the plural form in the msgid_plural. As the target, these translation units have an array of msgstr, representing the number of forms in the target language:

```plaintext
msgid "You have %d file"
msgid_plural "You have %d files"
msgstr [0] "Du har %d fil"
msgstr [1] "Du har %d filer"
```

The target language may have one or more forms (Japanese has one form, while Polish has 3 forms), and the logic for selecting which form to use for a parameter is defined in a PO header field, where nplurals defines the number of forms and plural defines a C-expression for evaluating which item in the msgstr array to use at run time:

```plaintext
"Plural-Forms: nplurals=2; plural=(n != 1);
"Plural-Forms: nplurals=3; plural=(n == 1 ? 0 : n%10>=2 && n%10<=4 && (n%100<10 || n%100>=20) ? 1 : 2;"
```

This is a typical example for a Germanic language, which has a special case when \textit{n} is 1. A more complex example is Polish, which has special cases for when \textit{n} is 1, and in addition some numbers ending in 2, 3 or 4:

```plaintext
"Plural-Forms: nplurals=3; plural=n==1 ? 0 : n%10>=2 && n%10<=4 && (n%100<10 || n%100>=20) ? 1 : 2;"
```
C-expressions are defined as “condition ? true_value : false_value”, where condition is an expression evaluating to true/false. In the above example, the first condition is “n%10<10 || n%10>=20” which if true gives the result 0, and if false gives the result of a second c-expression. For the second expression, the condition is “n%10==2 && n%10<=4 && (n%100<10 || n%100>=20)”, which if true gives the result 1, and if false gives the result 2. At run time, Gettext will use the msgstr associated with the index returned from this expression.

**Source string as primary key.** Another unique “feature” with Gettext and the PO format, is that it uses the source string (msgid) is primary identifier. This is different from other common resource formats such as Java Resource Bundles (Figure 9) and .NET Resources, which use some sort of logical id to map to the actual source string.

![Button Save As](button_saveAs.png) ![Message Username Invalid](msg_usernameInvalid.png)

**Figure 9: Java .properties use of logical ids**

Although use of the source string as id makes it easier for developers, as they do not have to handle external resource files, there are some disadvantages with this approach: First of all, there cannot be multiple translations of the same string within the same Gettext domain. This has led to ‘hacks’, where developers prefix the string with some context information, and this context information is then removed at runtime. In addition, if the original (English) string changes slightly because of spelling mistakes, tools has to do some sort of fuzzy matching to update the translated PO files.

Using source strings as ids is a challenge when updating the resource files when the source code has changed. If a string message has changed in the source code, there is no way for the tools to know which translation units are changed, and the old translation units are simply discarded and new ones are created. This makes change-tracking and version control on a translation-unit challenging, having to rely on fuzzy matching algorithms to try and maintain the translation history for a unit.

If source strings are used as ids, there is also no way of having different translations of similar strings in different contexts within the same catalog. Some projects (For example KDE and the GNOME Glib library) combine the context and the source string to form unique ids, and have custom functions to extract the context information from the source string at runtime.

### 2.1.4 Translation Tools

Open source localisation tools are mainly based around the PO file format. The most advanced of these is KBabel from the KDE project (Diehl et al., 2002). KBabel supports advanced features such as handling of plural forms in PO files, internal translation memory, dictionaries and catalog management (Costin and Kiefer, 2004). Other popular PO editors include GTranslator, Emacs PO Mode and poEdit.

Each translation project (For example KDE, GNOME and Fedora) has its own set of web based status pages, displaying statistics and information relating to the translation process. Recently, there has been a shift towards web based translation portals, where contributors can, in addition to viewing statistics, also translate using regular web browsers. The more sophisticated of these are Pootle and Rosetta (Figure 10). While Pootle is an independent project, covering translations from projects like OpenOffice, Mozilla and KDE, Rosetta is the translation portal for the Ubuntu Linux distribution.
2.1.5 Translation Reuse

The GNU Gettext tools have some support for translation memory through PO compendium files (Free Software Foundation, 2002). These files are of similar format to PO files and contain previously translated entries. PO compendiums are used to ensure consistency across applications and projects by leveraging previously translated segments. KBabel has an internal translation memory as well as support for PO compendiums and support for TMX. The TMX support is however limited and only usually only contain the original English term and the translated term. These glossaries are shared through collaborative wikis and web pages.

Present Gettext based localisation processes suffer from very limited translation reuse. This is mainly due to limitations of the PO format, providing little support for segmentation and abstraction of inline codes.

2.1.6 Terminology Management

There are no agreed on standards for handling terminology in open source localisation, and different language teams and projects use different approaches. Most commonly, teams share a glossary list, which sometimes contain definitions of terms, but usually only contain the original English term and the translated term. These glossaries are shared through collaborative wikis and web pages.
2.1.7 Version Control

As previously mentioned, most active open source projects have short release cycles. Localizable data is stored together with source code in version control systems such as CVS (Concurrent Versions System) and SVN (Subversion). Batch jobs take care of updating PO files from the original sources as updated source code is stored in the versioning system (KDE.org Team, 2004). Translators can view the status of translations though user friendly status pages on the web which are automatically generated from the PO files stored in the repository (Diehl et al.).

2.1.8 Limitations

Localisation of open source software is more or less limited to translation of software messages. Other aspects of the user interface such as icons, images, button sizes and layout are currently hard to customize for each language (Van Schouwen, 2004). This is largely due to the development model adopted by most open source projects, where the application is distributed in US English, and additional language packs can be downloaded containing the Gettext message catalog in the desired language.

Though widely used and accepted in the open source communities – and successfully deployed in activities well outside its original brief - the simple string table PO file format is reaching its limits, being unable to support the rich structures inherent in modern localisation. More specifically:

- PO lacks support for advanced meta-data, pre-translation and workflows;
- PO based localisation is limited to translation of textual content such as documentation and software messages, and the format does not support icons or binary images; and
- While PO has been successfully used for localisation of DocBook documents, the format is not a natural fit for the localisation of paragraph based text. In particular, the format has no support for segment based translation reuse, meta-data support is very limited, and inline tags cannot be extracted.

In addition, localisation of open source software and documentation is highly technical, and translators presently need advanced technical skills to be able to contribute to the process.

2.2 Localisation and Open Standards

2.2.1 A standard resource exchange format

The XLIFF Technical Committee (2003a) describes the purpose of XLIFF as “to define, through XML vocabularies, an extensible specification for the interchange of localisation information.” As shown in the following diagram, XLIFF enables tool vendors to focus on a single file format in contrast to traditional localisation tools which have to support a multitude of native formats:
Figure 12: XLIFF localisation workflow.

Vendors can create XLIFF filters for their proprietary file formats, and localisation engineers are not bound to use tools that support the proprietary format, but can use any tool that supports the XLIFF standard.

XLIFF supports custom defined workflow metadata during the localisation process. During the different stages of localisation, the XLIFF document might include data from Translation Memory, Machine Translation and Term Bases. Segments can be marked as ‘signed-off’, ‘needing review’ or similar, and documents can go through several localisation phases (e.g. rough-translation, review).

Similar to TMX, XLIFF allows for abstraction of inline codes such as text markers, inline images and references (Savourel, 2003). This means that meta-data and markup such as XML tags can be included in translations, allowing tools to display these as non-translatable markup elements.

XLIFF is a young standard, and since its introduction in 2002, some vendors have added support to parts of the standard in their tools, but many tools still provide no special XLIFF support – treating it as just another XML format (The XLIFF Technical Committee, 2003a). Many high-end localisation tools such as Trados and Alchemy Catalyst already have very good support for proprietary file formats such as Windows resources and Microsoft Word documents, and the need for XLIFF isn’t there as long as customers are sticking to these file formats.

Viswanadha and Scherer (2004) describe two distinct approaches to handling XLIFF in the development and localisation process:

1. The transient approach: Convert source formats to XLIFF in the localisation process, and only keep the source files in source control systems. On each iteration of the localisation process, an XLIFF roundtrip is needed, and XLIFF is simply a transient format used only in the localisation process.

2. The persistent approach: Keep XLIFF files in source control systems, and convert back to the original format at build time. Updated source files are merged with existing XLIFF documents, and meta-data in the XLIFF files are preserved. This approach requires intelligent filters to handle updates and merging of XLIFF files.

Some further work is being undertaken to improve the XLIFF standard. The XLIFF Segmentation Sub Committee is working on a standard for representing segmentation metadata in XLIFF to improve translation memory effectiveness (Jewtushenko, 2004). The existing XLIFF standard doesn’t specify how text is segmented into translation units and as different vendors and file formats use different algorithms for segmentation, it becomes increasingly difficult to leverage useful data from translation memory systems. In addition, The XLIFF Technical Committee is working on guides for canonical representations of common file formats (such as Java Properties and .NET resource bundles), promoting greater interoperability between tools.
2.2.2 XLIFF Document Structure

XLIFF is structured as a group of <file> elements, each representing an extracted document. Each <file> element contains a <header> element for storing meta-data relating to the original document, as well as localisation meta-data. Following the <header> element is a <body> element containing the localisable material, which may include textual or binary translation units. Textual data is represented as <trans-unit> elements, and binary data as <bin-unit> elements. Translation units can be grouped together using <group> elements, supporting hierarchical data structures.

```xml
<xliff version="1.1">
  <file original="myfile.ext" datatype="plaintext" source-language="en-US" target-language="nb-NO">
    <body>
      <trans-unit id="#1">
        <source>Have a nice day</source>
        <target>Ha en hyggelig dag</target>
      </trans-unit>
    </body>
    <file>
      <alt-trans>
        <target>Have a great day</target>
      </alt-trans>
    </file>
  </file>
</xliff>
```

**Figure 14: Minimal XLIFF document**

Translation Units. Localisable material is segmented into translation units, represented as <trans-unit> elements (for textual content) and <bin-unit> elements (for binary content). As GNU Gettext and other open source localisation solutions at present only deals with textual data, we will not explain handling of binary content in any further detail.

Each <trans-unit> element contains a <source> and a <target> element, representing the original and the translated resource. As a translator usually only handles the translation to one target language, an XLIFF document is strictly bi-lingual, allowing only a single <target> element for a translation unit. Other language translation suggestions (usually from translation memory or machine translation), stored in <alt-trans> elements (Figure 17), can however be included to provide guidance to translators.
XLIFF allows for abstraction of markup and other data in `<source>` and `<target>` elements. The following example shows the abstraction of the HTML `<b>` element within a translation unit:

```xml
<trans-unit id="#1">
  <source>This is a <b>&lt;b&gt;bold &lt;/b&gt;</b></source>
</trans-unit>
```

**Figure 15: Abstraction of inline codes**

**Context Information.** Additional context information for translation units can be stored in `<context>` elements, grouped in `<context-group>` elements. In hierarchical XLIFF files, it is also possible to define context groups for a set of translation units:

```xml
<group id="#1">
  <trans-unit id="#1">
    <source>How are you?</source>
    <context-group name='location data' purpose='location'>
      <context type='linenumber'>11</context>
    </context-group>
  </trans-unit>
  <trans-unit id="#1">
    <source>Have a nice day</source>
    <context-group name='location data' purpose='location'>
      <context type='linenumber'>12</context>
    </context-group>
  </trans-unit>
</group>
```

**Figure 16: Adding Context Information**

Context groups can be used for a range of purposes, including translation memory lookups (`'purpose' attribute set to 'match'`), translator information (`'purpose' attribute set to 'information'`) and location information (`'purpose' attribute set to 'location'`).

**Workflow Information.** XLIFF maintains processing and localisation workflow information in `<phase>` elements. The different phases are defined in the XLIFF header, as `<phase>` elements.
<xliff>
  <header>
    <phase-group>
      <phase name='review' process-name='Translation' contact-name='Joe Bloggs' contact-email='joebloggs@example.com'/>
      <phase name='pre-trans#1' process-name='TM Matching'/>
    </phase-group>
  </header>
  <body>
    <trans-unit id="#1">
      <source>Have a nice day</source>
      <target phase-name='review'>Ha en hyggelig dag</target>
      <alt-trans match-quality='70%' tool='company-TM'>Have a nice time</alt-trans>
      <target xml:lang='nb-NO' phase-name='pre-trans#1'>Ha det gøy</target>
      <alt-trans tool='company-MT'>Have a nice day</alt-trans>
      <target xml:lang='da' phase-name='pre-trans#1'>ha en fin dag</target>
      <alt-trans note from="John">blah</alt-trans>
    </trans-unit>
  </body>
</xliff>

Figure 17: Workflow tracking through <phase> elements

Meta-data. XLIFF provides a large set of standard attributes to support meta-data such as restrictions on string length and translator comments, as well as specifying data-type and resource-type for a translation unit.

Extensibility. XLIFF can be extended using custom namespaces, and the standard provides a number of extension points. In addition, standard attributes can be extended by prefixing attribute-values with ‘x-‘. While XLIFF allows for extensibility, it is recommended that standard specification-provided elements and attribute-values be used, enabling greater tool interoperability.

2.2.3 Translation Memory and Open Standards

Translation memory (TM) allows for reuse of previously translated segments. Translations are stored in a database, and as new documents are translated, similar or identical entries are suggested to translators based on these previous translations. Effective leveraging of previous translations relies upon a high degree of repetition in the projects undertaken; otherwise, only a limited degree of reuse is possible (Iverson, 2003). Successfully deployed TMs significantly reduce turnaround time on translation projects, reducing cost and time-to-market.

