XLIFF Version 2.0

Committee Specification Draft 01

16 April 2013

Specification URIs

This version:

http://docs.oasis-open.org/xliff/xliff-core/v2.0/csd01/xliff-core-v2.0-csd01.html (Authoritative)
http://docs.oasis-open.org/xliff/xliff-core/v2.0/csd01/xliff-core-v2.0-csd01.pdf
http://docs.oasis-open.org/xliff/xliff-core/v2.0/csd01/xliff-core-v2.0-csd01.xml

Previous version:

N/A

Latest version:

http://docs.oasis-open.org/xliff/xliff-core/v2.0/xliff-core-v2.0.html (Authoritative)
http://docs.oasis-open.org/xliff/xliff-core/v2.0/xliff-core-v2.0.pdf
http://docs.oasis-open.org/xliff/xliff-core/v2.0/xliff-core-v2.0.xml

Technical Committee:

OASIS XML Localisation Interchange File Format (XLIFF) TC

Chair:

Bryan Schnabel (bryan.s.schnabel@tektronix.com), Individual

Editors:

Tom Comerford (tom@supratext.com), Individual
David Filip (davidf@ul.ie), Localisation Research Centre
Rodolfo M. Raya (rmraya@maxprograms.com), Maxprograms
Yves Savourel (ysavourel@enlaso.com), ENLASO Corporation

Additional artifacts:

This prose specification is one component of a Work Product that also includes:

- XML schemas accessible from http://docs.oasis-open.org/xliff/xliff-core/v2.0/csd01/schemas/

Related Work:

This specification replaces or supersedes:

- *XLIFF Version 1.2.* 1 February 2008. OASIS Standard. http://docs.oasis-open.org/xliff/v1.2/os/xliff-core.html

Declared XML Namespaces:

- urn:oasis:names:tc:xliff:document:2.0
- urn:oasis:names:tc:xliff:matches:2.0
- urn:oasis:names:tc:xliff:glossary:2.0
- urn:oasis:names:tc:xliff:fs:2.0
- urn:oasis:names:tc:xliff:metadata:2.0

- urn:oasis:names:tc:xliff:resourcedata:2.0
- urn:oasis:names:tc:xliff:changetracking:2.0
- urn:oasis:names:tc:xliff:sizerestriction:2.0
- urn:oasis:names:tc:xliff:validation:2.0

Abstract:

This document defines version 2.0 of the XML Localisation Interchange File Format (XLIFF). The purpose of this vocabulary is to store localizable data and carry it from one step of the localization process to the other, while allowing interoperability between tools.

Status:

This document was last revised or approved by the OASIS XML Localisation Interchange File Format (XLIFF) TC on the above date. The level of approval is also listed above. Check the “Latest version” location noted above for possible later revisions of this document.

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

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

Citation format:

When referencing this specification the following citation format should be used:

[XLIFF v2.0]

XLIFF Version 2.0. 16 April 2013. OASIS Committee Specification Draft 01. http://docs.oasis-open.org/xliff/xliff-core/v2.0/csd01/xliff-core-v2.0-csd01.html.

Notices

All capitalized terms in the following text have the meanings assigned to them in the OASIS Intellectual Property Rights Policy (the "OASIS IPR Policy"). The full Policy may be found at the OASIS website.

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 section are included on all such copies and derivative works. However, this document itself may not be modified in any way, including by removing the copyright notice or references to OASIS, except as needed for the purpose of developing any document or deliverable produced by an OASIS Technical Committee (in which case the rules applicable to copyrights, as set forth in the OASIS IPR Policy, 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 OWNERSHIP RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

OASIS requests that any OASIS Party or any other party that believes it has patent claims that would necessarily be infringed by implementations of this OASIS Committee Specification or OASIS Standard, to notify OASIS TC Administrator and provide an indication of its willingness to grant patent licenses to such patent claims in a manner consistent with the IPR Mode of the OASIS Technical Committee that produced this specification.

OASIS invites any party to contact the OASIS TC Administrator if it is aware of a claim of ownership of any patent claims that would necessarily be infringed by implementations of this specification by a patent holder that is not willing to provide a license to such patent claims in a manner consistent with the IPR Mode of the OASIS Technical Committee that produced this specification. OASIS may include such claims on its website, but disclaims any obligation to do so.

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' procedures with respect to rights in any document or deliverable produced by an OASIS Technical Committee can be found on 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 implementers or users of this OASIS Committee Specification or OASIS Standard, can be obtained from the OASIS TC Administrator. OASIS makes no representation that any information or list of intellectual property rights will at any time be complete, or that any claims in such list are, in fact, Essential Claims.

The name "OASIS" is a trademark of OASIS, the owner and developer of this specification, and should be used only to refer to the organization and its official outputs. OASIS welcomes reference to, and implementation and use of, specifications, while reserving the right to enforce its marks against misleading uses. Please see http://www.oasis-open.org/policies-guidelines/trademark for above guidance.

Table of Contents

Appendixes

1 Introduction

XLIFF is the *XML Localisation Interchange File Format* designed by a group of software providers, localization service providers, and localization tools providers. It is intended to give any software provider a single interchange file format that can be understood by any localization provider.

All text is normative unless otherwise labeled.

1.1 Terminology

1.1.1 Key words

The key words *must*, *must not*, *required*, *shall*, *shall not*, *should*, *should not*, *recommended*, *may*, and *optional* are to be interpreted as described in [**RFC 2119**]. Note that for reasons of style, these words are not capitalized in this document.

1.1.2 Definitions

XLIFF Document
: XML document that declares the namespace `"urn:oasis:names:tc:xliff:document:2.0"` as its main namespace, has `<xliff>` as root element and complies with the XML Schemas that are part of this specification.

Translation
: A rendering of the meaning of source text, expressed in the target language.

1.1.3 Key concepts

XLIFF Core
: The core of XLIFF 2.0 consists of the minimum set of XML elements and attributes required to (a) prepare a document that contains text extracted from one or more files for localization, (b) allow it to be completed with the translation of the extracted text, and (c) allow the generation of translated versions of the original document.

  The XML namespace that corresponds to the core subset of XLIFF 2.0 is `"urn:oasis:names:tc:xliff:document:2.0"`.

XLIFF Module
: A module is an optional set of XML elements and attributes that stores information about a process applied to an XLIFF document and the data incorporated into the document as result of that process.

  Each official module defined for XLIFF 2.0 has its grammar defined in an independent XML Schema with a separate namespace.

1.2 Normative References

[**BCP 47**] M. Davis, *Tags for Identifying Languages*, *http://tools.ietf.org/html/bcp47* IETF (Internet Engineering Task Force).

[**RFC 2119**] S. Bradner, *Key words for use in RFCs to Indicate Requirement Levels*, *http://www.ietf.org/rfc/rfc2119.txt* IETF (Internet Engineering Task Force) RFC 2119, March 1997.

[**UAX 15**] M. Davis, K. Whistler, *UNICODE NORMALIZATION FORMS*, *http://www.unicode.org/reports/tr15/* Unicode Normalization Forms.

1.3 Non-Normative References

[**XML I18N BP**] *Best Practices for XML Internationalization*, 13 February 2008, *http://www.w3.org/TR/xml-i18n-bp/* W3C Working Group.

[**LDML**] *Unicode Locale Data Markup Language* *http://unicode.org/reports/tr35/*

[**SRX**] *Segmentation Rules eXchange* *http://www.gala-global.org/oscarStandards/srx/*

2 Detailed Specifications

XLIFF is a bilingual document format designed for containing text that needs translation, its corresponding translation and auxiliary data that makes the translation proccess possible.

At creation time, an XLIFF file *may* contain only text in source language. Translations expressed in target language *may* be added at a later time.

The root element of an XLIFF document is `<xliff>`. It contains a collection of `<file>` elements. Each `<file>` element contains a set of `<unit>` elements that contain the text to be translated in the `<source>` child of one or more `<segment>` elements. Translations are stored in the `<target>` child of each `<segment>` element.

2.1 General Processing Requirements

- A tool processing a valid XLIFF document that contains XLIFF-defined elements that it cannot handle *must* preserve those elements.
- A tool processing a valid XLIFF document that contains custom elements that it cannot handle *should* preserve those elements.

2.2 Elements

This section contains a description of all elements used in XLIFF 2.0.

2.2.1 Tree Structure

Legend:

1 = one
+ = one or more
? = zero or one
* = zero, one or more

```
<xliff> 1
|
+---<file> +
     |
     +---<skeleton> ?
     |
     +---<gls:glossary> ?
     |
     +---<mda:metadata> ?
     |
     +---<res:resourceData> *
     |
     +---<slr:profile> ?
     |
     +---<val:validation> ?
     |
     +---<any> *
     |
     +--- At least one of (<group>* or <unit>*)
     |
     +---<group> *
```

```
|   |
|   +--- At least one of (<group>* or <unit>*)
|   |
|   +---<mda:metadata> ?
|   |
|   +---<val:validation> ?
|   |
|   +---<any> *
|
+---<unit> *
    |
    +---<segment> +
    |   |
    |   +---<source> 1
    |   |
    |   +---<target> ?
    |   |
    |   +---<notes> ?
    |   |    |
    |   |    +---<note> +
    |   |
    |   +---<mtc:matches> ?
    |   |
    |   +---<mda:metadata> ?
    |   |
    |   +---<ctr:changeTrack> ?
    |   |
    |   +---<val:validation> ?
    |
    +---<ignorable> *
    |   |
    |   +---<source> 1
    |   |
    |   +---<target> ?
    |   |
    |   +---<mda:metadata> ?
    |
    +---<notes> ?
    |   |
    |   +---<note> +
    |
    +---<originalData> ?
    |       |
    |       +---<data> +
    |
    +---<mtc:matches> ?
    |
    +---<gls:glossary> ?
    |
    +---<mda:metadata> ?
    |
    +---<ctr:changeTrack> ?
    |
    +---<val:validation> ?
    |
    +---<any> *
```

2.2.2 Structural Elements

The structural elements used in XLIFF 2.0 are: `<xliff>`, `<file>`, `<skeleton>`, `<group>`, `<unit>`, `<segment>`, `<ignorable>`, `<notes>`, `<note>`, `<originalData>`, `<data>`, `<source>` and `<target>`.

2.2.2.1 xliff

Root element for XLIFF documents.

Contains:

- One or more `<file>` elements

Attributes:

- `version`, required
- `srcLang`, required
- `tgtLang`, optional
- attributes from any namespace, optional

Processing Requirements

- The `tgtLang` attribute must be present when the XLIFF document contains `<target>` elements that are children of `<segment>` or `<ignorable>`.

2.2.2.2 file

Container for localization material extracted from a single document/source.

Contains:

- Zero or one `<skeleton>` element followed by
- Zero or one `<gls:glossary>` element followed by
- Zero or one `<mda:metadata>` elements followed by
- Zero or one `<slr:profiles>` elements followed by
- Zero or one `<slr:data>` elements followed by
- Zero, one or more `<res:resourceData>` elements followed by
- Zero or one `<validation:validation>` elements followed by
- Zero, one or more elements from any namespace followed by
- One or more `<unit>` or `<group>` elements in any order.

Attributes:

- `id`, optional
- `original`, optional
- `srcDir`, optional
- `trgDir`, optional
- `fs:fs`, optional
- `fs:subFs`, optional
- `slr:storageRestriction`, optional
- `slr:sizeRestriction`, optional
- `slr:sizeInfo`, optional
- `slr:sizeInfoRef`, optional
- attributes from any namespace, optional

Processing Requirements

- The value of the optional `id` attribute *must* be unique among all `<file>` children of the enclosing `<xliff>` element.

- The `<file>` element *must not* have a `<skeleton>` child, if and only if the optional `skeleton` attribute is used.

2.2.2.3 skeleton

Container for untranslatable material pertaining to the parent `<file>` element.

Contains:

Either

- Untranslatable text
- XML elements from any namespace

or

- is empty.

Attributes:

- `href`, optional

Processing Requirements

- Tools procesing an XLIFF file that contains `<skeleton>` elements *must* keep those elements unchanged.
- Tools creating an XLIFF file that contains `<skeleton>` elements *must* leave the `<skeleton>` element empty if and only if the attribute `href` is used.

2.2.2.4 group

Provides a way to organize units into a structured hierarchy.

Contains:

- One or more `<unit>` or `<group>` elements in any order followed by
- Zero or one `<mda:metadata>` elements followed by
- Zero or one `<slr:data>` elements followed by
- Zero or one `<validation:validation>` elements followed by
- Zero, one or more elements from any namespace.

Attributes:

- `id`, optional
- `name`, optional
- `srcDir`, optional
- `trgDir`, optional
- `fs:fs`, optional
- `fs:subFs`, optional
- `slr:storageRestriction`, optional
- `slr:sizeRestriction`, optional
- `slr:sizeInfo`, optional
- `slr:sizeInfoRef`, optional
- attributes from any namespace, optional

2.2.2.5 unit

Extracted translatable text.

Contains:

- One or more or <ignorable> elements in any order followed by
- Zero or one <notes> element followed by
- Zero or one <originalData> element followed by
- Zero or one <mtc:matches> element followed by
- Zero or one <gls:glossary> element followed by
- Zero or one <mda:metadata> elements followed by
- Zero or one <slr:data> elements followed by
- Zero or one <validation:validation> elements followed by
- Zero, one or more elements from any namespace.

Attributes:

- id, required
- name, optional
- translate, optional
- srcDir, optional
- trgDir, optional
- fs:fs, optional
- fs:subFs, optional
- slr:storageRestriction, optional
- slr:sizeRestriction, optional
- slr:sizeInfo, optional
- slr:sizeInfoRef, optional
- attributes from any namespace, optional

Processing Requirements

- A <unit> must contain at least one element.
- The value of the id attribute must be unique among all <unit> children of the enclosing <file> element.
- A <unit> element is considered to be translated when all its children with translate attribute not set to no have the approved attribute set to yes.

2.2.2.6 segment

Minimum portion of translatable text.

Contains:

- One <source> element followed by
- Zero or one <target> element followed by
- Zero or one <notes> elements followed by
- Zero or one <mtc:matches> element followed by
- Zero or one <ctr:changeTrack> elements followed by
- Zero or one <validation:validation> elements followed by
- Zero or one <mda:metadata> elements.

Attributes:

- id, optional
- translate, optional
- approved, optional
- state, optional
- subState, optional
- attributes from any namespace, optional
- fs:fs, optional
- fs:subFs, optional

Processing Requirements

- The value of the optional `id` attribute *must* be unique among all `<segment>` and `<ignorable>` children of the enclosing `<unit>` element.

2.2.2.7 ignorable

Information that needs to be preserved but does not need to be modified/translated.

Contains:

- One `<source>` element followed by
- Zero or one `<target>` element followed by
- Zero or one `<mda:metadata>` elements.

Attributes:

- `id`, optional
- attributes from any namespace, optional
- `fs:fs`, optional
- `fs:subFs`, optional

Processing Requirements

- The value of the optional `id` attribute must be unique among all `<segment>` and `<ignorable>` children of the enclosing `<unit>` element.

2.2.2.8 notes

Collection of comments.

Contains:

- One or more `<note>` elements

Attributes:

- `fs:fs`, optional
- `fs:subFs`, optional

2.2.2.9 note

A comment that contains information about `<source>`, `<target>`, `<segment>` or `<unit>` elements.

Contains:

- Text

Attributes:

- `id`, optional
- `appliestTo`, optional
- `category`, optional
- `priority`, optional
- attributes from any namespace, optional
- `fs:fs`, optional
- `fs:subFs`, optional

2.2.2.10 originalData

Unit-level collection of original data for the inline codes.

Contains:

- One or more <data> elements

Attributes:

- fs:fs, optional
- fs:subFs, optional

2.2.2.11 data

Storage for the original data of an inline code.

Contains:

- Untranslatable text
- Zero, one or more <cp> elements.

Untranslatable text and <cp> elements may appear in any order.

Attributes:

- id, required
- dir, optional
- fs:fs, optional
- fs:subFs, optional

Processing Requirements

- The value of the id attribute must be unique among all <data> children of the enclosing <originalData> element.

2.2.2.12 source

Portion of text to be translated.

Contains:

- Text
- Zero, one or more <cp> elements
- Zero, one or more <ph> elements
- Zero, one or more <pc> elements
- Zero, one or more <sc> elements
- Zero, one or more <ec> elements
- Zero, one or more <mrk> elements
- Zero, one or more <sm> elements
- Zero, one or more elements

Text and inline elements may appear in any order.

Attributes:

- xml:lang, optional
- xml:space, optional
- dir, optional
- attributes from any namespace, optional
- fs:fs, optional
- fs:subFs, optional

Processing Requirements

- When a <source> element is a child of or <ignorable> and the optional xml:lang attribute is present, its value must be equal to the value of the srcLang attribute of the enclosing <xliff> element.

2.2.2.13 target

The translation of the sibling <source> element.

Contains:

- Text
- Zero, one or more <cp> elements
- Zero, one or more <ph> elements
- Zero, one or more <pc> elements
- Zero, one or more <sc> elements
- Zero, one or more <ec> elements
- Zero, one or more <mrk> elements
- Zero, one or more <sm> elements
- Zero, one or more elements

Text and inline elements may appear in any order.

Attributes:

- xml:lang, optional
- xml:space, optional
- dir, optional
- fs:fs, optional
- fs:subFs, optional
- attributes from any namespace, optional

Processing Requirements

- When a <target> element is a child of or <ignorable> and the optional xml:lang attribute is present, its value must be equal to the value of the tgtLang attribute of the enclosing <xliff> element.
- When a <target> child is added to a element, the value of its xml:space attribute *must* be set to `preserve` if the xml:space attribute of the sibling <source> element is set to `preserve`.

2.2.3 Inline Elements

The inline elements at the <source> or <target> level are: <cp>, <ph>, <pc>, <sc>, <ec>, <mrk>, <sm> and .

The elements at the <unit> level directly related to inline elements are: <originalData> and <data>.

2.2.3.1 cp

Represents a Unicode character that is invalid in XML.

Contains:

This element is always empty.

Parents:

<data>, <mrk>, <source>, <target> and <pc>

Attributes:

- hex, required
- fs:fs, optional
- fs:subFs, optional

Example:

```
<unit id="1">
 <segment>
  <source>Ctrl+C=<cp hex="0003"/></source>
 </segment>
</unit>
```

The example above shows a character U+0003 (Control C) as it must be represented in XLIFF.

Processing Requirements

- Writers *must* encode all invalid XML characters of the content using `<cp>`.
- Writers *must not* encode valid XML characters of the content using `<cp>`.

2.2.3.2 ph

Represents a standalone code of the original format.

Contains:

This element is always empty.

Parents:

`<source>`, `<target>`, `<pc>` and `<mrk>`

Attributes:

- `canCopy`, optional
- `canDelete`, optional
- `canReorder`, optional
- `copyOf`, optional
- `disp`, optional
- `equiv`, optional
- `id`, required.
- `dataRef`, optional
- `subFlows`, optional
- `subType`, optional
- `type`, optional
- `fs:fs`, optional
- `fs:subFs`, optional
- `slr:equivStorage`, optional
- `slr:sizeInfo`, optional
- `slr:sizeInfoRef`, optional

Example:

```
<unit id="1">
 <segment>
  <source>Number of entries: <ph id="1" dataRef="d1" />
<ph id="2" dataRef="d2"/>(These entries are only the ones matching the
current filter settings)</source>
 </segment>
 <originalData>
  <data id="d1">%d</data>
  <data id="d2">&lt;br/></data>
 </originalData>
```

```
</unit>
```

2.2.3.3 pc

Represents a well-formed spanning original code.

Contains:

- Text
- Zero, one or more `<cp>` elements
- Zero, one or more `<ph>` elements
- Zero, one or more `<pc>` elements
- Zero, one or more `<sc>` elements
- Zero, one or more `<ec>` elements
- Zero, one or more `<mrk>` elements
- Zero, one or more `<sm>` elements
- Zero, one or more `<em>` elements

Text and inline elements may appear in any order.

Parents:

`<source>`, `<target>`, `<pc>` and `<mrk>`

Attributes:

- `canCopy`, optional
- `canDelete`, optional
- `canOverlap`, optional
- `canReorder`, optional
- `copyOf`, optional
- `dispEnd`, optional
- `dispStart`, optional
- `equivEnd`, optional
- `equivStart`, optional
- `id`, required
- `dataRefEnd`, optional
- `dataRefStart`, optional
- `subFlowsEnd`, optional
- `subFlowsStart`, optional
- `subType`, optional
- `type`, optional
- `dir`, optional
- `fs:fs`, optional
- `fs:subFs`, optional
- `slr:storageRestriction`, optional
- `slr:sizeRestriction`, optional
- `slr:equivStorage`, optional
- `slr:sizeInfo`, optional
- `slr:sizeInfoRef`, optional

Example:

```
<unit id="1">
 <segment><pc id="1" dataRefStart="1" dataRefEnd="2">Important</pc>
text</source>
 </segment>
 <originalData>
  <data>&lt;B&gt;</data>
```

```
  <data>&lt;/B&gt;</data>
 </originalData>
</unit>
```

Processing Requirements

- The <pc> should not be used to represent standalone codes. Using a spanning code for standalone code may result in having text inside a span where the original format does not allow it.

