



XLIFF 1.2 Representation Guide for Java Resource Bundles

Committee Draft, 16 May 2006

This version:

<http://www.oasis-open.org/committees/xliff/documents/cd-xliff-profile-java-1.2-20060516.htm>

Latest version:

<http://www.oasis-open.org/committees/xliff/documents/cd-xliff-profile-java-1.2-20060516.htm>

Previous version:

<not applicable yet>

Editors:

Tony Jewtushenko<tony.jewtushenko@productinnovator.com>

Rodolfo M. Raya<rmraya@heartsome.net>

Notices

OASIS takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Information on OASIS's procedures with respect to rights in OASIS specifications can be found at the OASIS website. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementors or users of this specification, can be obtained from the OASIS Executive Director.

OASIS invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights which may cover technology that may be required to implement this specification. Please address the information to the OASIS Executive Director.

Copyright © OASIS Open 2005. All Rights Reserved.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this paragraph are included on all such copies and derivative works. However, this document itself may not be modified in any way, such as by removing the copyright notice or references to OASIS, except as needed for the purpose of developing OASIS specifications, in which case the procedures for copyrights defined in the OASIS Intellectual Property Rights document must be followed, or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by OASIS or its successors or assigns.

This document and the information contained herein is provided on an "AS IS" basis and OASIS DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Abstract

This document describes how Java Resource Bundles ([ListResourceBundle](#) and [PropertyResourceBundle](#)) should be coded when extracted to an XLIFF document.

Status

This document was last revised or approved by the XLIFF TC on the above date. The level of approval is also listed above. Check the 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-java-1.2-errata.htm

Table of Contents

1. [Introduction](#)
 - 1.1. [Purpose](#)
 - 1.2. [Transitional and Strict](#)
2. [General Considerations](#)
 - 2.1. [Java Resource Types](#)
 - 2.2. [ListResourceBundle](#)
 - 2.3. [PropertyResourceBundle](#)
 - 2.4. [Extraction Techniques](#)
 - 2.5. [Order Of Extraction](#)
 - 2.6. [Key Identifier](#)
 - 2.7. [Preserving Message Replaceables in Value](#)
 - 2.8. [Comments](#)
 - 2.9. [Sample Properties File represented as XLIFF](#)
 - 2.10. [Sample List Resources Java Class represented as XLIFF](#)

Appendices

A. [Contributions](#)

B. [References](#)

1. Introduction

As different tools may provide different filters to extract the content of Java Resource Bundles, 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 data contained in Java Resource Bundles as XLIFF content. It offers a collection of recommended mapping of Java Resource Bundles 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. General Considerations

This section discusses the general considerations to take in account when extracting Java Resource Bundle data.

2.1. Java Resource Types

Java's architecture supports the localization of application resources via the `java.util.ResourceBundle` abstract class. This class has two subclasses: `PropertyResourceBundle` and `ListResourceBundle`. A `ResourceBundle` contains key/value pairs, where each key uniquely identifies locale-specific objects in the bundle. Non-string resources, binary objects, must be stored in `ListResourceBundle`, since the `PropertyResourceBundle` can only contain `String` objects. String values can contain `MessageFormat` replaceables, which are delimited within `{ }`.

2.2. ListResourceBundle

A `ListResourceBundle` is a list of resources stored in a .class file. Compiled resources are stored in binary format, which means `ListResourceBundle` can contain localized resources of any data type.

Additional locales are created by copying the base .java file to a locale specific .java file identified by adding a locale suffix to its filename. After translation has been completed, the localized .java file is compiled into a .class file.