Translation Memory eXchange (TMX) is a mature XML standard for lossless exchange of TM data regardless of TM vendor and tool. Two levels of certification exist:

- Level 1 where no inline codes or markup is stored.
- Level 2 which incorporates inline codes.
A large number of vendors are supporting the standards, and an increasing number of tools have been certified by LISA (The Localisation Industry Standards Association) (Zetzsche, 2003).

Effective leveraging of translations relies on a common segmentation standard. This has recently been accomplished through the Segmentation Rule eXchange standards (Localisation Industry Standards Association, 2004), which tightly coupled with the TMX standard provide a complete standards based way of exchanging translation memory data.

Future work to improve the TMX standard includes support for XML schema (Localisation Industry Standards Association, 2004). XML schemas allows for extensible non-TMX markup inside TMX documents by using XML namespaces.

The Internet has opened up possibilities for global sharing of TMs. (Levitt, 2003) predicts Internet shared TMs will become a standard, as this may allow higher overall consistency and shorter turnaround time compared to traditional TM systems in which each translator has their own local TM.

TMX and XLIFF make decoupling of Translation tools and TMs possible (Foster, 2004b). XLIFF documents can be fed with TM entries before they are sent to translators by inserting alt-trans elements. Translators will then not need access to a TM since all relevant information is stored in the XLIFF document. When translation is complete, the translated entries can be included in the TM by converting the XLIFF document to TMX using a standard XML-based transformation script.

2.2.4 Terminology Management

Where translation memory is all about reusing translations, terminology management is about defining and managing concepts and definitions occurring in translatable material. Warburton (2005a) identifies the need for terminology management as early as in the content creation process: If terms are identified, defined and translated early, cost can be reduced and translators will have access to a consistent bi-lingual dictionary throughout the localisation process. The TermBase eXchange (TBX) standard provides a common file format for the exchange of terminology data. The standard prevent vendor lock-in, as the format, as TMX does for translation memory data, provide a rich file format for importing and exporting terminology data from terminology management systems.

To allow better linkage of terms in XML documents to TBX entries, LISA is currently drafting the TBX Link specification\(^2\). As this specification matures, it is reasonable to believe that XLIFF will incorporate support for it, providing direct support for linking terms in XLIFF translation units with corresponding entries in a TBX document.

2.2.5 Translation Web Services

The OASIS Translation Web Services technical committee are working on a specification for automating the communication between parties in the localisation process (The Translation Web Services Technical Committee, 2005). This specification includes methods for requesting localisation quotes, retrieving and submitting localisation

---

\(^2\) The first draft of the TBX-Link specification was announced on the XLIFF TC mailing list 5 May 2005: http://lists.oasis-open.org/archives/xliff/200505/msg00008.html
jobs, and querying the status of localisation jobs. In addition, the specification covers
querying language service providers for services and languages supported.

![Diagram of Translation Web Services](image)

Figure 18: Translation Web Services

( Palas and Karásek, 2003 )

### 2.3 Open Standards in open source localisation

XLIFF opens up a wide range of possibilities for open source localisation projects,
both in terms of workflows and localisation possibilities. Currently, the open source
community is using the PO format as a common file format for localisation. In addition to
its original use as the Gettext file format, the PO format is actively used in translating
XML based DocBook files (Diehl et al., 2002, Wang, 2004). Since the PO format does
not support the rich set of meta-data that DocBook provides, this solution is far from
optimal, and is only used because of good tool support for the PO file format. The main
advantages of incorporating XLIFF into open source localisation processes are outlined
below.

#### 2.3.1 Advantages of XLIFF and TMX

##### 2.3.1.1 Metadata

XLIFF allows for various types of metadata to be stored in documents. While PO
files allow a limited amount of metadata (e.g. general comments and references for each
individual segment, various header-specific metadata), XLIFF has an extensive range of
possible elements to specify for each translation segment as well as for the XLIFF file as
a whole (The XLIFF Technical Committee, 2003b). By using `maxwidth` and `minwidth`,
size constraints can be specified for strings as well as other measurable elements such as
images. Each segment and phase can have a translator associated with it, thus providing
direct support for workflows and integrated version management. If PO files are used,
this information is only available in CVS logs.

##### 2.3.1.2 Workflows

The PO format doesn’t allow for workflows beyond marking entries as fuzzy for
later revision. XLIFF specifies elements and attributes for workflow information, and
localisation can pass through multiple, defined phases (e.g. pre-translation, rough
translation, review), in which changes are documented in the XLIFF sources (The XLIFF
Technical Committee, 2003a). This can be utilized in e.g. release planning and quality
assurance processes in open source projects. Workflows can also integrate translation
memory as explained below.
2.3.1.3 Translation Memory Improvements

XLIFF and TMX allow for the possibility of shared translation memories in open source projects. At present this is hard to accomplish because of the limited information stored in the PO format. With XLIFF, multiple TM matches can be stored in the document, eliminating the need for client side TM technologies (Raya, 2004b). Sharing of TM is important to open source projects since contributors are spread geographically. Upon ‘checking out’ a file for translation, TM suggestions can be automatically inserted. When a translator has completed his/her work and returns the file to the repository, a TMX document containing the approved translations can be automatically processed and new pairs imported into the TM.

2.3.1.4 Decoupling of localisation technologies

Open source localisation is currently based on the Gettext utilities, and most localisation tools in this domain are focused solely around the PO file format (Foster, 2004a). By building open source localisation tools that support XLIFF, software projects can use technologies other than Gettext without this affecting the existing localisation process.

2.3.1.5 XML based processing

One advantage of XML over other file formats is the range of open and free tools and technologies available to process this format (Savourel, 2001). Many facilities exist to define parsing and transformation tools for specific formats. These can be used in the open source environment in a wide range of contexts; for example presenting summaries and data for the translation status pages of the various projects. By using simple XSLT and other XML transformation languages, intuitive summaries and reports can readily be generated from TMX and XLIFF files. These technologies can also be used by the translation tool to present user friendly reports at various stages of the localisation process; e.g. printable HTML documentation can be straightforwardly rendered from XLIFF sources for comparison and proofreading.

2.3.1.6 Localisation of non-textual elements

XLIFF is not limited to localisation of textual content, but can also handle binary data. This opens up a lot of possibilities for future enhancements of the localisation process. GNOME and KDE User Interface dialogs are currently stored in XML formats and can be encapsulated in XLIFF documents and localized using visual XLIFF tools, similar to the processing of Windows Resource files in tools such as Alchemy Catalyst (The XLIFF Technical Committee, 2003a). This however, requires architectural changes to the way localisation is handled at runtime. Currently Gettext handles all localisation of strings, but localizing other elements requires some redesign of how localisation is handled by the system.

2.3.2 Previous Research

Previous research on incorporating XLIFF into open source localisation workflows has been limited and the results remain restricted: The Heartsome XLIFF Translation Editor has XLIFF filters for the PO and DocBook file formats though these are proprietary implementations (Raya, 2004a). SUN Microsystems has XLIFF filters for these formats as well (Foster, 2004b), but these are also proprietary and not released to the wider community. SUN Microsystems is, however, in the process of open sourcing their localisation tools, but have been long delays in this process, and no release date has
yet been set. There are no signs of collaboration on the specification of the XLIFF filters, and no publicly available research has been done to standardize these filters.

2.4 Summary

XLIFF is a relatively new standard, and has yet to realise its full potential in tool implementations and processes – particularly within the open source community. As described in section 2.3.1, XLIFF has an impressive range of advantages compared to the PO file format. PO was designed for software message catalogues, but has been exploited in the absence of alternatives - to become a common file format for other localizable information because of its good tool support.

While there is a case for throwing away PO totally and replacing it with XLIFF, the vast number of existing projects that depend on Gettext and the PO format suggests a stronger case for keeping it. There is no urgent need to replace PO as a software message format, but adding another layer (XLIFF) makes it easier to create a common tool to support this format as well as formats such as DocBook XML. Introducing XLIFF based workflows and tool support will increase the level of translator efficiency by pre-translating segments and adding meta-data such as translator comments.

Successful implementation of an XLIFF based localisation process for open source projects relies on 3 major factors: A Translation Memory system, XLIFF filters for common file formats and XLIFF enabled translation tools. None of these presently exist as open source implementations, and further research is needed on how to best incorporate these in open source localisation processes. In addition, open source generally has poor terminology management and localisation process management, and further research is needed, investigating how open standards can improve open source localisation in these areas.
3 Research Approach

In working with open source related topics, we have, as academic researchers, a choice of two distinct approaches to handling the interaction with the open source community: First we have the observer approach, where the researcher simply observes the work happening in an open source environment, but do not take an active role as a contributor. Secondly, we have the interactive approach, where the researcher becomes an active part of a community, contributing to the greater good of open source. In this research we have a strong focus on the interactive approach, and seek to improve open source localisation as active contributors.

This approach to research fits well in with the Action research methodology. Action research requires both practical and academic outcomes (Coghlan and Brannick, 2001). The practical outcomes of this project include a model for improving open source localisation processes through the adoption of XLIFF. This is accomplished through the development of filters to convert between PO and XLIFF. To develop these filters, the requirement of academic outcomes is fulfilled, as we define a specification for mapping the PO format to XLIFF.

![Figure 19: The Action Research process](image.png)

Action research is defined as a 5 step iterative process. In the above figure, the development of the XLIFF 1.1 Representation Guide for Gettext PO is shown as a single action research cycle. The problem of mapping PO to XLIFF is further broken down into smaller identifiable objectives, each mapping to an action research cycle. By doing this, we ensure that we thoroughly cover all aspects relating to mapping PO to XLIFF, and at all stages get constructive feedback from the open source community.

3.1 The XLIFF Tools project

To support our dual goal of contributing back to the community, and collaboration with the open source community, we launched the XLIFF Tools project, hosted on
Freedesktop.org, a small collaborative open source developer portal. Under the XLIFF Tools umbrella, we initially launched two sub-projects: the definition of a XLIFF representation guide for PO, and xlifftool\(^3\) – a project aiming at implementing filters and tools to support XLIFF within Gettext based development and localization processes. To accomplish the project goals, we had three main requirements:

1. **A project Mailing list:** The mailing list is the main channel of communication for the XLIFF tools project. In addition, the public mailing list archives is an important source of information and a reference point for future research in the area of XLIFF and open source localization.

2. **A wiki based web site:** The web site has to functions. First of all, it is the primary source of information about the XLIFF Tools project, and provides pointers to the mailing list archives, the source code repository, and other information relating to the project. The web site, being a wiki, is also a method for collaboration, providing a simple way for contributors to share and collaborate on ideas.

3. **A source code repository:** The source code repository provides a way for contributors to retrieve the latest implementations of software related to the XLIFF Tools project, and provides a way for contributors to collaboratively work on developing the same software project.

The Freedesktop.org project was selected as the host for the XLIFF Tools project. We chose Freedesktop.org because of its status as a cross-desktop developer community, working on standards shared by major open source desktops such as GNOME and KDE. Further, Freedesktop.org was attractive because of its small but vibrant developer community, compared to the overwhelming number of projects hosted by the other identified alternative, SourceForge.net.

The XLIFF Tools project was launched 31 January 2005 by publishing an announcement to relevant mailing lists in the area of open source internationalisation and localization. Now, four months later, the project mailing list has over 25 subscribers, and the archive has passed 200 messages, proving the success of the project.

### 3.2 Tasks

<table>
<thead>
<tr>
<th>#</th>
<th>Task</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Launch XLIFF Tools project</td>
<td>This task consists of 1) Setting up the web-site, source code repository and mailing list, and 2) Publishing a project announcement to the relevant mailing-lists and standards bodies.</td>
</tr>
<tr>
<td>2</td>
<td>Identify key aspects of PO and other open source localization formats.</td>
<td>Investigate how localization is presently done in open source software through the GNU Gettext library and other tools; Identifying key strengths and weaknesses.</td>
</tr>
<tr>
<td>3</td>
<td>Investigate how open source localization formats can be supported in XLIFF.</td>
<td>Investigate how the various features of open source localization can be supported in XLIFF; also, investigate if there are aspects of open source localization that are not presently well supported in XLIFF.</td>
</tr>
<tr>
<td>4</td>
<td>Define XLIFF Representation</td>
<td>Create a guide for how to represent the various features of the PO file format in XLIFF.</td>
</tr>
</tbody>
</table>

\(^3\) See Appendix B for a general overview of xlifftool.
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>5 Implement tools to convert between PO and XLIFF</td>
<td>Identify other areas of open source localisation needing reform, and investigate how open standards can help in accomplishing this.</td>
</tr>
<tr>
<td>6 Investigate other standards</td>
<td>Prepare and present paper on this research for the 2nd Asia-Pacific workshop on Software Internationalisation, part of Australian Software Engineering Conference (ASWEC), 29 March - 1 April 2005, Brisbane, Australia.</td>
</tr>
<tr>
<td>7 Prepare ASWEC Workshop Paper</td>
<td>Summarize and analyse results of findings, prepare and organize reflections to form the honours dissertation.</td>
</tr>
<tr>
<td>8 Write honours dissertation</td>
<td></td>
</tr>
</tbody>
</table>

Table 1: Research Tasks

3.3 Verification and Testing of results

As this research is expected to result in a paper submission to the XLIFF Technical Committee, it is vital to get comments from experts in the field at the various stages of the research. This will ensure quality control, and the researcher will have valuable feedback to work with. Specifically, this will be done by approaching a network of contacts in the XLIFF Technical Committee and the OpenI18N Committee for comments and review. This stage of the research will be supported by the selected research approach. As the research is evaluated by experts in the field (evaluation phase of action research), the result will decide if additional cycles are needed to refine the solutions.

