XLIFF Core Version 2.2

Working Draft 01

22 April 2024

Specification URIs

This version:
https://docs.oasis-open.org/xliff/xliff-core/v2.2/wd01/xliff-core-v2.2-wd01.html (Authoritative)
https://docs.oasis-open.org/xliff/xliff-core/v2.2/wd01/xliff-core-v2.2-wd01.pdf
https://docs.oasis-open.org/xliff/xliff-core/v2.2/wd01/xliff-core-v2.2-wd01.xml

Previous version:
https://docs.oasis-open.org/xliff/xliff-core/v2.1/os/xliff-core-v2.1-os.html (Authoritative)
https://docs.oasis-open.org/xliff/xliff-core/v2.1/os/xliff-core-v2.1-os.pdf
https://docs.oasis-open.org/xliff/xliff-core/v2.1/os/xliff-core-v2.1-os.xml

Latest version:
https://docs.oasis-open.org/xliff/xliff-core/v2.2/xliff-core-v2.2.html (Authoritative)
https://docs.oasis-open.org/xliff/xliff-core/v2.2/xliff-core-v2.2.pdf
https://docs.oasis-open.org/xliff/xliff-core/v2.2/xliff-core-v2.2.xml

Technical Committee:
OASIS XML Localisation Interchange File Format (XLIFF) TC

Chairs:
Lucía Morado Vázquez (lucia.morado@unige.ch), University of Geneva
Yoshito Umaoka (yoshito_umaoka@us.ibm.com), IBM

Editors:
Rodolfo M. Raya (rmraya@maxprograms.com), Individual
Lucía Morado Vázquez (lucia.morado@unige.ch), University of Geneva

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

- XML schemas accessible from https://docs.oasis-open.org/xliff/xliff-core/v2.2/wd01/schemas/

Related Work:
This specification replaces or supersedes:

- XLIFF Version 2.1. Edited by David Filip, Tom Comerford, Soroush Saadatfar, Felix Sasaki and Yves Savourel. 13 February 2018. OASIS Standard. https://docs.oasis-open.org/xliff/xliff-core/v2.1/os/xliff-core-v2.1.html

Declared XML Namespaces:

- urn:oasis:names:tc:xliff:document:2.2
- 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:sizerestriction:2.0
- urn:oasis:names:tc:xliff:validation:2.0
- http://www.w3.org/2005/11/its
- urn:oasis:names:tc:xliff:itsm:2.0
- urn:oasis:names:tc:xliff:pgs:1.0

Abstract:

This document defines version 2.2 of the XML Localization 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 and among 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 https://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 (https://www.oasis-open.org/committees/xliff/ipr.php).

Note for any machine-readable content (aka Computer Language Definitions) declared Normative for this Work Product that is provided in separate plain text files, in the event of a discrepancy between any such plain text file and display content in the Work Product's prose narrative document(s), the content in the separate plain text file prevails.

Citation format:

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

[XLIFF-2.2]

XLIFF Version 2.2. Edited by Rodolfo M. Raya and Lucía Morado Vázquez 22 April 2024. Working Draft 01. https://docs.oasis-open.org/xliff/xliff-core/v2.2/wd01/xliff-core-wd01.html. Latest version: https://docs.oasis-open.org/xliff/xliff-core/v2.2/xliff-core.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 https://www.oasis-open.org/policies-guidelines/trademark for above guidance.

Table of Contents

Appendixes

1 Introduction

XLIFF is the XML Localization Interchange File Format designed by a group of multilingual content publishers, software providers, localization service providers, localization tools providers, and researchers. It is intended to give any multilingual content owner a single interchange file format that can be understood by any localization provider, using any conformant localization tool. While the primary focus is on being a lossless interchange format, usage of XLIFF as a processing format is neither encouraged nor discouraged or prohibited.

All text is normative unless otherwise labeled. The following common methods are used for labeling portions of this specification as informative and hence non-normative:

Appendices and sections marked as "(Informative)" or "Non-Normative" in title,
Notes (sections with the "Note" title),
Warnings (sections with the "Warning" title),
Examples (mainly example code listings, tree diagrams, but also any inline examples or illustrative exemplary lists in otherwise normative text),
Schema and other validation artifacts listings (the corresponding artifacts are normative, not their listings).

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 [**BCP 14**].

1.1.2 Definitions

Agent
: any application or tool that generates (creates), reads, edits, writes, processes, stores, renders or otherwise handles XLIFF documents.

  Agent is the most general application conformance target that subsumes all other specialized user agents disregarding whether they are defined in this specification or not.

Enrich, Enriching
: the process of associating module and extension based metadata and resources with the Extracted XLIFF payload

  Processing Requirements

  - Enriching MAY happen at the time of Extraction.

  Note

  Extractor knowledge of the native format is not assumed while Enriching.

Enricher, Enricher Agent
: any Agent that performs the Enriching process

Extract, Extraction
: the process of encoding localizable content from a native content or User Interface format as XLIFF payload, so that localizable parts of the content in the source language are available for translation into the target language along with the necessary context information

Extractor, Extractor Agent
: any Agent that performs the Extraction process

Merge, Merging
: the process of importing XLIFF payload back to the originating native format, based on the full knowledge of the Extraction mechanism, so that the localized content or User Interface strings replace the source language in the native format

Merger, Merger Agent
: an Agent that performs the Merge process

> ### Warning
>
> Unless specified otherwise, any Merger is deemed to have the same knowledge of the native format as the Extractor throughout the specification.
>
> Mergers independent of Extractors can succeed, but it is out of scope of this specification to specify interoperability for merging back without the full Extractor knowledge of the native format.

Modify, Modification
: the process of changing core and module XLIFF structural and inline elements that were previously created by other Writers

  Processing Requirements

  - XLIFF elements MAY be Modified and Enriched at the same time.

> ### Note
>
> Extractor or Enricher knowledge of the native format is not assumed while modifying.

Modifier, Modifier Agent
: an Agent that performs the Modification process

Translation, Translate
: a rendering of the meaning of the source text, expressed in the target language

Writer, Writer Agent
: an Agent that creates, generates, or otherwise writes an XLIFF document for whatever purpose, including but not limited to Extractor, Modifier, and Enricher Agents.

> ### Note
>
> Since XLIFF is intended as an exchange format rather than a processing format, many applications will need to generate XLIFF documents from their internal processing formats, even in cases when they are processing XLIFF documents created by another Extractor.

1.1.3 Key concepts

XLIFF Core
: The core of XLIFF 2.2 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 2.2 is `"urn:oasis:names:tc:xliff:document:2.2"`.

XLIFF-defined (elements and attributes)
: The following is the list of allowed schema URI prefixes for XLIFF-defined elements and attributes:

```
urn:oasis:names:tc:xliff:
http://www.w3.org/2005/11/its
```

However, the following namespaces are NOT considered XLIFF-defined for the purposes of the XLIFF 2.2 specification:

```
urn:oasis:names:tc:xliff:document:1.0
urn:oasis:names:tc:xliff:document:1.1
urn:oasis:names:tc:xliff:document:1.2
```

Elements and attributes from other namespaces are not XLIFF-defined.