2.2.3.4 sc

Start marker of a spanning original code.

Contains:

This element is always empty.

Parents:

<source>, <target>, <pc> and <mrk>

Attributes:

- canCopy, optional
- canDelete, optional
- canOverlap, optional
- canReorder, optional
- copyOf, optional
- disp, optional
- equiv, optional
- id, required
- isolated, optional
- dataRef, optional
- subFlows, optional
- subType, optional
- type, optional
- dir, optional
- fs:fs, optional
- fs:subFs, optional
- slr:storageRestriction, optional
- slr:sizeRestriction, optional
- slr:equivStorage, optional
- slr:sizeInfo, optional
- slr:sizeInfoRef, optional

Example:

```
<unit id="1">
 <segment>
  <source><sc id="1" type="fmt" subType="xlf:b"/>First sentence. </source>
 </segment>
 <segment>
  <source>Second sentence.<ec startRef="1" type="fmt" subType="xlf:b"/></source>
 </segment>
</unit>
```

Processing Requirements

- The values of the attributes canCopy, canDelete, canReorder and canOverlap must be the same as the values the ones in the <ec> element corresponding to this start marker.

- The attribute `isolated` must be set to `yes` when the `<ec>` element corresponding to this start marker is not in the same `<unit>`, and set to `no` otherwise.

- The `<sc>` / `<ec>` pair should not be used to represent standalone codes. Using a spanning code for standalone code may result in having text inside a span where the original format does not allow it.

2.2.3.5 ec

End marker of a spanning original code.

Contains:

This element is always empty.

Parents:

`<source>`, `<target>`, `<pc>` and `<mrk>`

Attributes:

- `canCopy`, optional
- `canDelete`, optional
- `canOverlap`, optional
- `canReorder`, optional
- `copyOf`, optional
- `disp`, optional
- `equiv`, optional
- `id`, optional
- `isolated`, optional
- `dataRef`, optional
- `startRef`, optional
- `subFlows`, optional
- `subType`, optional
- `type`, optional
- `fs:fs`, optional
- `fs:subFs`, optional
- `slr:equivStorage`, optional
- `slr:sizeInfo`, optional
- `slr:sizeInfoRef`, optional

Example:

```
<unit id="1">
 <segment>
  <source>Text in <sc id="1" dataRef="d1"/>bold <sc id="2" dataRef="d2"/>
and<ec startRef="1" dataRef="d3"/> italics<ec startRef="2" dataRef="d4"/>.</source>
 </segment>
 <originalData>
  <data id="d1">\b </data>
  <data id="d2">\i </data>
  <data id="d3">\b0 </data>
  <data id="d4">\i0 </data>
 </originalData>
</unit>
```

Processing Requirements

- The values of the attributes `canCopy`, `canDelete`, `canReorder` and `canOverlap` must be the same as the values the ones in the `<sc>` element corresponding to this end marker.

- The attribute `isolated` must be set to `yes` when the `<sc>` element corresponding to this end marker is not in the same `<unit>` and set to `no` otherwise.
- When the attribute `isolated` is set to `yes`, the attribute `id` must be used instead of the attribute `startRef`.
- The `<sc>` / `<ec>` pair should not be used to represent standalone codes. Using a spanning code for standalone code may result in having text inside a span where the original format does not allow it.

2.2.3.6 mrk

Represents an annotation.

Contains:

- Text
- Zero, one or more `<cp>` elements
- Zero, one or more `<ph>` elements
- Zero, one or more `<pc>` elements
- Zero, one or more `<sc>` elements
- Zero, one or more `<ec>` elements
- Zero, one or more `<mrk>` elements
- Zero, one or more `<sm>` elements
- Zero, one or more `<em>` elements

Text and inline elements may appear in any order.

Parents:

`<source>`, `<target>`, `<pc>` and `<mrk>`

Attributes:

- `id`, required
- `translate`, optional
- `type`, optional
- `ref`, optional
- `value`, optional
- `fs:fs`, optional
- `fs:subFs`, optional
- `slr:storageRestriction`, optional
- `slr:sizeRestriction`, optional
- attributes from any namespace, optional

See the Annotations section for more details and examples on how to use the `<mrk>` element.

2.2.3.7 sm

Start marker of an annotation.

Contains:

This element is always empty.

Parents:

`<source>`, `<target>`, `<pc>` and `<mrk>`

Attributes:

- `id`, required

- translate, optional
- type, optional
- ref, optional
- value, optional
- fs:fs, optional
- fs:subFs, optional
- slr:storageRestriction, optional
- slr:sizeRestriction, optional
- attributes from any namespace, optional

See the Annotations section for more details and examples on how to use the <sm> element.

2.2.3.8 em

End marker of an annotation.

Contains:

This element is always empty.

Parents:

<source>, <target>, <pc> and <mrk>

Attributes:

- startRef, required
- fs:fs, optional
- fs:subFs, optional

See the Annotations section for more details and examples on how to use the element.

2.3 Attributes

This section lists all the various attributes used in XLIFF core elements.

2.3.1 XLIFF Attributes

The attributes defined in XLIFF 2.0 are: appliesTo, approved, canCopy, canDelete, canOverlap, canReorder, category, copyOf, name, dataRef, dataRefEnd, dataRefStart, dir, disp, dispEnd, dispStart, equiv, equivEnd, equivStart, hex, href, id, isolated, original, priority, ref, startRef, skeleton, srcDir, srcLang, subFlows, subFlowsEnd, subFlowsStart, subType, subState, state, tgtLang, translate, trgDir, type, value and version.

2.3.1.1 approved

Approved - Indicates whether the holding element contains a translation suitable to be used when converting the XLIFF file to original format.

Value description: yes or no.

Default value: no

Used in: .

2.3.1.2 appliesTo

Comment target - Indicates the element to whom the content of the note applies.

Value description: source or target.

Default value: undefined.

Used in: <note>.

2.3.1.3 canCopy

Replication editing hint - Indicates whether or not the inline code can be copied.

Value description: yes if the code can be copied, no if the code should not be copied.

Default value: yes.

Used in: <pc>, <sc>, <ec>, <ph>.

2.3.1.4 canDelete

Deletion editing hint - Indicates whether or not the inline code can be deleted.

Value description: yes if the code can be deleted, no if the code should not be deleted.

Default value: yes.

Used in: <pc>, <sc>, <ec>, <ph>.

2.3.1.5 canOverlap

Code can overlap - Indicates that the spanning code where this attribute is used may enclose partial spanning codes (i.e. a start marker without its corresponding end marker, or a end marker without its corresponding start marker).

Value description: yes or no.

Default value: the default value for this attribute depends on the element in which it is used:

- When used in <pc>: no.
- When used in <sc> or <ec>: yes.

Used in: <pc>, <sc> and <ec>

Example:

```
<unit id="1">
 <segment>
  <source><pc id="1" dataRefStart="3" dataRefEnd="4" canOverlap="no"/>Bold,
<sc id="2" dataRef="1" canOverlap="yes"/>both</pc>,
italics<ec startRef="2" dataRef="2"/></source>
 </segment>
 <originalData>
  <data id="1">\i1 </data>
  <data id="2">\i0 </data>
  <data id="3">{\b </data>
  <data id="4">}</data>
 </originalData>
</unit>
```

2.3.1.6 canReorder

Re-ordering editing hint - Indicates whether or not the inline code can be re-ordered. See Editing Hints section for more details.

Value description: `yes` if the code can be re-ordered, `no` if the code should not be re-ordered.

Default value: `yes`.

Used in: `<pc>`, `<sc>`, `<ec>`, `<ph>`.

2.3.1.7 category

Category - Provides a way to categorize notes.

Value description: Text.

Default value: undefined

Used in: `<note>`.

2.3.1.8 copyOf

Reference to base code - Holds the `id` of the base code of a copied code.

Value description: NMTOKEN. The `id` value of the base code of which this code is a copy.

Default value: undefined

Used in: `<ph>`, `<pc>`, `<sc>`, `<ec>`.

Example:

```
<unit id="1">
 <segment>
  <source>Äter <pc id="1">katter möss</pc>?</source>
  <target>Do <pc id="1">cats</pc> eat <pc id="2"
copyOf="1">mice</pc>?</target>
 </segment>
</unit>
```

2.3.1.9 dir

Indicates the directionality of a content.

Value description: `ltr` (Left-To-Right) or `rtl` (Right-To-Left)

Default value: default values for this attribute depend on the element in which it is used:

- When used in `<source>`:

  The value of the `srcDir` attribute of the `<unit>` element in which the unit is located.

- When used in `<target>`:

  The value of the `trgDir` attribute of the `<unit>` element in which the unit is located.

- When used in `<pc>`, or `<sc>`:

  The value of the `dir` attribute of the `<source>` or `<target>` element in which the element is located.

- When used in `<data>`:

  The value `ltr`.

Used in: <source>, <target>, <data>, <pc> and <sc>.

2.3.1.10 disp

Display text - Holds an alternative user-friendly display representation of the original data of the inline code.

Value description: Text

Default value: undefined

Used in: <ph>, <sc>, <ec>.

Example:

```
<unit id="1">
 <segment>
  <source>Welcome back <ph id="1" disp="[UserName]" dataRef="d1"/>!</source>
 </segment>
 <originalData>
  <data id="d1">{1}</data>
 </originalData>
</unit>
```

Note

To provide a plain text equivalent of the code, use the equiv attribute.

2.3.1.11 dispEnd

Display text - Holds an alternative user-friendly display representation of the original data of the end marker of an inline code.

Value description: Text

Default value: undefined

Used in: <pc>.

Example:

```
<unit id="1">
 <segment>
  <source>Example of <pc id="1" dataRefStart="d1" dataRefEnd="d2"
dispStart="&lt;span>" dispEnd="&lt;/span>">formatted
text</pc>.</source>
 </segment>
 <originalData>
  <data id="d1">\cf1\ul\b\f1\fs24 </data>
  <data id="d2">\cf0\ulnone\b0\f0\fs22 </data>
 </originalData>
</unit>
```

In the example above, the dispStart and dispEnd attributes provide a more user-friendly representation of the original formatting codes.

Note

To provide a plain text equivalent of the code, use the equivEnd attribute.

2.3.1.12 dispStart

Display text - Holds an alternative user-friendly display representation of the original data of the start marker of an inline code.

Value description: Text

Default value: undefined

Used in: <pc>.

Example:

```
<unit id="1">
 <segment>
  <source>Example of <pc id="1" dataRefStart="d1" dataRefEnd="d2"
dispStart="&lt;span>" dispEnd="&lt;/span>">formatted
text</pc>.</source>
 </segment>
 <originalData>
  <data id="d1">\cf1\ul\b\f1\fs24 </data>
  <data id="d2">\cf0\ulnone\b0\f0\fs22 </data>
 </originalData>
</unit>
```

In the example above, the dispStart and dispEnd attributes provide a more user-friendly representation of the original formatting codes.

Note

To provide a plain text equivalent of the code, use the equivStart attribute.

2.3.1.13 equiv

Equivalent text - Holds a plain text representation of the original data of the inline code that may be used when generating a plain text representation of the content.

Value description: Text

Default value: an empty string.

Used in: <ph>, <sc>, <ec>.

Example:

```
<unit id="1">
 <segment>
  <source>Open <ph id="1" equiv="" dataRef="d1"/>File</source>
 </segment>
 <originalData>
  <data id="d1">&amp;</data>
 </originalData>
</unit>
```

In this example the equiv attribute of the <ph> element is used to indicate that the original data of the code can be ignored in the text representation of the string. This could, for instance, help a spell-checker tool to process the content as "Open File".

Note

To provide a user-friendly representation, use the `disp` attribute.

2.3.1.14 equivEnd

Equivalent text - Holds a plain text representation of the original data of the end marker of an inline code that may be used when generating a plain text representation of the content.

Value description: Text

Default value: an empty string

Used in: `<pc>`.

Example:

```
<unit id="1">
 <segment>
  <source>The jam made of <pc id="1" dataRefStart="d1" equivStart=""
dataRefEnd="d2" equivEnd="">lingonberries</pc> is quite tasty.</source>
 </segment>
 <originalData>
  <data id="d1">&lt;span class="link" onclick="linkTo('dbId5345')"></data>
  <data id="d2">&lt;/span></data>
 </originalData>
</unit>
```

Note

To provide a user-friendly representation, use the `dispEnd` attribute.

2.3.1.15 equivStart

Equivalent text - Holds a plain text representation of the original data of the start marker of an inline code that may be used when generating a plain text representation of the content.

Value description: Text

Default value: an empty string

Used in: `<pc>`.

Example:

```
<unit id="1">
 <segment>
  <source>The jam made of <pc id="1" dataRefStart="d1" equivStart=""
dataRefEnd="d2" equivEnd="">lingonberries</pc> is quite tasty.</source>
 </segment>
 <originalData>
  <data id="d1">&lt;span class="link" onclick="linkTo('dbId5345')"></data>
  <data id="d2">&lt;/span></data>
 </originalData>
</unit>
```

Note

To provide a user-friendly representation, use the `dispStart` attribute.

2.3.1.16 hex

Hexadecimal code point - Holds the value of a Unicode code point that is invalid in XML.

Value description: A canonical representation of the hexBinary XSD data type: Two hexadecimal digits to represent each octet of the Unicode code point. The allowed values are any of the values representing code points invalid in XML, between hexadecimal 0000 and 10FFFF (both included).

Default value: undefined

Used in: <cp>.

Example:

```
<cp hex="001A"/><cp hex="0003"/>
```

The example above shows a character U+001A and a character U+0003 as they should be represented in XLIFF.

2.3.1.17 href

href - a pointer to the location of an external skeleton file pertaining to the enclosing <file> element..

Value description: IRI.

Default value: undefined

Used in: <skeleton>.

2.3.1.18 id

Identifier - A character string used to identify an element.

Value description: NMTOKEN. The scope of the values for this attribute depend on the element in which it is used

- When used in <unit>:

  The value must be unique within the <file> element.

- When used in or <ignorable>:

  The value must be unique within the <unit> element.

- When used in <data>:

  The value must be unique within the <originalData> element.

- When used in <mrk>, <sm>, <pc>, <sc>, <ec> or <ph>:

  The value must be unique within the or <ignorable> elements and inline elements with the same id in both source and target must be corresponding elements.

Default value: undefined

Used in: <file>, <unit>, , <ignorable>, <match>, <data>, <sc>, <ec>, <ph>, <pc>, <mrk> and <sm>.

2.3.1.19 isolated

Orphan code flag - Indicates if the start or end marker of a spanning inline code is not in the same <unit> as its corresponding end or start marker.

Value description: `yes` if this start or end marker is not in the same `<unit>` as its corresponding end or start marker, `no` if both markers are in the same `<unit>`.

Default value: no

Used in: `<sc>`, `<ec>`.

Example:

```
<unit id="1">
 <segment>
  <source><pc id="1">Warning: File not found.</pc></source>
 </segment>
 <mtc:matches>
  <mtc:match>
   <segment>
    <source><sc id="1" isolated="yes"/>Warning:</source>
    <target><sc id="1" isolated="yes"/>Attention :</target>
   </segment>
  </mtc:match>
 </match>
</unit>
```

In the example above the `<sc>` elements have their `isolated` attribute set to `yes` because they do not have their corresponding `<ec>` elements.

2.3.1.20 name

Resource Name - The original identifier of the resource corresponding to the extracted `<unit>` or `<group>`. For example: the key in the key/value pair in a Java properties file, the ID of a string in a Windows string table, the index value of an entry in a database table, etc.

Value description: text string.

Default value: undefined.

Used in: `<unit>` and `<group>`.

2.3.1.21 dataRef

Original data reference - Holds the identifier of the `<data>` element that contains the original data for a given inline code.

Value description: An XSD NMTOKEN that must be the value of the `id` attribute of one of the `<data>` element listed in the same `<unit>` element.

Default value: undefined.

Used in: `<ph>`, `<sc>`, `<ec>`.

Example:

```
<unit id="1">
 <segment>
  <source>Error in '<ph id="1" dataRef="d1"/>'.</source>
  <target>Erreur dans '<ph id="1" dataRef="d1"/>'.</target>
 </segment>
 <originalData>
  <data id="d1">{0}</data>
```

```
 </originalData>
</unit>
```

The example above shows a <ph> element that has its original data stored outside the content, in a <data> element.

2.3.1.22 dataRefEnd

Original data reference - Holds the identifier of the <data> element that contains the original data for the end marker of a given inline code.

Value description: An XSD NMTOKEN that must be the value of the id attribute of one of the <data> element listed in the same <unit> element.

Default value: undefined.

Used in: <pc>.

Example:

```
<unit id="1">
 <segment>
  <source><pc id="1" dataRefStart="d1" dataRefEnd="d2">Efficiency<pc>
is the operative word here.</source>
  <target><pc id="1" dataRefStart="d1" dataRefEnd="d2">Efficacité<pc>
est le mot clé ici.</target>
 </segment>
 <originalData>
  <data id="d1">&lt;EM></data>
  <data id="d2">&lt;/EM></data>
 </originalData>
</unit>
```

The example above shows two <pc> elements with their original data stored outside the content, in two <data> elements.

2.3.1.23 dataRefStart

Original data reference - Holds the identifier of the <data> element that contains the original data for the start marker of a given inline code.

Value description: An XSD NMTOKEN that must be the value of the id attribute of one of the <data> element listed in the same <unit> element.

Default value: undefined.

Used in: <pc>.

Example:

```
<unit id="1">
 <segment>
  <source><pc id="1" dataRefStart="d1" dataRefEnd="d2">Efficiency<pc>
is the operative word here.</source>
  <target><pc id="1" dataRefStart="d1" dataRefEnd="d2">Efficacité<pc>
est le mot clé ici.</target>
 </segment>
 <originalData>
  <data id="d1">&lt;EM></data>
```

```
  <data id="d2">&lt;/EM></data>
 </originalData>
</unit>
```

The example above shows two <pc> elements with their original data stored outside the content, in two <data> elements.

2.3.1.24 original

Original File - A pointer to the location of the original document from which the content of the enclosing <file> element is extracted.

Value description: Text.

Default value: undefined

Used in: <file>.

2.3.1.25 priority

Priority - Provides a way to prioritize notes.

Value description: Integer 1-10.

Default value: 1

Used in: <note>.

Note

Please note that 1 is the highest priority that can be interpeted as an alert, e.g. an ITS Localization Note of the type alert. The best parctice is to use only one alert per an anotated element, and the full scale of 2-10 can be used for prioritizing notes of lesser importance than the alert.

2.3.1.26 ref

Reference - Holds a reference for the associated annotation.

Value description: A value of the XSD type anyURI. The semantics of the value depends on the type of annotation:

- When used in a term annotation, the value is referring to the resource providing information about the term.
- When used in a comment annotation, the value is referring to a <note> element.
- When used in a custom annotation, the value is defined by each custom annotation.

Default value: undefined

Used in: <mrk> or <sm>.

Example:

```
<unit id="1">
 <segment>
  <source>The <pc id="1">ref</pc> attribute of a term
annotation holds a <mrk id="m1" type="term"
ref="http://dbpedia.org/page/Uniform_Resource_Identifier">URI</mrk>
pointing to more information about the given term.</source>
```

```
 </segment>
</unit>
```

Processing Requirements

- The value of the `ref` attribute should point to an element that is a child of the `<unit>` element where the parent of the attribute is located. Pointing to an element located elsewhere may prevent proper execution of processing requirements.

2.3.1.27 startRef

Start marker reference - The `id` of the `<sc>` element or the `<sm>` element a given `<ec>` element or `<em>` element corresponds.

Value description: NMTOKEN.

Default value: undefined

Used in: `<ec>`, `<em>`.

Example:

```
<unit id="1">
 <segment>
  <source><sc id="1"/>Bold, <sc id="2"/>both<ec startRef="1"/>,
italics<ec startRef="2"/></source>
 </segment>
</unit>
```

2.3.1.28 skeleton

skeleton attribute - a pointer to the location of the file that contains untranslatable data for the enclosing `<file>` element.

Value description: IRI.

Default value: undefined

Used in: `<file>`.

2.3.1.29 srcDir

Indicates the directionality of the source content.

Value description: `ltr` (Left-To-Right) or `rtl` (Right-To-Left)

Default value: default values for this attribute depend on the element in which it is used:

- When used in `<file>`:

  The value `ltr`.

- When used in `<unit>`:

  The value of the `srcDir` attribute of the `<file>` element in which the unit is located.

Used in: `<unit>` and `<file>`.

2.3.1.30 srcLang

Source Language - The code of the language in which the text to be translated is expressed.

Value description: A language code as described in [BCP 47].

Default value: undefined

Used in: <xliff>.

2.3.1.31 subFlows

Sub-flows list - Holds a list of id attributes corresponding to the <unit> elements that contain the sub-flows for a given inline code.

Value description: A list of NMTOKEN values separated by spaces. Each value corresponds to the id attribute of a <unit> element.

Default value: undefined

Used in: <ph>, <sc>, <ec>.

Example:

See the example in the Sub-Flows section.

2.3.1.32 subFlowsEnd

Sub-flows list - Holds a list of id attributes corresponding to the <unit> elements that contain the sub-flows for the end marker of a given inline code.

Value description: A list of NMTOKEN values separated by spaces. Each value corresponds to the id attribute of a <unit> element.