The researcher seeks to collaborate with individuals in the open source community for testing and review of software artefacts (XLIFF filters for PO). To ensure compatibility with the vast amount of existing GNU Gettext based software, the filters will also be tested on a large amount of existing data (PO files) from projects such as GNOME, Fedora and KDE.

3.4 Ethical issues

Software artefacts will be released under an open source license compatible with the GNU General Public License (Free Software Foundation, 1991).

As this project seeks to involve the open source communities, it is important to give credit where credit is due. The researcher will ensure that contributors are acknowledged in accordance to open source practices, through CVS log messages as well as in the software documentation.
4 Adopting XLIFF in Open Source Localisation

In this section we examine in depth how XLIFF may be adopted within open source localisation processes. In doing so, we have chosen to focus solely on the PO file format. While there are a number of other file formats in use in the open source localisation process (indeed many that could gain more from converting to XLIFF than PO), PO remains the core format in the localisation process, and an essential target for any meaningful revision.

We start off by presenting results from the heart of this research – the XLIFF Representation Guide for Gettext PO (Section 4.1). In doing so, we also discuss difficulties and challenges encountered in this process. We then we look at how to incorporate XLIFF in the current open source development processes (Section 4.2), and finally, we examine the need for open source tool support for XLIFF (Section 4.3).

4.1 XLIFF Representation of the PO format

While earlier versions of the XLIFF standard provided considerable flexibility in the permissible attribute values - with many allowing free-form parsed character data - the XLIFF technical committee has recognised that this flexibility acts as a barrier to intelligent resource matching and translation reuse. Recent revisions to the standard have limited this practice, placing greater reliance on explicit enumeration of attribute values. In the subsequent discussion, there arise a number of situations in which the shell of our proposed mapping is readily defined and specified, but the resulting framework permits a range of implementations consistent with this specification. In each case we have tried to specify a model or reference implementation which we believe best represents the additional information contained in the PO format, while remaining consistent with the intent of the XLIFF standard.

The XLIFF standard and its structure were examined in some detail in Section 2.2.2, and similarly, a high-level overview of the PO format was presented in Section 2.1.3. Details of the actual mapping between PO and XLIFF are deferred to Appendix A (XLIFF 1.1 Representation Guide for Gettext PO). In this section, we will however, present the core findings and challenges identified in the process.

4.1.1 Overview

We have mapped each PO file to a XLIFF <file> element, with the datatype attribute set to ‘po’ (Figure 20 below). The PO header is mapped to a translation unit, or is stored in the XLIFF skeleton. Each additional Gettext domain defined in the PO file is encapsulated in a XLIFF <group> element, with the restype attribute set to ‘x-gettext-domain’.
4.1.2 The PO Header.

The PO header contains both technical and project-related meta-data, and can also contain user-defined variables. As seen in Figure 21, the header is structured as a normal translation unit, with the source field (msgid) left empty, and the PO header elements contained within the target field (msgstr). Additional meta-data, such as copyright and licensing information, is stored in comment fields.

Figure 21: PO Header

PO supports a set of project meta-data for which there is no existing XLIFF equivalent. XLIFF is merely interested in the localizable content of a file, whereas PO also contains meta-data about the localisation project and technical meta-data providing context.

In processes where XLIFF is simply used as an optional transient localisation format (Section 2.2.1), it is important that translators are able to change some of the information stored in the PO header – such as adding their name as a translator and changing the PO revision date. In addition, translators must be able to retrieve information such as the email address for bug reports. To accommodate this, the easiest
and most convenient approach is to store the PO header data in a `<trans-unit>` element, as shown in the following example:

```xml
<trans-unit id="#1" retype="x-gettext-domain-header">
  <source>
    Project-Id-Version: MyPackage 1.0
    Report-Msgid-Bugs-To: foo@example.com
    POT-Creation-Date: 2004-11-11 04:29+0900
    PO-Revision-Date: 2005-02-01 12:00+0900
    Last-Translator: Foo Bar foo@example.com
    Language-Team: My Language LL@li.org
    MIME-Version: 1.0
    Content-Type: text/plain; charset=utf-8
    Content-Transfer-Encoding: 8bit
    Plural-Forms: nplurals=2; plural=n>1;
    X-Custom-Variable: value
  </source>
  <target>
    ... modified header ...
  </target>
  <note from="po-file">
    Translation for MyPackage.
    Copyright (C) 2005 Foo Bar
    This file is distributed under the same license as MyPackage.
    Foo Bar <foo@example.com>, 2005.
  </note>
</trans-unit>
```

Figure 22: PO header as a `<trans-unit>` element

While this approach has some downsides, it nevertheless provides a way to combine PO and XLIFF based localisation processes:

- Translators would need to know the purpose and format of PO headers even though they are using XLIFF based localisation tools
- The PO header is not a conceptual translation unit, and therefore treating it as one is not consistent with the XLIFF Specification.

In processes where XLIFF is used as a persistent format throughout the development and localisation process, it is possible to eliminate the need for the PO header in the localisation process. Project – and localisation related meta-data can be stored elsewhere, and the technical meta-data needed for converting PO to MO at build time can be automatically generated.

4.1.3 Translation Units

Not surprisingly, we suggest mapping each PO translation unit to a XLIFF `<trans-unit>` element, with the `<source>` element representing the PO `msgid` field, and the `<target>` element representing the PO `msgstr` field. While this base-case is trivial, there are several features of the PO format that require further consideration:

Plural forms. XLIFF does not have any concept of pluralisation of translation units, but we suggest simulating this through utilizing XLIFF’s hierarchical structure of `<group>` and `<trans-unit>` elements. As shown below, a group of plurals can be grouped in a XLIFF `<group>` element, and each plural form then be represented as one `<trans-unit>` element within that group.
The fuzzy flag. The fuzzy flag is Gettext’s way of marking a translation as unfinished, and denotes that it needs review by a translator. This flag can easily be mapped to the approved attribute in XLIFF, where this attribute is set to ‘no’ if the fuzzy flag is present or when the PO entry is not translated.

Comments. PO files have two types of comments: Translator comments and extracted comments. The extracted comments occur both in PO Templates and PO files, and represent comments related to a translation unit, which are added by developers in the source code. Translator comments occur only in PO files, as they are added by translators in the localisation process. We suggest mapping these to XLIFF <note> elements, and/or XLIFF <context> elements, as shown in the following example:

Source code references. Each PO entry contains a list of references, referring to the location (source file + line number) the translation unit was extracted from. The current XLIFF specification has some support for this concept through a set of predefined values (‘sourcefile’ and ‘linenumber’) for the ‘type’ attribute in <context> elements:

Figure 23: Plural Translation Unit in XLIFF

Figure 24: Handling of PO comments in XLIFF
Problems do however occur when a PO file has more than one reference for a single translation unit. In these situations it would be tempting to simply add a `<context-group>` element for each PO reference, but there is a problem with this approach: The XLIFF specification states that context groups within the same hierarchical level in the XLIFF file must be uniquely named\(^4\), though it would be beneficial and more logical to have all PO references in a single context group. A workaround for this might be to not use the native XLIFF approach, but rather store PO references in `<context>` elements where the 'type' attribute is set to a custom value such as 'x-location':

\[^4\] A small ambiguity in the XLIFF specification was found while discussing this. The issue is being raised with the XLIFF Technical Committee: http://lists.oasis-open.org/archives/xliff/200505/msg00000.html
identifier, and this value cannot be used as the ‘resname’ attribute for translation units, because of the limitation of XML attribute values. To overcome this, we suggest using hash-values to create unique identifiers, using the following two rules:

- For non-plural Translation Units, use a string hash of `domain_name + "::" + msgid`. If the Translation Unit is in the default domain, use ‘messages’ as the domain name.
- For plural Translation Units, use a string hash of `domain_name "::" + msgid + "::plural[" + n + "]"`, where n is the plural index of msgstr.

Handling Escape Sequences. In source code it is often necessary to use escape sequences to represent characters such as the newline character (\u2028), the horizontal tab character (\u0009) and non-ASCII Unicode characters. We suggest representing these in XLIFF using the intended characters, and not the escape sequences. For example:

```
msgid ""
"Please Enter the following Data:\n"
"\t- First Name\n"
"\t- Last Name\n"
msgstr ""
```

Figure 28: PO entry with escape sequences

This entry should be represented in XLIFF as follows, replacing the escape sequences with the intended characters:

```
<trans-unit>
  <source>Please Enter the following Data:
    - First Name
    - Last Name
</source>
</trans-unit>
```

Figure 29: XLIFF translation unit with converted escape sequences

4.2 Gettext and XLIFF in the Development Process

Up until now we have primarily focused on a general mapping between PO and XLIFF. While this is beneficial in itself, the success of adopting XLIFF in open source localisation rely how well this format can be integrated in development processes and tools. In this section, we first examine the most commonly used Gettext based localisation workflow, and then extend our discussion to consider incorporating XLIFF in open source localisation. Finally we discuss the need for open source translation tools that support XLIFF and present recent developments in the area.

Most GNU based open source projects use automated build systems based on GNU Autotools. These systems support integration with Gettext, automatically extracting translatable strings from source-code and, in some cases, merge previously translated string-tables using fuzzy-matching. These build systems are composed of small unix tools, called from macros and build scripts. The Gettext toolkit comes with a set of tools for this purpose:

<table>
<thead>
<tr>
<th>Tool</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>xgettext</td>
<td>Extracts messages from source code into PO Template (POT) files</td>
</tr>
<tr>
<td>msginit</td>
<td>Initializes a POT file to a language specific PO file</td>
</tr>
<tr>
<td>msgmerge</td>
<td>Merges a translated PO file with a newly extracted POT files by adding new translation units, marking obsolete entries (entries that have been</td>
</tr>
</tbody>
</table>
removed from source code) and updating extracted comments and source code references.

<table>
<thead>
<tr>
<th>msgfmt</th>
<th>Generates a binary MO file from a translated PO file.</th>
</tr>
</thead>
</table>

Table 2: Most commonly used tools in the Gettext package

4.2.1 Current Gettext based workflow

The following diagram gives an overview of how Gettext is integrated in the development process and build system:

![Workflow Diagram](image)

Figure 30: Current PO based workflow

1. Messages are extracted from source files to PO String table Templates (POT files) using the xgettext tool.

2. For each language, previously translated PO files are merged (2b) with the freshly extracted PO Template file using the msgmerge tool. If there is no existing PO file for the target language, the POT file is initialized (2a) for the target language using the msginit tool. In addition, entries are at this stage in some cases matched against PO Compendium files containing completed translations from across projects.

3. Translators retrieve PO files from the repository, and translate using PO based localisation tools. Translated PO files are then committed back to the repository.

4. When a maintainer creates a release, Machine Object (MO) files are generated from translated PO files using the msgfmt tool, and included in the distributed package.

5. Applications retrieve translated messages from MO files at runtime.

While this diagram gives a good overview of the process, it is important to note that it is only a representation a common way of using Gettext in the development process, and most projects use variations of this workflow. For example, in many projects it is
common that merging of PO files (Step 2) is done by translators in the translation process. In addition, many project use the intltool package to extract strings to PO from other file types (Desktop entries, User Interface dialogs and XML files) for localisation, and this tool also takes care of merging the back the translated entries. It is also important to note, in the diagram above, that Steps 1 through 3 often are often iterated many times in a single release cycle, providing a way for translators to incrementally translate an application while it is being developed.

4.2.2 Optional XLIFF workflow in current processes

One approach of incorporating XLIFF in open source is to leave the present processes and workflows untouched, and simply use XLIFF as an optional file format for use by translators in the localisation process. With this approach, PO files are converted to XLIFF for translation, and then back-converted to PO when translation is complete:

A limitation with this approach is that every time the PO sources are updated (this happens quite frequently in open source development processes), new XLIFF files would

---

5 po2xlf and xlf2po have been developed by the researcher as part of the XLIFF Tools project. For a general overview of the toolkit, see Appendix B.
have to be generated, loosing the rich meta-data stored in the old XLIFF sources. With this approach, PO is the persistent file format (Section 2.2.1), and XLIFF is simply a transient file format used in the localisation process. To overcome this, we need an approach where XLIFF is stored in the repository as a persistent format.

### 4.2.3 Native XLIFF based workflow

In the previous approach, PO files are stored in the repository, and XLIFF is merely used as a temporary file format in the localisation process. The following approach proposes changes to the build system to allow the use of XLIFF as the persistent file format in the build systems and localisation process, totally eliminating the need for the PO format in the localisation process:

**Figure 32: Fully XLIFF based workflow**

1. Extract messages from source code to PO template using the `xgettext` tool.
2. Convert the PO Template file to XLIFF using the `po2xlf` tool.
3. Initialize (a) (using `xlfpoinit`) or update (b) (using `xlfpomerge`) the XLIFF file for each language.
4. Optionally pre-process the XLIFF file for each language (For example adding translation suggestions).
5. Translate XLIFF file for each language using XLIFF based localisation tools.
6. When a release is prepared, convert the translated XLIFF files back to PO (using `xlf2po`).
7. Create MO files from PO files for distribution, using the `msgfmt` tool.