XLIFF Document
: Any XML document that declares the namespace `"urn:oasis:names:tc:xliff:document:2.2"` as its main namespace, has `<xliff>` as the root element and complies with the XML Schemas and the declared Constraints that are part of this specification.

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 2.2 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).

[**HTML5**] Ian Hickos, Robin Berjon, Steve Faulkner, Travis Leithead, Erika Doyle Navara, Edward O'Connor, Silvia Pfeiffer *HTML5. A vocabulary and associated APIs for HTML and XHTML*, *http://www.w3.org/TR/html5/* W3C Recommendation 28 October 2014.

[**ITS**] David Filip, Shaun McCance, Dave Lewis, Christian Lieske, Arle Lommel, Jirka Kosek, Felix Sasaki, Yves Savourel *Internationalization Tag Set (ITS) Version 2.0*, *http://www.w3.org/TR/its20/* W3C Recommendation 29 October 2013.

[**NOTE-datetime**] M. Wolf, C. Wicksteed, *Date and Time Formats*, *http://www.w3.org/TR/NOTE-datetime* W3C Note, 15th September 1997.

[**BCP 14**] S. Bradner, *Key words for use in RFCs to Indicate Requirement Levels*, *https://tools.ietf.org/search/bcp14* IETF (Internet Engineering Task Force) RFC 2119, March 1997.

[**RFC 3987**] M. Duerst and M. Suignard, *Internationalized Resource Identifiers (IRIs)*, *https://www.ietf.org/rfc/rfc3987.txt* IETF (Internet Engineering Task Force) RFC 3987, January 2005.

[**RFC 7303**] H. Thompson and C. Lilley, *XML Media Types*, *https://www.tools.ietf.org/html/rfc7303* IETF (Internet Engineering Task Force) RFC 7303, July 2014.

[**UAX #9**] M. Davis, A. Lanin, A. Glass, *UNICODE BIDIRECTIONAL ALGORITHM*, *http://www.unicode.org/reports/tr9/tr9-35.html* Unicode Bidirectional Algorithm, May 18, 2016.

[**UAX #15**] M. Davis, K. Whistler, *UNICODE NORMALIZATION FORMS*, *http://www.unicode.org/reports/tr15/tr15-44.html* Unicode Normalization Forms, February 24, 2016.

[**Unicode**] The Unicode Consortium, *The Unicode Standard*, *http://www.unicode.org/versions/Unicode9.0.0/* Mountain View, CA: The Unicode Consortium, June 21, 2016.

[**XML**] W3C, *Extensible Markup Language (XML) 1.0*, *http://www.w3.org/TR/xml/* (Fifth Edition) W3C Recommendation 26 November 2008.

[**XML namespace**] W3C, *Schema document for namespace http://www.w3.org/XML/1998/namespace* *http://www.w3.org/2001/xml.xsd* [http://www.w3.org/2009/01/xml.xsd]. at https://docs.oasis-open.org/xliff/xliff-core/v2.2/wd01/schemas/informativeCopiesOf3rdPartySchemas/w3c/xml.xsd in this distribution

[**XML Catalogs**] Norman Walsh, *XML Catalogs*, *https://www.oasis-open.org/committees/download.php/14809/xml-catalogs.html* OASIS Standard V1.1, 07 October 2005.

[**XML Schema**] W3C, *XML Schema*, *refers to the two part standard comprising [XML Schema Structures] and [XML Schema Datatypes]* (Second Editions) W3C Recommendations 28 October 2004.

[**XML Schema Datatypes**] W3C, *XML Schema Part 2: Datatypes*, *http://www.w3.org/TR/xmlschema-2/* (Second Edition) W3C Recommendation 28 October 2004.

[**XML Schema Structures**] W3C, *XML Schema Part 1: Structures*, *https://www.w3.org/TR/xmlschema-1/* (Second Edition) W3C Recommendation 28 October 2004.

1.3 Non-Normative References

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

[**SRX**] *Segmentation Rules eXchange* *http://www.unicode.org/uli/pas/srx/*

[**UAX #29**] M. Davis, *UNICODE TEXT SEGMENTATION*, *http://www.unicode.org/reports/tr29/* Unicode text Segmentation.

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

[**ICU MessageFormat**] *ICU MessageFormat Class* *https://unicode-org.github.io/icu/userguide/format_parse/messages/#messageformat*

[**Grammatical Genders**] *List of languages by type of grammatical genders* *https://en.wikipedia.org/wiki/List_of_languages_by_type_of_grammatical_genders#More_than_three_grammatical_genders*

[**CLDR Plural Spec**] *The CLDR spec for Plural Rules:* *https://cldr.unicode.org/index/cldr-spec/plural-rules*

[**CLDR Plural Lang**] *Language Plural Rules (all languages):* *http://www.unicode.org/cldr/charts/latest/supplemental/language_plural_rules.html*

[**CLDR Plural**] *CLDR plural files (`plurals.xml` & `ordinals.xml` in `core.zip`):* *https://www.unicode.org/Public/cldr/44/*

[**ICU4C API**] *ICU4C APIs:* *https://unicode-org.github.io/icu-docs/apidoc/released/icu4c/classicu_1_1PluralRules.html*

[**ICU4J API**] *ICU4J APIs:* *https://unicode-org.github.io/icu-docs/apidoc/released/icu4j/com/ibm/icu/text/PluralRules.html*

2 Conformance

1. *Document Conformance*

    a. XLIFF is an XML vocabulary, therefore conformant XLIFF Documents MUST be well formed and valid [XML] documents.

    b. Conformant XLIFF documents MUST be valid instances of the official Core XML Schema (https://docs.oasis-open.org/xliff/xliff-core/v2.2/wd01/schemas/xliff_core_2.2.xsd) that is a part of this multipart Work Product.

    c. As not all aspects of the XLIFF specification can be expressed in terms of XML Schemas, conformant XLIFF documents MUST also comply with all relevant elements and attributes definitions, normative usage descriptions, and Constraints specified in this specification document.

    d. XLIFF documents MAY contain custom extensions, as defined in the Extension Mechanisms section.

2. *Application Conformance*

    a. XLIFF Writers MUST create conformant XLIFF Documents to be considered XLIFF compliant.

    b. Agents processing conformant XLIFF Documents 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 documents, provided that the elements that contain custom extensions are not removed according to XLIFF Processing Requirements or the extension's own processing requirements.

    c. All Agents MUST comply with Processing Requirements for otherwise unspecified Agents or without a specifically set target Agent.

    d. Specialized Agents defined in this specification - this is Extractor, Merger, Writer, Modifier, and Enricher Agents - MUST comply with the Processing Requirements targeting their specifically defined type of Agent on top of Processing Requirements targeting all Agents as per point c. above.

    e. XLIFF is a format explicitly designed for exchanging data among various Agents. Thus, a conformant XLIFF application MUST be able to accept XLIFF Documents it had written after those XLIFF Documents were Modified or Enriched by a different application, provided that:

        i. The processed files are conformant XLIFF Documents,

        ii. in a state compliant with all relevant Processing Requirements.