[Listing 1](#) shows a sample `ListResourceBundle` class named "DiskResources.java" and a version for "es" locale.

```
import java.awt.Rectangle;
import java.util.ListResourceBundle;

public class DiskResources extends ListResourceBundle {
    public Object[][] getContents() {
        return contents;
    }
    static final Object[][] contents = {
        {"key1", "Cabinet {0} contains {1} folders."},
        {"key2", "Folder {0} contains {1,choice,0#no files|1#one file|
1<{1,number,integer} files}."},
        {"key3", "Folder \"{0}\" is empty."},
        {"key4", "File \"My Stuff\" deleted."},
        {"key5", "Added {0,number} files."},
        {"key6", new Rectangle(10,25,100,150)},
        {"key7", "No files were removed while processing " +
                "current folder."}
    };
}

-----

import java.awt.Rectangle;
import java.util.ListResourceBundle;

public class DiskResources_es extends ListResourceBundle {
    public Object[][] getContents() {
        return contents;
    }
    static final Object[][] contents = {
        {"key1", "El gabinete {0} contiene {1} carpetas."},
        {"key2", "La carpeta {0} contiene {1,choice,0#cero archivos|1#un archivo|
1<{1,number,integer} archivos}."},
        {"key3", "La carpeta \"{0}\" está vacía."},
        {"key4", "Archivo \"Mis cosas\" eliminado."},
        {"key5", "{0,number} archivos añadidos."},
        {"key6", new Rectangle(10,25,100,150)},
        {"key7", "Ningún archivo fue eliminado al procesar " +
                "la carpeta actual."}
    };
}
```

```
};  
}
```

Listing 1 - Sample List Resource file

2.3. PropertyResourceBundle

[PropertyResourceBundle](#) is a concrete implementation of [ResourceBundle](#) class that stores translatable [String](#) resources in plain text files via name-value pairs using "name=value" syntax.

The resources of a [PropertyResourceBundle](#) are stored in files with the extension ".properties". Properties files strings are fetched at run-time, they are not compiled. The keys are case-sensitive and should be unique within the .properties file.

[Listing 2](#) below is a sample [PropertyResourceBundle](#) .properties file with four localizable strings.

```
# Copyright information  
key1=Copyright \u00A9 2006 FARO Inc.  
key2=Box 12 is {0,number} inches high.  
key3=Box '{0}' is blue.  
key4=Boxes are built in three sizes: small, \  
medium and large.
```

Listing 2 - Sample .properties file

2.4.Extraction Techniques

Resource Bundles are Java code and not XML. Therefore, XSL transformation standards cannot directly be utilized to convert the resource bundles into XLIFF. XSLT could be used to transform the translated XLIFF back to the native Java, or to manipulate an intermediate XML or initial XLIFF file in preparation for manual or automated Computer Aided Translation (CAT), but software best practices favour using the same technology for both extraction and recomposition.

The class [java.util.Properties](#) provides methods for converting .properties files to XML format. [Listing 3](#) shows the XML version of the .properties file from [Listing 2](#).

```
<?xml version="1.0" encoding="UTF-8"?>  
<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">  
<properties>  
<comment>Listing 2</comment>  
<entry key="key4">Boxes are built in three sizes: small, medium and large.</entry>  
<entry key="key3">Box '{0}' is blue.</entry>  
<entry key="key2">Box 12 is {0,number} inches high.</entry>  
<entry key="key1">Copyright © 2006 FARO Inc. </entry>  
</properties>
```

Listing 3 - Java resources in XML format

XML files generated using class [Properties](#) can be converted to XLIFF using XSL transformations or specialized XML filters. Processing of Java resources stored in XML format is beyond the scope of this document.

2.4.1. Using Filters

It is necessary to extract the Java Resource Bundles- List and Property - to XLIFF by developing a

custom filter application. Such tools can be written using a variety of programming and scripting languages such as Perl, Python, C, C++, C#, Java, etc..

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

The following guidelines should be adhered to:

- Each .properties file is represented in a separate `<file>` element in an XLIFF document
- The `datatype` attribute of a `<file>` element is set to `javapropertyresourcebundle` or `javalistresourcebundle`, depending on the type of resource
- The `<body>` element contains a separate `<trans-unit>` element for each key-value pair
- The `translate` attribute of `<trans-unit>` element is set to `no` when the value string is a null string
- The value string becomes the content of the child `<source>` element

Characters that require a special representation in a [ResourceBundle](#) should be parsed by the XLIFF filter and represented appropriately in the XLIFF file. The following cases should be contemplated when designing a filter:

- Single quotes, double quotes, backslashes and other special characters need to be prefixed with a `\` character in [ListResourceBundle](#) classes. Filters should use unescaped versions of these characters in the XLIFF file.
- Properties files are written using ISO 8859-1 character set. All characters not supported by this character set are encoded using the `\uXXXX` Java notation for Unicode. Filters should replace encoded characters with their appropriate Unicode versions in the XLIFF file.
- Long strings can be concatenated using `'+'` operator in [ListResourceBundle](#) classes (see the last entry in [Listing 1](#)). Filters should automatically merge concatenated strings.
- A `\` character at the end of a line in a .properties file indicates that the text to translate continues in the following line (see the last entry in [Listing 2](#)). Filters should automatically merge split lines.