$xlfpoinit$ and $xlfpomerge$ have been developed by the researcher as part of the XLIFF Tools project. For a general overview of the toolkit, see Appendix B.
In this new workflow we have replaced PO with XLIFF as the persistent format, storing XLIFF files in the version control repository. The need for PO – other than an intermediate format for communicating with GNU Gettext - is eliminated. PO files are never edited by developers or translators, being used only as a temporary format in the software development process. Developers, localisation engineers and translators can then all benefit from XLIFF in all stages of the development process.

To support this approach, we have introduced two new tools: `xlfpoinit` and `xlfpomerge`. These tools serve the same purpose as their PO equivalents (`msginit` and `msgmerge`), and provide a way to manage updating sources in the localisation process.

By using this approach, open source projects can leverage all the benefits of XLIFF based localisation, but still use Gettext internally as the resource format. This approach also builds a platform for further expansion of the localisation process in open source project, where other source formats (such as DocBook and Desktop Entry files) can be converted to XLIFF for localisation.

### 4.2.4 Gettext Integration

Building on the approach presented above, it would now be possible to integrate XLIFF within the GNU Gettext toolkit, eliminating the need for custom XLIFF filters and tools in the build system and development process:

1. Extract messages from source code to a language-neutral XLIFF file, using the modified `xgettext` tool.
2. Initialize (a) or update (b) the XLIFF file for each language, using the modified `msginit` and `msgmerge` tools.
3. Optionally pre-process the XLIFF file for each language (For example adding translation suggestions).
4. Translate XLIFF file for each language using XLIFF based localisation tools.

Figure 33: XLIFF Gettext integration
5. Create MO files from XLIFF files for distribution, using the modified `msgfmt` tool.

This approach totally eliminates the need for the PO format, as the Gettext tools would work natively with XLIFF. We have not prioritized further researching this approach, as our goal with this research is to propose XLIFF as a common resource format for all open source processes, and the ‘native approach’ described above meet our needs in this area.

### 4.3 Tool Support

The PO file format has a stranglehold on open source localisation, mainly because of its simplicity, but also because of good tool support. KBabel is by far the most feature-rich PO editor, and includes support for Translation Memories, spell checking, plural-forms and PO catalog management. A large obstacle for the adoption of XLIFF in open source localisation is the lack of good open source localisation tools supporting the format. While KBabel has some support for XLIFF\(^7\), and Transolution\(^8\) promising to become a fully fledged open source translation suite, the present PO tools still dominates, both in terms of usability and features. This is set to change shortly, with SUN Microsystems releasing their Localisation Tools and filters to the community, through the Open Language Tools project. There have, however, been very long delays in the open sourcing of this project - and only time will tell when these tools will be made available.

In addition to the typical rich client PO editors, there has recently been an increasing interest in web based translation portals such as Pootle and Rosetta. These portals allow contributors to translate PO files using regular web browsers. Though these interfaces lack the richness of modern user interfaces (limited to dynamic HTML pages), they attract contributors because of the inherent simplicity, - not having to deal with the technical issues often involved in open source localisation, such as version control repositories. These portals allow for greater control of the localisation process, and can be further improved by adopting other open standards in addition to XLIFF, - which we will discuss in the following section.

\(^7\) KBabel’s data structures are based on the PO format, hence, the support for XLIFF is limited to the richness of the PO format.

\(^8\) More information on Transolution available at [http://transolution.python-hosting.com/](http://transolution.python-hosting.com/)
5 Localisation Process Improvements through Open Standards

In the previous section we examined in detail how open source localisation processes based around GNU Gettext and the PO file format could benefit from the use of XLIFF. In addition to defining a reference mapping between the two formats, we examined how XLIFF could be incorporated into build systems – ensuring that localisable resources are stored in XLIFF as opposed to PO in the version control repositories. Using XLIFF stored as the main localisation resource format provide the foundation for what we are about to present.

In this section we present possible solutions to the more challenging issues facing open source localisation – the actual localisation process. We do this by further examining common open source localisation processes (background presented in Section 2.1), and propose high level solutions based on standards such as TMX, TBX and Translation Web Services. Some of these standards are still in draft form (Translation Web Services, TBX), while others have been around for a while, and are accepted as industry standards (TMX). Common to all these standards, is the fact that they have emerged as results of needs in the commercial localisation industry, and promote best practices in the field.

Some of the themes we discuss in this section are not revolutionary or new in commercial localisation settings (in fact, we draw examples from some of the available proprietary tools). They are however new in the setting of open source and collaborative localisation, and contribute to meeting many of the challenges facing open source localisation:

1. Limited Translation reuse
2. Ad-hoc based Terminology Management
3. High level of technical skills needed to contribute
4. Error prone due to highly manual processes
5. No standardized way of contributing to the localisation process – different project use different approaches, but often same people contributing.
6. Limited support for quality control in the localisation process

We will discuss and propose solutions to all these issues. In Section 5.1 we discuss how we can increase translation reuse (1) through centralized translation memory repositories. Section 5.2 presents possible solutions for improving terminology management (2) in open source, through the adoption of standards such as TBX within an XLIFF-based localisation process. Finally, in Section 5.3 we discuss the need for service based localisation workflows, and simplification of the localisation process through process automation and well defined communication protocols (4, 5 and 6). The main aim of this discussion is not to present the right solution (we don’t at this point present full solutions, - just ‘sketches’), but to identify ways in which workflows based on open standards can meet the challenges facing traditional open source localisation processes.
5.1 Shared Translation Memory Repositories

As described in Section 2.1.5, current Gettext based localisation practices provide very limited support for translation reuse, mainly due to the limited support for segmentation and abstraction of inline codes in the PO format. However, large open source projects, like KDE, GNOME and Linux distributions, are ideal eco-systems for centralized translation memory databases, due to the vast amount of translatable data and the large number of contributors.

It is tempting to look at each of these large open source projects from an industry viewpoint. Localisation projects are Localisation Providers; community contributors are translators working for the provider, and software maintainers are localisation customers purchasing services from the provider. The providers retrieve sources for translation from customers. Ideally, these sources can now be matched against the providers’ translation memory, and sent to translators for completion. When a translation is complete, the providers feed the translation memory with the updated translation, increasing the value of the translation memory, which is critical intellectual property to the provider. In contrast to commercial localisation providers though, open source TM assets could be shared across projects, further increasing the possibilities of positive matches. In addition, TM assets could be made available for download in form of TMX archives. These files could then be imported into client localisation tools supporting the TMX format.

The following figure depicts a typical workflow for incorporating a centralized Translation Memory in the localisation process:

![Figure 34: XLIFF workflow with TMX based Translation Memory](image)

In the above figure, XLIFF documents are retrieved from the version control repository (1), and submitted to the TM system for pre-translation (2). In the pre-translation process, exact and fuzzy matches are added to the XLIFF documents. In the translation process, the translator can then accept or reject the translation suggestions, and further translate the XLIFF documents (3). Approved entries in the translated XLIFF documents are then exported to TMX (4), and imported to the TM repository (5). Independent of this, translated documents are committed to the version control repository.
With the upcoming segmentation support addition\(^9\) to the XLIFF standard, the XLIFF format will now provide extensive support for segmentation, in addition to the existing support for abstraction of inline codes. Combining XLIFF with the Segmentation Rules eXchange (SRX) standard and TMX, it will now be increasingly feasible to implement efficient Translation Memory support based on open XML standards. With widely published rules for text segmentation (based on the SRX standard), efficiently leveraging translations from across project boundaries becomes possible, as data from different TMX enabled translation memories can share common segmentation rules.

### 5.2 Terminology Management

As with TM matching, not much standardization has been done with terminology management in open source localisation (Section 2.1.6). Warburton (2005b) identifies the advantages early identification and definitions of terms in the localisation process. The XLIFF format provides support for this in the file format, as presented in the following figures:

**Figure 35:** Translation Unit without identified terms

Here we could identify the term “path separator” by encapsulating it in a `<mrk>` element:

```xml
<trans-unit id="1">
  <source>Please make sure the path separator is set to ‘/’</source>
</trans-unit>
```

**Figure 36:** Translation Unit with terms identified

Identification of terms should ideally occur at a very early stage in the localisation process, before any translation work is done. This makes it possible for all members of language teams to standardize on the terminology first for a particular project, before any translation work is done. As English is used as the source language in most open source projects, identification of terms can be done synchronously with the development effort, in a process similar to what we depict in the following diagram:

**Figure 37:** Marking terms in XLIFF documents

Here, an XLIFF document (ideally a language neutral document) is retrieved from the version control repository (1). Terms are identified and marked in the XML document (2), before the file is committed to the repository again (3). In addition, a mechanism should exist that allows terms to be identified after a translation process is started.

\(^9\) The XLIFF segmentation proposal is at time of writing being finalized, and is expected to be formally accepted, by voting in the XLIFF Technical Committee, in June 2005.
The process of extracting terms could be automated, by checking a document against a list of pre-existing terms. It is important to note however, that different projects might use different definitions for identical terms, and that such a process does not take into account new terms that are not present in the predefined list. This could be improved by having a semi-automated process, where potential terms were automatically flagged, and a ‘review’ phase could be introduced to identify new terms and approve the terms automatically extracted in the process.

Obviously, the process of identifying terms is only valuable if definitions of these terms exist and are properly managed, which the job of terminology management systems (also referred to as terminology databases or term bases). Availability of such systems become increasingly important in open source localisation, where often many contributors at different locations work on translating projects for a specific language. In open source, this is presently commonly handled through sharing dictionaries using collaborative systems such as wikis.

The problem with using web pages or wikis for sharing terminology databases is that the data cannot be easily utilized by software. The ideal situation would be to have the terminology data in a format that tools in the localisation process could easily exchange and use to automatically lookup terms that are identified in the source file. That’s where the TermBase eXchange standard (Section 2.2.4) comes in. By adopting the TBX standard for terminology management in open source software, it is possible to automatically include terminology definitions for terms present in a XLIFF document, and ship this with translations. This again highlights the advantage of XLIFF over PO, as we in XLIFF can include data from translation memory and terminology databases. Utilizing XLIFF in this way, as project bundles, is already proven by industry, with localisation tools such as RC-WINTRANS (Figure 38 below) using this approach.

---

![Diagram](image.png)

**Figure 38: RC-WINTRANS use of XLIFF project files**

### 5.3 Service based Localisation Workflows

In the previous two sections we described ways of incorporating centralized Translation Memories as well as Terminology Management in open source localisation. Current localisation practices in projects such as the Fedora and KDE localisation...
projects require translators to manually retrieve PO files from version control repositories for translation. Having translators retrieve source files directly from the development repositories eliminates the possibility of doing any automated server side processing. In this section we identify a need for a higher level of abstraction in the localisation processes – automated workflow systems, to accommodate many of the aspects we have discussed so far.

We propose a generalized model (Figure 39 below) based on interoperable communication protocols such as XML-RPC or SOAP Web Services, that allows for communication between clients (rich clients like KBabel, or web based systems like Pootle) and upstream translation projects (such as KDE, GNOME and Fedora). This would abstract away the need for translators to handle technical issues such as CVS, and would further allow for better control and management of the localisation process.

![Figure 39: Localisation Infrastructure based on standards based XML formats](image)

This model is based on XLIFF files (and other project-related meta-data) being stored in a version control repository, similarly to how we in current practices store PO in the repositories. Using this workflow, TM entries and Terminology definitions (as described in Section 5.1 and 5.2) could be added to the XLIFF archives before they are sent to translators (B and C), and similarly new entries could be added to the TM before committing the translations to the repository (A).

Perhaps the most useful set of tools in current open source localisation practices are the catalog managers (Section 2.1.4), in form of translation status pages and tools like KBabel’s Catalog Manager. These tools present translation tasks and status reports in user friendly and intuitive ways, and in addition often provide ways of directly retrieving sources from version control repositories. Recently, with portals such as Pootle, Rosetta and IRMA, these status pages have been extended to become fully fledged localisation solutions, where translations can be performed in a web based environment. Expanding on these concepts, in the figure above, we propose a web services based interface (D) to the Localisation Infrastructure system, enabling any client supporting the protocol (such as web based translation portals, rich client localisation tools) to communicate with the upstream localisation project.

The OASIS Translation Web Services (Trans-WS) committee aims to develop a web services based interface for automating communication between localisation vendors and clients. This specification includes methods for requesting localisation quotes, retrieving and submitting localisation jobs, and querying the status of localisation jobs. In addition, the specification covers querying language service providers for which services and languages are supported.
Open source localisation processes based on community contribution do not fit very well in within the scope of the Trans-WS specification. In open source projects, translators (community contributors) are the active party, providing their services in terms of translating an application to a specific language, based on various motivating factors. There is however emerging needs for standardized and automated processes in open source localisation, where community contributors can query, retrieve and submit translations. Currently this can be done by manually checking the translation status pages for a project with a web browser, or alternatively, in the case of the KDE project, by using a tool such as the KBabel Catalog Manager to interact with the version control system to retrieve this information.

The reason for proposing a Web Services based protocol such as SOAP or XML-RPC, is the flexibility it gives in implementing clients. By implementing a standards based interface, custom ‘Catalog Manager’ type application can be used to retrieve translation projects (collection of XLIFF files), which then in turn can be translated in any XLIFF enabled editor. The same interface could then be used to develop web based translation portals, rich client catalog managers, and localisation suites. Currently this is a big obstacle for translation portals such as Pootle, Rosetta and IRMA, as there are no ways of committing and updating translations upstream in a standardized way.