3. *Backwards Compatibility*

    a. Conformant applications are REQUIRED to support XLIFF 2.0 and 2.1.

    b. Conformant applications are NOT REQUIRED to support XLIFF 1.2 or previous versions.

Note

XLIFF documents conformant to this specification are not and cannot be conformant to XLIFF 1.2 or earlier versions. If an application needs to support for whatever business reason both XLIFF 2 and XLIFF 1.2 or earlier, these will need to be supported as separate functionalities.

3 Fragment Identification

Because XLIFF documents do not follow the usual behavior of XML documents when it comes to element identifiers, this specification defines how Agents MUST interpret the fragment identifiers in IRIs pointing to XLIFF documents.

Note

Note that some identifiers may change during the localization process. For example `<data>` elements may be re-grouped or not depending on how tools treat identical original data.

Constraints

- A fragment identifier MUST match the following format:

```
<expression>      ::= "#" ["/"] <selector>
                      {<selectorSeparator> <selector>}
<selector>        ::= [<prefix> <prefixSeparator>] <id>
<prefix>          ::= NMTOKEN
<id>              ::= NMTOKEN
<prefixSeparator> ::= "="
<selectorSeparator>  ::= "/"
```

- There MUST NOT be two identical prefixes in the expression.
- When used, the following selectors MUST be declared in this order: file selector, group selector and unit selector.
- The selectors for modules or extensions, `<note>`, `<segment>` or `<ignorable>` or source inline elements, target inline elements and `<data>` have the following constraints:
  - Only one of them MAY be used in the expression.
  - The one used MUST be the last selector of the expression.

Warning

Note that due to the above Constraints, referencing fragments using third party namespaces within Modules or extensions (including but not limited to XLIFF Core or the Metadata Module) is not possible. This is to restrict the complexity of the fragment identification mechanism, as it would otherwise have potentially unlimited depth.

3.1 Selectors for Core Elements

- The prefix `f` indicates a `<file>` id and the value of that id is unique among all `<file>` `id` attribute values within the enclosing `<xliff>` element.
- The prefix g indicates a `<group>` id and the value of that id is unique among all `<group>` `id` attribute values within the enclosing `<file>` element.
- The prefix u indicates a `<unit>` id and the value of that id is unique among all `<unit>` `id` attribute values within the enclosing `<file>` element.
- The prefix n indicates a `<note>` id and the value of that id is unique among all `<note>` `id` attribute values within the immediate enclosing `<file>`, `<group>`, or `<unit>` element.

- The prefix `d` indicates a `<data>` id and the value of that id is unique among all `<data>` `id` attribute values within the enclosing `<unit>` element.
- The prefix `t` indicates an id for an inline element in the `<target>` element and the value of that id is unique within the enclosing `<unit>` element (with the exception of the matching inline elements in the `<source>`).
- No prefix indicates an id for a `<segment>` or an `<ignorable>` or an inline element in the `<source>` element and the value of that id is unique within the enclosing `<unit>` element (with the exception of the matching inline elements in the `<target>`).

3.2 Selectors for Modules and Extensions

A selector for a module or an extension uses a registered prefix and the value of that id is unique within the immediate enclosing `<file>`, `<group>` or `<unit>` element.

Constraints

- The prefix of a module or an extension MUST be an NMTOKEN longer than 1 character and MUST be defined in the module or extension specification.
- The prefix of a module or an extension MUST be registered with the XLIFF TC.
- A given module or extension namespace URI MUST be associated with a single prefix.
- A prefix MAY be associated with more than one namespace URI (to allow for example different versions of a given module or extension to use the same prefix).

See also the constraints related to how IDs need to be specified in extensions (which applies for modules as well).

3.3 Relative References

Fragment identifiers that do not start with a character `/` (U+002F) are relative to their location in the document, or to the document being processed.

Any unit, group or file selector missing to resolve the relative reference is obtained from the immediate enclosing unit, group or file elements.

3.4 Examples

Given the following XLIFF document:

```
<xliff xmlns="urn:oasis:names:tc:xliff:document:2.2" version="2.2"
    srcLang="en" trgLang="fr">
  <file id="f1">
    <notes>
      <note id="n1">note for file.</note>
    </notes>
    <unit id="u1">
      <my:elem xmlns:my="myNamespaceURI" id="x1">data</my:elem>
      <notes>
        <note id="n1">note for unit</note>
      </notes>
      <segment id="s1">
        <source><pc id="1">Hello <mrk id="m1" type="term">World</mrk>!</pc>
            </source>
```

```
        <target><pc id="1">Bonjour le <mrk id="m1" type="term">Monde</mrk>
            ! </pc></target>
      </segment>
    </unit>
  </file>
</xliff>
```

You can have the following fragment identifiers:

- `#f=f1/u=u1/1` refers to the element `<pc id="1">` of the source content of the element `<unit id="u1">`.
- `#f=f1/u=u1/t=1` refers to the element `<pc id="1">` of the target content of the element `<unit id="u1">`.
- `#f=f1/n=n1` refers to the element `<note id="n1">` of the element `<file id="f1">`.
- `#f=f1/u=u1/n=n1` refers to the element `<note id="n1">` of the element `<unit id="u1">`.
- `#f=f1/u=u1/s1` refers to the element `<segment id="s1">` of the element `<unit id="u1">`.
- Assuming the extension defined by the namespace URI `myNamespaceURI` has registered the prefix `myprefix`, the expression `#f=f1/u=u1/myprefix=x1` refers to the element `<my:element id="x1">` of the element `<unit id="u1">`.

4 The Core Specification

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

At creation time, an XLIFF document MAY contain only text in the source language. Translations expressed in the 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. Typically, 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.

4.1 General Processing Requirements

- An Agent processing a valid XLIFF Document that contains XLIFF-defined elements and attributes that it cannot handle MUST preserve those elements and attributes.
- An Agent processing a valid XLIFF Document that contains custom elements and attributes that it cannot handle SHOULD preserve those elements and attributes.

4.2 Elements

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

4.2.1 Tree Structure

Legend:

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

```
<xliff>
|
+---<notes> ?
| |
| +---<note> +
|
+---<mda:metadata> ?
| |
| +---<mda:metagroup> +
|   |
|   +---At least one of (<mda:metagroup> OR <mda:meta>)
|
+---<file> +
  |
  +---<skeleton> ?
  | |
  | +---<other> *
  |
  +---<other> *
  |
  +---<notes> ?
```

```
| |
| +---<note> +
|
+---At least one of (<unit> OR <group>)
    | |
    | +---<unit>
    |   |
    |   +---<other> *
    |   |
    |   +---<notes> ?
    |   | |
    |   | +---<note> +
    |   |
    |   +---<originalData> ?
    |   | |
    |   | +---<data> +
    |   |
    |   +---At least one of (<segment> OR <ignorable>)
    |       | |
    |       | +---<segment>
    |       |   |
    |       |   +---<source> 1
    |       |   |
    |       |   +---<target> ?
    |       |
    |       +---<ignorable>
    |           |
    |           +---<source> 1
    |           |
    |           +---<target> ?
    |
    +---<group>
        |
        +---<other> *
        |
        +---<notes> ?
        | |
        | +---<note> +
        |
        +---At least one of (<unit> OR <group>)
```

4.2.2 Structural Elements

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

4.2.2.1 xliff

Root element for XLIFF documents.