Default value: undefined

Used in: <pc>.

Example:

See the example in the Sub-Flows section.

2.3.1.33 subFlowsStart

Sub-flows list - Holds a list of id attributes corresponding to the <unit> elements that contain the sub-flows for the start marker of a given inline code.

Value description: A list of NMTOKEN values separated by spaces. Each value corresponds to the id attribute of a <unit> element.

Default value: undefined

Used in: <pc>.

Example:

See the example in the Sub-Flows section.

2.3.1.34 subType

subType - Indicates the secondary level type of an inline code.

Value description:

The value is composed of a prefix and a sub-value separated by a character : (U+003A).

The prefix is a string uniquely identifying a collection of values for a specific authority. The sub-value is any string value defined by an authority.

The prefix `xlf` is reserved for this specification, and the following sub-values are defined:

`xlf:lb` - Line break
`xlf:pb` - Page break
`xlf:b` - Bold
`xlf:i` - Italics
`xlf:u` - Underlined
`xlf:var` - Variable

Other prefixes and sub-values may be defined by the users.

Default value: Undefined

Used in: `<pc>`, `<sc>`, `<ec>` and `<ph>`

Processing Requirements

- If the attribute `subType` is used, the attribute `type` must be specified as well.

2.3.1.35 subState

subState - indicates a user-defined status for the `<segment>` element.

Value description:

The value is composed of a prefix and a sub-value separated by a character : (U+003A).

The prefix is a string uniquely identifying a collection of values for a specific authority. The sub-value is any string value defined by an authority.

The prefix `xlf` is reserved for this specification.

Other prefixes and sub-values may be defined by the users.

Default value: Undefined

Used in: `<segment>`

Processing Requirements

- If the attribute `subState` is used, the attribute `state` *must* be specified as well.

2.3.1.36 state

Type - indicates the state of the translation of a segment.

Value description: The value must be set to one of the following values:

`initial` - indicates the segment is in its initial state.
`translated` - indicates the segment has been translated.
`reviewed` - indicates the segment has been reviewed.
`final` - indicates the segment is finalized and ready to be used.

One can further specify the state of the translation using the `subState` attribute.

Default value: `initial`

Used in: `<segment>`

2.3.1.37 tgtLang

Target Language - The code of the language in which the translated text is expressed.

Value description: A language code as described in [BCP 47].

Default value: undefined

Used in: <xliff>.

2.3.1.38 translate

Translate - Indicates whether or not the source text text of a element should be translated.

Value description: yes or no.

Default value: yes

Used in: , <unit> and <mrk>.

2.3.1.39 trgDir

Indicates the directionality of the target content.

Value description: ltr (Left-To-Right) or rtl (Right-To-Left)

Default value: default values for this attribute depend on the element in which it is used:

- When used in <file>:

  The value ltr.

- When used in <unit>:

  The value of the trgDir attribute of the <file> element in which the unit is located.

Used in: <unit> and <file>.

2.3.1.40 type

Type - Indicates the type of an element.

Value description: allowed values for this attribute depend on the element in which it is used.

- When used in <pc>, <sc>, <ec> or <ph>:

  The value must be set to one of the following values:

  fmt - Formatting (e.g. a <b> element in HTML)
  ui - User interface element
  quot - Inline quotation
  link - Link (e.g. an <a> element in HTML)
  image - Image or graphic
  other - Type of element not covered by any of the other top-level types.

  One can further specify the type of a code using the subType attribute.

  Default value: Undefined

- When used in <mrk> or <sm>:

  One of the following values: generic, comment, term, or a user-defined value that is composed of a prefix and a value separated by a character : (U+003A).

  Default value: generic

Used in: <pc>, <sc>, <ec>, <mrk> and <ph>

2.3.1.41 value

Value - Holds a value for the associated annotation.

Value description: Text.

- When used in a term annotation, the value is a definition of the term.
- When used in a comment annotation, the value is the text of the comment.
- When used in a custom annotation, the value is defined by each custom annotation.

Default value: undefined

Used in: `<mrk>`.

2.3.1.42 version

XLIFF version - The version attribute is used to specify the format version of the XLIFF document. This corresponds to the version number of the XLIFF specification that is being adhered to.

Value description: Text.

Default value: 2.0

Used in: `<xliff>`.

2.3.2 XML namespace

The attributes from XML namespace used in XLIFF 2.0 are xml:lang and xml:space.

2.3.2.1 xml:lang

Language - The xml:lang attribute specifies the language variant of the text of a given element. For example: `xml:lang="fr-FR"` indicates the French language as spoken in France.

Value description: A language code as described in [BCP 47].

Default value: undefined

Used in: `<source>`, `<target>`.

2.3.2.2 xml:space

White spaces - The xml:space attribute specifies how white spaces (ASCII spaces, tabs and line-breaks) should be treated.

Value description: `default` or `preserve`. The value `default` signals that an application's default white-space processing modes are acceptable for this element; the value `preserve` indicates the intent that applications preserve all the white space. This declared intent is considered to apply to all elements within the content of the element where it is specified, unless overridden with another instance of the xml:space attribute. For more information see the section on xml:space in the XML specification.

Default value: `default`

Used in: `<source>`, `<target>`.

2.4 CDATA sections

CDATA sections (`<![CDATA[...]]>`) are allowed in XLIFF content, but on output they *may* be changed into normal escaped content.

Note that avoiding CDATA sections is considered a best practice from the internationalization viewpoint [XML I18N BP].

Processing Requirements

- Readers *must* process CDATA sections.
- Writers *may* preserve the original CDATA sections.

2.5 XML Comments

XML comments (`<!--...-->`) are allowed in XLIFF content, but they are ignored in the parsed content.

For example:

```
<source>Text content <!--IMPORTANT-->that is important</source>
```

and

```
<source>Text content that is important</source>
```

are identical after parsing and correspond to the same following parsed content:

```
Text content that is important
```

To annotate a section of the content with a comment that is recognized and preserved by XLIFF user agents, use the `<note>` element, or the `<mrk>` element.

Processing Requirements

- Readers *must* ignore XML comments. That is the XLIFF parsed content is the same whether or not there is an XML comment in the document.
- Writers *may* preserve XML comments on output.

2.6 Inline Content

The XLIFF inline content defines how to encode the content extracted from the original source. The content includes the following types of data:

- Text -- Textual content.
- Inline codes -- Sequences of content that are not linguistic text, such as formatting codes, variable placeholders, etc.

  For example: the element `<b>` in HTML, or the placeholder `{0}` in a Java string.
- Annotations -- Markers that delimit a span of the content and carry or point to information about the specified content.

  For example: a flag indicating that a given section of text is not intended for translation, or an element indicating that a given expression in the text is a term associated with a definition.

There are two elements that contain inline markup in XLIFF: `<source>` and `<target>`.

In some cases, data directly associated with inline elements *may* also be stored at the `<unit>` level.

2.6.1 Text

The XLIFF inline markup does not prescribe how to represent normal text, besides that it *must* be valid XML.

2.6.1.1 Characters invalid in XML

Because the content represented in XLIFF can be extracted from anywhere, including software resources and other material that can contain control characters, XLIFF needs to be able to represent all Unicode code points.

However, XML does not have the capability to represent all Unicode code points, and does not provide any official mechanism to escape the forbidden code points.

To remedy this, the inline markup provides the <cp> element.

The syntax and semantic of <cp> in XLIFF are similar to the ones of <cp> in the Unicode Locale Data Markup Language [LDML].

2.6.2 Inline Codes

The specification takes into account two types of codes:

Original code
: An *original code* is a code that exists in the original document being extracted into XLIFF.

Added code
: An *added code* is a code that does not exist in the original document, but has been added to the content at some point after extraction.

Any code (original or added) belongs to one of the two following categories:

Standalone
: A *standalone* code is a code that corresponds to a single position in the content. An example of such code is the
 element in HTML.

Spanning
: A *spanning* code is a code that encloses a section of the content using a start and an end marker. There are two kinds of spanning codes:

  - Codes that can overlap, that is: they can enclose a non-closing or a non-opening spanning code. Such codes do not have an XML-like behavior. For example the RTF code \b1...\b0 is a spanning code that is allowed to overlap.
  - Codes that *must not* overlap, that is: they cannot enclose a partial spanning code and have an XML-like behavior at the same time. An example of such code is the <emphasis>...</emphasis> element in DocBook.

  When the opening or closing marker of a spanning code does not have its corresponding closing or opening marker in the same unit, it is an *orphan code*.

2.6.2.1 Representation of the codes

Spanning codes present a set of challenges in XLIFF:

First, because the code format of the original data extracted to XLIFF does not need to be XML, spanning codes can overlap.

For example, in the following RTF content, the format markers are in a sequence: start bold, start italics, end bold, end italics. This does not translate into a well-formed mapping.

```
Text in \b bold \i and\b0  italics\i0
```

Another challenge is the possible effect of segmentation: A spanning code can start in one segment and end in another.

For example, in the following HTML content, the segmentation splits the text independently of the codes so the starting and ending tags of the <B>...</B> element end up in different parts of the <unit> element:

```
[Sentence <B>one. ][Sentence two.][ ][Sentence</B> three.]
```

Finally, a third potential cause of complication is that the start or the end markers of a spanning code can become orphans if their segment is used outside of its original <unit>.

For example, an entry with bold text can be broken down into two segments:

```
Segment 1 = "<b>Warning found: "
Segment 2 = "The file is read-only</b>"
```

And later, one of the segments can be re-used outside its original <unit>, for instance as a translation candidate:

```
New segment = "<b>Warning found - see log</b>"
Fuzzy match = "<b>Warning found: "
```

Because of these use cases, the representation of a spanning code cannot always be mapped to a similar spanning element in XLIFF.

When taking into account these issues, the possible use cases and their corresponding XLIFF representations are as follow:

Table 1. Inline code use cases

Use Case	Example of Representation
Standalone code	<ph id='1'/>
Well-formed spanning code	<pc id='1'>text</pc>
Start marker of spanning code	<sc id='1'/>
End marker of spanning code	<ec startRef='1'/>
Orphan start marker of spanning code	<sc id='1' isolated='yes'/>
Orphan end marker of spanning code	<ec id='1' isolated='yes'/>

2.6.2.2 Usage of <pc> and <sc>/<ec>

A spanning code *must* be represented using a <sc> element and a <ec> element if the code is not well-formed or orphan.

For example, the following RTF content has two spans of formatting:

```
Text in \b bold \i and\b0  italics\i0
```

They can only be represented using two pairs of <sc> and <ec> elements:

```
Text in <sc id="1">\b </sc>bold
 <sc id="2">\i </sc>and<ec startRef="1">\b0 </ec> italics<ec startRef="2">\i0</ec>
```

If the spanning code is well-formed it *may* be represented using either a single <pc> element or using a pair of <sc> and a <ec> elements.

For example, the following RTF content has a single span of formatting:

```
Text in \b bold\b0 .
```

It can be represented using either notations:

```
Text in <pc id="1" canOverlap="yes" dataRefStart="c1" dataRefEnd="c2">bold</pc>.
```

```
Text in <sc id="1" dataRef="c1"/>bold<ec startRef="1" dataRef="c2"/>.
```

Processing Requirements

- When both the <pc> and the <sc>/<ec> representations are possible, writers *may* use either one as long as all the information of the inline code (e.g. original data, sub-flow indicators, etc.) are preserved.
- When converting representation between a pair of <sc> and <ec> elements and a <pc> element or vice-versa, the user agents *must* map their attributes as shown in the following table:

  Table 2. Mapping between attributes

<pc> attributes	<sc> attributes	<ec> attributes
id	id	startRef / id
type	type	type
dispStart	disp	
dispEnd		disp
equivStart	equiv	
equivEnd		equiv
subFlowsStart	subFlows	
subFlowsEnd		subFlows
dataRefStart	dataRef	
dataRefEnd		dataRef
	isolated	isolated
canCopy	canCopy	canCopy
canDelete	canDelete	canDelete
canReorder	canReorder	canReorder
copyOf	copyOf	copyOf
canOverlap	canOverlap	canOverlap
dir	dir	

- Readers *must* be able to handle any types of inline code representation.

2.6.2.3 Storage of the original data

Most of the time, inline codes correspond to an original construct in the format from which the content was extracted. This is the *original data*.

XLIFF tries to abstract and normalize as much as possible the extracted content because this allows a better re-use of the material across projects. Some tools require access to the original data in order to create the translated document back into its original format. Others do not.

2.6.2.3.1 No storage of the original data

In this option, the original data of the inline code is not preserved inside the XLIFF document.

The tool that created the initial XLIFF document is responsible for providing a way to re-create the original format properly when merging back the content.

For example, for the following HTML content:

```
This <B>naked mole rat</B> is <B>pretty ugly</B>.
```

one possible XLIFF representation is the following:

```
<unit id="1">
 <segment>
  <source>This <pc id="1">naked mole rat</pc>
is <pc id="2">pretty ugly</pc>.</source>
  <target>Cet <pc id="1">hétérocéphale</pc>
est <pc id="2">plutôt laid</pc>.</target>
 </segment>
</unit>
```

2.6.2.3.2 Storage of the original data

In this option, the original data of the inline code is stored in a structure that resides outside the content (i.e. outside `<source>` or `<target>`) but still inside the `<unit>` element.

The structure is an element `<originalData>` that contains a list of `<data>` entries uniquely identified within the `<unit>` by an `id` attribute. In the content, each inline code using this mechanism includes a `dataRef` attribute that points to a `<data>` element where its corresponding original data is stored.

For example, for the following HTML content:

```
This <B>naked mole rat</B> is <B>pretty ugly</B>.
```

The following XLIFF representation stores the original data:

```
<unit id="1">
 <segment>
  <source>This <pc id="1" dataRefStart="d1" dataRefEnd="d2">naked mole rat</pc>
is <pc id="2" dataRefStart="d1" dataRefEnd="d2">pretty ugly</pc>.</source>
  <target>Cet <pc id="1" dataRefStart="d1" dataRefEnd="d2">hétérocéphale</pc>
est <pc id="2" dataRefStart="d1" dataRefEnd="d2">plutôt laid</pc>.</target>
 </segment>
 <originalData>
  <data id="d1">&lt;B></data>
  <data id="d2">&lt;/B></data>
 </originalData>
</unit>
```

Note

This mechanism allows to re-use identical original data by pointing to the same `<data>` element.

2.6.2.4 Adding Codes

When processing a content, there are possible cases when new inline codes need to be added.

For example, in the following HTML help content, the text has the name of a button in bold:

```
Press the <b>Emergency Stop</b> button
to interrupt the count-down sequence.
```

In the translated version, the original label needs to remain in English because the user interface, unlike the help, is not translated. However, for convenience, a translation is also provided and emphasized using another style. That new formatting needs to be added:

```
Appuyez sur le bouton <b>Emergency Stop</b> (<i>Arrêt d'urgence</i>)
pour interrompre le compte à rebours.
```

Having to split a single formatted span of text into several separate parts during translation, can serve as another example. For instance, the following sentence in Swedish uses bold on the names of two animals:

```
Äter <b>katter möss</b>?
```

But the English translation separates the two names and therefore needs to duplicate the bold codes.

```
Do <b>cats</b> eat <b>mice</b>?
```

Processing Requirements

- User agents *may* add inline codes.
- The `id` value of the added code *must* be different from all `id` values in both source and target content of the unit where the new code is added.
- Merging tools *may* ignore added inline codes when re-creating the extracted content in its original format.

There are several ways to add codes:

2.6.2.4.1 Duplicating an existing code

One way to create a new code is to duplicate an existing one (called the *base code*).

If the base code is associated with some original data: the new code simply use these data.

For example, the translation in the following unit, the second inline code is a duplicate of the first one:

```
<unit id="1">
 <segment>
  <source>Äter <pc id="1" dataRefStart="d1" dataRefEnd="d2">katter
möss</pc>?</source>
  <target>Do <pc id="1" dataRefStart="d1" dataRefEnd="d2">cats</pc>
eat <pc id="2" dataRefStart="d1" dataRefEnd="d2">mice</pc>?</target>
 </segment>
 <originalData>
  <data id="d1">&lt;b></data>
```

```
  <data id="d2">&lt;/b></data>
 </originalData>
</unit>
```

If the base code has no associated data, the new code *must* use the `copyOf` attribute to indicate the `id` of the base code. This allows the merging tool to know what original data to re-use.

For example, the translation in the following unit, the second inline code is a duplicate of the first one:

```
<unit id="1">
 <segment>
  <source>Esznek <pc id="1">a magyarok svéd húsgombócot</pc>?</source>
  <target>Do <pc id="1">Hungarians</pc> eat
<pc id="2" copyOf="1">Swedish meatballs</pc>?</target>
 </segment>
</unit>
```

Processing Requirements

- User agents *must not* clone a code that has its `canCopy` attribute is set to `no`.
- The `copyOf` attribute *must* be used when, and only when, the base code has no associated original data.

2.6.2.4.2 Creating a brand-new code

Another way to add a code is to create it from scratch. For example, this can happen when the translated text requires additional formating.

For example, in the following unit, the UI text needs to stay in English, and is also translated into French as an hint for the reader. The French translation for the UI text is formatted in italics:

```
<unit id="1">
 <segment>
  <source>Press the <pc id="1" dataRefStart="d1" dataRefEnd="d2">Emergency Stop</pc>
button to interrupt the count-down sequence.</source>
  <target>Appuyez sur le bouton
<pc id="1" dataRefStart="d1" dataRefEnd="d2">Emergency Stop</pc>
(<pc id="2" dataRefStart="n2" dataRefEnd="n2">Arrêt d'urgence</pc>) pour interrompre
le compte à rebours.</target>
 </segment>
 <originalData>
  <data id="d1">&lt;b></data>
  <data id="d2">&lt;/b></data>
  <data id="n1">&lt;i></data>
  <data id="n2">&lt;/i></data>
 </originalData>
</unit>
```

2.6.2.4.3 Converting text into a code

Another way to add a code is to convert part of the extracted text into code. In some cases the inline code can be created after extraction, using part of the text content. This can be done, for instance, to get better matches from an existing TM, or better candidates from an MT system.

For example, it can happen that a tool extracting a Java properties file to XLIFF is not sophisticated enough to treat HTML or XML snippets inside the extracted text as inline code:

```
# text property for the widget 'next'
nextText: Click <ui>Next</ui>
```

Resulting XLIFF content:

```
<unit id="1">
 <segment>
  <source>Click &lt;ui>Next&lt;/ui></source>
 </segment>
</unit>
```

But another tool, later in the process, can be used to process the initial XLIFF document and detect additional inline codes. For instance here the XML elements such as `<ui>`.

The original data of the new code is the part of the text content that is converted as inline code.

```
<unit id="1">
 <segment>
  <source>Click <pc id="1" dataRefStart="d1" dataRefEnd="d2">Next</pc></source>
 </segment>
 <originalData>
  <data id="d1">&lt;ui></data>
  <data id="d2">&lt;/ui></data>
 </originalData>
</unit>
```

Warning

Converting XLIFF text content into original data for inline code might need a tool-specific process as the tool which did the initial extraction could have applied some conversion to the original content to create the XLIFF content (e.g. un-escape special characters).

2.6.2.5 Removing Codes

When processing a content, there are some possible cases when existing inline codes need to be removed.

For an example the translation of a sentence can result in grouping of several formatted parts into a single one. For instance, the following sentence in English uses bold on the names of two animals:

```
Do <b>cats</b> eat <b>mice</b>?
```

But the Swedish translation group the two names and therefore needs only a single bolded part.

```
Äter <b>katter möss</b>?
```

Processing Requirements

- User agents *may* remove a given inline code only if its `canDelete` attribute is set to `yes`.
- When removing a given inline code, the user agents *must* remove its associated original data, except if the original data is shared with another inline code that remains in the unit. Note that having to delete the original data is unlikely because such original data is likely to be associated to an inline code in the source content.

There are several ways to remove codes:

2.6.2.5.1 Deleting a code

One way to remove a code is to delete it from the extracted content. For example, in the following unit, the translated text does not use the italics formatting. It is removed from the target content, but the original data are preserved because they are still used in the source content.

```
<unit id="1">
 <segment>
  <source>I read <pc id="1" dataRefStart="d1" dataRefEnd="d2">Little
House on the Prairie</pc> to my children.</source>
  <target>子供に「大草原の小さな家」を読みました。</target>
 </segment>
 <originalData>
  <data id="d1">&lt;i></data>
  <data id="d2">&lt;/i></data>
 </originalData>
</unit>
```

2.6.2.5.2 Converting a code into text

Another way to remove an inline code is to convert it into text content. This is likely to be a rare use case. It is equivalant to deleting the code, with the addition to place the original data for the given code into the content, as text. This can be done, for example, to get better matches from an existing TM, or better candidates from an MT system.

For instance, the following unit has an inline code corresponding to a variable place-holder. A tool can temporarily treat this variable as text to get better matches from an existing TM.

```
<unit id="1">
 <segment>
  <source>Cannot find '<ph id="1" dataRef="d1"/>'.</source>
 </segment>
 <originalData>
  <data id="d1">%s</data>
 </originalData>
</unit>
```

The modified unit would end up like as shown below. Note that because the original data was not associated with other inline code it has been removed from the unit:

```
<unit id="1">
 <segment>
  <source>Cannot find '%s'.</source>
 </segment>
</unit>
```

Warning

Converting the original data of an inline code into text content might need a tool-specific process as the tool which did the initial extraction could have applied some conversion to the original content.