Defining a standard interface for interacting with open source localisation repositories also opens up possibilities of composing services in a unique way. This is especially useful for community driven Linux distributions such as Ubuntu and the Fedora Project, where the distributions are composed by combining software from a multitude of projects. For example, in the Fedora project, it would be beneficial if translators could automatically retrieve translations tasks for the current GNOME or KDE release, without having to manually track each project independently.

In building an infrastructure targeting open source localisation, we cannot assert that localisation contributors have high-speed and reliable internet connections, or even have internet connections at all. In developing countries, it is often more feasible to arrange translation gatherings (nick-named ‘translate-a-thons’), where contributors come together in school computer labs or similar and translate open source software using thin client web based translation tools, and one computer acting as a server running a localisation portal such as Pootle. The proposed model also caters for this, as similarly, local servers could be set up, or sets of translation tasks could be retrieved upfront. In addition, the proposed localisation model (Figure 39) also supports the current approach with contributors directly interacting with the version control repositories, retrieving and committing sources for translation in their XLIFF based localisation tool of choice (E). We see it as important to support this simple approach, as it would eliminate the danger
of a single point of failure, and in addition keep contributors that are hesitant to change their current localisation practices happy.

The overall aim in proposing a service based localisation infrastructure is to enhance and simplify the localisation process from a translator point-of-view. The technical level of knowledge needed to contribute to current open source localisation processes, at worst, turn potential contributors away, and in other cases decrease the translator efficiency. With structured approaches to new technologies (translation memory, terminology management), these issues must be addressed, as these technologies introduce another level of complexity – and may if not addressed potentially cause more pain than gain for translators.
6 Conclusions

In Section 4 we presented the main contribution of this thesis – a bridge between XLIFF, the industry standard for exchange of localisable data, and GNU Gettext, the de facto standard in open source localisation. This was accomplished by developing an XLIFF representation guide for the Gettext PO file format, followed by a discussion of how best to incorporate XLIFF within present open source development processes. In this section we specifically targeted the PO format, as it is being used (some would say exploited) as a common resource format in the localisation process for a number of file types in open source. Our goal has been, however, not to simply eliminate PO in favour of XLIFF in Gettext based localisation, but rather to propose XLIFF as the standard resource format for all open source localisation, much in the same way as PO is being used (or indeed, over-used) at present. This research has only fulfilled part of that goal, as only the PO format has been thoroughly covered. To make XLIFF an attractive option in open source localisation, tools and filters to convert between other common file formats, especially DocBook and other formats presently converted to PO for localisation, are needed.

By adopting XLIFF as the common resource format in open source localisation presently based on Gettext, we have also accomplished a de-coupling of technologies in the localisation and internationalisation process. Developers are no longer limited to using GNU Gettext, and can investigate using other alternatives without affecting the localisation process. Such decoupling brings forth a unique opportunity for further research in the area of localisation technologies in the development process.

During the course of this research, we have noticed an increasing interest from the open source community in regards to adopting XLIFF in localisation processes presently based on Gettext and the PO format. Developers from leading localisation projects (including Ubuntu’s Rosetta, Gnome, KDE, Fedora and Pootle) have expressed an interest in XLIFF as a replacement for PO. It is, however, too early to determine if, and when, these new standards will be adopted in these projects, as much of the underlaying infrastructure and tools (mainly XLIFF filters and localisation tools) do not yet exist. In addition, despite all the deficiencies and limitations of Gettext and PO, current localisation practises are to a large extent successful - with projects like KDE and GNOME being localised to over 80 languages, and the benefits of switching to XLIFF once the infrastructure is in place will have to be overwhelming to justify such a move.

In Section 5 we looked beyond XLIFF, discussing possible future opportunities for open source localisation processes, further extending the bridge between open source and industry standards. The discussion was built on the foundation laid in the previous section, having XLIFF as the common resource format, and focused on three main areas in need of better solutions in open source: translation reuse, terminology management and service based localisation workflows.

Translation reuse in open source has up until now only been addressed in an ad hoc fashion, through PO Compendiums and the KBabel PO editor, and there are no structures in place for sharing translation memory data. In addition, the data fed into these translation memories are not ideal, as current PO based processes have no support for segmentation, alignment and abstraction of inline codes and markup. Adopting XLIFF and TMX in these processes provide a foundation for building quality translation
memories, storing quality data, which can in turn increase translation efficiency in years to come. There are, however, no existing open source TMX certified Translation Memory systems available, and further research and development is needed in this area.

Terminology management is, arguably, at least from a quality and consistency point of view, more important than translation reuse. In current open source processes, terminology is, at best, as with translation memory, handled in an ad hoc fashion. Terms are not identified prior to the translation process, and if terminology management is undertaken at all, language teams simply use combined bi-lingual glossaries of words that have been previously identified in the translation process. The localisation industry has identified and acknowledged the importance of proper terminology management, and has agreed on a standard file format, TBX, for exchanging terminology. Further research is needed in defining workflows incorporating terminology in open source, and further, developing open source terminology management systems supporting these standards.

As with many other industries, the localisation industry is looking at ways of automating the business processes through service oriented architectures based around SOAP Web Services. This work is now being formalized through the OASIS Translation Web Services committee, developing a specification providing interfaces for automating communication between localisation vendors and customers. As the commercial interest in open source continues to grow, Translation Web Services can be of great benefit to open source software. Open source software vendors can use this channel to localise resources (which is now based on XLIFF) through commercial localisation service providers. There is, however, a need for service oriented architectures in the collaborative open source localisation environment, as this process is not covered by the Trans-WS specification. In open source localisation, translators are the active part (the localisation provider), providing services to an open source localisation project (the client), and research is needed, in defining a service oriented architecture for these environments. The goal of this would be, as with industry standards in the field, to provide an automated system eliminating many of the trivial and non-localisation related tasks relating to the process; In other words, the goal must be to let the translator concentrate on the translation and not the surrounding technicalities.

The standards we have covered in this thesis, except for TMX, are still fairly young, and are still in a process of maturing. XLIFF 1.1 was approved as an OASIS Committee Specification in November 2003, but is still being finalized before submission for approval as an OASIS standard. Revision 1.1 of the standard limited the amount of free-form meta-data allowed through depreciation of the <prop> element, and provided a larger set of pre-defined attribute values. As free-form data is being restricted in the XLIFF namespace, the specification now recommends using other namespaces within the document for storing additional information, further expanding the possible usage areas of the format. This has recently been demonstrated through the work on the XLIFF Representation Guide for HTML, where common (X)HTML attributes are added to the translation units, further enriching the set of available meta data.

As the localisation industry adopts service based architectures and Translation Web Services, we believe there will be a strong focus on the linkage between standards based XML file formats. The beginning of this can already be seen, through the inclusion of elements from the XLIFF namespace in the Trans-WS specification, and it is reasonable to believe that there will also be a stronger linkage between XLIFF and TBX in future revisions of the standard, providing a way of linking terms identified in the XLIFF document with definitions in the TBX document.
If the open source communities adopt the standards based file formats discussed in this thesis, not only will it benefit the localisation processes, but the adoption will provide a unique playground for these standards to continue maturing in an open environment. This will not only benefit the open source communities, but also the commercial localisation industry.

As a conclusion to this thesis, the XLIFF resource format is mature and rich enough to provide a valuable replacement for the GNU Gettext PO format. But for this to happen, development of localisation tools supporting XLIFF is needed, and further, development of XLIFF filters for other file types that currently use PO in the localisation process. Open source localisation can also benefit from standards beyond XLIFF, and through further research, the localisation process can be enriched through standardized management of translation reuse and terminology. To fully take advantage of this, structured localisation workflows needs to be developed, and these can benefit from the development in the area of Translation Web Services, providing a way of automating the localisation process.
Appendix A: XLIFF Representation Guide for Gettext PO
XLIFF 1.1 Representation Guide for Gettext PO

Working Draft 7 June 2005

Document identifier:

wd-xliff-profile-po

Editor:

Asgeir Frimansson <asgeirf@redhat.com>

Abstract:

This document defines a guide for mapping the GNU Gettext PO (Portable Object) file format to XLIFF (XML Localisation Interchange File Format).

Status of this Document:

This document is a Preliminary Working Draft of the committee. It is an OASIS draft document for review by OASIS members and other interested parties. Comments may be sent to <xliff-comment@lists.oasis-open.org>

This document may be updated, replaced, or rendered obsolete by other documents at any time. It may also be discarded without further follow up. It is inappropriate to use this document as reference material other than "work in progress".

This document is developed in collaboration with the XLIFF Tools Project [XLIFF Tools].

Copyright © 2005 The Organization for the Advancement of Structured Information Standards [OASIS]
Appendix A: XLIFF Representation Guide for Gettext PO

1. Introduction

As different tools may provide different filters to extract the content of Gettext Portable Object (PO) 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 PO data in XLIFF. It offers a collection of recommended mappings of all features of PO that developers of XLIFF filters can implement, and users of XLIFF utilities can rely on to insure a better interoperability between tools.

2. Overview of the PO file format

Because the Gettext PO format is not a defined standard - nor is the format well documented, we will in this section present an overview of the features and design of the PO file format.

2.1. PO and POT

There are two types of PO files: PO Template files (POTs) and Language specific PO files (POs). POTs contain a skeleton header, followed by the extracted translation units. POTs are generated by the xgettext extraction tool and are not meant to be edited by humans. POTs are converted into Language Specific POs by the msginit tool, and these files are then edited by translators.

When source code is updated, a new POT is generated for the project, and the changes from previous versions are incorporated into the existing translations by using the msgmerge tool. This tool inserts new translation units into the existing PO files, marks translation units no longer in use as obsolete, and updates any references and extracted comments.

Transoded PO files are converted to binary resource files, known as MO (Machine Object) files, by the msgfmt tool. The Gettext library use MO files at run time; hence PO files are only used in the development and localisation process.

2.2. General Structure

A PO file starts with a header, followed by a number of translation units.

# SOME DESCRIPTIVE TITLE.
# Copyright (C) YEAR THE PACKAGE'S COPYRIGHT HOLDER
# This file is distributed under the same license as the PACKAGE
# package.
# FIRST AUTHOR <EMAIL@ADDRESS>, YEAR.
#
#, fuzzy
msgid ""
msgstr ""
"Project-Id-Version: PACKAGE-NAME VERSION\n"
"Report-Msgid-Bugs-To: BUG-EMAIL-ADDR <EMAIL@ADDRESS>\n"
2.3. Header

The PO header follows a similar structure to PO translation units, but is distinguished by its empty source element (msgid). The header variables are contained in the headers' target (msgstr) element, with newline character representations (\n) separating each variable.

The initial comment lines (comments are lines starting with "# ") usually contains a copyright notice as well as licensing information, followed by a list of all translators that has been involved in translating the specific PO file.

The header skeleton in a POT file is initially marked with the fuzzy flag (flags are comma separated entries on lines starting with ", ") . This flag is removed when the header variables are filled in and the POT file is initialized to a language-specific PO file.
### Table 1. Predefined PO Header variables

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project-Id-Version</td>
<td>Application name and version</td>
</tr>
<tr>
<td>Report-Msgid-Bugs-To</td>
<td>Mailing list or contact person for reporting errors in translation units.</td>
</tr>
<tr>
<td>POT-Creation-Date</td>
<td>Date POT file was generated. Automatically filled in by Gettext</td>
</tr>
<tr>
<td>PO-Revision-Date</td>
<td>Time stamp when PO file was last edited by a translator</td>
</tr>
<tr>
<td>Last-Translator</td>
<td>Contact information for last translator editing the file.</td>
</tr>
<tr>
<td>Language-Team</td>
<td>Name of language team that translated this file.</td>
</tr>
<tr>
<td>MIME-Version</td>
<td>MIME version used for specifying Content-Type</td>
</tr>
<tr>
<td>Content-Type</td>
<td>MIME content type and character set for this file</td>
</tr>
<tr>
<td>Content-Transfer-Encoding</td>
<td>MIME transfer encoding</td>
</tr>
<tr>
<td>Plural-Forms</td>
<td>Number of plural forms in target language, and c-expression for evaluating which plural form to use for a parameter.</td>
</tr>
</tbody>
</table>

In addition to these predefined variables, the PO header can contain custom user-defined variables of the same format.

#### 2.4. Translation Units

```plaintext
# Translator Comment
#. Extracted Comment
#: myfile.c:12 myfile.c:32
#, flag
msgid "Original String"
msgstr "Translated String"
```

PO translation units use the source string (msgid) as primary id, and contain the translation in the msgstr field. In addition to this, PO translation units contain other meta-data, explained in further detail in the following sections.

#### 2.4.1. Source and Target

The `msgid` and `msgstr` contains the source and target string of a translation unit.

The actual content of `msgid` and `msgstr` is a concatenation of the strings enclosed by quotes (U+0022 characters) on each line. For example:

```plaintext
msgid ""
"My name is "
"
"%s. \n"
"What is"
"
"your name?"
```

is exactly the same as:
Appendix A: XLIFF Representation Guide for Gettext PO

2.4.2. Translator Comments

```plaintext
# This is a comment line

# This is another comment line
```

Translator comments are lines starting with "# " (U+0023 + U+0020). These comments are added by translators, and are not present in POT files.

