XLIFF 1.2 Representation Guide for Gettext PO

Committee Draft, 16 May 2006

This version:
http://www.oasis-open.org/committees/xliff/documents/cd-xliff-profile-po-1.2-20060516.htm

Latest version:
http://www.oasis-open.org/committees/xliff/documents/cd-xliff-profile-po-1.2-20060516.htm

Previous version:
<not applicable yet>

Editor:

Asgeir Frimannsson <asgeirf@redhat.com>

Tony Jewtushenko <tony.jewtushenko@productinnovator.com>

Rodolfo M. Raya <rmraya@heartsome.net>

Abstract

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

Status

This document was last revised or approved by the XLIFF TC on the above date. The level of approval is also listed above. Check the current location noted above for possible later revisions of this document. This document is updated periodically on no particular schedule.

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

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

The non-normative errata page for this specification is located at www.oasis-open.org/committees/xliff/documents/xliff-profile-po-1.2-errata.htm

Notices

Table of Contents

Appendixes

---

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.

1.2 Transitional and Strict

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

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

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

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

- **Strict** - All deprecated elements and attributes are not allowed. Obsolete items from previous versions of XLIFF are deprecated and should not be used when writing new XLIFF documents.

Use this to validate XLIFF documents that you create.

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

2. 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 contains 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.

Translated 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"
"POT-Creation-Date: YEAR-MO-DA HO:MI+ZONE\n"
"PO-Revision-Date: YEAR-MO-DA HO:MI+ZONE\n"
"Last-Translator: FULL NAME <EMAIL@ADDRESS>\n"
"Language-Team: LANGUAGE <EMAIL@ADDRESS>\n"
"MIME-Version: 1.0\n"
"Content-Type: text/plain; charset=CHARSET\n"
"Content-Transfer-Encoding: 8bit\n"
"Plural-Forms: nplurals=2; plural=n!=1;\n"
"X-User-Defined-Var: VALUE\n"
# Translator Comment
#. Extracted Comment
#: myfile.c:12
#, flag
msgid "Original String 1"
msgstr "Translated String 1"
# Translator Comment
#. Extracted Comment
```

```
#: myfile.c:23
#, flag
msgid "Original String 2"
msgstr "Translated String 2"
```

2.3. Header

```
# French Translation for MyApplication.
# Copyright (C) 2005 John Developer
# This file is distributed under the same license as the MyApp package.
# John Developer <john@example.com>, 2005.
# Joe Translator <joe@example.com>, 2005.
#
msgid ""
msgstr ""
"Project-Id-Version: MyApp 1.0\n"
"Report-Msgid-Bugs-To: MyApp List <myapp-list@example.com>\n"
"POT-Creation-Date: 2005-04-27 13:15+0900\n"
"PO-Revision-Date: 2005-04-27 13:45+0900\n"
"Last-Translator: Joe Translator <joe@example.com>\n"
"Language-Team: French Team <fr-list@example.com>\n"
"MIME-Version: 1.0\n"
"Content-Type: text/plain; charset=UTF-8\n"
"Content-Transfer-Encoding: 8bit\n"
"Plural-Forms: nplurals=2; plural=(n!=1);\n"
"X-Generator: KBabel 1.9\n"
```

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

Variable Name	Description
`Project-Id-Version`	Application name and version
`Report-Msgid-Bugs-To`	Mailing list or contact person for reporting errors in translation units.
`POT-Creation-Date`	Date POT file was generated. Automatically filled in by Gettext
`PO-Revision-Date`	Time stamp when PO file was last edited by a translator
`Last-Translator`	Contact information for last translator editing the file.
`Language-Team`	Name of language team that translated this file

Variable Name	Description
MIME-Version	MIME version used for specifying Content-Type
Content-Type	MIME content type and character set for this file
Content-Transfer-Encoding	MIME transfer encoding
Plural-Forms	Number of plural forms in target language, and c-expression for evaluating which plural form to use for a parameter.

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

2.4. Translation Units

```
# 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:

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

is exactly the same as:

```
msgid "My name is %s. \nWhat is your name?"
```

2.4.2. Translator Comments

```
# 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

```
#. 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:

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

This would become:

```
#. 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

```
#: 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

Flag Name	Description
`fuzzy`	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.
`no-wrap`	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.
`X``-format`, where *`X`* is any of the following: • `awk` • `c` • `csharp` • `elips` • `gcc-internal` • `java` • `librep` • `lisp` • `objc` • `object-pascal` • `perl` • `perl-brace` • `php` • `python` • `qt` • `scheme` • `sh` • `smalltalk` • `tcl` • `ycp`	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.
`no-`*`X`*`-format`, where *`X`* is any of the items in the list above.	Indicates that Gettext is to skip the format check for this translation unit. This flag has to be set by developers in the source code.