Contains:

- Zero or one <notes> element followed by
- Zero or one <*mda:metadata*> element followed by
- One or more <file> elements

Attributes:

- `version`, REQUIRED
- `srcLang`, REQUIRED
- `trgLang`, OPTIONAL
- `xml:space`, OPTIONAL
- attributes from other namespaces, OPTIONAL

Constraints

- The `trgLang` attribute is REQUIRED if and only if the XLIFF document contains `<target>` elements that are children of `<segment>` or `<ignorable>`.

Note

The use of attributes from XLIFF modules MUST be in accordance with the constraints specified in the corresponding modules.

4.2.2.2 file

Container for localization material extracted from an entire single document, or another high level self contained logical node in a content structure that cannot be described in the terms of documents.

Note

Sub-document artifacts such as particular sheets, pages, chapters and similar are better mapped onto the `<group>` element. The `<file>` element is intended for the highest logical level. For instance a collection of papers would map to a single XLIFF Document, each paper will be represented with one `<file>` element, whereas chapters and subsections will map onto nested `<group>` elements.

Contains:

- Zero or one `<skeleton>` element followed by
- elements from other namespaces, OPTIONAL
- Zero or one `<notes>` element followed by
- One or more `<unit>` or `<group>` elements in any order.

Attributes:

- `id`, REQUIRED
- `canResegment`, OPTIONAL
- `original`, OPTIONAL
- `translate`, OPTIONAL
- `srcDir`, OPTIONAL
- `trgDir`, OPTIONAL
- `xml:space`, OPTIONAL
- attributes from other namespaces, OPTIONAL

Constraints

- The following XLIFF Module elements are explicitly allowed by the wildcard `other`:

  - Zero or one `<mda:metadata>` elements
  - Zero or one `<res:resourceData>` element
  - Zero or one `<slr:profiles>` elements
  - Zero or one `<slr:data>` elements
  - Zero or one `<val:validation>` elements
  - Zero, one, or more `<its:provenanceRecords>` elements

- Module and Extension elements MAY be used in any order.

> **Note**
>
> The use of attributes from XLIFF modules MUST be in accordance with the constraints specified in the corresponding modules.

4.2.2.3 skeleton

Container for non-translatable material pertaining to the parent `<file>` element.

Contains:

Either

- Non-translatable text
- elements from other namespaces

or

- is empty.

Attributes:

- `href`, OPTIONAL

Constraints

- The attribute `href` is REQUIRED if and only if the `<skeleton>` element is empty.

Processing Requirements

- Modifiers and Enrichers processing an XLIFF document that contains a `<skeleton>` element MUST NOT change that element, its attributes, or its content.
- Extractors creating an XLIFF document with a `<skeleton>` element MUST leave the `<skeleton>` element empty if and only if they specify the attribute `href`.

4.2.2.4 group

Provides a way to organize units into a structured hierarchy.

Note that this is especially useful for mirroring a source format's hierarchical structure.

Contains:

- elements from other namespaces, OPTIONAL
- Zero or one `<notes>` element followed by
- Zero, one or more `<unit>` or `<group>` elements in any order.

Attributes:

- `id`, REQUIRED
- `name`, OPTIONAL
- `canResegment`, OPTIONAL
- `translate`, OPTIONAL
- `srcDir`, OPTIONAL
- `trgDir`, OPTIONAL
- `type`, OPTIONAL
- `xml:space`, OPTIONAL

- attributes from other namespaces, OPTIONAL

Constraints

- The following XLIFF Module elements are explicitly allowed by the wildcard `other`:

  - Zero or one <*mda:metadata*> elements
  - Zero or one <*slr:data*> elements
  - Zero or one <*val:validation*> elements
  - Zero, one, or more <*its:provenanceRecords*> elements

- Module and Extension elements MAY be used in any order.

> **Note**
>
> The use of attributes from XLIFF modules MUST be in accordance with the constraints specified in the corresponding modules.

4.2.2.5 unit

Static container for a dynamic structure of elements holding the extracted translatable source text, aligned with the translated text.

Contains:

- elements from other namespaces, OPTIONAL
- Zero or one `<notes>` elements followed by
- Zero or one `<originalData>` element followed by
- One or more `<segment>` or `<ignorable>` elements in any order.

Attributes:

- `id`, REQUIRED
- `name`, OPTIONAL
- `canResegment`, OPTIONAL
- `translate`, OPTIONAL
- `srcDir`, OPTIONAL
- `trgDir`, OPTIONAL
- `xml:space`, OPTIONAL
- `type`, OPTIONAL
- attributes from other namespaces, OPTIONAL

Constraints

- A `<unit>` MUST contain at least one `<segment>` element.

- The following XLIFF Module elements are explicitly allowed by the wildcard `other`:

  - Zero or one <*mtc:matches*> elements
  - Zero or one <*gls:glossary*> elements
  - Zero or one <*mda:metadata*> elements
  - Zero or one <*res:resourceData*> elements
  - Zero or one <*slr:data*> elements
  - Zero or one <*val:validation*> elements
  - Zero, one, or more <*its:locQualityIssues*> elements
  - Zero, one, or more <*its:provenanceRecords*> elements

- Module and Extension elements MAY be used in any order.

Note

The use of attributes from XLIFF modules MUST be in accordance with the constraints specified in the corresponding modules.

4.2.2.6 segment

This element is a container to hold in its aligned pair of children elements the minimum portion of translatable source text and its translation in the given Segmentation.

Contains:

- One `<source>` element followed by
- Zero or one `<target>` element

Attributes:

- `id`, OPTIONAL
- `canResegment`, OPTIONAL
- `state`, OPTIONAL
- `subState`, OPTIONAL
- attributes from the namespace `urn:oasis:names:tc:xliff:pgs:1.0`, OPTIONAL, provided that the Constraints specified in the *Plural, Gender, and Select Module* are met.

4.2.2.7 ignorable

Part of the extracted content that is not included in a segment (and therefore not translatable). For example tools can use `<ignorable>` to store the white space and/or codes that are between two segments.

Contains:

- One `<source>` element followed by
- Zero or one `<target>` element

Attributes:

- `id`, OPTIONAL

4.2.2.8 notes

Collection of comments.

Contains:

- One or more `<note>` elements

4.2.2.9 note

This is an XLIFF specific way how to present end user readable comments and annotations. A note can contain information about `<source>`, `<target>`, `<unit>`, `<group>`, `<file>` or `<xliff>` elements.