2.4.3. Extracted Comments

```plaintext
#. This is an extracted comment
#. This is another extracted comment
```

Extracted comments are lines starting with ",. " (U+0023 + U+002E). These comments are extracted from the source code. Source-code comments are normally extracted if they are on the same line as the source string, or on the line immediately preceding it, as in the following c-example:

```c
/* This comment will be extracted */
gettext("Hello World");
```

This would become:

```plaintext
#. This comment will be extracted
msgid "Hello World"
msgstr ""
```

When updating a PO file from a new POT file, existing extracted comments in the language specific PO file are discarded, and the extracted comments present in the POT file are inserted in the existing PO file.

2.4.4. References

```plaintext
#: myfile.c:1 myfile.c:23 otherfile.c:1
#: otherfile.c:34
```

References are identified by lines starting with "#: " (U+0023 + U+003A). References are space separated lists of locations (sourcefile:linenumber) specifying where the translation unit is found in a source file.

As each `msgid` has to be unique within a PO domain, a single translation unit can contain multiple references; one for each location where the string is found in the source code.

Similar to extracted comments, when updating a PO file from a new POT file, existing references in the language specific PO file are discarded, and the references present in the POT file are inserted in the existing PO file.

2.4.5. Flags

Flags are identified by lines starting with "#, " (U+0023 + U+002C). Multiple flags are separated by commas.
Flags are used both as processing instructions by the Gettext tools, and by translators to indicate that a translation unit is unfinished or "fuzzy".

### Table 2. Flag values and descriptions

<table>
<thead>
<tr>
<th>Flag Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fuzzy</td>
<td>Indicates that a translation units needs review by a translator. This flag is inserted by the gettext tools when a translation unit changes, or when the translation unit does not pass the format check. The flag is also commonly used by translators to mark a translation unit as unfinished. Note that entries marked as fuzzy are not included when PO files are compiled to binary MO files.</td>
</tr>
<tr>
<td>no-wrap</td>
<td>Indicates that the text in the msgid field is not to be wrapped at page with (usually 80 characters) which it usually is. Note that this does not affect the wrapping of the actual source string, only the representation of it in the PO file. This flag is set by developers in the source code, or by adding a command-line flag when invoking the Gettext tools.</td>
</tr>
<tr>
<td>X-format, where X is any of the following:</td>
<td>Indicates that Gettext is to do a format check on the translation unit to validate that both msgid and msgstr contains valid parameter values according to the source format. This flag is automatically Inserted by the Gettext extraction tool.</td>
</tr>
<tr>
<td>awk</td>
<td></td>
</tr>
<tr>
<td>c</td>
<td></td>
</tr>
<tr>
<td>csharp</td>
<td></td>
</tr>
<tr>
<td>elips</td>
<td></td>
</tr>
<tr>
<td>gcc-internal</td>
<td></td>
</tr>
<tr>
<td>java</td>
<td></td>
</tr>
<tr>
<td>librep</td>
<td></td>
</tr>
<tr>
<td>lisp</td>
<td></td>
</tr>
<tr>
<td>objc</td>
<td></td>
</tr>
<tr>
<td>object-pascal</td>
<td></td>
</tr>
<tr>
<td>perl</td>
<td></td>
</tr>
<tr>
<td>perl-brace</td>
<td></td>
</tr>
<tr>
<td>php</td>
<td></td>
</tr>
<tr>
<td>python</td>
<td></td>
</tr>
<tr>
<td>qt</td>
<td></td>
</tr>
<tr>
<td>scheme</td>
<td></td>
</tr>
<tr>
<td>sh</td>
<td></td>
</tr>
<tr>
<td>smalltalk</td>
<td></td>
</tr>
<tr>
<td>tcl</td>
<td></td>
</tr>
<tr>
<td>ycp</td>
<td></td>
</tr>
</tbody>
</table>
Flags (except **fuzzy**) are inserted and overridden by developers in source code, by adding them to a comment immediately preceding the call to `gettext`, as in the following example:

```c
/* xgettext:no-c-format */
printf(_("Hello World"));
```

Since the Gettext call here is inside a `printf` function call, the `gettext` tools will automatically assume this is a **c-format** string. But in this example the developer overrides that, and specifies it is not so, which would generate the following PO translation unit:

```
#, no-c-format
msgid "Hello World"
msgstr ""
```

### 2.4.6. Plural Forms

Gettext, in addition to supporting normal translation units with a single `msgid` and `msgstr`, support **plural form** translation units. These translation units contain the **singular** English form in the `msgid` field, and the **plural** form in the `msgid_plural`. As the target, these translation units have an array of `msgstr`, representing the number of forms in the target language:

```c
msgid "You have %d file"
msgid_plural "You have %d files"
msgstr[0] "Du har %d fil"
msgstr[1] "Du har %d filer"
```

The target language may have one or more forms (Japanese has one form, while Polish has 3 forms), and the logic for selecting which form to use for a parameter is defined in a PO header field, where `nplurals` defines the number of forms and `plural` contains a c-expression for evaluating which item in the `msgstr` array to use at run time:

```
"Plural-Forms: nplurals=2; plural=(n != 1);
"
```

This is a typical example for a Germanic language, which has a special case when `n` is 1. A more complex example is Polish, which has special cases for when `n` is 1, and in addition some numbers ending in 2, 3 or 4:

```
"Plural-Forms: nplurals=3; 
"plural=n==1 ? 0 : n%10>=2 && n%10<=4 && (n%100<10 || n%100>=20) ? 1 : 2;
"
```

C-expressions are defined as `condition ? true_value : false_value` where `condition` is an expression evaluating to true/false. In the above example, the first
condition is \( n=1 \) which if true gives the result 0, and if false gives the result of a second c-expression. For the second expression, the condition is \( n\%10\geq 2 \quad \&\& \quad n\%10\leq 4 \quad \&\& \quad (n\%100<10 \quad || \quad n\%100\geq 20) \), which if true gives the result 1, and if false gives the result 2. At run time, Gettext will use the \texttt{msgstr} with the index returned from this expression.

### 2.4.7. Obsolete Translation Units

Obsolete entries are translation units that are no longer present in the source-files, and are therefore commented out when a PO file is updated. These entries are re-used by Gettext only if the translation-unit re-appears in the project, and are also used for fuzzy matching by the 'msgmerge' tool. Obsolete entries are marked with "#~" (U+0023 + U+007E), as in the following example:

```plaintext
# This is a translator comment
#: msgid ""
#: "Please enter the following details:\n"
#: "  - First Name\n"
#: "  - Last Name\n"
#: msgstr ""
#: "Venligst fyll inn følgende data:\n"
#: "  - Fornavn\n"
#: "  - Etternavn\n"
```

### 2.5. Domains

One single PO file normally represents one MO file, known as a Gettext \texttt{domain}, but the PO format also allows for representing multiple domains in a single PO file. This is done by adding the \texttt{domain} keyword followed by the domain name, as in the following example:

```plaintext
domain "domain_1"

msgid "hello world"
msgid "hei verden"

domain "domain_2"

msgid "hello world"
msgid "hei verden"
```

The above example would produce two MO files, \texttt{domain_1.mo} and \texttt{domain_2.mo}. If no domain is specified, translation units belong to the default domain \texttt{messages}.

A PO header is bound to a domain, so each domain has its own header.

Having multiple domains in a single PO file is very rare; in fact, the authors have never seen this in use.

### 3. General Considerations

This section discusses the general considerations to take in account when extracting data from PO files.
3.1. PO flavours

Because of good open source tool support, the PO file format has been used as a common file format for the extraction of localisable data from a number of different source formats, including XML-based document-formats such as Docbook. This guide mainly covers representation of PO files generated from the GNU Gettext toolkit - targeting only localisation of software messages.

It is fully possible to apply this guide to PO files extracted from XML formats. However, it is highly recommended to use native XLIFF filters wherever possible, and not use PO as a middle-format in these processes.

3.2. Source and Target Languages

The PO file format does not provide a way of identifying the source and target language within a file. By GNU standards, GNU software is written in American English (en-US), and this is reflected in Gettext by only having support for Germanic plural forms in the source language. It is therefore recommended to set the source-language attribute to en-US by default.

POSIX locale names typically use the form language[_territory][.codeset][@modifier], where language is an ISO 639 language code, territory is an ISO 3166 country code, and codeset is a character set or encoding identifier like ISO-8859-1 or UTF-8.

Locale names (through use of the source-language, target-language and xml:lang attributes), should, - as specified in the XLIFF specification, use [RFC 3066], and not variants of the POSIX form.

3.3. Translation Unit Ids

The PO file format is different from most other software localisation resource formats in that it does not use ID based translation units. Gettext use the source string as the primary id, meaning that within a Gettext domain, a source string must be unique.

When representing a PO translation unit in XLIFF we cannot use the source string as the value for the id or resname attribute because of the limitations of XML attribute values. Many localisation tools rely on these attributes for leveraging, updates and alignment, hence not providing a solution for this may cause interoperability problems.

We suggest the following approach for providing unique resname attribute values for translation units:

- For non-plural Translation Units, use a string hash of domain_name + ":" + msgid. If the Translation Unit is in the default domain, use "messages" as the domain name.
- For plural Translation Units, use a string hash of domain_name + ":" + msgid + ":plural[" + n + "]", where n is the plural index of msgstr.
It is however possible to use the PO format with logical ids, though this approach is not much used. To support this, filters may add an optional function (specified by a command-line flag or similar) to use msgid as the logical id, and then put the value of msgstr in the <source> element.

For example:

<table>
<thead>
<tr>
<th>msgid &quot;HELLO_WORLD&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>msgstr &quot;Hello World!&quot;</td>
</tr>
</tbody>
</table>

would be mapped to:

```
<trans-unit id="1" resname="HELLO_WORLD" xml:space="preserve">
  <source>Hello World!</source>
</trans-unit>
```

After translation, the translated entry would be inserted as msgstr. For example:

```
<trans-unit id="1" resname="HELLO_WORLD" xml:space="preserve">
  <source>Hello World!</source>
  <source>Hei verden!</source>
</trans-unit>
```

would be back-converted to PO as:

<table>
<thead>
<tr>
<th>msgid &quot;HELLO_WORLD&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>msgstr &quot;Hei Verden!&quot;</td>
</tr>
</tbody>
</table>

### 3.4. Handling of Escape Sequences in Software Messages

Software messages commonly use escape sequences for representing common control characters like newline ('\n'), horizontal tabs ('\t'), and others. When converting to XLIFF, these sequences can either be preserved, or filters may choose to replace escape sequences with the intended character representation.

For example, the following C source code fragment:

```c
printf("Please Enter the following Data:\n\t- First Name\n\t- Last Name\n");
```

would be represented in PO as:

```c
msgid "Please Enter the following Data:\n\t- First Name\n\t- Last Name\n"
msgstr ""
```

This fragment could be presented in XLIFF by preserving the escape sequences:

```
<source>Please Enter the following Data:\n\t- First Name\n\t- Last Name\n</source>
```

which could be further enhanced by encapsulating escape characters in <ph> elements:

```
<source>Please Enter the following Data:<ph id='1' ctype='lb'>\n</ph><ph id='2' ctype='x-ht'>\t</ph>- First Name<ph id='3' ctype='lb'>\n</ph></source>
```
Or, the filters could replace escape sequences with the intended characters:

```
Please Enter the following Data:
- First Name
- Last Name
```

The recommended approach, as also depicted in the table below, is as follows:

- **Escape Sequences representing ASCI Control Characters**, except '\n' (Linefeed LF - U+000A), '\r' (Carriage Return CR - U+000D) and '\t' (Horizontal Tabulator HT - U+0009), should remain as escaped sequences in XLIFF. The escape sequences should be abstracted in `<ph>` elements, with the c-type attribute set to x-ch-NN where NN is the name of the ASCI control character.
- The Control Character '\t' (Horizontal Tabulator HT - U+0009) should be converted to the intended Unicode representation (U+0009).
- The Control Character '\n' (Linefeed LF - U+000A) should be converted to the intended Unicode representation (U+000A).
- The Gettext tools discourages use of the '\r' (Carriage Return CR - U+000D) escape sequence. Filters may choose to implement support for Mac and DOS/Windows style line endings by replacing DOS/Windows ('\r\n') and Older Mac ('\r') line endings with Unix ('\n') line endings. Filters could store information about the original line endings encoding, and use this information to insert the correct line endings on back-conversion.
- All other escaped characters should be converted to the intended Unicode representation.

In addition, characters in a PO file that are not supported by the XML specification (For example Vertical Tabulator VT - U+000B) should be abstracted in a similar way to control characters.