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:

```
/* 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:

```
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);\n"
```

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>=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` 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:

```
# 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 *domain*, but the PO format also allows for representing multiple domains in a single PO file. This is done by adding the `domain` keyword followed by the domain name, as in the following example:

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

The above example would produce two MO files, `domain_1.mo` and `domain_2.mo`. If no domain is specified, translation units belong to the default domain `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:

```
msgid "HELLO_WORLD"
msgstr "Hello World!"
```

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>
 <target>Hei verden!</target>
</trans-unit>
```

would be back-converted to PO as:

```
msgid "HELLO_WORLD"
msgstr "Hei Verden!"
```

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:

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

would be represented in PO as:

```
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- First Name\n\t- Last Name\n</source>
```

which could be further enhanced by encapsulating escape characters in XLIFF `<ph>` or `<x/>` elements:

```
<source>Please Enter the following Data:<x id='1' ctype='lb'/>\
<x id='2' ctype='x-ht'/>- First Name<x id='3' ctype='lb'/>\
<x id='4' ctype='x-ht'/>- First Name<x id='5' ctype='lb'/>\
</source>
```

Or, the filters could replace escape sequences with the intended characters:

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

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>` or `<x/>` 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

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

[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 in later versions of Gettext handled similar to '\b', '\f' and '\r' escape sequences.

Although most of the XLIFF inline tags are represented in the TMX standard, the <x/> tag is not. TMX is a standard to exchange Translation Memory (TM) data created by Computer Aided Translation (CAT) and localization tools. If you plan to store or deliver XLIFF text content using TMX, you may wish to use the <ph> approach for encapsulating escape sequences or you will need to represent <x/> tags in some alternate way in TMX.

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

```
<?xml version="1.0" ?>
<xliff version="1.2" xmlns="urn:oasis:names:tc:xliff:document:1.">
 <file original="filename.po" source-language="en-US" datatype="po">
 <body>

 <trans-unit>
 ... PO header for default domain...
</trans-unit>
... translation units for default domain...
<group restype="x-gettext-domain" resname="domain-name">
... header and translation units for domain 'domain-name'...
</group>
</body>
</file>
</xliff>
```

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

For example:

```
# French Translation for MyApplication.
# Copyright (C) 2005 John Developer
# This file is distributed under the same license as the MyApp package.
# John Developer <john@example.com>, 2005.
# Joe Translator <joe@example.com>, 2005.
#, fuzzy
msgid ""
msgstr ""
"Project-Id-Version: MyApp 1.0\n"
"Report-Msgid-Bugs-To: MyApp List <myapp-list@example.com>\n"
"POT-Creation-Date: 2005-04-27 13:15+0900\n"
"PO-Revision-Date: 2005-04-27 13:45+0900\n"
"Last-Translator: Joe Translator <joe@example.com>\n"
"Language-Team: French Team <fr-list@example.com>\n"
"MIME-Version: 1.0\n"
"Content-Type: text/plain; charset=UTF-8\n"
"Content-Transfer-Encoding: 8bit\n"
"Plural-Forms: nplurals=2; plural=(n!=1);\n"
"X-Generator: KBabel 1.9\n"
```

would be mapped to:

```
<trans-unit id="1" restype="x-gettext-domain-header" approved="no"
xml:space="preserve">
 <source>Project-Id-Version: MyApp 1.0
Report-Msgid-Bugs-To: MyApp List <myapp-list@example.com>
POT-Creation-Date: 2005-04-27 13:15+0900
PO-Revision-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>
 <target>Project-Id-Version: MyApp 1.0
Report-Msgid-Bugs-To: MyApp List <myapp-list@example.com>
POT-Creation-Date: 2005-04-27 13:15+0900
PO-Revision-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
</target>
 <note from="po-translator">Copyright (C) 2005 John Developer
This file is distributed under the same license as the MyApp package.
John Developer <john@example.com>, 2005.
Joe Translator <joe@example.com>, 2005.</note>
</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:

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

would be mapped to:

```
<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:

```
msgid "%d file deleted"
```

```
msgid_plural "%d files deleted"
msgstr[0] "%d fil slettet"
msgstr[1] "%d filer slettet"
```

would be mapped to:

```
<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.