2.6.2.6 Editing Hints

XLIFF provides some information about what editing operations are applicable to inline codes:

- A code can be deleted: That is, the code element as well as its original data (if any are attached) are removed from the document. This hint is represented with the `canDelete` attribute. The default value is `yes`: deletion is allowed.

  For example, the following extracted C string has the code `<ph id='1'/>` set to be not deletable because removing the original data (the variable placeholder `%s`) from the string would result in an error when running the application:

- A code can be copied: That is, the code is used as a *base code* for adding another inline code. See Section 2.6.2.4.1, "Duplicating an existing code" for more details. This hint is represented with the `canCopy` attribute. The default value is `yes`: copy is allowed.

- A code can be re-ordered: That is, a given code can be moved before or after another inline code. This hint is represented with the `canReorder` attribute. The default value is `yes`: re-ordering is allowed.

Note that often those properties are related and appear together. For example, the code in the firt unit shown below is a variable placeholder that has to be preserved and cannot be duplicated, and when several of such variables are present, as in the second unit, they cannot be re-ordered:

```
<unit id="1">
 <segment>
  <source>Can't open '<ph id="1" dataRef="d1" canCopy="no" canDelete="no"/>'.</source>
 </segment>
 <originalData>
  <data id="d1">%s</data>
 </originalData>
</unit>
<unit id="2">
 <segment>
  <source>Number of <ph id="1" dataRef="d1" canCopy="no" canDelete="no" canReorder="no"/
<ph id="2" dataRef="d2" canCopy="no" canDelete="no" canReorder="no"/>.</source>
 </segment>
 <originalData>
  <data id="d1">%s</data>
  <data id="d2">%d</data>
 </originalData>
</unit>
```

See the Target Content Modification section for additional details on editing.

Processing Requirements

- Extraction tools *should* set the `canDelete`, `canCopy` and `canReorder` attributes for the codes that need to be treated differently than with the default settings.
- User agents *should not* delete inline codes that have their attribute `canDelete` set to `no`.
- User agents *should not* replicate inline codes that have their attribute `canCopy` set to `no`.
- When the attribute `canReorder` is set to `no`, the attributes `canCopy` and `canDelete` *must* also be set to `no`.

2.6.3 Annotations

An annotation is an element that associates a section of the content with some metadata information.

Annotation *may* be created by the tool that generates the initial XLIFF document, or by any other user agent later in the process. For example, after a first tool creates the document, a second tool can annotate the source content with terminological information.

Annotations are represented using the <mrk> element, or the pair of <sm> and elements.

Processing Requirements

- When a user agent removes a <mrk> element or a pair of <sm> / elements and the ref attribute is present, it *must* check whether or not the URI pointed by the ref attribute is within the same <unit> as the removed element. If it is and no other element has a reference to the pointed element, the user agent *must* remove the pointed element.

2.6.3.1 Type of Annotations

There are several pre-defined types of annotation and user agents can also define custom types.

2.6.3.1.1 Translate Annotation

This annotation is used to indicate whether a span of content is translatable or not.

Usage:

- The id attribute is required
- The translate attribute is required and set to yes or no
- The type attribute is optional and set to generic (this is the default value)

For example:

```
He saw his <mrk id="m1" translate="no">doppelgänger</mrk>.
```

Note

This annotation overrides the translate attribute set at the level.

Note

The translate attribute can also be used at the same time as another type of annotation. For example:

```
He saw his <mrk id="m1" translate="no" type="term">doppelgänger</mrk>.
```

2.6.3.1.2 Term Annotation

This annotation is used to mark up a term in the content, and possibly associate information to it.

Usage:

- The id attribute is required
- The type attribute is required and set to term
- The value attribute is optional and contains a short definition of the term
- The ref attribute is optional and contains a URI pointing to information on the term
- The translate attribute is optional and set to yes or no

For example:

```
<unit id="1">
 <segment>
  <source>He is my <mrk id="m1" type="term">doppelgänger</mrk>.</source>
  <target>Il est mon <mrk id="m1" type="term">alter ego</mrk>.</target>
 </segment>
</unit>
```

2.6.3.1.3 Comment Annotation

This annotation is used to associate a span of content with a comment.

Usage:

- The `id` attribute is required
- The `type` attribute is required and set to `comment`
- If the `value` attribute is present it contains the text of the comment, otherwise the `ref` attribute *must* be present and contains the id value (in URI format) of a `<note>` element that holds the comment.
- The `translate` attribute is optional and set to `yes` or `no`

For example, here with the `value` attribute:

```
The <mrk id="m1" type="comment"
 value="Possible values: Printer or Stacker"><ph id="1" dataRef="d1"/></mrk>
has been enabled.
```

And here using the `ref` attribute:

```
<unit id="1">
 <segment>
  <source>You use your own namespace.</source>
  <target>Vous pouvez utiliser votre propre <mrk id="m1"
type="comment" ref="#n1">namespace</mrk>.</target>
  <notes>
   <note id="n1" appliesTo="target">Please check the
translation for 'namespace'. On also can use 'espace de nom',
but I think most technical manuals use the English
term.</note>
  </notes>
 </segment>
</unit>
```

2.6.3.1.4 Custom Annotation

The `<mrk>` element can be used to implement custom annotations.

A custom annotation *must not* provide the same functionality as a pre-defined annotation.

Usage:

- The `id` attribute is required
- The `type` attribute is required and set to a unique user-defined value.
- The `translate` attribute is optional and set to `yes` or `no`
- The use and semantics of the `value` and `ref` attributes are user-defined.

For example:

```
One of the earliest surviving works of literature is
<mrk id="m1" type="myCorp:isbn" value="978-0-14-44919-8">The
Epic of Gilgamesh</mrk>.
```

2.6.3.2 Splitting Annotations

Annotations can overlap spanning inline codes or other annotations. They also can be split by segmentation. Because of this, a single annotation span can be represented using a pair of `<sm>` and `<em>` elements instead of a single `<mrk>` element.

For example, one can have the following content:

```
<unit id="1">
 <segment>
  <source>Sentence A. <mrk id="m1" type="comment"
value="Comment for B and C">Sentence B. Sentence C.</mrk></source>
 </segment>
</unit>
```

After a user agent performs segmentation, the annotation element `<mrk>` is changed to a pair of `<sm>` and `<em>` elements:

```
<unit id="1">
 <segment>
  <source>Sentence A. </source>
 </segment>
 <segment>
  <source><sm id="m1" type="comment"
value="Comment for B and C"/>Sentence B. </source>
 </segment>
 <segment>
  <source>Sentence C.<em startRef="m1"/></source>
 </segment>
</unit>
```

2.6.4 Sub-Flows

A sub-flow is a section of text embedded inside an inline code, or inside another section of text.

For example, the following HTML content includes two sub-flows: The first one is the value of the `title` attribute ("`Start button`"), and the second one is the value of the `alt` attribute ("`Click here to start!`"):

```
Click to start: <img title="Start button"
 src="btnStart.png" alt="Click here to start!"/>
```

Another example is the following DITA content where the footnote "`A Palouse horse is the same as an Appaloosa.`" is defined at the middle of a sentence:

```
Palouse horses<fn>A Palouse horse is the same as
 an Appaloosa.</fn> have spotted coats.
```

In XLIFF, each sub-flow is stored in its own `<unit>` element, and the `subFlows` attribute is used to indicate the location of the embedded content.

Therefore the HTML content of the example above can be represented like below:

```
<unit id="1">
 <segment>
  <source>Start button</source>
 </segment>
</unit>
<unit id="2">
 <segment>
  <source>Click here to start!</source>
```

```
 </segment>
</unit>
<unit id="3">
 <segment>
  <source>Click to start: <ph id="1" subFlows="1 2"/></source>
 </segment>
</unit>
```

Processing Requirements

- The extraction tool *should* store each sub-flow in its own `<unit>` element.
- An inline code containing or delimiting one or more sub-flows *must* have an attribute `subFlows` that holds a list of the identifiers of the `<unit>` elements where the sub-flows are stored.
- The extraction tool *may* order the `<unit>` elements of the sub-flows and the `<unit>` element of their parent in any order it sees fit. User agents coming later in the process *must* keep the `<unit>` elements in their initial order.

2.6.5 White Spaces

While white spaces can be significant or insignificant in the original format, they are always treated as significant when stored as original data in XLIFF.

Processing Requirements

- For the elements `<sc>`, `<ec>`, `<ph>` and `<data>`: The white spaces of their content *must* be preserved in all cases, even if the value for `xml:space` is set or inherited as `default`.
- For the inline content and all other inline elements: The white spaces *must* be preserved if the value for `xml:space` is set to `preserve`, and they *may* be preserved if the value is set to `default`.

2.6.6 Bidirectional Text

Text directionality in XLIFF content is defined by inheritance. Source and target content can have different directionality.

The initial directionality for both the source and the target content is defined in the `<file>` element, using the optional attributes `srcDir` for the source and `trgDir` for the target. The default value for both attributes is `ltr`.

The `<unit>` element has also the two optional attributes `srcDir` and `trgDir`. Their values *must* be either `ltr` or `rtl`.The default value of the `srcDir` is inherited from the value of the `srcDir` attribute in the parent `<file>` element. The default value of the `trgDir` attribute is inherited from the value of the `trgDir` attribute in the parent `<file>` element.

The `<source>` element has an optional attribute `dir` with a value `ltr` or `rtl`. The default value is inherited from the value of the `<unit>` in which the element is located.

The `<target>` element has an optional attribute `dir` with a value `ltr` or `rtl`. The default value is inherited from the value of the `<unit>` in which the element is located.

The `<pc>` and `<sc>` elements have an optional attribute `dir` with a value `ltr` or `rtl`. The default value is inherited from the parent `<source>` or `<target>` in which the element is located.

Adding bidirectional information in the text content is done using the Unicode bidirectional control characters.

In addition, the `<data>` element has an optional attribute `dir` with a value `ltr` or `rtl` that is not inherited. The default value is `ltr`.

2.6.7 Target Content Modification

This section defines the rules tools need to follow when working with the target content of a given segment in order to provide interoperability throughout the whole process.

The extraction tool can create the initial target content as it sees fit.

The merging tool is assumed to have the same level of processing and native format knowledge as the extraction tool. Providing an interoperable way to convert native documents into XLIFF with one tool and back to the native format with another tool without the same level of knowledge is outside the scope of this specification.

The user agents modifying the target content of a document between the extraction and the merging tools ensure interoperability by applying specific rules. These rules are separated into two cases: When there is an existing target and when there is no existing target.

2.6.7.1 Without an Existing Target

When there is no existing target, the processing requirements for a given segment are the following:

Processing Requirements

- User agents *may* leave the segment without a target.
- User agents *may* create a new target as follow:
  - User agents *may* add translatable text.
  - User agents *must* put all non-removable inline codes in the target.
  - User agents *must* preserve the order of all the non-reorderable inline codes.
  - User agents *may* put any removable inline code in the target.
  - User agents *may* add inline codes.
  - User agents *may* add or remove annotations.
  - User agents *may* split the segment into two segments.
  - User agents *may* join the segment with the following one.
  - User agents *may* convert any `<pc>` element into a pair of `<sc>` and `<ec>` elements.
  - User agents *may* convert, if it is possible, any pair or `<sc>` and `<ec>` elements into a `<pc>` element.

2.6.7.2 With an Existing Target

When working with a segment with content already in the target, user agents *must* choose one of the three behaviors described below:

Processing Requirements

- User agents *may* leave the existing target unchanged.
- User agents *may* modify the existing target as follow:
  - User agents *may* add or modify translatable text.
  - User agents *must* preserve all non-removable inline codes, regardless whether or not they exist in the source.

    - User agents *must* preserve any non-reorderable inline codes in the existing target.
    - User agents *must not* add any non-reorderable inline codes to the target.
    - User agents *must not* split the segment. The reason to not allow splitting of segments with content in the target node is because there is no guarantee that the content in the two nodes are linguistically in the same order, performing that operation would pose a risk to the integrity of the content.
    - User agents *may* remove any removable inline codes in the target.
    - User agents *may* add inline codes (including copying any cloneable inline codes of the existing target).
    - User agents *may* add or remove annotations.
    - User agents *may* join the segment with the following segment.
    - User agents *may* convert any `<pc>` element into a pair of `<sc>` and `<ec>` elements.
    - User agents *may* convert, if it is possible, any pair or `<sc>` and `<ec>` elements into a `<pc>` element.
- User agents *may* delete the existing target and start over as if working without an existing target.

2.6.8 Content Comparison

This specification defines two types of content equality:

- Equality type A: Two contents are equals if their normalized forms are equal.
- Equality type B: Two contents are equals if, in their normalized forms and with all inline code markers replaced by the value of their `equiv` attributes, the resulting strings are equal.

A content is normalized when:

- The text nodes are in Unicode Normalized Form C defined in the Unicode Annex #15: Unicode Normalization Forms [LDML].
- All annotation markers are removed.
- All pairs of `<sc>` and `<ec>` elements that can be converted into a `<pc>` element, are converted.
- All adjacent text nodes are merged into a single text node.
- For all the text nodes with the white space property set to `default`, all adjacent white spaces are collapsed into a single space.

2.7 Extension Mechanisms

XLIFF 2.0 offers two mechanisms for storing custom data in an XLIFF document:

1. Using the Metadata module for storing custom data in elements defined by the official XLIFF specification.
2. Using the standard XML namespace mechanism for storing data in elements or attributes defined in a custom XML Schema.

Both mechanisms can be used simultaneously.

2.7.1 Extension Points

The following elements allow storing custom data in <mda:metadata> elements or in elements from a custom XML namespace:

- <file>
- <group>
- <unit>

Custom extensions using <mda:metadata> are allowed in:

-
- <ignorable>
- <mtc:match>

The following elements accept custom attributes:

- <file>
- <group>
- <unit>
-
- <ignorable>
- <source>
- <target>
- <note>
- <mrk>
- <sm>

2.7.2 Processing Requirements

- A user extension, whether implemented using <mda:metadata> or using a custom namespace, *must not* provide the same functionality as an existing XLIFF core or module feature, however it *may* complement an extensible XLIFF core feature or module feature or provide a new funtionality at provided extensin points.
- Tools *must not* rely on user extensions (either in the Metadata module or custim namespace based) other than the ones possibly defined in <skeleton> to create the translated version of the original document.
- User agents that do not support a given custom namespace based user extension *should* preserve that extension without modifications, except if the element where the extension is located is removed.

2.8 Segmentation

In the context of XLIFF, a segment is a content which is either a unit of extracted text, or has been created from a unit of extracted text by means of a segmentation mechanism such as sentence boundary detection. For example, a segment can be a title, the text of a menu item, a paragraph or a sentence in a paragraph.

In the context of XLIFF, other types representations sometimes called "segmentation" can be represented using annotations. For example: the terms in a segment can be identified and marked up using the term annotation.

XLIFF does not specify how segmentation is carried out, only how to represent its result. The Segmentation Rules eXchange standard [SRX] provides detailed information about segmenting a content.

2.8.1 Segments Representation

In XLIFF each segment is represented by a element.

A <unit> can be made of a single .

Each element has one <source> element that contains the source content and one optional <target> element that can be empty or contain the translation of the source content at a given state.

Content parts between segments are represented with the <ignorable> element, which has the same content model as .

For example:

```
<unit id="1">
 <segment>
  <source>First sentence.</source>
  <target>Première phrase.</target>
 </segment>
 <ignorable>
  <source> </source>
 </ignorable>
 <segment>
  <source>Second sentence.</source>
 </segment>
</unit>
```

2.8.2 Segments Order

Some applications (e.g. aligner tools) may create segmented content where the target segments are not in the same order as the source.

To be able to map order differences, the <target> element has an optional order attribute that indicates its position in the sequence of segments (and inter-segments). Its value is an integer from 1 to N, including the parts between segments.

For example, the following HTML documents have the same paragraph with three sentences in different order:

```
<p lang='en'>Sentence A. Sentence B. Sentence C.</p>
```

```
<p lang='fr'>Phrase B. Phrase C. Phrase A.</p>
```

The XLIFF representation of the content, after segmentation and alignment, would be:

```
<unit id="1">
 <segment id="1">
  <source>Sentence A.</source>
  <target order="5">Phrase A.</target>
 </segment>
 <ignorable>
  <source> </source>
 </ignorable>
 <segment id="2">
  <source>Sentence B.</source>
  <target order="1">Phrase B.</target>
 </segment>
```

```
 <ignorable>
  <source> </source>
 </ignorable>
 <segment id="3">
  <source>Sentence C.</source>
  <target order="3">Phrase C.</target>
 </segment>
</unit>
```

2.8.3 Segmentation Modification

Segments within the same `<unit>` may be split, joined or moved as needed.

When modifying the segmentation, the user agents must make sure the integrity of the relations between segments and associated data (such as candidate translations, comments, etc.) is preserved.

Processing Requirements

- When merging, joining, or moving `<segment>` or `<ignorable>` elements, the user agent *must* update, add or remove the `order` attributes of the `<target>` elements as needed.
- [[TODO: PRs for how elements and attributes in segment, source, target should behave when re-segmenting.]]

3 Conformance

XLIFF is an XML vocabulary, thereafter conformant XLIFF documents *must* be well formed and valid XML documents.

Conformant XLIFF documents *must* be valid instances of the official Core XML Schema that is part of this XLIFF specification.

As not all aspects of the XLIFF specification can be expressed in terms of XML Schemas, conformant XLIFF documents *must* also comply with all requirements indicated in this specification document.

Applications that generate XLIFF documents *must* generate conformant XLIFF documents to be considered XLIFF compliant.

XLIFF documents *may* contain custom extensions, as defined in the Extension Mechanisms section. Applications that process conformant XLIFF files that contain custom extensions are not *required* to understand and process non-XLIFF elements or attributes. However, conformant applications *should* preserve existing custom extensions when processing conformant XLIFF files, provided that the elements that contain custom extensions are not removed according to XLIFF processing requirements or the extension's own processing requirements.

XLIFF is a format explicitly designed for exchanging data. Thus, a conformant XLIFF application *must* be able to accept XLIFF files it generated after those XLIFF files have been processed by a different tool, provided that the processed files are conformant XLIFF documents.

Appendix A XML Schemas and Catalog

The grammar of XLIFF 2.0 is defined using eight (8) XML Schemas and one (1) XML catalog. The module schemas are referenced from their respective modules.