### Table 3. Handling of Common Escape Sequences

<table>
<thead>
<tr>
<th>Escape Sequence</th>
<th>Intended Character</th>
<th>PO representation</th>
<th>XLIFF representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>?</td>
<td>? (U+003F)</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>'</td>
<td>' (U+0027)</td>
<td>'</td>
<td>'</td>
</tr>
<tr>
<td>&quot;</td>
<td>&quot; (U+0022)</td>
<td>&quot;</td>
<td>&quot;</td>
</tr>
<tr>
<td>\</td>
<td>\ (U+005C)</td>
<td>\</td>
<td>\</td>
</tr>
<tr>
<td>\a</td>
<td>BEL (U+0007) [a]</td>
<td>\a</td>
<td>&lt;ph ctype=&quot;x-ch-bel&quot;&gt;\a&lt;/ph&gt;</td>
</tr>
<tr>
<td>\b</td>
<td>BS (U+0008) [a]</td>
<td>\b [b]</td>
<td>&lt;ph ctype=&quot;x-ch-bs&quot;&gt;\b&lt;/ph&gt;</td>
</tr>
<tr>
<td>\f</td>
<td>FF (U+000C) [a]</td>
<td>\f [b]</td>
<td>&lt;ph ctype=&quot;x-ch-ff&quot;&gt;\f&lt;/ph&gt;</td>
</tr>
<tr>
<td>Escape Sequence</td>
<td>Intended Character</td>
<td>PO representation</td>
<td>XLIFF representation</td>
</tr>
<tr>
<td>----------------</td>
<td>-------------------</td>
<td>-------------------</td>
<td>----------------------</td>
</tr>
<tr>
<td>\n</td>
<td>LF (U+000A)</td>
<td>\n</td>
<td>LF [c]</td>
</tr>
<tr>
<td>\r</td>
<td>CR (U+000D)</td>
<td>\r [b]</td>
<td>LF [c]</td>
</tr>
<tr>
<td>\t</td>
<td>HT (U+0009)</td>
<td>\t</td>
<td>HT</td>
</tr>
</tbody>
</table>
| \v             | VT (U+000B) [a]   | VT[d]             | &lt;ph ctype="x-ch-
|                |                   | VT">\v&lt;/ph&gt;   |

[a] These characters cannot be used in XML. For more information, see Section 2.2 in the XML Specification [XML 1.0].

[b] Throws a Gettext Warning when used: "xgettext: internationalized messages should not contain the `X' escape sequence" where X is `\b', `\f' or `\r'.

[c] See bullet point above on handling Windows and Mac line endings.

d] Is from version 0.15 of Gettext handled similar to `\b', `\f' and `\r' escape sequences.

3.5. Character Set Conversion

The Content-Type PO header field specifies the character encoding used in the PO file. This field is used at run time by Gettext to provide character set conversion to the character set used by the application.

When extracting data from PO files, filters should use the Content-Type information to provide conversion to UTF-8 for storing data in XLIFF. On back-conversion, filters should also honour this field when re-creating the PO file.

3.6. Extracting from POT files

POT files are automatically generated by the Gettext tools, and is nothing but a simple string table containing the extracted translation units. POTs are much simpler than POs, which are modified by humans and contain additional meta-data (Translator comments, Header information).

If PO is not used in the localisation process, it would in many situations be more feasible to convert directly from POT to XLIFF, and not use language-specific PO files at all in the localisation process.

When converting from POT, the header can be ignored, as the header stored in POT is simply a skeleton header. When back-converting to PO, the filter can insert the necessary PO header elements (MIME elements and optionally plural forms definitions), providing all data needed to produce the language specific MO files.

When plural translation units exist in the POT file, it is important to note that it is impossible to send off a language neutral XLIFF file to translators. Filters need to insert the correct number of <trans-unit> elements for a plural group, and hence, filters need information on how many plural forms there are in a target language.
4. General Structure

Each PO file maps to one XLIFF <file> element. XLIFF representations of PO files should have the datatype attribute set to `po`, and the original attribute set to the name of the PO file.

The XLIFF may encapsulate the meta-data from the PO header in a <trans-unit> element, or store the header in a skeleton file.

The XLIFF <body> element contains translation units, which may be grouped by PO domains using hierarchical <group> elements.

5. Detailed Mapping

5.1. Header

There are two recommended approaches to handling the PO header in XLIFF: Leaving the header out of the XLIFF file, or treating the header as a translation unit. Both approaches are described below.

5.1.1. Approach 1: Leave header out

The information contained in the PO header is not needed in the localisation process, and can be left out of the XLIFF file.

When converting POT files, it is possible to completely ignore the PO header, as described in Section 3.6, “Extracting from POT files”.

5.1.2. Approach 2: Use a <trans-unit> element

This approach involves storing the whole PO header as a XLIFF <trans-unit> element; with the restype attribute set to `x-gettext-domain-header`. In PO the header is identified by a empty source field (msgid), and the header is stored in the target field...
Appendix A: XLIFF Representation Guide for Gettext PO

(msgstr). In converting to XLIFF, we copy the value of msgstr to both <source> and <target>, ensuring that translators can modify the header without losing track of the original content. Translator comments and the fuzzy flag is handled the same way as other translation units.

For example:

```xml
<trans-unit id="1" retype="x-gettext-domain-header" approved="no"
xml:space="preserve">
<source>
 John Developer <john@example.com>, 2005.
</source>
</trans-unit>
```

would be mapped to:

```xml
<trans-unit id="1" retype="x-gettext-domain-header" approved="no"
xml:space="preserve">
<source>
 Project-Id-Version: MyApp 1.0
 Report-Msgid-Bugs-To: MyApp List <mvp-app-list@example.com>
 POT-Creation-Date: 2005-04-27 13:15+0900
 PO-Creation-Date: 2005-04-27 13:45+0900
 Last-Translator: Joe Translator <joe@example.com>
 Language-Team: French Team <fr-list@example.com>
 MIME-Version: 1.0
 Content-Type: text/plain; charset=UTF-8
 Content-Transfer-Encoding: 8bit
 Plural-Forms: nplurals=2; plural=(n!=1)
 X-Generator: KBabel 1.9
</source>
</trans-unit>
```
The content of the PO header can hardly be seen as translatable data, hence this approach is not fully faithful to the XLIFF specification. However, this approach is recommended as a “lesser-of-evils” approach in that it allows translators to modify PO header information - which is necessary in many Gettext based localisation processes.

5.2. Translation Units

5.2.1. Non-Plurals

Each PO entry maps to a XLIFF <trans-unit> element, and contains the source string (msgid) in the <source> element, and the translation (msgstr) in the <target> element. White space and formatting should be preserved by setting the xml:space attribute to preserve.

For example:

<table>
<thead>
<tr>
<th>msgid</th>
<th>&quot;hello world&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>msgstr</td>
<td>&quot;hei verden&quot;</td>
</tr>
</tbody>
</table>

would be mapped to:

```xml
<trans-unit id="1" xml:space="preserve">
  <source>hello world</source>
  <target>hei verden</target>
</trans-unit>
```

5.2.2. Plurals

Each plural PO entry maps to a XLIFF <group> element with the restype attribute set to x-gettext-plurals, and contains one <trans-unit> element for each plural form in the target language.

For example:

<table>
<thead>
<tr>
<th>msgid</th>
<th>&quot;%d file deleted&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>msgid_plural</td>
<td>&quot;%d files deleted&quot;</td>
</tr>
<tr>
<td>msgstr[0]</td>
<td>&quot;%d fil slettet&quot;</td>
</tr>
<tr>
<td>msgstr[1]</td>
<td>&quot;%d filer slettet&quot;</td>
</tr>
</tbody>
</table>

would be mapped to:

```xml
<group restype="x-gettext-plurals">
  <trans-unit id="1[0]" xml:space="preserve">
    <source>%d file deleted</source>
    <target>%d fil slettet</target>
  </trans-unit>
  <trans-unit id="1[1]" xml:space="preserve">
    <source>%d files deleted</source>
    <target>%d filer slettet</target>
  </trans-unit>
</group>
```

When the target language has more than two plural forms, the plural source (msgid_plural) should be used in the <source> element for all translation units except the first.
Appendix A: XLIFF Representation Guide for Gettext PO

For example:

```plaintext
msgid "untranslated-singular"
msgid_plural "untranslated-plural"
msgstr[0] "translated-form-0"
msgstr[1] "translated-form-0"
...
msgstr[n] "translated-form-n"
```

would be mapped to:

```xml
<group restype="x-gettext-plurals">
  <trans-unit id="1[0]" xml:space="preserve">
    <source>untranslated-singular</source>
    <target>translated-form-0</target>
  </trans-unit>
  <trans-unit id="1[1]" xml:space="preserve">
    <source>untranslated-plural</source>
    <target>translated-form-1</target>
  </trans-unit>
  ...
  <trans-unit id="1[n]" xml:space="preserve">
    <source>untranslated-plural</source>
    <target>translated-form-n</target>
  </trans-unit>
</group>
```

When only one form exists for the target language (For example Japanese, Chinese, Korean), the plural group should include a second `<trans-unit>` element with the `translate` attribute set to `no`. This element should contain the original plural source (`msgid_plural`) in the `<source>` element, and is needed when back-converting to PO to create the `msgid_plural` field.

For example:

```plaintext
msgid "untranslated-singular"
msgid_plural "untranslated-plural"
msgstr[0] "translated-form-0"
```

would be mapped to:

```xml
<group restype="x-gettext-plurals">
  <trans-unit id="1[0]" xml:space="preserve">
    <source>untranslated-singular</source>
    <target>translated-form-0</target>
  </trans-unit>
  <trans-unit id="1[1]" xml:space="preserve" translate="no">
    <source>untranslated-plural</source>
  </trans-unit>
</group>
```

It is important to be aware of the implications of plural forms when extracting data from language neutral POT files, as described in Section 3.6, “Extracting from POT files”.

62
5.2.3. Obsolete Entries

Obsolete entries should not be included in the XLIFF file, and can be stored in a skeleton or ignored.

5.3. Translator Comments

Translator comments in PO have the same function as <note> elements in XLIFF - providing a way for people involved in the localisation process to include comments relating to a translation unit.

It is possible to map each translator comment to a <note> element, specifying that the comment is extracted from the PO file using the from attribute. Multi-line comments are concatenated, each line separated by a newline character.

For example:

```plaintext
# This is a comment that
# goes over multiple lines
msgid "hello world"
msgstr ""
```

could be mapped to:

```xml
<trans-unit id="1">
  <source>hello world</source>
  <note from="po-translator">This is comment that goes over multiple lines</note>
</trans-unit>
```

Optionally, translator comments can be mapped to <context> elements with the type attribute set to x-po-transcomment. For example:

```plaintext
# This is a comment that
# goes over multiple lines
msgid "hello world"
msgstr ""
```

could be mapped to:

```xml
<trans-unit id="1">
  <source>hello world</source>
  <context-group name="po-entry" purpose="information">
    <context type="x-po-trancomment">This is comment that goes over multiple lines</context>
  </context-group>
</trans-unit>
```

It is up to the individual filter implementer to decide which approach (if not both) to use.

5.4. Extracted Comments

Extracted comments in PO are comments extracted from source code, and provide a way for developers to add comments relating to a translation unit. They can be mapped to XLIFF in a similar fashion to Translator Comments.
It is possible to map each extracted comment to a `<note>` element, specifying that the comment is extracted from the PO file, representing a developer comment, using the `from` attribute. Multi-line comments are concatenated, each line separated by a newline character.

For example:

```xml
<trans-unit id="1">
  <source>hello world</source>
  <note from="developer">This is comment that goes over multiple lines</note>
</trans-unit>
```

Optionally, extracted comments can be mapped to `<context>` elements with the `type` attribute set to `x-po-autocomment`. The surrounding `<context-group>` element (same context group as the Translator Comment as described above) would have the `name` attribute set to `po-entry` and the `purpose` attribute set to `information`.

For example:

```xml
<trans-unit id="1">
  <source>hello world</source>
  <context-group name="po-entry" purpose="information">
    <context type="x-po-autocomment">This is comment that goes over multiple lines</context>
  </context-group>
</trans-unit>
```

As with Translator Comments, it is up to the individual filter implementer to decide which approach (if not both) to use.

### 5.5. References

Each reference is mapped to two `<context>` elements, one specifying the source file (`context-type attribute` set to `sourcefile`) and the other representing the location in the source file (`context-type attribute` set to `linenumber`).

Each reference is in addition grouped in a `<context-group>` element, with the `name` attribute set to `po-reference`, and the `purpose` attribute set to `location`.

For example:
Appendix A: XLIFF Representation Guide for Gettext PO

### 5.6. Flags

#### 5.6.1. fuzzy

The `fuzzy` flag in PO maps to the `approved` attribute of a `<trans-unit>` element in XLIFF. The `approved` attribute is set to `no` if the `fuzzy` flag is present, and is set to `yes` if the flag is absent.

For example:

```plaintext
#: example.c:34 otherfile.c:233
msgid "Hello world"
msgstr ""

msgid "Hello world!"
msgstr "Hei verden!"
```

should be mapped to:

```xml
<trans-unit id="1" approved="no">
  <source>hello world</source>
</trans-unit>
<trans-unit id="2" approved="yes">
  <source>Hello world!</source>
  <target>Hei Verden!</target>
</trans-unit>
```

If the `msgstr` field is empty and the `fuzzy` flag is absent, the translation unit is still marked as not approved. When the `msgstr` field contains data and the `fuzzy` flag is set, the `state` attribute of the `<target>` element is set to `needs-review-translation`.

For example:

```plaintext
#, fuzzy
msgid "Hello world"
msgstr ""

msgid "Hello world!"
msgstr "Hei verden!"
```
should be mapped to:

```xml
<trans-unit id="1" approved="no">
    <source>Hello world</source>
</trans-unit>
<trans-unit id="2" approved="no">
    <source>Hello world!</source>
    <target state="needs-review-translation">Hei Verden!</target>
</trans-unit>
```

When back-converting to PO, the fuzzy flag is set unless the approved attribute of the translation unit is set to yes.

### 5.6.2. no-wrap

The no-wrap flag only controls the visual layout of a translation unit in the PO file, and not the actual content. Hence, this flag has no meaning in an XLIFF file and can be ignored by filters.

Note that it is possible, when back-converting to PO, to honour the no-wrap flag. This can be done by implementing the same formatting rules as the Gettext tools:

- Leave the first line (same line as the msgid/msgstr keyword) blank.
- Only split lines when encountering the newline character ("\n"); Do not word-wrap long lines.