Contains:

- Text

Attributes:

- `id`, OPTIONAL
- `appliesTo`, OPTIONAL
- `category`, OPTIONAL

- priority, OPTIONAL
- ref, OPTIONAL
- attributes from other namespaces, OPTIONAL

Example:

```
<unit id="18">
  <notes>
    <note id="1" ref="#18-0">Change text to lower case</note>
  </notes>
  <segment state="initial" id="18-0">
    <source>Create Memories from Existing Translations</source>
  </segment>
</unit>
```

Note

When the ref attribute points to a element, by default the <note> content applies to its <source> child, unless the optional appliesTo attribute is set to target.

Note

The use of attributes from XLIFF modules MUST be in accordance with the constraints specified in the corresponding modules.

4.2.2.10 originalData

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

Contains:

- One or more <data> elements

4.2.2.11 data

Storage for the original data of an inline code.

Contains:

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

Non-translatable text and <cp> elements MAY appear in any order.

Attributes:

- id, REQUIRED
- dir, OPTIONAL
- xml:space, OPTIONAL, the value is restricted to preserve on this element

4.2.2.12 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

Constraints

- When a <source> element is a child of or <ignorable>, the explicit or inherited value of the OPTIONAL xml:lang attribute MUST be equal to the value of the srcLang attribute of the enclosing <xliff> element.

4.2.2.13 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
- order, OPTIONAL

Constraints

- When a <target> element is a child of or <ignorable>, the explicit or inherited value of the OPTIONAL xml:lang MUST be equal to the value of the trgLang attribute of the enclosing <xliff> element.

4.2.3 Inline Elements

The XLIFF Core 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>.

4.2.3.1 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

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 has to 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>`.

4.2.3.2 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
- attributes from other namespaces, OPTIONAL

Example:

```
<unit id="1">
  <originalData>
    <data id="d1">%d</data>
    <data id="d2">&lt;br/></data>
  </originalData>
  <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>
</unit>
```

Constraints

- The following XLIFF Module attributes are explicitly allowed by the wildcard `other`:

  - attributes from the namespace `urn:oasis:names:tc:xliff:fs:2.0`, OPTIONAL, provided that the Constraints specified in the *Format Style Module* are met.
  - attributes from the namespace `urn:oasis:names:tc:xliff:sizerestriction:2.0`, OPTIONAL, provided that the Constraints specified in the *Size and Length Restriction Module* are met.

- No other attributes MUST be used.

Processing Requirements

- Extractors MUST NOT use the <ph> element to represent spanning codes.

  Rationale: Using a standalone placeholder code for a spanning code does not allow for controlling the span (for instance tag order and data integrity) when modifying inline content and is in direct contradiction to the business logic described in Representation of the codes and normative statements included in Usage of <pc> and <sc>/<ec>

  Note

  It is possible although not advised to use <ph> to mask non translatable inline content. The preferred way of protecting portions of inline content from translation is the Core Translate Annotation. See also discussion in the *ITS Module section on representing translatability inline*.

4.2.3.3 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 elements

Text and inline elements may appear in any order.

Parents:

- `<source>`
- `<target>`
- `<pc>`
-`<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
- attributes from other namespaces, OPTIONAL

Example:

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

Constraints

- The following XLIFF Module attributes are explicitly allowed by the wildcard `other`:

  - attributes from the namespace `urn:oasis:names:tc:xliff:fs:2.0`, OPTIONAL, provided that the Constraints specified in the *Format Style Module* are met.
  - attributes from the namespace `urn:oasis:names:tc:xliff:sizerestriction:2.0`, OPTIONAL, provided that the Constraints specified in the *Size and Length Restriction Module* are met.

- No other attributes MUST be used.

Processing Requirements

- Extractors MUST NOT use the `<pc>` element to represent standalone codes.

  Rationale: Using a spanning code for a standalone code can easily result in having text inside a span where the original format does not allow it.

4.2.3.4 sc

Start 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
- `dataRef`, OPTIONAL
- `dir`, OPTIONAL
- `disp`, OPTIONAL
- `equiv`, OPTIONAL
- `id`, REQUIRED
- `isolated`, OPTIONAL
- `subFlows`, OPTIONAL
- `subType`, OPTIONAL
- `type`, OPTIONAL
- attributes from other namespaces, 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>
```

Constraints

- The following XLIFF Module attributes are explicitly allowed by the wildcard `other`:

  - attributes from the namespace `urn:oasis:names:tc:xliff:fs:2.0`, OPTIONAL, provided that the Constraints specified in the *Format Style Module* are met.
  - attributes from the namespace `urn:oasis:names:tc:xliff:sizerestriction:2.0`, OPTIONAL, provided that the Constraints specified in the *Size and Length Restriction Module* are met.

- No other attributes MUST be used.

- 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 code.

- If the OPTIONAL attribute `isolated` is present, its value MUST be set to `yes` when the `<ec>` element corresponding to this start marker is not in the same `<unit>`. When the corresponding `<ec>` element is present in the same `<unit>`, the attribute value MUST be set to `no`.

Processing Requirements

- Extractors MUST NOT use the `<sc>` / `<ec>` pair to represent standalone codes.

  Rationale: Using a spanning code for a standalone code can easily result in having text inside a span where the original format does not allow it.

4.2.3.5 ec

End 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
- `dataRef`, OPTIONAL
- `dir`, OPTIONAL
- `disp`, OPTIONAL
- `equiv`, OPTIONAL
- `id`, OPTIONAL
- `isolated`, OPTIONAL
- `startRef`, OPTIONAL
- `subFlows`, OPTIONAL
- `subType`, OPTIONAL
- `type`, OPTIONAL
- attributes from other namespaces, OPTIONAL

Example:

```
<unit id="1">
  <originalData>
    <data id="d1">\b </data>
    <data id="d2">\i </data>
    <data id="d3">\b0 </data>
    <data id="d4">\i0 </data>
  </originalData>
  <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>
</unit>
```

Constraints

- The following XLIFF Module attributes are explicitly allowed by the wildcard `other`:

  - attributes from the namespace `urn:oasis:names:tc:xliff:fs:2.0`, OPTIONAL, provided that the Constraints specified in the *Format Style Module* are met.
  - attributes from the namespace `urn:oasis:names:tc:xliff:sizerestriction:2.0`, OPTIONAL, provided that the Constraints specified in the *Size and Length Restriction Module* are met.

- No other attributes MUST be used.
- The values of the attributes `canCopy`, `canDelete` and `canOverlap` MUST be the same as the values the ones in the `<sc>` element corresponding to this end code.
- The value of the attribute `canReorder` MUST be `no` if the value of `canReorder` is `firstNo` in the `<sc>` element corresponding to this end code.
- The attribute `isolated` MUST be set to `yes` if and only if the `<sc>` element corresponding to this end code is not in the same `<unit>` and set to `no` otherwise.
- If and only if the attribute `isolated` is set to `yes`, the attribute `id` MUST be used instead of the attribute `startRef` that MUST be used otherwise.
- If and only if the attribute `isolated` is set to `yes`, the attribute `dir` MAY be used, otherwise the attribute `dir` MUST NOT be used on the `<ec>` element.

Processing Requirements

- Extractors MUST NOT use the `<sc>` / `<ec>` pair to represent standalone codes.

  Rationale: Using a spanning code for a standalone code can easily result in having text inside a span where the original format does not allow it.

4.2.3.6 mrk

Represents an annotation pertaining to the marked span.

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
- attributes from other namespaces, OPTIONAL

Constraints

- The [XML namespace] MUST NOT be used at this extension point.
- The following XLIFF Module attributes are explicitly allowed by the wildcard `other`:

  - attributes from the namespace `urn:oasis:names:tc:xliff:fs:2.0`, OPTIONAL, provided that the Constraints specified in the *Format Style Module* are met.
  - attributes from the namespace `urn:oasis:names:tc:xliff:sizerestriction:2.0`, OPTIONAL, provided that the Constraints specified in the *Size and Length Restriction Module* are met.
  - attributes from the namespace `http://www.w3.org/2005/11/its`, OPTIONAL, provided that the Constraints specified in the *ITS Module* are met.
  - attributes from the namespace `urn:oasis:names:tc:xliff:itsm:2.1`, OPTIONAL, provided that the Constraints specified in the *ITS Module* are met.

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

4.2.3.7 sm

Start marker of an annotation where the spanning marker cannot be used for well-formedness reasons.

Contains:

This element is always empty.

Parents:

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

Attributes:

- id, REQUIRED
- translate, OPTIONAL
- type, OPTIONAL
- ref, OPTIONAL
- value, OPTIONAL
- attributes from other namespaces, OPTIONAL

Constraints

- The [XML namespace] MUST NOT be used at this extension point.
- The following XLIFF Module attributes are explicitly allowed by the wildcard `other`:

  - attributes from the namespace `urn:oasis:names:tc:xliff:fs:2.0`, OPTIONAL, provided that the Constraints specified in the *Format Style Module* are met.
  - attributes from the namespace `urn:oasis:names:tc:xliff:sizerestriction:2.0`, OPTIONAL, provided that the Constraints specified in the *Size and Length Restriction Module* are met.
  - attributes from the namespace `http://www.w3.org/2005/11/its`, OPTIONAL, provided that the Constraints specified in the *ITS Module* are met.
  - attributes from the namespace `urn:oasis:names:tc:xliff:itsm:2.1`, OPTIONAL, provided that the Constraints specified in the *ITS Module* are met.

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

4.2.3.8 em

End marker of an annotation where the spanning marker cannot be used for well-formedness reasons.

Contains:

This element is always empty.

Parents:

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

Attributes:

- `startRef`, REQUIRED

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

4.3 Attributes

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

4.3.1 XLIFF Attributes

The attributes defined in XLIFF 2.2 are: `appliesTo`, `canCopy`, `canDelete`, `canOverlap`, `canReorder`, `canResegment`, `category`, `copyOf`, `dataRef`, `dataRefEnd`, `dataRefStart`, `dir`, `disp`, `dispEnd`, `dispStart`, `equiv`, `equivEnd`, `equivStart`, `hex`, `href`, `id`, `isolated`, `name`, `order`, `original`, `priority`, `ref`, `srcDir`, `srcLang`, `startRef`, `state`, `subFlows`, `subFlowsEnd`, `subFlowsStart`, `subState`, `subType`, `trgLang`, `translate`, `trgDir`, `type`, `value` and `version`.

4.3.1.1 appliesTo

Comment target - indicates the element to what the content of the note applies.

Value description: `source` or `target`.

Default value: undefined.

Used in: `<note>`.

4.3.1.2 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 is not intended to be copied.

Default value: `yes`.

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

4.3.1.3 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 is not allowed to be deleted.

Default value: `yes`.

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

4.3.1.4 canOverlap

Code can overlap - indicates whether or not the spanning code where this attribute is used can enclose partial spanning codes (i.e. a start code without its corresponding end code, or an end code without its corresponding start code).

Value description: `yes` or `no`.

Default value: default values for this attribute depend 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">
  <originalData>
    <data id="1">\i1 </data>
    <data id="2">\i0 </data>
    <data id="3">{\b </data>
    <data id="4">}</data>
  </originalData>
  <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>
</unit>
```

4.3.1.5 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` in case the code can be re-ordered, `firstNo` when the code is the first element of a sequence that cannot be re-ordered, `no` when it is another element of such a sequence.

Default value: `yes`.

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

For the normative Usage Description see Constraints and Processing Requirements in the Editing Hints section.

4.3.1.6 canResegment

Can resegment - indicates whether or not the source text in the scope of the given `canResegment` flag can be reorganized into a different structure of `<segment>` elements within the same parent `<unit>`.

Value description: `yes` or `no`.

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

- When used in `<file>`:

  The value `yes`.

- When used in any other element:

  The value of the `canResegment` attribute of its parent element.

Used in: `<file>` `<group>` `<unit>`, and `<segment>`.

4.3.1.7 category

Category - provides a way to categorize notes.

Value description: Text.

Default value: undefined

Used in: `<note>`.

4.3.1.8 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>
```

4.3.1.9 dataRef

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

Value description: An [XML Schema Datatypes] 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">
  <originalData>
    <data id="d1">{0}</data>
  </originalData>
  <segment>
    <source>Error in '<ph id="1" dataRef="d1"/>'.</source>
    <target>Erreur dans '<ph id="1" dataRef="d1"/>'.</target>
```

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

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

4.3.1.10 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 [XML Schema Datatypes] 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">
  <originalData>
    <data id="d1">&lt;EM></data>
    <data id="d2">&lt;/EM></data>
  </originalData>
  <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>
</unit>
```

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

4.3.1.11 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 [XML Schema Datatypes] 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">
  <originalData>
    <data id="d1">&lt;EM></data>
    <data id="d2">&lt;/EM></data>
  </originalData>
  <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>
</unit>
```

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

4.3.1.12 dir

Directionality - indicates the directionality of content.

Value description: `ltr` (Left-To-Right), `rtl` (Right-To-Left), or `auto` (determined heuristically, based on the first strong directional character in scope, see [UAX #9]).

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

- When used in a <pc>, <sc>, or <ec> element that has a <source> element as its parent:

  The value of the `srcDir` attribute of the <unit> element, in which the elements are located.

- When used in a <pc>, <sc>, or <ec> element that has a <target> element as its parent:

  The value of the `trgDir` attribute of the <unit> element, in which the elements are located.

- When used in a <pc>, <sc>, or <ec> element that has a <pc> element as its parent:

  The value of the `dir` attribute of the parent <pc> element.

- When used in <data>:

  The value `auto`.