A.1 XML Schemas Tree

```
Core XML Schema
   |
   +---Candidates Module XML Schema
   |
   +---Glossary Module XML Schema

   |
   +---Metadatata Module XML Schema
   |
   +---Resource Data Module XML Schema
   |
   +---Change Tracking Module XML Schema
   |
   +---Size and Length Restriction Module XML Schema
   |
   +---Validation Module XML Schema
```

A.2 XML Catalog

The catalog listed below for reading convenience is accessible at http://docs.oasis-open.org/xliff/xliff-core/v2.0/csd01/schemas/catalog.xml.

```
<?xml version="1.0" encoding="UTF-8"?>
<catalog xmlns="urn:oasis:names:tc:entity:xmlns:xml:catalog">
    <uri name="urn:oasis:names:tc:xliff:document:2.0" uri="xliff_core_2.0.xsd"/>
    <uri name="urn:oasis:names:tc:xliff:changetracking:2.0" uri="modules/change_tracking
    <uri name="urn:oasis:names:tc:xliff:glossary:2.0" uri="modules/glossary.xsd"/>
    <uri name="urn:oasis:names:tc:xliff:matches:2.0" uri="modules/matches.xsd"/>
    <uri name="urn:oasis:names:tc:xliff:metadata:2.0" uri="modules/metadata.xsd"/>
    <uri name="urn:oasis:names:tc:xliff:resourcedata:2.0" uri="modules/resource_data.xs
    <uri name="urn:oasis:names:tc:xliff:sizerestriction:2.0" uri="modules/size_restrict
    <uri name="urn:oasis:names:tc:xliff:validation:2.0" uri="modules/validation.xsd"/>
</catalog>
```

A.3 Core XML Schema

The schema listed below for reading convenience is accessible at http://docs.oasis-open.org/xliff/xliff-core/v2.0/csd01/schemas/xliff_core_2.0.xsd.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified"
```

```
  xmlns:xlf="urn:oasis:names:tc:xliff:document:2.0"
  xmlns:ctr="urn:oasis:names:tc:xliff:changetracking:2.0"
  xmlns:gls="urn:oasis:names:tc:xliff:glossary:2.0"
  xmlns:mtc="urn:oasis:names:tc:xliff:matches:2.0"
  xmlns:mda="urn:oasis:names:tc:xliff:metadata:2.0"
  xmlns:res="urn:oasis:names:tc:xliff:resourcedata:2.0"
  xmlns:slr="urn:oasis:names:tc:xliff:sizerestriction:2.0"
  xmlns:val="urn:oasis:names:tc:xliff:validation:2.0"
  targetNamespace="urn:oasis:names:tc:xliff:document:2.0">

  <xs:import namespace="http://www.w3.org/XML/1998/namespace"
    schemaLocation="http://www.w3.org/2001/xml.xsd"/>
  <xs:import namespace="urn:oasis:names:tc:xliff:changetracking:2.0"
    schemaLocation="modules/change_tracking.xsd"/>
  <xs:import namespace="urn:oasis:names:tc:xliff:glossary:2.0"
    schemaLocation="modules/glossary.xsd"/>
  <xs:import namespace="urn:oasis:names:tc:xliff:matches:2.0"
    schemaLocation="modules/matches.xsd"/>
  <xs:import namespace="urn:oasis:names:tc:xliff:metadata:2.0"
    schemaLocation="modules/metadata.xsd"/>
  <xs:import namespace="urn:oasis:names:tc:xliff:resourcedata:2.0"
    schemaLocation="modules/resource_data.xsd"/>
  <xs:import namespace="urn:oasis:names:tc:xliff:sizerestriction:2.0"
    schemaLocation="modules/size_restriction.xsd"/>
  <xs:import namespace="urn:oasis:names:tc:xliff:validation:2.0"
    schemaLocation="modules/validation.xsd"/>

  <!-- Attribute definitions -->

  <xs:simpleType name="yesNo">
    <xs:restriction base="xs:string">
      <xs:enumeration value="yes"/>
      <xs:enumeration value="no"/>
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="dirValue">
    <xs:restriction base="xs:string">
      <xs:enumeration value="ltr"/>
      <xs:enumeration value="rtl"/>
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="appliesTo">
    <xs:restriction base="xs:string">
      <xs:enumeration value="source"/>
      <xs:enumeration value="target"/>
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="userDefinedValue">
    <xs:restriction base="xs:string">
      <xs:pattern value="[^\s:]+:[^\s:]+"/>
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="attrType_type">
    <xs:restriction base="xs:string">
```



```
    <xs:pattern value="(fmt|quot|link|image|other)"/>
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="typeForMrkValues">
  <xs:restriction base="xs:NMTOKEN">
    <xs:enumeration value="generic"/>
    <xs:enumeration value="comment"/>
    <xs:enumeration value="term"/>
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="attrType_typeForMrk">
  <xs:union memberTypes="xlf:typeForMrkValues xlf:userDefinedValue"/>
</xs:simpleType>

<!-- Structural elements -->

<xs:element name="xliff">
  <xs:complexType>
    <xs:sequence>
      <xs:element minOccurs="1" maxOccurs="unbounded" ref="xlf:file"/>
    </xs:sequence>
    <xs:attribute name="version" fixed="2.0" use="required"/>
    <xs:attribute name="srcLang" use="required"/>
    <xs:attribute name="tgtLang" use="optional"/>
    <xs:anyAttribute namespace="##any" processContents="skip"/>
  </xs:complexType>
</xs:element>

<xs:element name="file">
  <xs:complexType>
    <xs:sequence>
      <xs:element minOccurs="0" maxOccurs="1" ref="xlf:skeleton"/>
      <xs:element minOccurs="0" maxOccurs="1" ref="gls:glossary"/>
      <xs:element minOccurs="0" maxOccurs="1" ref="mda:metadata"/>
      <xs:element minOccurs="0" maxOccurs="1" ref="slr:profiles"/>
      <xs:element minOccurs="0" maxOccurs="1" ref="slr:data"/>
      <xs:element minOccurs="0" maxOccurs="unbounded" ref="res:resourceData"/>
      <xs:element minOccurs="0" maxOccurs="1" ref="validation:validation"/>
      <xs:any maxOccurs="unbounded" minOccurs="0" namespace="##other"
        processContents="skip"/>
      <xs:choice minOccurs="1" maxOccurs="unbounded">
        <xs:element ref="xlf:unit"/>
        <xs:element ref="xlf:group"/>
      </xs:choice>
    </xs:sequence>
    <xs:attribute name="id" use="optional" type="xs:NMTOKEN"/>
    <xs:attribute name="original" use="optional"/>
    <xs:attribute name="srcDir" use="optional" type="xlf:dirValue"
      default="ltr"/>
    <xs:attribute name="trgDir" use="optional" type="xlf:dirValue"
      default="ltr"/>
    <xs:attribute ref="fs:fs" use="optional"/>
    <xs:attribute ref="fs:subFs" use="optional"/>
    <xs:attribute ref="slr:storageRestriction" use="optional"/>
    <xs:attribute ref="slr:sizeRestriction" use="optional"/>
    <xs:attribute ref="slr:sizeInfo" use="optional"/>
```

```
    <xs:attribute ref="slr:sizeInfoRef" use="optional"/>
    <xs:anyAttribute namespace="##any" processContents="skip"/>
  </xs:complexType>
</xs:element>

<xs:element name="skeleton">
  <xs:complexType mixed="true">
    <xs:sequence>
      <xs:any maxOccurs="unbounded" minOccurs="0" namespace="##other"
        processContents="skip"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="group">
  <xs:complexType>
    <xs:sequence>
      <xs:choice minOccurs="1" maxOccurs="unbounded">
        <xs:element ref="xlf:unit"/>
        <xs:element ref="xlf:group"/>
      </xs:choice>
      <xs:element minOccurs="0" maxOccurs="1" ref="mda:metadata"/>
      <xs:element minOccurs="0" maxOccurs="1" ref="slr:data"/>
      <xs:element minOccurs="0" maxOccurs="1" ref="validation:validation"/>
      <xs:any maxOccurs="unbounded" minOccurs="0" namespace="##other"
        processContents="skip"/>
    </xs:sequence>
    <xs:attribute name="id" use="optional" type="xs:NMTOKEN"/>
    <xs:attribute name="name" use="optional"/>
    <xs:attribute name="srcDir" use="optional" type="xlf:dirValue"/>
    <xs:attribute name="trgDir" use="optional" type="xlf:dirValue"/>
    <xs:attribute ref="fs:fs" use="optional"/>
    <xs:attribute ref="fs:subFs" use="optional"/>
    <xs:attribute ref="slr:storageRestriction" use="optional"/>
    <xs:attribute ref="slr:sizeRestriction" use="optional"/>
    <xs:attribute ref="slr:sizeInfo" use="optional"/>
    <xs:attribute ref="slr:sizeInfoRef" use="optional"/>
    <xs:anyAttribute namespace="##any" processContents="skip"/>
  </xs:complexType>
</xs:element>

<xs:element name="unit">
  <xs:complexType>
    <xs:sequence>
      <xs:choice minOccurs="1" maxOccurs="unbounded">
        <xs:element ref="xlf:segment"/>
        <xs:element ref="xlf:ignorable"/>
      </xs:choice>
      <xs:element minOccurs="0" maxOccurs="1" ref="xlf:notes"/>
      <xs:element minOccurs="0" maxOccurs="1" ref="xlf:originalData"/>
      <xs:element minOccurs="0" maxOccurs="1" ref="mtc:matches"/>
      <xs:element minOccurs="0" maxOccurs="1" ref="gls:glossary"/>
      <xs:element minOccurs="0" maxOccurs="1" ref="mda:metadata"/>
      <xs:element minOccurs="0" maxOccurs="1" ref="slr:data"/>
      <xs:element minOccurs="0" maxOccurs="1" ref="validation:validation"/>
      <xs:any maxOccurs="unbounded" minOccurs="0" namespace="##other"
        processContents="skip"/>
    </xs:sequence>
```

```
    <xs:attribute name="id" use="required" type="xs:NMTOKEN"/>
    <xs:attribute name="name" use="optional" />
    <xs:attribute name="translate" use="optional" type="xlf:yesNo" default="yes"/>
    <xs:attribute name="srcDir" use="optional" type="xlf:dirValue"/>
    <xs:attribute name="trgDir" use="optional" type="xlf:dirValue"/>
    <xs:attribute ref="fs:fs" use="optional"/>
    <xs:attribute ref="fs:subFs" use="optional"/>
    <xs:attribute ref="slr:storageRestriction" use="optional"/>
    <xs:attribute ref="slr:sizeRestriction" use="optional"/>
    <xs:attribute ref="slr:sizeInfo" use="optional"/>
    <xs:attribute ref="slr:sizeInfoRef" use="optional"/>
    <xs:anyAttribute namespace="##any" processContents="skip"/>
  </xs:complexType>
</xs:element>

<xs:element name="segment">
  <xs:complexType>
    <xs:sequence>
      <xs:element minOccurs="1" maxOccurs="1" ref="xlf:source"/>
      <xs:element minOccurs="0" maxOccurs="1" ref="xlf:target"/>
      <xs:element minOccurs="0" maxOccurs="1" ref="xlf:notes"/>
      <xs:element minOccurs="0" maxOccurs="1" ref="mtc:matches"/>
      <xs:element minOccurs="0" maxOccurs="1" ref="ctr:changeTrack"/>
      <xs:element minOccurs="0" maxOccurs="1" ref="validation:validation"/>
      <xs:element minOccurs="0" maxOccurs="1" ref="mda:metadata"/>
    </xs:sequence>
    <xs:attribute name="id" use="optional" type="xs:NMTOKEN"/>
    <xs:attribute name="translate" use="optional" type="xlf:yesNo" default="yes"/>
    <xs:attribute name="approved" type="xlf:yesNo" default="no"/>
    <xs:attribute name="state" use="optional" default="initial"/>
    <xs:attribute name="subState" use="optional"/>
    <xs:anyAttribute namespace="##any" processContents="skip"/>
    <xs:attribute ref="fs:fs" use="optional"/>
    <xs:attribute ref="fs:subFs" use="optional"/>
  </xs:complexType>
</xs:element>

<xs:element name="ignorable">
  <xs:complexType>
    <xs:sequence>
      <xs:element minOccurs="1" maxOccurs="1" ref="xlf:source"/>
      <xs:element minOccurs="0" maxOccurs="1" ref="xlf:target"/>
      <xs:element minOccurs="0" maxOccurs="1" ref="mda:metadata"/>
    </xs:sequence>
    <xs:attribute name="id" use="optional" type="xs:NMTOKEN"/>
    <xs:anyAttribute namespace="##any" processContents="skip"/>
    <xs:attribute ref="fs:fs" use="optional"/>
    <xs:attribute ref="fs:subFs" use="optional"/>
  </xs:complexType>
</xs:element>

<xs:element name="originalData">
  <xs:complexType>
    <xs:sequence>
      <xs:element minOccurs="1" maxOccurs="unbounded" ref="xlf:data"/>
    </xs:sequence>
    <xs:attribute ref="fs:fs" use="optional"/>
    <xs:attribute ref="fs:subFs" use="optional"/>
```

```
  </xs:complexType>
</xs:element>

<xs:element name="data">
  <xs:complexType mixed="true">
    <xs:sequence>
      <xs:element minOccurs="0" maxOccurs="unbounded" ref="xlf:cp"/>
    </xs:sequence>
    <xs:attribute name="id" use="required" type="xs:NMTOKEN"/>
    <xs:attribute name="dir" use="optional" type="xlf:dirValue"/>
    <xs:attribute ref="fs:fs" use="optional"/>
    <xs:attribute ref="fs:subFs" use="optional"/>
  </xs:complexType>
</xs:element>

<xs:element name="notes">
  <xs:complexType>
    <xs:sequence>
      <xs:element minOccurs="1" maxOccurs="unbounded" ref="xlf:note"/>
    </xs:sequence>
    <xs:attribute ref="fs:fs" use="optional"/>
    <xs:attribute ref="fs:subFs" use="optional"/>
  </xs:complexType>
</xs:element>

<xs:element name="note">
  <xs:complexType mixed="true">
    <xs:attribute name="id" use="optional" type="xs:NMTOKEN"/>
    <xs:attribute name="appliesTo" use="optional" type="xlf:appliesTo"/>
    <xs:attribute name="category" use="optional"/>
    <xs:attribute name="priority" use="optional"/>
    <xs:anyAttribute namespace="##any" processContents="skip"/>
    <xs:attribute ref="fs:fs" use="optional"/>
    <xs:attribute ref="fs:subFs" use="optional"/>
  </xs:complexType>
</xs:element>

<xs:element name="source">
  <xs:complexType mixed="true">
    <xs:group ref="xlf:inline" minOccurs="0" maxOccurs="unbounded"/>
    <xs:attribute ref="xml:lang" use="optional"/>
    <xs:attribute ref="xml:space" use="optional"/>
    <xs:attribute name="dir" use="optional" type="xlf:dirValue"/>
    <xs:anyAttribute namespace="##any" processContents="skip"/>
    <xs:attribute ref="fs:fs" use="optional"/>
    <xs:attribute ref="fs:subFs" use="optional"/>
  </xs:complexType>
</xs:element>

<xs:element name="target">
  <xs:complexType mixed="true">
    <xs:group ref="xlf:inline" minOccurs="0" maxOccurs="unbounded"/>
    <xs:attribute ref="xml:lang" use="optional"/>
    <xs:attribute ref="xml:space" use="optional"/>
    <xs:attribute name="dir" use="optional" type="xlf:dirValue"/>
    <xs:attribute ref="fs:fs" use="optional"/>
    <xs:attribute ref="fs:subFs" use="optional"/>
    <xs:anyAttribute namespace="##any" processContents="skip"/>
```

```
  </xs:complexType>
</xs:element>

<!-- Inline elements -->

<xs:group name="inline">
  <xs:choice>
    <xs:element minOccurs="0" maxOccurs="unbounded" ref="xlf:sc"/>
    <xs:element minOccurs="0" maxOccurs="unbounded" ref="xlf:ec"/>
    <xs:element minOccurs="0" maxOccurs="unbounded" ref="xlf:ph"/>
    <xs:element minOccurs="0" maxOccurs="unbounded" ref="xlf:pc"/>
    <xs:element minOccurs="0" maxOccurs="unbounded" ref="xlf:cp"/>
    <xs:element minOccurs="0" maxOccurs="unbounded" ref="xlf:mrk"/>
    <xs:element minOccurs="0" maxOccurs="unbounded" ref="xlf:sm"/>
    <xs:element minOccurs="0" maxOccurs="unbounded" ref="xlf:em"/>
  </xs:choice>
</xs:group>

<xs:element name="sc">
  <!-- Start Code -->
  <xs:complexType mixed="false">
    <xs:attribute name="canCopy" use="optional" type="xlf:yesNo" default="yes"/>
    <xs:attribute name="canDelete" use="optional" type="xlf:yesNo" default="yes"/>
    <xs:attribute name="canOverlap" use="optional" type="xlf:yesNo"/>
    <xs:attribute name="canReorder" use="optional" type="xlf:yesNo" default="yes"/>
    <xs:attribute name="copyOf" use="optional" type="xs:NMTOKEN"/>
    <xs:attribute name="disp" use="optional"/>
    <xs:attribute name="equiv" use="optional"/>
    <xs:attribute name="id" use="required" type="xs:NMTOKEN"/>
    <xs:attribute name="isolated" use="optional" type="xlf:yesNo" default="no"/>
    <xs:attribute name="dataRef" use="optional"/>
    <xs:attribute name="subFlows" use="optional" type="xs:NMTOKENS"/>
    <xs:attribute name="subType" use="optional" type="xlf:userDefinedValue"/>
    <xs:attribute name="type" use="optional" type="xlf:attrType_type"/>
    <xs:attribute name="dir" use="optional" type="xlf:dirValue"/>
    <xs:attribute ref="fs:fs" use="optional"/>
    <xs:attribute ref="fs:subFs" use="optional"/>
    <xs:attribute ref="slr:storageRestriction" use="optional"/>
    <xs:attribute ref="slr:sizeRestriction" use="optional"/>
    <xs:attribute ref="slr:equivStorage" use="optional"/>
    <xs:attribute ref="slr:sizeInfo" use="optional"/>
    <xs:attribute ref="slr:sizeInfoRef" use="optional"/>
  </xs:complexType>
</xs:element>

<xs:element name="ec">
  <!-- End Code -->
  <xs:complexType mixed="false">
    <xs:attribute name="canCopy" use="optional" type="xlf:yesNo" default="yes"/>
    <xs:attribute name="canDelete" use="optional" type="xlf:yesNo" default="yes"/>
    <xs:attribute name="canOverlap" use="optional" type="xlf:yesNo"/>
    <xs:attribute name="canReorder" use="optional" type="xlf:yesNo" default="yes"/>
    <xs:attribute name="copyOf" use="optional" type="xs:NMTOKEN"/>
    <xs:attribute name="disp" use="optional"/>
    <xs:attribute name="equiv" use="optional"/>
    <xs:attribute name="id" use="optional" type="xs:NMTOKEN"/>
    <xs:attribute name="isolated" use="optional" type="xlf:yesNo" default="no"/>
    <xs:attribute name="dataRef" use="optional"/>
```

```
      <xs:attribute name="startRef" use="optional"/>
      <xs:attribute name="subFlows" use="optional" type="xs:NMTOKENS"/>
      <xs:attribute name="subType" use="optional" type="xlf:userDefinedValue"/>
      <xs:attribute name="type" use="optional" type="xlf:attrType_type"/>
      <xs:attribute ref="fs:fs" use="optional"/>
      <xs:attribute ref="fs:subFs" use="optional"/>
      <xs:attribute ref="slr:equivStorage" use="optional"/>
      <xs:attribute ref="slr:sizeInfo" use="optional"/>
      <xs:attribute ref="slr:sizeInfoRef" use="optional"/>
    </xs:complexType>
  </xs:element>

  <xs:element name="ph">
    <!-- Placeholder -->
    <xs:complexType mixed="false">
      <xs:attribute name="canCopy" use="optional" type="xlf:yesNo" default="yes"/>
      <xs:attribute name="canDelete" use="optional" type="xlf:yesNo" default="yes"/>
      <xs:attribute name="canReorder" use="optional" type="xlf:yesNo" default="yes"/>
      <xs:attribute name="copyOf" use="optional" type="xs:NMTOKEN"/>
      <xs:attribute name="disp" use="optional"/>
      <xs:attribute name="equiv" use="optional"/>
      <xs:attribute name="id" use="required" type="xs:NMTOKEN"/>
      <xs:attribute name="dataRef" use="optional"/>
      <xs:attribute name="subFlows" use="optional" type="xs:NMTOKENS"/>
      <xs:attribute name="subType" use="optional" type="xlf:userDefinedValue"/>
      <xs:attribute name="type" use="optional" type="xlf:attrType_type"/>
      <xs:attribute ref="fs:fs" use="optional"/>
      <xs:attribute ref="fs:subFs" use="optional"/>
      <xs:attribute ref="slr:equivStorage" use="optional"/>
      <xs:attribute ref="slr:sizeInfo" use="optional"/>
      <xs:attribute ref="slr:sizeInfoRef" use="optional"/>
    </xs:complexType>
  </xs:element>

  <xs:element name="pc">
    <!-- Paired Code -->
    <xs:complexType mixed="true">
      <xs:group ref="xlf:inline" minOccurs="0" maxOccurs="unbounded"/>
      <xs:attribute name="canCopy" use="optional" type="xlf:yesNo" default="yes"/>
      <xs:attribute name="canDelete" use="optional" type="xlf:yesNo" default="yes"/>
      <xs:attribute name="canOverlap" use="optional" type="xlf:yesNo"/>
      <xs:attribute name="canReorder" use="optional" type="xlf:yesNo" default="yes"/>
      <xs:attribute name="copyOf" use="optional" type="xs:NMTOKEN"/>
      <xs:attribute name="dispEnd" use="optional"/>
      <xs:attribute name="dispStart" use="optional"/>
      <xs:attribute name="equivEnd" use="optional"/>
      <xs:attribute name="equivStart" use="optional"/>
      <xs:attribute name="id" use="required" type="xs:NMTOKEN"/>
      <xs:attribute name="dataRefEnd" use="optional"/>
      <xs:attribute name="dataRefStart" use="optional"/>
      <xs:attribute name="subFlowsEnd" use="optional" type="xs:NMTOKENS"/>
      <xs:attribute name="subFlowsStart" use="optional" type="xs:NMTOKENS"/>
      <xs:attribute name="subType" use="optional" type="xlf:userDefinedValue"/>
      <xs:attribute name="type" use="optional" type="xlf:attrType_type"/>
      <xs:attribute name="dir" use="optional" type="xlf:dirValue"/>
      <xs:attribute ref="fs:fs" use="optional"/>
      <xs:attribute ref="fs:subFs" use="optional"/>
      <xs:attribute ref="slr:storageRestriction" use="optional"/>
```



```
    <xs:attribute ref="slr:sizeRestriction" use="optional"/>
    <xs:attribute ref="slr:equivStorage" use="optional"/>
    <xs:attribute ref="slr:sizeInfo" use="optional"/>
    <xs:attribute ref="slr:sizeInfoRef" use="optional"/>
  </xs:complexType>
</xs:element>

<xs:element name="cp">
  <!-- Code Point -->
  <xs:complexType mixed="false">
    <xs:attribute name="hex" use="required"/>
    <xs:attribute ref="fs:fs" use="optional"/>
    <xs:attribute ref="fs:subFs" use="optional"/>
  </xs:complexType>
</xs:element>