For example:

```
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:

```
<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:

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

would be mapped to:

```
<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”.

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:

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

could be mapped to:

```
<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 `context-type` attribute set to `x-po-transcomment`. For example:

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

could be mapped to:

```
<trans-unit id="1">
 <source>hello world</source>
 <context-group name="po-entry" purpose="information">
 <context 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:

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

could be mapped to:

```
<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 `context-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:

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

could be mapped to:

```
<trans-unit id="1">
 <source>hello world</source>
 <context-group name="po-entry" purpose="information">
 <context 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:

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

would be mapped to:

```
<trans-unit id="1">
 <source>hello world</source>
 <context-group name="po-reference" purpose="location">
 <context context-type="sourcefile">example.c</context>
 <context context-type="linenumber">34</context>
 </context-group>
 <context-group name="po-reference" purpose="location">
 <context context-type="sourcefile">otherfile.c</context>
 <context context-type="linenumber">233</context>
 </context-group>
</trans-unit>
```

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:

```
#, fuzzy
msgid "Hello world"
msgstr ""
msgid "Hello world!"
msgstr "Hei verden!"
```

should be mapped to:

```
<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:

```
msgid "Hello world"
```

```
msgstr ""
#, fuzzy
msgid "Hello world!"
msgstr "Hei verden!"
```

should be mapped to:

```
<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:

```
<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:

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

in favour of word-wrapping similar to this:

```
msgid "As prompted on the following screen, please enter "
"the following details:\n"
" - First Name\n"
" - Last Name\n"
msgstr "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>` or `<x/>` elements with the `c-type` attribute set to the value mapping to the format flag (see the table below). For example:

```
#, c-format
msgid "Hello %s, your score is %d."
msgstr "Hei %s, du har %d poeng."
```

Here the parameters `%s` and `%d` can be extracted:

```
<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

Flag Name	`c-type` value
`awk-format`	`x-awk-param`
`c-format`	`x-c-param`
`csharp-format`	`x-csharp-param`
`elisp-format`	`x-elisp-param`
`gcc-internal-format`	`x-gcc-internal-param`
`java-format`	`x-java-param`
`librep-format`	`x-librep-param`
`lisp-format`	`x-lisp-param`
`objc-format`	`x-objc-param`

Flag Name	c-type value
object-pascal-format	x-object-pascal-param
perl-format	x-perl-param
perl-brace-format	x-perl-brace-param
php-format	x-php-param
python-format	x-python-param
qt-format	x-qt-param
scheme-format	x-scheme-param
sh-format	x-sh-param
smalltalk-format	x-smalltalk-param
tcl-format	x-tcl-param
ycp-format	x-ycp-param

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

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

If we here in the target language wanted to write:

```
Score: %d. Name: %s
```

we would have to specify the position of the parameters:

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

Most XLIFF editors does not provide a way for translators to edit the content of <ph> elements, and with <x/> elements the content is fully abstracted, meaning this logic would have to be implemented in the filters.

For example, in the following PO fragment:

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

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

```
<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:

```
<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:

```
#, c-format
msgid "Hello %s, your score is %d."
msgstr "Score is %2$d for %1$s"
```

Take note that the 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:

```
<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
- Doug Domeny
- Karl Eichwalder
- Asgeir Frimannsson
- Tim Foster
- David Fraser
- Paul Gampe
- Bruno Haible
- James M. Hogan
- Rodolfo M. Raya
- Peter Reynolds
- Yves Savourel
- Bryan Schnabel
- Tony Jewtushenko

B. Examples of converted PO files

We have provided the following two examples of PO files converted to XLIFF:

- A simple PO Template file [example.pot] converted to XLIFF [example.xlf].
- A partially translated PO file [example_nb_NO.po] converted to XLIFF [example_nb-NO.xlf].

References

[GNU Gettext] *The GNU Gettext Manual* http://www.gnu.org/software/gettext/manual

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

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

[XML 1.0] *Extensible Markup Language (XML) 1.0 (Third Edition)*. W3C (World Wide Web Consortium), Feb 2004

[XLIFF 1.1] *XLIFF 1.1 Specification*. OASIS XLIFF Technical Committee, October 2003.

[XLIFF 1.2] *XLIFF 1.2 Specification*. OASIS XLIFF Technical Committee, May 2006.

[XLIFF Tools] *The XLIFF Tools Project* http://xliff-tools.freedesktop.org/