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

4.3.1.13 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">
  <originalData>
    <data id="d1">{1}</data>
  </originalData>
  <segment>
    <source>Welcome back <ph id="1" disp="[UserName]" dataRef="d1"/>!
        </source>
  </segment>
</unit>
```

Note

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

4.3.1.14 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">
  <originalData>
    <data id="d1">\cf1\ul\b\f1\fs24 </data>
    <data id="d2">\cf0\ulnone\b0\f0\fs22 </data>
  </originalData>
  <segment>
    <source>Example of <pc id="1" dataRefStart="d1" dataRefEnd="d2"
        dispStart="&lt;span>" dispEnd="&lt;/span>">
        formatted text</pc>.</source>
  </segment>
</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.

4.3.1.15 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">
  <originalData>
    <data id="d1">\cf1\ul\b\f1\fs24 </data>
    <data id="d2">\cf0\ulnone\b0\f0\fs22 </data>
  </originalData>
  <segment>
    <source>Example of <pc id="1" dataRefStart="d1" dataRefEnd="d2"
        dispStart="&lt;span>" dispEnd="&lt;/span>">
```

```
        formatted text</pc>.</source>
  </segment>
</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.

4.3.1.16 equiv

Equivalent text - holds a plain text representation of the original data of the inline code that can 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">
  <originalData>
    <data id="d1">&amp;</data>
  </originalData>
  <segment>
    <source>Open <ph id="1" equiv="" dataRef="d1"/>File</source>
  </segment>
</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.

4.3.1.17 equivEnd

Equivalent text - holds a plain text representation of the original data of the end marker of an inline code that can 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">
  <originalData>
    <data id="d1">&lt;span class="link" onclick="linkTo('dbId5345')">
        </data>
```

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

Note

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

4.3.1.18 equivStart

Equivalent text - holds a plain text representation of the original data of the start marker of an inline code that can 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">
  <originalData>
    <data id="d1">&lt;span class="link" onclick="linkTo('dbId5345')">
        </data>
    <data id="d2">&lt;/span></data>
  </originalData>
  <segment>
    <source>The jam made of <pc id="1" dataRefStart="d1" equivStart=""
        dataRefEnd="d2" equivEnd="">lingonberries</pc> is quite
        tasty.</source>
  </segment>
</unit>
```

Note

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

4.3.1.19 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 [XML Schema Datatypes] 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 have to be represented in XLIFF.

4.3.1.20 href

Hyperlink reference - 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>`.

4.3.1.21 id

Identifier - a character string used to identify an element.

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

- When used in a `<file>` element:

  The value MUST be unique among all `<file>` `id` attribute values within the enclosing `<xliff>` element.

- When used in `<group>` elements:

  The value MUST be unique among all `<group>` `id` attribute values within the enclosing `<file>` element.

- When used in `<unit>` elements:

  The value MUST be unique among all `<unit>` `id` attribute values within the enclosing `<file>` element.

- When used in `<note>` elements:

  The value MUST be unique among all `<note>` `id` attribute values within the immediate enclosing `<file>`, `<group>`, or `<unit>` element.

- When used in `<data>` elements:

  The value MUST be unique among all `<data>` `id` attribute values within the enclosing `<unit>` element.

- When used in `<segment>`, `<ignorable>`, `<mrk>`, `<sm>`, `<pc>`, `<sc>`, `<ec>`, or `<ph>` elements:

  - The inline elements enclosed by a `<target>` element MUST use the duplicate `id` values of their corresponding inline elements enclosed within the sibling `<source>` element if and only if those corresponding elements exist.

  - Except for the above exception, the value MUST be unique among all of the above within the enclosing `<unit>` element.

Note

All of the above defined uniqueness scopes ignore Module and Extension data. It would be impossible to impose those uniqueness requirements onto Module or Extension data. As Core only Modifiers could inadvertently cause conflicts with Modules or Extensions based data they cannot access. Modules and Extensions reusing Core need to specify their own uniqueness scopes for the xlf:id. In general, Modules and Extensions are advised to mimic the Core uniqueness requirement within their specific wrapper elements enclosing the reused Core elements or attributes, yet Module or Extensions are free to set wider uniqueness scopes if it makes business sense.

Default value: undefined

Used in: `<file>`, `<group>`, `<unit>`, `<note>`, `<segment>`, `<ignorable>`, `<data>`, `<sc>`, `<ec>`, `<ph>`, `<pc>`, `<mrk>` and `<sm>`.

4.3.1.22 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 code.

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

Default value: `no`.

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

Example:

```
<file id="f2" xmlns:abc="urn:abc">
  <unit id="1">
    <mtc:matches>
      <mtc:match id="tc01" ref="seg2">
        <source><sc id="1" isolated="yes"/>Warning:</source>
        <target><sc id="1" isolated="yes"/>Attention :</target>
      </mtc:match>
    </mtc:matches>
    <segment id="seg2">
      <source><pc id="1">Warning: File not found.</pc></source>
    </segment>
  </unit>
</file>
```

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

4.3.1.23 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.

Default value: undefined.

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

4.3.1.24 order

target order - indicates the order, in which to compose the target content parts.

Value description: A positive integer.

Default value: implicit, see below

When order is not explicitly set, the `<target>` `order` corresponds to its sibling `<source>`, i.e. it is not being moved anywhere when composing target content of the enclosing `<unit>` and the implicit `order` value is of that position within the `<unit>`.

Used in: `<target>`.

Constraints

- The value of the `order` attribute MUST be unique within the enclosing `<unit>` element.
- The value of each of the `order` attributes used within a `<unit>` element MUST NOT be higher than N, where N is the number of all current `<segment>` and `<ignorable>` children of the said `<unit>` element.

See the Segments Order section for the normative usage description.

4.3.1.25 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: IRI.

Default value: undefined

Used in: `<file>`.

4.3.1.26 priority

Priority - provides a way to prioritize notes.

Value description: Integer 1-10.

Default value: 1

Used in: `<note>`.

Note

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

4.3.1.27 ref

Reference - holds a reference for the associated element.

Value description: A value of the [XML Schema Datatypes] type anyURI. The semantics of the value depends on where the attribute is used:

- When used in a <note> element, the URI value is referring to a `<segment>`, `<source>` or `<target>` element within the same enclosing `<unit>`.

  When used in a term annotation, the URI value is referring to a resource providing information about the term.

- When used in a *translation candidates annotation*, the URI value is referring to an external resource providing information about the translation candidate.

- When used in a comment annotation, the value is referring to a `<note>` element within the same enclosing `<unit>`.

- When used in a custom annotation, the value is defined by each custom annotation.

Default value: undefined

Used in: `<note>`, `<mrk>` and `<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>
```

4.3.1.28 srcDir

Source directionality - indicates the directionality of the source content.

Value description: `ltr` (Left-To-Right), `rtl` (Right-To-Left), , or `auto` (determined heuristically, based on the first strong directional character in scope, see [UAX #9]).

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

- When used in `<file>`:

  The value `auto`.

- When used in any other element:

  The value of the `srcDir` attribute of its parent element.

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

4.3.1.29 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>`.