<xs:element name="mrk">
  <!-- Annotation Marker -->
  <xs:complexType mixed="true">
    <xs:group ref="xlf:inline" minOccurs="0" maxOccurs="unbounded"/>
    <xs:attribute name="id" use="required" type="xs:NMTOKEN"/>
    <xs:attribute name="translate" use="optional" type="xlf:yesNo" default="yes"/>
    <xs:attribute name="type" use="optional" type="xlf:attrType_type"/>
    <xs:attribute name="ref" use="optional" type="xs:anyURI"/>
    <xs:attribute name="value" use="optional"/>
    <xs:attribute ref="fs:fs" use="optional"/>
    <xs:attribute ref="fs:subFs" use="optional"/>
    <xs:attribute ref="slr:storageRestriction" use="optional"/>
    <xs:attribute ref="slr:sizeRestriction" use="optional"/>
    <xs:anyAttribute namespace="##any" processContents="skip"/>
  </xs:complexType>
</xs:element>

<xs:element name="sm">
  <!-- Start Annotation Marker -->
  <xs:complexType mixed="false">
    <xs:attribute name="id" use="required" type="xs:NMTOKEN"/>
    <xs:attribute name="translate" use="optional" type="xlf:yesNo" default="yes"/>
    <xs:attribute name="type" use="optional" type="xlf:attrType_type"/>
    <xs:attribute name="ref" use="optional" type="xs:anyURI"/>
    <xs:attribute name="value" use="optional"/>
    <xs:attribute ref="fs:fs" use="optional"/>
    <xs:attribute ref="fs:subFs" use="optional"/>
    <xs:attribute ref="slr:storageRestriction" use="optional"/>
    <xs:attribute ref="slr:sizeRestriction" use="optional"/>
    <xs:anyAttribute namespace="##any" processContents="skip"/>
  </xs:complexType>
</xs:element>

<xs:element name="em">
  <!-- End Annotation Marker -->
  <xs:complexType mixed="false">
    <xs:attribute name="startRef" use="required"/>
    <xs:attribute ref="fs:fs" use="optional"/>
    <xs:attribute ref="fs:subFs" use="optional"/>
  </xs:complexType>
</xs:element>
```

```
</xs:schema>
```

Appendix B Translation Candidates Module

The source text of a document can be pre-processed against various translation resources (TM, MT, etc.) to provide translation candidates. This module provides an XLIFF capability to store lists of possible translations along with information about the similarity of the match, the quality of the translation, its provenance, etc.

B.1 Module Specification

B.1.1 Module Namespace

The namespace for the Translation Candidates module is: `urn:oasis:names:tc:xliff:matches:2.0`

B.1.2 Module Elements

The elements defined in the Translation Candidates module are: `<matches>` and `<match>`.

B.1.2.1 Tree Structure

Legend:

1 = one
+ = one or more
? = zero or one

```
<matches> 1
|
+---<match> +
    |
    +---<xlf:source> 1
    |
    +---<xlf:target> 1
    |
    +---<xlf:originalData> ?
    |   |
    |   +---<xlf:data> +
    |
    +---<mda:metadata> ?
```

B.1.2.2 matches

Collection of matches retrieved from any leveraging system (MT, TM, etc.)

Contains:

- One or more `<match>` elements

B.1.2.3 match

A potential translation suggested for the enclosing `<unit>` or `<segment>` element.

Contains:

- One <source> element followed by
- One <target> element followed by
- Zero or one <originalData> element followed by
- Zero or one <mda:metadata> element.

Attributes:

- id, optional
- origin, optional
- similarity, optional
- matchQuality, optional
- matchSuitability, optional
- type, optional
- reference, optional
- attributes from any namespace, optional

Processing Requirements

- When a <target> element is a child of <match> and the reference attribute is set to "yes", the optional xml:lang attribute's value does not need to be equal to the value of the tgtLang attribute of the enclosing <xliff> element.

B.1.3 Module Attributes

The attributes defined in the Translation Candidates module are: id, similarity, matchQuality, matchSuitability, origin, type, subtype and reference.

B.1.3.1 id

Identifier - A character string used to identify a <match> element.

Value description: NMTOKEN.

Default value: undefined

The value must be unique within the <matches> element.

Used in: <match>.

B.1.3.2 similarity

Similarity - indicates the similarity level between the content of the <source> child of a <match> element and the translatable text being matched.

Value description: a decimal number between 0.0 and 100.0.

Default value: undefined

Used in: <match>.

B.1.3.3 matchQuality

Match quality - indicates the quality of the <target> child of a <match> element based on an external benchmark or metric.

Value description: a decimal number between 0.0 and 100.0.

Default value: undefined

Used in: <match>.

Note

This attribute can carry a human review based metrics score, a Machine Translation self-reported confidence score etc.

B.1.3.4 matchSuitability

Match suitability - indicates the general suitability and relevance of its <match> element with the regard to populating the <target> child of the enclosing element based on various external benchmarks or metrics pertaining to both the <source> child of the enclosing and the <target> child of the <match>.

This attribute is intended to carry a value that can be combined from values provided in similarity and matchQuality attributes based on an external algorithm.

Value description: a decimal number between 0.0 and 100.0.

Default value: undefined

Used in: <match>.

Note

This attribute is also useful for mapping match-quality as specified in XLIFF 1.2.

Processing Requirements

- Tools processing this module *must* make use of matchSuitability for match ordering purposes if the attribute is specified.

B.1.3.5 origin

Match origin - indicates the data origin of a <match> element.

Value description: Text.

Default value: undefined

Used in: <match>.

B.1.3.6 type

Type - Indicates the type of a <match> element, it gives the value providing additional information on how the match was generated or qualifying further the relevance of the match. The list of pre-defined values is general and user-specific information can be added using the subtype attribute.

Value description:

Table B.1. Standard Values

Value	Description
am	Assembled Match: match generated by assembling parts of different translations.
mt	Machine Translation: candidate generated by a machine translation system.
icm	In Context Match: match for which the context is the same as the context of the source content. E.g. the source text for both contents is also preceded by an identical source segment, or both appear as level 2 headings.
idm	Identifier-based Match: match that has an identifier identical to the source content. For example the previous translation of a given UI component with the same ID.
tb	Term Base: match obtained from a terminological database, i.e. the whole segment matches with a term base entry.
tm	Translation Memory: just an ordinary translation memory match.
other	match of a top level type not covered by any of the above definitions

- *Default value:* tm

Used in: `<match>`

B.1.3.7 subtype

Subtype - Indicates the subtype, i.e. a secondary level type, of a `<match>` element.

Value description:

The value is composed of a prefix and a sub-value separated by a character : (U+003A). The prefix is a string uniquely identifying a collection of values for a specific authority. The sub-value is any string value defined by an authority.

The prefix xlf is reserved for this specification, but no sub-values are defined for it at this time. Other prefixes and sub-values may be defined by the users.

- *Default value:* undefined

Used in: `<match>`

Processing Requirements

- If the attribute `subtype` is used, the attribute `type` *must* be specified as well.
- If the attribute `type` is modified, the attribute `subtype` *must* be updated or deleted.

B.1.3.8 reference

Reference - Indicates that the `<match>` is to be used as a reference language. For example, a German translation to be used as reference by a Luxembourgish translator.

Value description: Yes or No.

Default value: No.

Used in: <match>

B.1.4 XML Schema

The schema listed below for reading convenience is accessible at http://docs.oasis-open.org/xliff/xliff-core/v2.0/csd01/schemas/modules/matches.xsd.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified"
    xmlns:mtc="urn:oasis:names:tc:xliff:matches:2.0"
    xmlns:mda="urn:oasis:names:tc:xliff:metadata:2.0"
    xmlns:xlf="urn:oasis:names:tc:xliff:document:2.0"
    targetNamespace="urn:oasis:names:tc:xliff:matches:2.0">

    <xs:import namespace="urn:oasis:names:tc:xliff:document:2.0"
        schemaLocation="../xliff_core_2.0.xsd"/>
    <xs:import namespace="urn:oasis:names:tc:xliff:metadata:2.0"
        schemaLocation="metadata.xsd"/>

    <xs:simpleType name="similarity">
        <xs:restriction base="xs:decimal">
            <xs:minInclusive value="0.0"/>
            <xs:maxInclusive value="100.0"/>
        </xs:restriction>
    </xs:simpleType>

    <xs:simpleType name="typeValues">
        <xs:restriction base="xs:string">
            <xs:enumeration value="am"/>
            <xs:enumeration value="ebm"/>
            <xs:enumeration value="idm"/>
            <xs:enumeration value="ice"/>
            <xs:enumeration value="mt"/>
            <xs:enumeration value="tm"/>
        </xs:restriction>
    </xs:simpleType>

    <!-- Elements for holding translaton candidates -->

    <xs:element name="matches">
        <xs:complexType>
            <xs:sequence>
                <xs:element minOccurs="1" maxOccurs="unbounded"
                    ref="mtc:match"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>

    <xs:element name="match">
        <xs:complexType>
            <xs:sequence>
                <xs:element minOccurs="1" maxOccurs="1" ref="xlf:source"/>
                <xs:element minOccurs="1" maxOccurs="1" ref="xlf:target"/>
                <xs:element minOccurs="0" maxOccurs="1" ref="xlf:originalData"/>
                <xs:element minOccurs="0" maxOccurs="1" ref="mda:metadata"/>
```

```
            </xs:sequence>
            <xs:attribute name="id" use="optional" type="xs:NMTOKEN"/>
            <xs:attribute name="type" use="optional" type="mtc:typeValues"/>
            <xs:attribute name="origin" use="optional"/>
            <xs:attribute name="similarity" use="optional" type="mtc:similarity"/>
            <xs:anyAttribute namespace="##any" processContents="skip"/>
        </xs:complexType>
    </xs:element>

</xs:schema>
```

Appendix C Glossary Module

Simple glossaries, consisting of a list of terms with a definition or translation, can be optionally embedded in an XLIFF document using the namespace mechanism to include elements from the Glossary module.

C.1 Module Specification

C.1.1 Module Namespace

The namespace for the Glossary module is: `urn:oasis:names:tc:xliff:glossary:2.0`

C.1.2 Module Elements

The elements defined in the Glossary module are: `<glossary>`, `<glossentry>`, `<term>`, `<translation>` and `<definition>`.

C.1.2.1 Tree Structure

Legend:

1 = one
+ = one or more
? = zero or one

```
<glossary> 1
|
+---<glossentry> +
     |
     +---<term> 1
     |
     +---<translation> ?
     |
     +---<definition> ?
```

C.1.2.2 glossary

Container for a list of glossary terms.

Contains:

- One or more `<glossentry>` elements.

C.1.2.3 glossentry

Glossary entry.

Contains:

One `<term>` element followed by
Zero or one `<translation>` element followed by
Zero or one `<definition>` element.

Processing Requirements

- A `<glossentry>` element must contain a `<translation>` or a `<definition>` element to be valid.

C.1.2.4 term

A term in the glossary, expressed in the source language of the enclosing `<xliff>` element.

Contains:

Plain text.

C.1.2.5 translation

A translation of the sibling `<term>` element expressed in the target language of the enclosing `<xliff>` element.

Contains:

Plain text.

C.1.2.6 definition

Optional definition in plain text for the term stored in the sibling `<term>` element.

Contains:

- Plain text.

Attributes

- `source`, required

C.1.3 Module Attributes

The attribues defined in the Glossary module are: `source`.

C.1.4 XML Schema

The schema listed below for reading convenience is accessible at http://docs.oasis-open.org/xliff/xliff-core/v2.0/csd01/schemas/modules/glossary.xsd.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
    xmlns:gls="urn:oasis:names:tc:xliff:glossary:2.0"
    targetNamespace="urn:oasis:names:tc:xliff:glossary:2.0">

    <xs:import namespace="http://www.w3.org/XML/1998/namespace"
        schemaLocation="http://www.w3.org/2001/xml.xsd"/>

    <!-- Elements for holding simple glossary data -->

    <xs:element name="glossary">
        <xs:complexType mixed="false">
            <xs:sequence>
                <xs:element minOccurs="1" maxOccurs="unbounded" ref="gls:glossentry" />
            </xs:sequence>
        </xs:complexType>
    </xs:element>

    <xs:element name="glossentry">
        <xs:complexType mixed="false">
```

```
            <xs:sequence>
                <xs:element minOccurs="1" maxOccurs="1" ref="gls:term"/>
                <xs:element minOccurs="0" maxOccurs="1" ref="gls:translation" />
                <xs:element minOccurs="0" maxOccurs="1" ref="gls:definition" />
            </xs:sequence>
        </xs:complexType>
    </xs:element>

    <xs:element name="term">
        <xs:complexType mixed="true"/>
    </xs:element>

    <xs:element name="translation">
        <xs:complexType mixed="true"/>
    </xs:element>

    <xs:element name="definition">
        <xs:complexType mixed="true">
            <xs:attribute name="source" use="required" />
        </xs:complexType>
    </xs:element>

</xs:schema>
```

Appendix D Format Style Module

This is intended as a namespace mechanism to carry inside an XLIFF document information needed for generating a quick at a glance html preview of XLIFF content using a predefined set of simple html formatting elements.

D.1 Module Specification

Format Style module consists of just two attributes: `fs` and `subFs`. It does not specify any elements.

Format Style allows most structural and inline XLIFF core elements to convey basic formatting information using a predefined subset of HTML formatting elements. It primarily enables the generation of HTML pages or snippets for preview and review purposes. It *must* not be used to prescibe a roundtrip to a source document format.

The `fs` attribute holds the name of an HTML formatting element. If additional style information is needed, the optional `subFs` attribute is provided.

D.1.1 Module Namespace

The namespace for the Format style module is: `urn:oasis:names:tc:xliff:fs:2.0`

D.1.2 Module Attributes

The attributes defined in the Format Style module are: `fs`, `subFs`.

D.1.2.1 fs

Format style attribute, fs - allows most structural and inline XLIFF core elements to convey basic formatting information using a predefined subset of HTML formatting elements (for example, HTML elements names like <script> are not included). It enables the generation of HTML pages or snippets for preview and review purposes. This attribute is not intended to facilitate round-tripping back into the original format. If additional style information is needed, the optional `subFs` attribute is provided.

Value description:

code	computer code fragment
col	table column
colgroup	table column group
dd	definition description
del	deleted text
div	generic language/style container
dl	definition list
dt	definition term
em	emphasis
h1	heading
h2	heading
h3	heading
h4	heading
h5	heading
h6	heading
head	document head
hr	horizontal rule
html	document root element
i	italic text style
img	image
label	form field label text
legend	fieldset legend
li	list item
ol	ordered list
p	paragraph
pre	preformatted text
q	short inline quotation
s	strike-through text style
samp	sample program output, scripts, etc.
select	option selector
small	small text style
span	generic language/style container
strike	strike-through text
strong	strong emphasis
sub	subscript
sup	superscript
table	
tbody	table body
td	table data cell
tfoot	table footer
th	table header cell
thead	table header
title	document title
tr	table row
tt	teletype or monospaced text style
u	underlined text style
ul	unordered list

Table D.1. fs attribute values

Default value: undefined.

Used in: <file>, <unit>, , <ignorable>, <notes>, <note>, <originalData>, <data>, <source>, <target>, <cp>, <sc>, <ec>, <ph>, <pc>, <mrk>, <sm> and .

*Example:*To facilitate HTML preview, fs can be applied to XLIFF like this like:

```
<xliff>
 <file fs:fs="html">
  <unit id="1" fs:fs="div">
   <segment>
    <source fs:fs="p">Mick Jones renewed his interest in the Vintage  <pc id="1" fs:fs=
  </segment>
  </unit>
 </file>
</xliff>
```

With an XSL stylesheet like this:

```
<xsl:template match="node()|@*">
 <xsl:copy>
  <xsl:apply-templates select="@*|node()"/>
 </xsl:copy>
</xsl:template>


<xsl:template match="*" priority="2">
 <xsl:choose>
  <xsl:when test="@fs:fs">
   <xsl:element name="{@fs:fs}">
    <xsl:apply-templates />
   </xsl:element>
  </xsl:when>
  <xsl:otherwise>
   <xsl:apply-templates />
  </xsl:otherwise>
 </xsl:choose>
</xsl:template>
```

You can generate a an HTML page like this:

```
<html>
      <div>
         <p>Mick Jones renewed his interest in the Vintage  <strong>'72 Telecaster Thin
         </p>
      </div>
</html>
```

D.1.2.2 subFs

Sub-format style, subFs - allows extra metadata, like URL for example, to be added in concert with the fs attribute.

The subFs *must* only be used to carry attribute name/value comma-delimited pairs for attributes that are valid for the HTML element identified by the accompanying fs. Example: `fs:fs="img" fs:subFs="src,smileface.png"`.

Commas (,) and backslashes (\) in the value part of the pair must be escaped with a backslash.

The `subFs` attribute is not intended to facilitate round-tripping back into the original format.

Value description: IRI.

Default value: undefined.

Used in: `<file>`, `<unit>`, `<segment>`, `<ignorable>`, `<notes>`, `<note>`, `<originalData>`, `<data>`, `<source>`, `<target>`, `<cp>`, `<sc>`, `<ec>`, `<ph>`, `<pc>`, `<mrk>`, `<sm>` and `<em>`.

Processing Requirements

- If the attribute `subFs` is used, the attribute `fs` *must* be specified as well.

Appendix E Metadata Module

The Metadata module provides a mechanism for storing custom metadata using elements that are part of the official XLIFF specification.

E.1 Module Specification

E.1.1 Module Namespace

The namespace for the Metadata module is: `urn:oasis:names:tc:xliff:metadata:2.0`

E.1.2 Module Elements

The elements defined in the Metadata module are: `<metadata>`, `<metagroup>`, and `<meta>`.

E.1.2.1 Tree Structure

Legend:

1 = one
+ = one or more

```
<metadata> 1
|
+---<metagroup> +

       |
       +---<meta> +
```

E.1.2.2 metadata

Container for metadata associated with the enclosing element.

Contains:

- One or more `<metagroup>` elements

E.1.2.3 metagroup

Provides a way to organize metadata into a structured hierarchy.

Contains:

- One or more `<meta>` elements

Attributes

- `category`, optional

E.1.2.4 meta

Contains:

- Untranslatable text

Attributes

- `type`, required

E.1.3 Module Attributes

The attribues defined in the Metadata module are: `category` and `type`.

E.1.3.1 category

category - indicates a category for metadata contained in the enclosing `<metagroup>` element.

Default value: `undefined`

Used in: `<metagroup>`.

E.1.3.2 type

type - indicates the type of metadata contained by the enclosing element.

Default value: `undefined`

Used in: `<meta>`.

E.1.4 XML Schema

The schema listed below for reading convenience is accessible at http://docs.oasis-open.org/xliff/xliff-core/v2.0/csd01/schemas/modules/metadata.xsd.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
        xmlns:mda="urn:oasis:names:tc:xliff:metadata:2.0"
        targetNamespace="urn:oasis:names:tc:xliff:metadata:2.0">

    <xs:import namespace="http://www.w3.org/XML/1998/namespace"
            schemaLocation="http://www.w3.org/2001/xml.xsd"/>


    <!-- Elements for holding custom metadata -->

    <xs:element name="metadata">
        <xs:complexType mixed="false">
            <xs:sequence>
                <xs:element minOccurs="1" maxOccurs="unbounded" ref="mda:metagroup" />
            </xs:sequence>
        </xs:complexType>
    </xs:element>

    <xs:element name="metagroup">
        <xs:complexType>
            <xs:sequence>
                <xs:element minOccurs="1" maxOccurs="unbounded" ref="mda:meta" />
            </xs:sequence>
            <xs:attribute name="category" use="optional"/>
        </xs:complexType>
    </xs:element>
```

```
    <xs:element name="meta">
        <xs:complexType mixed="true">
            <xs:attribute name="type" use="required" type="xlf:attrType_type"/>
        </xs:complexType>
    </xs:element>

</xs:schema>
```

Appendix F Resource Data Module

The Resource Data module provides a mechanism for referencing external resource data that *may* need to be modified or used as contextual reference during translation.

F.1 Module Specification

F.1.1 Module Namespace

The namespace for the Resource Data module is: `urn:oasis:names:tc:xliff:resourceData:2.0`

F.1.2 Module Elements

The elements defined in the Resource Data module are: `<resourceData>`, `<source>`, `<target>`, and `<reference>`.

F.1.2.1 Tree Structure

Legend:

? = zero or one
* = zero, one or more

```
<resourceData>  *
|
+---<source> ?
|
+---<target> ?
|
+---<reference> *
```

F.1.2.2 resourceData

Specifies the references to external resource data that may need to be modified or used as contextual reference during translation.

Contains:

- Zero or One `<source>` element followed by
- Zero or One `<target>` element followed by
- Zero, one, or more `<reference>` elements

Attributes:

- `id`, optional
- `mimeType`, required
- `context`, optional
- attributes from any namespace, optional

Processing Requirements

- If a user agent does not understand how to process the `mimeType`, or the file it references, the Resource Data module can be ignored, but still must be preserved.

- The value of the optional id attribute must be unique among all <resourceData> children of the enclosing <file> element.
- The required mimeType attribute should only be modified or removed if the referenced files are modified or removed.
- For each instance of <resourceData> containing only <source>:
  - User agents may leave <resourceData> unchanged, e.g. they are not required to create <target> or <reference>.
  - User agents may create <target> or <reference> as a siblings of <source>.

F.1.2.3 source

References the actual resource data that may need to be modified or used as contextual reference during translation.

Contains:

- This element is always empty.