For example:

```xml
<trans-unit id="1" approved="yes">
    <source>As prompted on the following screen, please enter the following details:
    - First Name
    - Last Name
</source>
    <target>Venligst fyll inn følgende data når du kommer til neste skjermbilde:
    - Fornavn
    - Etternavn
</target>
</trans-unit>
```

would when back-converted be formatted as:

```plaintext
msgid ""
"As prompted on the following screen, please enter the following details:\n" - First Name\n" - Last Name\nmsgstr ""
"Venligst fyll inn følgende data når du kommer til neste skjermbilde:\n" - Fornavn\n" - Etternavn\n```

in favour of word-wrapping similar to this:

```plaintext
msgid "As prompted on the following screen, please enter:
" - First Name\n" - Last Name\nmsgstr ""
"Venligst fyll inn følgende data når du kommer til neste skjermbilde:\n" - Fornavn\n" - Etternavn\n```
How the no-wrap flag is stored (if it is honoured) in the localisation process, is up to the individual filter implementers.

5.6.3. x-format

The x-format flag (For example: c-format, java-format, php-format) specifies that the Gettext is to do some format checks before accepting the translation, ensuring that the parameters present in the source string (msgid) is there in the translated entry (msgstr). This format check is done by the Gettext tools after translation, when generating MO files, or when merging a PO file with a newly extracted POT file.

This flag can be honoured by extracting parameters to <ph> elements with the c-type attribute set to the format flag (see the table below). For example:

```plaintext
msgid "Hello $s, your score is $d."
msgstr "Hei $s, du har $d poeng."
```

Here the parameters $s and $d can be extracted:

```plaintext
<trans-unit id="1" approved="yes">
  <source>Hello <ph id="1" ctype="x-c-param">$s</ph>, your score is <ph id="2" ctype="x-c-param">$d</ph></source>
  <target>Hei <ph id="1" ctype="x-c-param">$s</ph>, du har <ph id="2" ctype="x-c-param">$d</ph> poeng.</target>
</trans-unit>
```

Table 4. Recommended c-type attribute values

<table>
<thead>
<tr>
<th>Flag Name</th>
<th>c-type value</th>
</tr>
</thead>
<tbody>
<tr>
<td>awk-format</td>
<td>x-awk-param</td>
</tr>
<tr>
<td>c-format</td>
<td>x-c-param</td>
</tr>
<tr>
<td>csharp-format</td>
<td>x-csharp-param</td>
</tr>
<tr>
<td>elisp-format</td>
<td>x-elisp-param</td>
</tr>
<tr>
<td>gcc-internal-format</td>
<td>x-gcc-internal-param</td>
</tr>
<tr>
<td>java-format</td>
<td>x-java-param</td>
</tr>
<tr>
<td>libreps-format</td>
<td>x-libreps-param</td>
</tr>
<tr>
<td>lisp-format</td>
<td>x-lisp-param</td>
</tr>
<tr>
<td>obj-c-format</td>
<td>x-obj-c-param</td>
</tr>
<tr>
<td>object-pascal-format</td>
<td>x-object-pascal-param</td>
</tr>
<tr>
<td>perl-format</td>
<td>x-perl-param</td>
</tr>
<tr>
<td>perl-brace-format</td>
<td>x-perl-brace-param</td>
</tr>
<tr>
<td>php-format</td>
<td>x-php-param</td>
</tr>
</tbody>
</table>
Appendix A: XLIFF Representation Guide for Gettext PO

<table>
<thead>
<tr>
<th>Flag Name</th>
<th>c-type value</th>
</tr>
</thead>
<tbody>
<tr>
<td>python-format</td>
<td>x-python-param</td>
</tr>
<tr>
<td>qt-format</td>
<td>x-qt-param</td>
</tr>
<tr>
<td>scheme-format</td>
<td>x-scheme-param</td>
</tr>
<tr>
<td>sh-format</td>
<td>x-sh-param</td>
</tr>
<tr>
<td>smalltalk-format</td>
<td>x-smalltalk-param</td>
</tr>
<tr>
<td>tcl-format</td>
<td>x-tcl-param</td>
</tr>
<tr>
<td>ycp-format</td>
<td>x-ycp-param</td>
</tr>
</tbody>
</table>

For some source formats special consideration is needed when reordering parameters. For example:

```plaintext
Hello %s, your score is %d.
```

If we here in the target language wanted to write:

```plaintext
Score: %d Name: %s
```

we would have to specify the position of the parameters:

```plaintext
Score is %2$d for %1$s
```

Most XLIFF editors does not provide a way for translators to edit the content of the `<ph>` elements, meaning this logic would have to be implemented in the filters.

For example, in the following PO fragment:

```plaintext
#, c-format
msgid "Hello %s, your score is %d."
msgstr ""
```

the extraction filter could insert necessary ordering-tags when converting to XLIFF:

```xml
<trans-unit id="1" approved="no">
  <source>Hello <ph id="1" ctype="x-c-param">%1$s</ph>, your score is <ph id="2" ctype="x-c-param">%2$d</ph>.</source>
</trans-unit>
```

The translator could then safely re-order the parameters:

```xml
<trans-unit id="1" approved="yes">
  <source>Hello <ph id="1" ctype="x-c-param">%1$s</ph>, your score is <ph id="2" ctype="x-c-param">%2$d</ph>.</source>
  <target>Score is <ph id="2" ctype="x-c-param">%2$d</ph> for <ph id="1" ctype="x-c-param">%1$s</ph>.</target>
</trans-unit>
```

and the back converted PO file would then become:

```plaintext
#, c-format
msgid "Hello %s, your score is %d."
msgstr "Score is %2$d for %1$s"
```
Appendix A: XLIFF Representation Guide for Gettext PO

Take note that the parameters in msgid are replaced with the original parameters on back-conversion.

It is recommended to implement support for extracting parameters only if support for parameter re-ordering is also implemented.

5.6.4. no-X-format

no-X-format (For example: no-c-format, no-php-format) flags can be ignored as they have no functional use and are ignored by the Gettext tools. These flags are added by developers in source code to override the automatic insertion of x-format flags.

5.7. Domains

If multiple domains are present in a PO file, it is recommended to group each domain in a <group> element with the restype attribute set to x-gettext-domain and the resname attribute set to the name of the domain. For Example:

```
domain "domain_1"
msgid "hello world"
msgstr "hei verden"
domain "domain_2"
msgid "hello world"
msgstr "hei verden"
```

should be mapped to:

```
<group restype="x-gettext-domain" resname="domain_1">
  <trans-unit id="1">
    <source>hello world</source>
    <target>hei verden</target>
  </trans-unit>
</group>

<group restype="x-gettext-domain" resname="domain_2">
  <trans-unit id="2">
    <source>hello world</source>
    <target>hei verden</target>
  </trans-unit>
</group>
```

In many cases a domain is not specified for the first translation units of a PO file (They are said to belong to the default domain 'messages'). It is recommended to not group these translation units, but rather have them as children of the <body> element, only grouping domains when the domain keyword is found. For Example:

```
msgid "hello world"
msgstr "hei verden"

domain "domain_2"
msgid "hello world"
msgstr "hei verden"
```
should be mapped to:

```xml
<trans-unit id="1">
  <source>hello world</source>
  <target>hei verden</target>
</trans-unit>
<group restype="x-gettext-domain" resname="domain_2">
  <trans-unit id="2">
    <source>hello world</source>
    <target>hei verden</target>
  </trans-unit>
</group>
```

### A. Contributions

The following people have contributed to this document:

- Josep Condal
- Fredrik Corneliusson
- Karl Eichwalder
- Asgeir Frimannsson
- Tim Foster
- David Fraser
- Paul Gampe
- Bruno Haible
- Rodolfo M. Raya
- Yves Savourel

### References

- [OASIS] Organization for the Advancement of Structured Information Standards Website.
- [XML 1.0] *Extensible Markup Language (XML) 1.0 (Third Edition)*. W3C (World Wide Web Consortium), Feb 2004
Appendix B: Overview of xlifftool

The xlifftool package aims to become a key tool for managing localisable resources in open source development processes and build systems. xlifftool is built around the same concepts as intltool [1], but use XLIFF as opposed to PO as the common resource format.

xlifftool is based around the concept of extracting localisable data from supported source formats to target-language neutral XLIFF templates. These files are initialized to target-language specific XLIFF files and can be stored in the version control repository. To keep the resource files updated, xlifftool provides tools to merge newly generated XLIFF templates with existing XLIFF resource files. After translation, the resource files are (usually at release-time) back-converted as translations of the original source files.

The current release of xlifftool only supports the GNU Gettext PO format, but we hope to implement support for other common open source formats (including Desktop Entries and several XML based formats such as DocBook, Glade/Qt UI dialogs and XHTML) in future releases of this product.

For the latest information on this package, visit the xlifftool wiki page at:

http://xliff-tools.freedesktop.org/wiki/Projects/xlifftool


Filters and tools supporting GNU Gettext

On the following pages, we give an overview of the four tools supporting GNU Gettext and the PO format: xlf2po, po2xlf, xlfpointit and xlfpomerge.
Appendix B: Overview of `xlifftool`

**po2xlf**

**Purpose**
To convert PO and POT files to XLIFF

**Description**
The `po2xlf` tool can be used for two purposes: To convert POT files to language neutral XLIFF Template files, and to convert existing PO files to XLIFF. When generating XLIFF Template files, the PO header and existing translations, are ignored.

**Usage**
`po2xlf [OPTIONS] INPUTFILE [OUTPUTFILE]`

If INPUTFILE is -, standard input is read. If no OUTPUTFILE is specified, output is written to standard output

**Options**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--embed-header</td>
<td>Include the PO header as a translation unit</td>
</tr>
<tr>
<td>--po-charset[=UTF-8]</td>
<td>Specify the character set used in the PO file.</td>
</tr>
<tr>
<td>--source-language</td>
<td>Specify the source language</td>
</tr>
<tr>
<td>--target-language</td>
<td>Specify the target language</td>
</tr>
<tr>
<td>--template</td>
<td>Create a XLIFF template only, ignoring any translated entries.</td>
</tr>
<tr>
<td>--help</td>
<td>Display usage information and exit</td>
</tr>
<tr>
<td>--version</td>
<td>Display version information and exit</td>
</tr>
</tbody>
</table>
Appendix B: Overview of xlf2po

**xlf2po**

**Purpose**
To back-convert XLIFF files to PO

**Description**
The xlf2po tool converts translated XLIFF representations of PO files (files generated with po2xlf) back to the original PO format, adding the necessary PO header fields.

**Usage**

```bash
xlf2po [OPTIONS] INPUTFILE [OUTPUTFILE]
```

If INPUTFILE is -, standard input is read. If no OUTPUTFILE is specified, output is written to standard output.

**Options**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--extract-notes</td>
<td>Extract XLIFF <code>&lt;note&gt;</code> elements to PO comments</td>
</tr>
<tr>
<td>--help</td>
<td>Display usage information and exit</td>
</tr>
<tr>
<td>--version</td>
<td>Display version information and exit</td>
</tr>
</tbody>
</table>
Appendix B: Overview of xliff tool

**xlfpoinit**

**Purpose**
To initialize XLIFF representations of POT files to language-specific XLIFF files

**Description**
The `xlfpoinit` tool initializes an XLIFF Template of a POT file for a specific target-language. In this process, the correct number of plural translation units are inserted (specified with the `--nplurals` and `--plural` attributes if not available in the internal database).

**Usage**

`xlfpoinit [OPTIONS] INPUTFILE [OUTPUTFILE]`

If INPUTFILE is -, standard input is read. If no OUTPUTFILE is specified, output is written to standard output

**Options**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--lang</td>
<td>The target language code.</td>
</tr>
<tr>
<td>--nplurals</td>
<td>Number of plurals for target language</td>
</tr>
<tr>
<td>--plural</td>
<td>C-expression for determining plural forms</td>
</tr>
<tr>
<td>--print-table</td>
<td>Displays the list of languages with plural forms information available in the internal plural database</td>
</tr>
<tr>
<td>--help</td>
<td>Display usage information and exit</td>
</tr>
<tr>
<td>--version</td>
<td>Display version information and exit</td>
</tr>
</tbody>
</table>
Appendix B: Overview of xliff tool

xlfpomerge

Purpose
Merges translated XLIFF representation of PO files with updated XLIFF representations of POT files.

Note: This tool is still ‘under construction’.

Description
The purpose of xlfpomerge is to merge two XLIFF-represented PO files: translated.xlf is a XLIFF file that has been previously processed with xlfpoinit. template.xlf is a XLIFF file generated by xlf2po, with the ‘--template’ flag set. Translation Units (TUs) in template.xlf that are not in translated.xlf are added, TUs in translated.xlf that are not in template.xlf are deleted, and PO references and extracted comments are updated.

Usage
xlfpomerge [OPTIONS] translated.xlf template.xlf [OUTPUTFILE]

If template.xlf is -, standard input is read. If no OUTPUTFILE is specified, output is written to standard output.

Options

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--keep-obsolete</td>
<td>Keeps obsolete translation units in translated.xlf</td>
</tr>
<tr>
<td>--help</td>
<td>Display usage information and exit</td>
</tr>
<tr>
<td>--version</td>
<td>Display version information and exit</td>
</tr>
</tbody>
</table>
References


KDE.org Team (2004) *What is Scripty?*  