2.5. Order of Extraction

The flow of the extracted data in the XLIFF document should be in the same order as the flow of data in the original [ResourceBundle](#). The extraction order should reflect the order of the data in the source document, and the author is responsible to group logical parts of the text together as much as possible.

2.6. Key Identifier

The identifier used for matching, leveraging, and other ID-related functions is stored in the `resname` attribute. The key of the resource becomes the `resname` attribute value of the `<trans-unit>` element.

The required `id` attribute of the XLIFF `<trans-unit>` element is an identifier allowing extraction tools to merge back the data. The value of the `id` attribute is serially assigned, according the retrieval order, to maintain consistency and uniqueness.

2.7. Preserving Message Replaceables in Value

Variable data is represented in Java strings using special descriptors surrounded by `{` and `}`. Those

descriptors, called "replaceables", are processed by the `MessageFormat` class at run-time to display variable data in the appropriate format for the selected locale. `MessageFormat` replaceables should be enclosed in `<ph>` elements by the filter that generates the XLIFF representation. If Skeleton files are used, a replaceable can optionally be represented with an `<x/>` tag instead of a `<ph>` element.

Replaceables of type `choice` include a parameter with two or more translatable strings. The string to display is selected at run-time according to a given condition. Replaceables of type `choice` should be encapsulated in `<ph>` elements, enclosing translatable strings in `<sub>` elements. Filter designers can parse the parameter with strings and hide the logic from the translator by using more than one `<sub>` element or expose the whole parameter, giving the translator the possibility to change the selection criteria according to the grammar of the target language. [Listing 4](#) shows a replaceable of type `choice` and its two possible representations.

```
key2=Folder {0} contains {1,choice,0#no files|1#one file|1<{2,number,integer}files}.
-----
<source>Folder <ph id="1">{0}</ph> contains
  <ph id="2">{1,choice,0#<sub>no files</sub>|1#<sub>one file</sub>|
1&lt;{2,number,integer}<sub>files</sub></ph>.
</source>

<source>Folder <ph id="1">{0}</ph> contains
  <ph id="2">{1,choice,<sub>0#no files|1#one file|
1&lt;{2,number,integer}files</sub></ph>.
</source>
```

Listing 4 - XLIFF representation of choice-type replaceables

Best localization practices recommend using class `java.text.MessageFormat` to produce concatenated messages in language-neutral way. However, it is possible to concatenate translatable strings and variables using a '+' operator. Filters should encapsulate variables and operators using either `<ph>` or `<x/>` tags as shown in [Listing 5](#).

```
String title = getBookTitle();
int bookNumber = 23801;

public Object[][] getContents() {
    return contents;
}
static final Object[][] contents = {
    {"key1", "Title: " + title + " - Number: " + bookNumber},
};
-----
<source>Title: <ph id="1"> + title + </ph> - Number: <ph id="2"> +
bookNumber</ph></source>
```

Listing 5 - Strings and variables concatenated.

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 `<ph>` elements for encapsulating replaceables. Otherwise, you will need to represent `<x/>` tags in some alternate way in TMX.

2.8. Comments

Lines that start with a # character are comments in .properties files. These lines do not contain translatable text. Filter designers may optionally concatenate consecutive comment lines and include them as context information in one <note> element in the first <trans-unit> generated after extracting the comments.

Comments in classes derived from `ListResourceBundle` are subject to Java coding rules. If an XLIFF filter tool finds comments inside a run of localizable text in a class that extends `ListResourceBundle`, the comment should be preserved by being treated as inline code and enclosed in a <ph> element. If Skeleton files are used, the comment can optionally be represented with an <x/> tag instead of a <ph> element.