Attributes:

- href, required
- xml:lang, optional
- attributes from any namespace, optional

Processing Requirements

- When the optional xml:lang attribute is present, its value must be equal to the value of the srcLang attribute of the enclosing <xliff> element.
- When the context attribute of <resourceData> is set to yes:
  - User agents may create <source> if not already present.
  - User agents should not modify <source>.
  - User agents may remove <source>.
- When the context attribute of <resourceData> is set to no:
  - <source> must be present.
  - User agents must not modify <source>.
  - User agents must not remove <source>.

F.1.2.4 target

References the localized counterpart of the sibling <source> element.

Contains:

- This element is always empty.

Attributes:

- href, required
- xml:lang, optional
- attributes from any namespace, optional

Processing Requirements

- When the optional `xml:lang` attribute is present, its value must be equal to the value of the `tgtLang` attribute of the enclosing `<xliff>` element.
- When the `context` attribute of `<resourceData>` is set to yes:
  - User agents may create `<target>` if not already present.
  - User agents should not modify `<target>`.
  - User agents may remove `<target>`.
- When the `context` attribute of `<resourceData>` is set to no:
  - User agents may create `<target>` if not already present.
  - User agents may leave `<target>` unchanged.
  - User agents may modify `<target>`.
  - User agents may delete `<target>` and start over as if working without an existing `<target>`. However `<target>` cannot be deleted without being replaced.

F.1.2.5 reference

References contextual data relating to the sibling `<source>` and `<target>` elements, such as a German screenshot for a Luxembourgish translator.

Contains:

- This element is always empty.

Attributes:

- `href`, required
- `xml:lang`, optional
- attributes from any namespace, optional

Processing Requirements

- When the optional `xml:lang` attribute is present, its value does not need to be equal to the value of the `srcLang` or `tgtLang` attribute of the enclosing `<xliff>` element.
- User agents may create `<reference>` if not already present.
- User agents should not modify `<reference>`.
- User agents may remove `<reference>`.

F.1.3 Module Attributes

The attributes defined in the Resource Data module are: `id`, `xml:lang`, `mimeType`, `context`, `href`, and `resourceDataId`.

F.1.3.1 id

Identifier - A character string used to identify a `<resourceData>` element.

Value description: NMTOKEN

Default value: undefined

Used in:<resourceData>

F.1.3.2 xml:lang

Language - The xml:lang attribute specifies the language variant of the text of a given element. For example: `xml:lang="fr-FR"` indicates the French language as spoken in France.

Value description: A language code as described in [BCP 47].

Default value: undefined

Used in: <source>, <target>, and <reference>.

F.1.3.3 mimeType

MIME type, mimeType - indicates the type of a resource object. This generally corresponds to the content type of [RFC 2045] [http://tools.ietf.org/rfc/rfc2045.txt], the MIME specification; e.g. mimeType="text/xml" indicates the resource data is a text file of XML format.

Value description: A MIME type. An existing MIME type *must* be used from a list of standard values [http://www.iana.org/assignments/media-types].

Default value: undefined

Used in:<resourceData>

Note

If you cannot use any of the standard MIME type values as specified above, a new MIME type can be registered according to [RFC 2048] [http://tools.ietf.org/rfc/rfc2048.txt].

F.1.3.4 context

Contextual Information - Indicates whether an external resource is to be used for context only and not modified.

Value description: yes or no

Default value: yes

Used in:<resourceData>

F.1.3.5 href

Hypertext Reference, href - IRI referencing an external resource.

Value description: IRI.

Default value: undefined

Used in:<source>, <target>, and <reference>

F.1.3.6 resourceDataId

Resopurce Data Reference - Holds a reference to an associated <resourceData> element.

Value description: An XSD NMTOKEN that must be the value of the id attribute of the <resourceData> element being referenced.

Default value: undefined

Used in: `<xlf:unit>`

F.1.4 Examples:

In this example, referenced XML contains resource data for a user interface control. The control is the container for the text “Load Registry Config” and needs to be resized to accommodate the increased length of the string due to translation. The `res:resourceDataId` attribute in the `<unit>` element refers to the appropriate `<resourceData>` element, which references the data to be modified. The name attribute of the `<unit>` element serves as the key for a user agent to associate `<source>` and `<target>` text with resource data contained in the referenced XML.

```
<file id="1">
  <res:resourceData id="r1" mimeType="text/xml" context="no">
    <res:source href="resources\en\registryconfig.resources.xml" />
    <res:target href="resources\de\registryconfig.resources.xml" />
  </res:resourceData>
  <unit id="1" name="130;WIN_DLG_CTRL_" res:resourceDataId="r1">
    <segment id="1" state="translated">
      <source>Load Registry Config</source>
      <target>Registrierungskonfiguration laden</target>
    </segment>
  </unit>
</file>
```

In this example, referenced images are used for context when translating. The `res:ResourceDataRef` attribute in the `<unit>` element refers to the appropriate `<resourceData>` element, which references the image to be displayed to the user.

```
<file id="1">
  <res:resourceData id="r1" mimeType="image/jpeg" context="yes">
    <res:source href="resources\en\registryconfig.resources.jpg" />
    <res:target href="resources\de\registryconfig.resources.jpg" />
  </res:resourceData>
  <unit id="1" name="133;WIN_DLG_CTRL_" res:ResourceDataId="r1">
    <segment id="1" state="translated">
      <source>Remove Registry Config</source>
      <target>Registrierungskonfiguration entfernen</target>
    </segment>
  </unit>
</file>
```

F.1.5 XML Schema

The schema listed below for reading convenience is accessible at http://docs.oasis-open.org/xliff/xliff-core/v2.0/csd01/schemas/modules/resource_data.xsd.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
    xmlns:res="urn:oasis:names:tc:xliff:resourceData:2.0"
    targetNamespace="urn:oasis:names:tc:xliff:resourceData:2.0">

    <xs:import namespace="http://www.w3.org/XML/1998/namespace"
        schemaLocation="http://www.w3.org/2001/xml.xsd"/>
```

```
    <xs:element name="resourceData">
      <xs:complexType>
        <xs:sequence>
          <xs:element minOccurs="0" maxOccurs="1" ref="res:source"/>
          <xs:element minOccurs="0" maxOccurs="1" ref="res:target"/>
          <xs:element minOccurs="0" maxOccurs="unbounded" ref="res:reference"/>
        </xs:sequence>
        <xs:attribute name="id" use="optional" type="xs:NMTOKEN"/>
        <xs:attribute name="mimeType" use="required"/>
        <xs:attribute name="context" use="optional"/>
        <xs:anyAttribute namespace="##any" processContents="skip"/>
      </xs:complexType>
    </xs:element>

    <xs:element name="source">
      <xs:complexType mixed="false">
        <xs:attribute name="href" use="required"/>
        <xs:attribute ref="xml:lang" use="optional"/>
        <xs:anyAttribute namespace="##any" processContents="skip"/>
      </xs:complexType>
    </xs:element>

    <xs:element name="target">
      <xs:complexType mixed="false">
        <xs:attribute name="href" use="required"/>
        <xs:attribute ref="xml:lang" use="optional"/>
        <xs:anyAttribute namespace="##any" processContents="skip"/>
      </xs:complexType>
    </xs:element>

    <xs:element name="reference">
      <xs:complexType mixed="false">
        <xs:attribute name="href" use="required"/>
        <xs:attribute ref="xml:lang" use="optional"/>
        <xs:anyAttribute namespace="##any" processContents="skip"/>
      </xs:complexType>
    </xs:element>

</xs:schema>
```

Appendix G Change Tracking Module

The Change Tracking module is used to store revision information for XLIFF elements and attributes.

G.1 Module Specification

G.1.1 Module Namespace

The namespace for the Change Tracking module is: `urn:oasis:names:tc:xliff:changeTracking:2.0`

G.1.2 Module Elements

The elements defined in the Change Tracking module are: `<changeTrack>`, `<revisions>`, `<revision>`, and `<item>`.

G.1.2.1 Tree Structure

Legend:

? = zero or one
+ = one or more

```
<changeTrack>  ?
|
+---<revisions> +
    |
    +---<revision> +
        |
        +---<item> +
```

G.1.2.2 changeTrack

Parent container for change tracking information associated with the enclosing element.

Contains:

- One or more `<revisions>` elements.

G.1.2.3 revisions

Container for logical groups of revisions associated with the enclosing element.

Contains:

- One or more `<revision>` elements.

Attributes:

- `appliesTo`, required
- `nid`, optional
- `currentVersion`, optional
- attributes from any namespace, optional

Processing Requirements

- User agents *may* create `<revisions>` elements with attributes.
- User agents *should not* modify `<revisions>` and its attributes defined in this module, except in the case where the `currentVersion` attribute is used. This attribute *should* be updated when a new revision becomes the most current.
- User agents *should not* remove `<revisions>` and its attributes defined in this module.
- When the `appliesTo` attribute refers to an element that is a multiple instance within the enclosing element, the `nid` attribute *must* be used to reference an individual instance if and only if the referenced instance has an id. Using `<notes>` as an example:

```
<notes>
  <note id="n1">new note</note>
  <note id="n2">another note</note>
</notes>
<changeTrack>
  <revisions appliesTo="note" nid="n1">
    <revision><item property="content">old note</item></revision>
  </revisions>
</changeTrack>
```

G.1.2.4 revision

Container for specific revisions associated with the enclosing element.

Contains:

- One or more `<item>` elements.

Attributes:

- `author`, optional
- `datetime`, optional
- `version`, optional
- `checksum`, optional
- attributes from any namespace, optional

Processing Requirements

- User agents *may* create `<revision>` elements with attributes.
- User agents *should not* modify `<revision>` and its attributes defined in this module.
- User agents *may* remove `<revision>` and its attributes defined in this module if and only if there is more than one instance of `<revision>` present. For example, a user agent can decide to preserve only the most current revision.
- When the `checksum` attribute is used, a user agent *should* compute and store the checksum on the initial instance of the element and any revisions being tracked. It *should* also detect that the current checksum of an element is out of date and allow the changes to be committed as a new revision or be reverted back to the latest tracked state.

G.1.2.5 item

Container for a specific revision associated with the enclosing element.

Contains:

- Text.

Attributes:

- `property`, required
- attributes from any namespace, optional

Processing Requirements

- User agents *may* create `<item>` elements with attributes.
- User agents *should not* modify `<item>` and its attribute defined in this module.
- User agents *should not* remove `<item>` and its attribute defined in this module, unless they are removed as part of a `<revision>` elemnent removed according to its own processiing requirements.

G.1.3 Module Attributes

The attributes defined in the Change Tracking module are: `appliesTo`, `author`, `checksum`, `currentVersion`, `datetime`, `nid`, `property`, and `version`.

G.1.3.1 appliesTo

appliesTo – indicates the XLIFF element to which the change track information applies.

Value description: Any valid XLIFF element name.

Default value: undefined

Used in:`<revisions>`

G.1.3.2 author

author - indicates the user or agent that created or modified the referenced element or its attributes.

Value description: Text.

Default value: undefined

Used in:`<revision>`.

G.1.3.3 checksum

checksum – a checksum of the revision data used to detect changes made by non-compliant XLIFF user agents, such as a manual search and replace operation using a text editor.

Value description: A CRC-32 checksum of the revision data.

Note

[Editorial Note: ITU reference along with Joachim's pseudocode must be added here..]

Default value: undefined

Used in:`<revision>`, any XLIFF element that accepts attributes from any namespace.

G.1.3.4 currentVersion

currentVersion - holds a reference to the most current version of a revision.

Value description: An XSD NMTOKEN that must be the value of the `version` attribute of one of the `<revision>` elements listed in the same `<revisions>` element.

Default value: none

Used in:`<revisions>`.

G.1.3.5 datetime

Date and Time, datetime - indicates the date and time the referenced element or its attributes were created or modified.

Value description: Date in [ISO 8601] [http://www.w3.org/TR/NOTE-datetime] format.

Default value: undefined

Used in:`<revision>`.

G.1.3.6 nid

Original data reference, nid - holds a reference to a single instance of an element that has multiple instances within the enclosing element.

Value description: An XSD NMTOKEN that must be the value of the `id` attribute of a single instance of an element that is a multiple instance within the enclosing element.

Default value: undefined

Used in:`<revisions>`

G.1.3.7 property

property – the type of revision data.

Value description: The value *must* be either `content` to signify the content of an element, or the name of the attribute relating to the revision data.

Default value: none

Used in:`<item>`.

G.1.3.8 version

version - indicates the version of the referenced element or its attributes that were created or modified.

Value description: NMTOKEN.

Default value: undefined

Used in:`<revision>`.

G.1.4 Examples:

The following example shows simple change tracking for `<source>`, `<target>`, and `<notes>`. The core elements serve as the current version and previous versions are stored in the Change Tracking module.

```
<unit id="1">
  <segment id="1">
    <source ctr:author="system" ctr:datetime="2013-02-15T10:00:00+8:00">
     Hello World</source>
    <target ctr:author="Erik" ctr:datetime="2013-02-17T11:00:00+8:00">
     Hallo Welt</target>
    <notes>
      <note id="n1" category="instruction" ctr:author="Ryan"
       ctr:datetime="2013-02-15T10:30:00+8:00">A formal greeting</note>
      <note id="n2" category="request" ctr:author="Erik"
       ctr:datetime="2013-02-17T11:30:00+8:00">Please Review</note>
    </notes>
    <changeTrack>
      <revisions appliesTo="source">
        <revision author="system" datetime="2013-01-15T10:00:00+8:00">
          <item property="content">Hello</item>>
        </revision>
        <revision author="system" datetime="2012-12-15T10:00:00+8:00">
          <item property="content">Hi</item>
        </revision>
      </revisions>
      <revisions appliesTo="target">
        <revision author="Erik" datetime="2013-01-17T11:00:00+8:00">
          <item property="content">Hallo</item>
        </revision>
        <revision author="Erik" datetime="2012-12-17T11:00:00+8:00">
          <item property="content">Gruesse</item>
        </revision>
      </revisions>
      <revisions appliesTo="note" nid="n1">
        <revision author="Ryan" datetime="2012-12-15T10:30:00+8:00">
          <item property="content">An informal greeting</item>
          <item property="category">comment</item>
        </revision>
      </revisions>
    </changeTrack>
  </segment>
</unit>
```

The following example shows change tracking for <source>, <target>, and <notes> using a checksum to detect changes that may have been made outside of an XLIFF aware user agent. Current versions and previous versions are both stored in the Change Tracking module. A checksum on a core element and the most current version of its revision in the Change Tracking module can be checked to determine the existence of an external revision, for example the addition of an exclamation point in the target string.

```
<unit id="1">
    <segment id="1">
      <source ctr:checksum="4A17B156">Hello World</source>
      <!-- checksum is no longer valid due to the addition of an exclamation point -->
      <target ctr:checksum="50281475">Hallo Welt!</target>
      <notes>
        <note category="instruction" id="n1" ctr:checksum="59890430">A formal greeting</
        <note category="comment" id="n2" ctr:checksum="80E3EB9D">Please Review</note>
```

```
      </notes>
      <changeTrack>
        <revisions appliesTo="source" currentVersion="r1">
          <revision author="system" datetime="2013-02-15T10:00:00+8:00"
           version="r1" checksum="4A17B156">
            <item property="content">Hello World</item>>
          </revision>
          <revision author="system" datetime="2013-01-15T10:00:00+8:00"
           version="r2" checksum="F7D18982">
            <item property="content">Hello</item>>
          </revision>
          <revision author="system" datetime="2012-12-15T10:00:00+8:00"
           version="r3" checksum="F7D18982">
            <item property="content">Hi</item>
          </revision>
        </revisions>
        <revisions appliesTo="target" currentVersion="r1">
          <revision author="Erik" datetime="2013-02-17T11:00:00+8:00"
           version="r1" checksum="50281475">
            <item property="content">Hallo Welt</item>
          </revision>
          <revision author="Erik" datetime="2013-01-17T11:00:00+8:00"
           version="r2" checksum="78B31ED5">
            <item property="content">Hallo</item>
          </revision>
          <revision author="Erik" datetime="2012-12-17T11:00:00+8:00"
           version="r3" checksum="089D3926">
            <item property="content">Gruesse</item>
          </revision>
        </revisions>
        <revisions appliesTo="note" nid="n1" currentVersion="r1">
          <revision author="Ryan" datetime="2013-02-15T10:30:00+8:00"
           version="r1" checksum="59890430">
            <item property="content">A formal greeting</item>
            <item property="category">instruction</item>
          </revision>
          <revision author="Ryan" datetime="2012-12-15T10:30:00+8:00"
           version="r2" checksum="885537C1">
            <item property="content">An informal greeting</item>
            <item property="category">comment</item>
          </revision>
        </revisions>
      </changeTrack>
    </segment>
  </unit>
```

G.1.5 XML Schema

The schema listed below for reading convenience is accessible at http://docs.oasis-open.org/xliff/xliff-core/v2.0/csd01/schemas/modules/change_tracking.xsd.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
    xmlns:ctr="urn:oasis:names:tc:xliff:changeTrack:2.0"
    targetNamespace="urn:oasis:names:tc:xliff:changeTrack:2.0">
```

```
    <xs:import namespace="http://www.w3.org/XML/1998/namespace"
        schemaLocation="http://www.w3.org/2001/xml.xsd"/>

    <xs:element name="changeTrack">
      <xs:complexType>
        <xs:sequence>
          <xs:element minOccurs="1" maxOccurs="unbounded" ref="ctr:revisions"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>

    <xs:element name="revisions">
      <xs:complexType>
        <xs:sequence>
          <xs:element minOccurs="1" maxOccurs="unbounded" ref="ctr:revision"/>
        </xs:sequence>
        <xs:attribute name="appliesTo" use="required" type="xlf:appliesTo"/>
        <xs:attribute name="nid" use="optional" type="xs:NMTOKEN"/>
        <xs:attribute name="currentVersion" use="optional"/>
        <xs:anyAttribute namespace="##any" processContents="skip"/>
      </xs:complexType>
    </xs:element>

    <xs:element name="revision">
      <xs:complexType>
        <xs:sequence>
          <xs:element minOccurs="1" maxOccurs="unbounded" ref="ctr:item"/>
        </xs:sequence>
        <xs:attribute name="author" use="optional"/>
        <xs:attribute name="datetime" use="optional"/>
        <xs:attribute name="version" fixed="2.0" use="required"/>
        <xs:attribute name="checksum" use="optional"/>
        <xs:anyAttribute namespace="##any" processContents="skip"/>
      </xs:complexType>
    </xs:element>

    <xs:element name="item">
      <xs:complexType mixed="true">
        <xs:attribute name="property" use="required"/>
        <xs:anyAttribute namespace="##any" processContents="skip"/>
      </xs:complexType>
    </xs:element>

</xs:schema>
```

Appendix H Size Restriction Module

The Size Restriction module provides a mechanism to annotate the XLIFF content with information on storage and general size restrictions.

H.1 Module Specification

H.1.1 Introduction

The restriction framework has support for two distinct types of restrictions; storage size restrictions and general size restriction. The reason for this is that it is often common to have separate restrictions between storage and display / physical representation of data. Since it would be impossible to define all restrictions here a concept of restriction profile is introduced. The profiles for storage size and general size are independent. The information related to restriction profiles are stored in the processing invariant part of the XLIFF file like the `<xlf:file>`, `<xlf:group>` and `<xlf:unit>` elements and contained within elements defined in this module. The information regarding the specific restrictions are stored on the processing invariant parts and on the inline elements as attributes or attributes referencing data in the elements defined in this module. To avoid issues with segmentation no information regarding size restrictions is present on `<xlf:segment>`, `<xlf:source>` and `<xlf:target>` elements. The module defines a namespace for all the elements and attributes it introduce, in the rest of the module specification elements and atributes are in this namespace unless stated otherwise. In other parts of the XLIFF specification the prefix "slr" is used to refer to this modules namespace. For clarity the prefix "xlf" will be used for core XLIFF elements and attributes. Profile names use the same namespace like naming conventions as parts of the XLIFF Core specification. The name should be composed of two components separated by a colon. <authority>:<name>. The authority "xliff" is reserved for profiles defined by the OASIS XLIFF Technical Committee.

H.1.2 Module Namespace

The namespace for the Size and Length restriction module is: `urn:oasis:names:tc:xliff:sizerestriction:2.0`

H.1.3 Module Elements

The elements defined in the Size and Length restriction module are: `<profiles>`, `<normalization>` and `<data>`.

H.1.3.1 Tree Structure

Legend:

1 = one
+ = one or more
? = zero or one

```
<profiles> 1
|
+---<normalization> ?

<data> +
```

H.1.3.2 profiles

This element selects the restriction profiles to use in the document. If no storage or general profile is specified the default values (empty) of those elements will disable restriction checking in the file.

Contains:

- Zero or one `<normalization>` element followed by
- elements from any namespace, optional

Attributes:

- `generalProfile`, optional
- `storageProfile`, optional

Processing Requirements

- Any overall configuration or settings related to the selected profile *must* be placed in child elements of this element.
- Data not related to the configuration of the selected profiles *must not* be placed in this element.

H.1.3.3 normalization

This element is used to hold the attributes specifying normalization form to apply to storage and size restrictions defined in the standard profiles.

Contains:

- empty element

Attributes:

- `general`, optional
- `storage`, optional

Processing Requirements

- If this element is not present no normalization should be performed for the standard profiles.
- Other profiles *may* use this element in its specified form but *must not* add new extensions to it.

H.1.3.4 data

This elements act as a container for data needed by the specified profile to check the part of the XLIFF document that is a sibling or descendant of a sibling of this element. It is not used by the default profiles.

Contains:

- elements from any namespace, optional

Attributes:

- `profile`, required
- attributes from any namespace, optional

Processing Requirements

- Third party profiles *must* place all data in this element instead of using other extension points if the data serves no other purpose in the processing of the document.
- Data not used by the specified profile *must not* be placed in this element.

H.1.4 Module Attributes

The attributes defined in the Size and Length restriction module are: `storageProfile`, `generalProfile`, `storage`, `general`, `profile`, `storageRestriction`, `sizeRestriction`, `equivStorage` , `sizeInfo` and `sizeInfoRef`.

H.1.4.1 storageProfile

This attribute specifies which profile should be used to check storage size restrictions. Empty means that no restrictions should be applied.

Value description: Name of restriction profile to use for storage size restrictions.

Default value: empty

Used in:`<profiles>`.

H.1.4.2 generalProfile

This attribute specifies which profile should be used to check general size restrictions. Empty means that no restrictions should be applied.

Value description: Name of restriction profile to use for general size restrictions.

Default value: empty

Used in:`<profiles>`.

H.1.4.3 storage

This attribute specifies the normalization to apply for storage size restrictions. The only normalization forms C and D as specified by the Unicode Consortium are supported, see Unicode Standard Annex #15 [http://unicode.org/reports/tr15/].

Value description: normalization to apply.

Table H.1. Values

Value	Description
none	No additional normalization *should* be done, content should be used as represented in the document. It is possible that other agents have already done some type of normalization when modifying content. This means that this setting could give different results depending on what agent(s) are used to preform a specific action on the document.
nfc	Normalization Form C *should* be used
nfd	Normalization Form D *should* be used

Default value: "none"

Used in: `<normalization>`.

H.1.4.4 general

This attribute specifies the normalization to apply for general size restrictions. The only normalization forms C and D as specified by the Unicode Consortium are supported, see Unicode Standard Annex #15 [http://unicode.org/reports/tr15/].

Value description: normalization to apply.

Table H.2. Values

Value	Description
none	No additional normalization *should* be done, content should be used as represented in the document. It is possible that other agents have already done some type of normalization when modifying content. This means that this setting could give different results depending on what agent(s) are used to preform a specific action on the document.
nfc	Normalization Form C *should* be used
nfd	Normalization Form D *should* be used

Default value: "none"

Used in:`<normalization>`.

H.1.4.5 profile

This attribute is used on the `<data>` element to indicate what profile the contents of that element apply to.

Value description: Name of a restriction profile

Default value: undefined

Used in:`<data>`.

H.1.4.6 storageRestriction

This attribute specifies the storage restriction to apply to the collection descendants of the element it is defined on.

Value description: Interpretation of the value is dependent on selected `storageProfile`. It *must* represent the restriction to apply to the indicated sub part of the document.

Default value: undefined

Used in: `<file>`, `<group>`, `<unit>`, `<mrk>`, `<sm>`, `<pc>` and `<sc>`.

H.1.4.7 sizeRestriction

This attribute specifies the size restriction to apply to the collection descendants of the element it is defined on.

Value description: Interpretation of the value is dependent on selected `generalProfile`. It *must* represent the restriction to apply to the indicated sub part of the document.

Default value: undefined

Used in: `<file>`, `<group>`, `<unit>`, `<mrk>`, `<sm>`, `<pc>` and `<sc>`.

H.1.4.8 equivStorage

This attribute provide a means to specify how much storaqge space an inline element will use in the native format. This size contribution is then added to the size contributed by the textual parts. This attribute is only allowed on the `<ec>` element if that elemnt has the `isolated` attribute set to `yes`. Otherwise the attribute on the paired `<sc>` element also cover its partner `<ec>` element.

Value description: Interpretation of the value is dependent on selected `storageProfile`. It *must* represent the equivalent storage size represented by the inline element.

Default value: undefined

Used in: `<pc>`, `<sc>`, `<ec>`, `<ph>` and

H.1.4.9 sizeInfo

This attribute is used to associate profile specific information to inline elements so that size information can be decoupled from the native format or represented when the native data is not available in the XLIFF document. It can be used on both inline elements and structural elements to provide information on things like GUI dialog or control sizes, expected padding or margins to consider for size, what font is used for contained text and so on. This attribute is only allowed on the `<ec>` element if that elemnt has the `isolated` attribute set to `yes`. Otherwise the attribute on the paired `<sc>` element also cover its partner `<ec>` element.

Value description: Interpretation of the value is dependent on selected `generalProfile`. It *must* represent information related to how the element it is attached to contributes to the size of the text or entity in which it occurs or represents.

Default value: undefined

Restriction: at most one of this attribute and `sizeInfoRef` can occur. Both cannot be specified at the same time.

Used in: `<file>`, `<group>`, `<unit>`, `<pc>`, `<sc>` and `<ec>` and `<ph>`,

H.1.4.10 sizeInfoRef

This attribute is used to point to data that provide the same function as the `sizeInfo` attribute does, but with the data stored outside the inline content of the XLIFF segment. This attribute is only allowed on the `<ec>` element if that elemnt has the `isolated` attribute set to `yes`. Otherwise the attribute on the paired `<sc>` element also cover its partner `<ec>` element.

Value description: a reference to data that provide the same information that could be put in a `sizeInfo` attribute. The reference *must* point to an element in a `<data>` element that is a sibling to the element this attribute is attached to or a sibling to one of its ancestors.

Default value: undefined

Restriction: at most one of this attribute and sizeInfo can occur. Both cannot be specified at the same time.

Used in: `<file>`, `<group>`, `<unit>`, `<pc>`, `<sc>` and `<ec>` and `<ph>`,

H.1.5 Standard profiles

H.1.5.1 General restriction profile ”xliff:codepoints”

This profile implements a simple string length restriction based on the number of Unicode code points. It is possible to specify if normalization should be applied using the `<normalization>` element and the `general` attribute. This profile make use of the following attributes from this module:

H.1.5.1.1 sizeRestriction

The value of this attribute holds the ”maximum” or ”minimum and maximum” size of the string. Either size must be an integer. The maximum size can also be ’*’ to denote that there is no maximum restriction. If only a maximum is specified it is implied that the minimum is 0 (empty string). The format of the value is the optional minimum size and a coma followed by a maximum size (”[minsize,]maxsize”). The default value is ’*’ which evaluates to a string with unbounded size.

H.1.5.1.2 sizeInfo

The value of this attribute is an integer representing how many code points the element it is set on should be considered to contribute to the total size. If empty the default for all elements is 0.

H.1.5.2 Storage restriction profiles ”xliff:utf8”, ”xliff:utf16” and ”xliff:utf32”

These three profiles define the standard size restriction profiles for the common Unicode character encoding schemes. It is possible to specify if normalization should be applied using the `<normalization>`element and the `storage`. All sizes are represented in 8bit bytes. The size of text for these profiles is the size of the text converted to the selected encoding without any byte order marks attached. The encodings are specified by the Unicode Consortium in chapter 2.5 of the Unicode specification [http://www.unicode.org/versions/Unicode6.2.0/ch02.pdf].

Table H.3. Profiles

Name	Description
xliff:utf8	The number of 8bit bytes needed to represent the string encoded as UTF-8 as specified by the Unicode consortium.
xliff:utf16	The number of 8bit bytes needed to represent the string encoded as UTF-16 as specified by the Unicode consortium.
xliff:utf32	The number of 8bit bytes needed to represent the string encoded as UTF-32 as specified by the Unicode consortium.

These profiles make use of the following attributes from this module:

H.1.5.2.1 storageRestriction

The value of this attribute holds the ”maximum” or ”minimum and maximum” size of the string. Either size must be an integer. The maximum size can also be ’*’ to denote that there is no maximum restriction. If only a maximum is specified it is implied that the minimum is 0 (empty string). The format of the value is the optional minimum size and a coma followed by a maximum size (”[minsize,]maxsize”). The default value is ’*’ which evaluates to a string with unbounded size.

H.1.5.2.2 equivStorage

The value of this attribute is an integer representing how many bytes the element it is set on should be considered to contribute to the total size. If empty the default is 0. The `<cp>` is always converted to its representation in the profiles encoding and the size of that representation is used as the size contributed by the `<cp>`.

H.1.6 Third party profiles

The general structure of this module together with the extensibility mechanisms provided, should lend itself to most size restriction schemes. For example to represent two dimensional data a profile might adopt using a coordinate style for the values of the general restriction attributes. Like ”{x,y}” to represent width and height or ”{{x1,y1},{x2,y2}}” to represent a bounding box. It should also be possible to embed information necessary to drive a display simulator and attach that data to text to do device specific checking. Or to provide font information and to glyph based general size checking.

H.1.7 Conformance

To claim conformance to the XLIFF size and length restriction module a tool must meet the following criteria.

- *must* Follow the schema of the XLIFF Core specification and the extensions provided in this module

- *must* follow all processing requirements set forth in this module specification regarding the general use of elements and attributes
- *must* support all standard profiles with normalization set to ”none”
- *should* support all standard profiles with all modes of normalization
- *may* support additional third party profiles for storage or general restrictions
- *must* provide at least one of the following
  - Add size and length restriction information to a document
  - If it supports the specified profile(s) in the document it *must* provide a way to check if the size and length restrictions in the document are met according to the profile(s)

H.1.8 XML Schema

The schema listed below for reading convenience is accessible at http://docs.oasis-open.org/xliff/xliff-core/v2.0/csd01/schemas/modules/size_restriction.xsd.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
        xmlns:slr="urn:oasis:names:tc:xliff:sizerestriction:2.0"
        targetNamespace="urn:oasis:names:tc:xliff:sizerestriction:2.0">

    <xs:import namespace="http://www.w3.org/XML/1998/namespace"
            schemaLocation="http://www.w3.org/2001/xml.xsd"/>

    <!-- Attributes for size and length restriction used on core elements-->

    <xs:attribute name="equiv-storage" type="xs:string"/>
    <xs:attribute name="size-info" type="xs:string"/>
    <xs:attribute name="size-info-ref" type="xs:NMTOKEN"/>
    <xs:attribute name="size-restriction" type="xs:string"/>
    <xs:attribute name="storage-restriction" type="xs:string"/>

    <!-- Elements for size and length restriction -->

    <xs:element name="profiles">
        <xs:complexType mixed="false">
            <xs:sequence>
                <xs:element minOccurs="0" maxOccurs="1" ref="slr:normalization" />
                <xs:any maxOccurs="unbounded" minOccurs="0" namespace="##other" process
            </xs:sequence>
            <xs:attribute name="generalProfile" use="optional"/>
            <xs:attribute name="storageProfile" use="optional"/>
        </xs:complexType>
    </xs:element>

    <xs:element name="normalization">
        <xs:complexType mixed="false">
            <xs:attribute name="general" use="optional"/>
   <xs:attribute name="storage" use="optional"/>
        </xs:complexType>
    </xs:element>

    <xs:element name="data">
        <xs:complexType mixed="false">
            <xs:sequence>
                <xs:any maxOccurs="unbounded" minOccurs="0" namespace="##other" process
            </xs:sequence>
```

```
            <xs:attribute name="profile" use="required"/>
            <xs:anyAttribute namespace="##any" processContents="skip"/>
        </xs:complexType>
    </xs:element>

</xs:schema>
```

Appendix I Validation Module

This module defines a specific set of validation rules that can be applied to target text both globally and locally. Further constraints can be defined that allow rules to be applied to target text based on conditions in the source text or disabled to override a global scope.

I.1 Module Specification

I.1.1 Module Namespace

The namespace for the Validation module is: `urn:oasis:names:tc:xliff:validation:2.0`

I.1.2 Module Elements

The elements defined in the Validation module are: `<validation>` and `<rule>`.

I.1.2.1 Tree Structure

Legend:

? = zero or one
+ = one or more

```
<validation>  ?
|
+---<rule> +
```

I.1.2.2 validation

Parent container for a list of rules and constraints to apply to the target text of the enclosing element.

Contains:

- One or more `<rule>` elements.

Attributes:

- attributes from any namespace, optional

Processing Requirements

- When the `<validation>` element occurs at the `<file>` level, rules *must* be applied to all `<target>` elements within the scope of that `<file>` element, except where overrides are specified at the `<group>` or `<unit>` level.
- When `<validation>` occurs at the `<group>` level, rules must be applied to all `<target>` elements within the scope of `<group>`, except where overrides are specified at the `<unit>` level.
- When `<validation>` occurs at the `<unit>` level, rules must be applied to all `<target>` elements within the scope of `<unit>`.
- When `<validation>` occurs at the `<segment>` level, rules must be applied to the `<target>` element within the scope of the `<segment>`.

I.1.2.3 rule

A specific rule and constraint to apply to the target text of the enclosing element.

Contains:

- This element is always empty.

Attributes:

- `startsWith`, optional
- `endsWith`, optional
- `occurrences`, optional
- `mustLoc`, optional
- `noLoc`, optional
- `disabled`, optional
- `existsInSource`, optional
- `normalization`, optional
- attributes from any namespace, optional

Processing Requirements

- Exactly one of the following attributes:
  - `startsWith`
  - `endsWith`
  - `occurrences`
  - `mustLoc`
  - `noLoc`

  must be used in any one `<rule>` element.
- User agents *may* create `<rule>` elements.
- User agents *must not* modify attributes defined in this module present in any `<rule>` element.
- User agents *must not* remove either `<rule>` elements or their attributes defined in this module.

I.1.3 Module Attributes

The attributes defined in the Validation module are: `startsWith`, `endsWith`, `occurrences`, `mustLoc`, `noLoc`, `disabled`, `existsInSource`, `normalization`,

I.1.3.1 startsWith

Target text starts with, startsWith - is a test to verify that the target text starts with a defined value.

Value description: Text

Default value: none

Used in:`<rule>`

Processing Requirements

- The target text *must* start with the value of `startsWith`.

I.1.3.2 endsWith

Target text ends with, endsWith - is a test to verify that the target text ends with a defined value.

Value description: Text

Default value: none

Used in:<rule>

Processing Requirements

- The target text *must* end with the value of endsWith.

I.1.3.3 occurrences

occurrences - is a test to verify that a certain number of occurences of a specified string exists in the target text.

Value description: Text.

Characters left parenthesis ((U+0028), right parenthesis) (U+0029), and quotation mark " (U+0022) *must* be escaped by enclosing within a pair of quotation marks, " (U+0022). The value *must* follow this pattern: "(string)(integer)".

For example: In this occurrences="( )(0)", the value patern prohibits any occurences of two non-breaking spaces next to each other. Here occurrences="(:)(1)", exactly one occurrence of the colon is enforced.

Default value: none

Used in:<rule>

Processing Requirements

- The target text *must* contain the integer number of occurences of the string as specified in occurrences.

I.1.3.4 mustLoc

Must localize, mustLoc - is a test for the presence of a string (substring) in the source text and a verification that it does not exist in the target text. Alternatively it can be used to verify presence of a prescribed replacement string in the target text.

Value description: Text.

Characters left parenthesis ((U+0028), right parenthesis) (U+0029), and quotation mark " (U+0022) *must* be escaped by enclosing within a pair of quotation marks, " (U+0022). The value *must* follow one of two patterns: either mustLoc="string" or mustLoc="(string)(string)", where the prescribed replacement string is enclosed within the second pair of parentheses.

Default value: none

Used in:<rule>

Processing Requirements

- When mustLoc contains only one string from the source text, for example: mustLoc="hello world"; the target text *must not* contain that string.
- When mustLoc contains a string from the source text and a replacement string for the target text, for example: mustLoc="(Hello world)(Hallo Welt)"; the target text *must* contain that replacement string.

I.1.3.5 noLoc

Not to localize, noLoc - is a test for the presence of a string (substring) in the source text and verification that it exists also in the target text.

Value description: Text

Default value: none

Used in:<rule>

Processing Requirements

- The target text *must* contain the string specified by the value of noLoc.

I.1.3.6 disabled

Disabled rule, disabled – determines whether a rule *must* or *must not* be applied within the scope of its enclosing element. For example, a rule defined at the <file> level may be disabled at the <unit> level.

This attribute is provided to allow for overriding execution of rules set at higher levels, see <validation>.

Value description: yes or no

Default value: no

Used in:<rule>

Processing Requirements

- When the disabled attribute is set to yes, the rule *must not* be applied within the scope of its enclosing element.

I.1.3.7 existsInSource

Exists in source, existsInSource – determines whether a target text rule depends on the rule's being satisfied in the source text or not. For example, with this attribute set to yes a test would check if the target text ends with a non-breaking space if and only if the source text ends with a non-breaking space. The default value no means that the rule will always be enforced in target text, irrespective if the source text satisfies it. This attribute is only relevant when used with one of the following attributes: startsWith, endsWith, or occurrences, attributes, it *must not* be used with either the mustLoc attribute or with the noLoc attribute.

Using existsInSource set to no will have no impact on execution of rules, except for overriding rules behavior set with existsInSource on a higher level.

Value description: yes or no

Default value: no

Used in:<rule>

Processing Requirements

- When using the existsInSource attribute, exactly one of the following attributes:
  - startsWith
  - endsWith
  - occurrences

  must be present within the same <rule> element.
- Within one <rule> element, with the existsInSource attribute set to yes, the rule *must* be applied to the target text if and only if the the rule is satisfied in the source text.

- When using one of the following attributes:
  - `mustLoc`
  - `noLoc` attribute.

  the `existsInSource` attribute *must not* be present within the same `<rule>` element.

I.1.3.8 normalization

normalization - specifies the normalization type to apply when validating a rule. Only the normalization forms C and D as specified by the Unicode Consortium are supported, see Unicode Standard Annex #15 [http://unicode.org/reports/tr15/].

Value description: The allowed values are are listed in the table below along with their corresponding types of normalization to be applied.

Table I.1. Values

Value	Description
none	No normalization *should* be done.
nfc	Normalization Form C *must* be used.
nfd	Normalization Form D *must* be used.

Default value: nfc

Used in:`<rule>`

I.1.4 Example:

```
<file>
  <validation>
    <rule noLoc="Hello" />
  </validation>
  <unit>
    <source>[Hello World from the XLIFF TC]"</source>
    <target>[Hallo Welt vom XLIFF TC]"</target>
    <validation>
      <rule startsWith="[" existsInSource="yes" />
      <rule endsWith="]" existsInSource="yes" />
      <rule occurrences="(&amp;nbsp;&amp;nbsp;)(0)" />
      <rule mustLoc="(World)(Welt)" />
      <rule noLoc="Hello" disabled="yes" />
    </validation>
  </unit>
</file>
```

I.1.5 XML Schema

The schema listed below for reading convenience is accessible at http://docs.oasis-open.org/xliff/xliff-core/v2.0/csd01/schemas/modules/validation.xsd.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
    xmlns:val="urn:oasis:names:tc:xliff:validation:2.0"
```

```
    targetNamespace="urn:oasis:names:tc:xliff:validation:2.0">

    <xs:import namespace="http://www.w3.org/XML/1998/namespace"
        schemaLocation="http://www.w3.org/2001/xml.xsd"/>

    <xs:element name="validation">
      <xs:complexType>
        <xs:sequence>
          <xs:element minOccurs="1" maxOccurs="unbounded" ref="val:rule"/>
        </xs:sequence>
        <xs:anyAttribute namespace="##any" processContents="skip"/>
      </xs:complexType>
    </xs:element>

    <xs:element name="rule">
      <xs:complexType mixed="false">
        <xs:attribute name="startsWith" use="optional"/>
        <xs:attribute name="endsWith" use="optional"/>
        <xs:attribute name="occurrences" use="optional"/>
        <xs:attribute name="mustLoc" use="optional"/>
        <xs:attribute name="noLoc" use="optional"/>
        <xs:attribute name="disabled" use="optional"/>
        <xs:attribute name="existsInSource" use="optional"/>
        <xs:attribute name="normalization" use="optional"/>
        <xs:anyAttribute namespace="##any" processContents="skip"/>
      </xs:complexType>
    </xs:element>

</xs:schema>
```

Appendix J Acknowledgements (Non-Normative)

The following individuals have participated in the creation of this specification and are gratefully acknowledged:

- Amaya, Victor - Oracle
- Chapman, Helena - IBM
- Coady, Shirley - MultiCorpora R&D Inc.
- Comerford, Tom - Individual
- Estreen, Fredrik - Lionbridge
- Filip, David - Localisation Research Centre
- King, Ryan - Microsoft
- Lieske, Christian - SAP AG
- Loomis, Steven - IBM
- Michael, Alan - Microsoft
- Morado Vazquez, Lucia - Localisation Research Centre
- O'Donnell, Kevin - Microsoft
- Ow, Michael - IBM
- Prause, Ingo - SDL
- Raya, Rodolfo - Maxprograms
- Ryoo, Jung Woo - Oracle
- Savourel, Yves - ENLASO Corporation
- Schnabel, Bryan - Individual
- Schurig, Joachim - Lionbridge
- Stahlschmidt, Uwe - Microsoft
- Waldhör, Klemens - TAUS
- Walters, David - IBM
- Wasala, Asanka - Localisation Research Centre