In `ListResourceBundle` resources, if an XLIFF filter finds comments following a key value pair, those comments can be associated with the corresponding <trans-unit> element using a <note> element.

2.9 Sample Properties File represented as XLIFF:

[Listing 6](#) below contains the XLIFF representation of the .properties file shown in [Listing 2](#).

```
<?xml version="1.0" encoding="UTF-8"?>
<xliff version="1.2"
  xmlns="urn:oasis:names:tc:xliff:document:1.2"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="urn:oasis:names:tc:xliff:document:1.2 xliff-core-1.2-
strict.xsd">
<file original="sample.properties"
  source-language="en"
  datatype="javapropertyresourcebundle">
<body>
  <trans-unit id="0" resname="key1">
    <source xml:lang="en">Copyright © 2006 FARO Inc. </source>
    <note>Copyright information</note>
  </trans-unit>
  <trans-unit id="1" resname="key2">
    <source xml:lang="en">Box 12 is <ph id="1">{0,number}</ph> inches
high.</source>
  </trans-unit>
  <trans-unit id="2" resname="key3">
    <source xml:lang="en">Box ' ' <ph id="1">{0}</ph>' ' is blue.</source>
  </trans-unit>
  <trans-unit id="3" resname="key4">
    <source xml:lang="en">Boxes are built in three sizes: small, medium and
large.</source>
  </trans-unit>
</body>
</file>
</xliff>
```

Listing 6 - XLIFF representation of a .properties file

2.9 Sample List Resources Java Class represented as XLIFF:

[Listing 7](#) shows the XLIFF document with localizable data extracted from the class "DiskResources.java" included in [Listing 1](#).

```

<?xml version="1.0" encoding="UTF-8"?>
<xliff version="1.2"
  xmlns="urn:oasis:names:tc:xliff:document:1.2"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="urn:oasis:names:tc:xliff:document:1.2 xliff-core-1.2-
strict.xsd">
<file original="DiskResources.java"
  source-language="en"
  datatype="javalistresourcebundle">
<body>
  <trans-unit id="0" resname="key1">
    <source>Cabinet <ph id="1">{0}</ph> contains <ph id="2">{1}</ph>
folders.</source>
  </trans-unit>
  <trans-unit id="1" resname="key2">
    <source>Folder <ph id="1">{0}</ph> contains
    <ph id="2">{1,choice,<sub>0#no files|1#one file|1&lt;{0,number,integer}
files}</sub>
    </ph>.</source>
  </trans-unit>
  <trans-unit id="2" resname="key3">
    <source>Folder '<ph id="1">{0}</ph>' is empty.</source>
  </trans-unit>
  <trans-unit id="3" resname="key4">
    <source>File "My Stuff" deleted.</source>
  </trans-unit>
  <trans-unit id="4" resname="key5">
    <source>Added <ph id="1">{0,number}</ph> files.</source>
  </trans-unit>
  <trans-unit id="5" resname="key7">
    <source>No files were removed while processing current folder.</source>
  </trans-unit>
</body>
</file>
</xliff>

```

Listing 7 - XLIFF representation of a List Resource Bundle

A. Contributions

The following people have contributed to this document:

- Eiju Akahane, IBM
- Doug Domeny, Ektron
- Tony Jewtushenko, Product Innovator Ltd.
- Milan Karásek, Moravia-IT
- Christian Lieske, SAP AG
- Matthew Lovatt, Oracle Corporation
- Magnus Martikainen, SDL International
- David Pooley, SDL International
- Rodolfo M. Raya, Heartsome Holdings Pte Ltd.
- John Reid, Novell
- Peter Reynolds, Idiom Technologies, Inc.
- Florian Sachse, Pass Engineering GmbH
- Bryan Schnabel
- David Walters, IBM
- Shigemichi Yazawa
- Andrzej Zydron, XML-Intl

B. References

[ISO]

[International Organization for Standardization](#) Web site.

[Java]

[Core Java Internationalization](#) Web Site

[Documentation of ResourceBundle Class](#)

[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.

[Unicode]

[Unicode Consortium](#) Web site.