4.3.1.30 startRef

Start code or 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>
```

4.3.1.31 state

State - 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.

The 4 defined states constitute a simple linear state machine that advances in the above given order. No particular workflow or process is prescribed, except that the three states more advanced than the default `initial` assume the existence of a translation within the segment. One can further specify the state of the translation using the `subState` attribute.

Default value: `initial`

Used in: `<segment>`

Processing Requirements

- When the optional `state` attribute is added to a `<segment>` element, its value MUST be set to `initial` if the element doesn't have a `<target>` child. All valid values MAY be used when a `<target>` child is present.
- Writers updating the attribute `state` MUST also update or delete `subState`.

Note

`state` is an OPTIONAL attribute of segments with a default value and segmentation can change as the XLIFF roundtrip progresses, hence implementers don't have to make explicit use of the attribute. However setting of the attribute is advantageous if a workflow needs to make use of Advanced Validation methods.

4.3.1.32 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.

4.3.1.33 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.

4.3.1.34 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.

4.3.1.35 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>`

Constraints

- If the attribute `subState` is used, the attribute `state` MUST be explicitly set.

Processing Requirements

- Writers updating the attribute `state` MUST also update or delete `subState`.

4.3.1.36 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 sub-values for a specific authority. The sub-value is any string value defined by the 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>`

Constraints

- If the attribute `subType` is used, the attribute `type` MUST be specified as well.
- The reserved `xlf:` prefixed values map onto the `type` attribute values as follows:

  For `xlf:b`, `xlf:i`, `xlf:u`, `xlf:lb`, and `xlf:pb`, the REQUIRED value of the `type` attribute is `fmt`.
  For `xlf:var`, the REQUIRED value of the `type` attribute is `ui`.

Processing Requirements

- Modifiers updating the attribute `type` MUST also update or delete `subType`.

4.3.1.37 trgLang

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>`.

4.3.1.38 translate

Translate - indicates whether or not the source text in the scope of the given `translate` flag is intended for translation.

Value description: `yes` or `no`.

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

- When used in `<file>`:

  The value `yes`.

- When used in any other admissible structural element (`<group>` or `<unit>`):

  The value of the `translate` attribute of its parent element.

- When used in annotations markers `<mrk>` or `<sm>`:

  The value of the `translate` attribute of the innermost `<mrk>` or `<unit>` element, in which the marker in question is located.

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

4.3.1.39 trgDir

Target directionality - indicates the directionality of the target content.

Value description: `ltr` (Left-To-Right), `rtl` (Right-To-Left), or `auto` (determined heuristically, based on the first strong directional character in scope, see [UAX #9]).

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

- When used in `<file>`:

  The value `auto`.

- When used in any other element:

  The value of the `trgDir` attribute of its parent element.

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

4.3.1.40 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
  `quote` - Inline quotation (as opposed to a block citation)
  `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.

  Example:

```
<segment>
  <source><pc id="q1" type="quote">Blázen,
      chce dobýt to nu v takovém po así</pc>, dodal slovy svého
      oblíbeného imaginárního autora.</source>
  <target><pc id="q1" type="quote">Madman, he wants to conquer the
      pole in this weather</pc>, offered he the words of his
      favourite imaginary playwright.</target>
</segment>
```

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 sub-value separated by a character `:` (U+003A).

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

  Default value: `generic`

- When used in `<group>` or `<unit>`:

  A value that 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 sub-values for a specific authority. The sub-value is any string value defined by the authority. The prefix `xlf` is reserved.

  Default value: undefined

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

Processing Requirements

- Modifiers updating the attribute `type` on `<pc>`, `<sc>`, `<ec>`, or `<ph>` MUST also update or delete `subType`.

4.3.1.41 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>` and `<sm>`.

4.3.1.42 version

XLIFF Version - is used to specify the version of the XLIFF Document. This corresponds to the version number of the XLIFF specification that the XLIFF document adheres to. For this specification, the version is 2.2.

Value description: 2.0, 2.1 or 2.2

Used in: `<xliff>`.

4.3.2 XML namespace

The attributes from XML namespace used in XLIFF 2.2 are: xml:lang and xml:space.

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

- When used in a `<source>` element:

  The value set in the `srcLang` attribute of the enclosing `<xliff>` element.

- When used in a `<target>` element:

  The value set in the `trgLang` attribute of the enclosing `<xliff>` element.

- When used in any other element:

  The value of the `xml:lang` attribute of its parent element.

Used in: `<source>`, `<target>` and where extension attributes are allowed.

4.3.2.2 xml:space

White spaces - the xml:space attribute specifies how white spaces (ASCII spaces, tabs and line-breaks) are to 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 values for this attribute depend on the element in which it is used:

- When used in `<data>`:

  The value `preserve`.

- When used in `<xliff>`:

  The value `default`.

- When used in any other element:

  The value of the `xml:space` attribute of its parent element.

Used in: `<xliff>`, `<file>`, `<group>`, `<unit>`, `<source>`, `<target>`, and `<data>`.

4.4 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

- Agents MUST process CDATA sections.
- Writers MAY preserve the original CDATA sections.

4.5 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

- Agents 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.

4.6 XML Processing Instructions

XML Processing Instructions [XML] (see specifically http://www.w3.org/TR/REC-xml/#sec-pi) are an XML mechanism to "allow documents to contain instructions for applications." XML Processing Instructions are allowed in XLIFF content but they are ignored in the parsed content in the same sense as XML Comments.

Processing Requirements

- Agents MUST NOT use Processing Instructions as a means to implement a feature already specified in XLIFF Core or Modules.
- Writers SHOULD preserve XML Processing Instructions in an XLIFF document.

Warning

Note that Agents using Processing Instructions to implement XLIFF Core or Module features are not compliant XLIFF applications disregarding whether they are otherwise conformant.

Warning

Although this specification encourages XLIFF Agents to preserve XML Processing Instructions, it is not and cannot be, for valid processing reasons, an absolute protection and it is for instance highly unlikely that Processing Instructions could survive an XLIFF roundtrip at the `<segment>` level or lower. Hence implementers are discouraged from using XML Processing Instructions at the `<segment>` and lower levels.

4.7 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 in an `<originalData>` element.

4.7.1 Text

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

4.7.1.1 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 [Unicode].

However, XML does not have the capability to represent all Unicode code points [Unicode], 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].

4.7.2 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 `<br/>` 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 cannot 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.

4.7.2.1 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'/>

4.7.2.2 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:

```
<unit id="1">
  <originalData>
    <data id="d1">\b </data>
    <data id="d2">\i </data>
    <data id="d3">\b0 </data>
    <data id="d4">\i0 </data>
  </originalData>
  <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>
</unit>
```

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, Extractors and Modifiers 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, Modifiers 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 (see <ec>)
type	type	type
subType	subType	subType
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	dir

- Agents MUST be able to handle any of the above two types of inline code representation.

4.7.2.3 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.

4.7.2.3.1 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>
```

4.7.2.3.2 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">
  <originalData>
    <data id="d1">&lt;B></data>
    <data id="d2">&lt;/B></data>
  </originalData>
  <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>
</unit>
```

Note

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

4.7.2.4 Adding Codes

When processing 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

- Modifiers 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.
- Mergers MAY ignore added inline codes when merging the translated content back into the original format.

There are several ways to add codes:

4.7.2.4.1 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 uses the same data.

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

```
<unit id="1">
  <originalData>
    <data id="d1">&lt;b></data>
    <data id="d2">&lt;/b></data>
  </originalData>
  <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>
</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

- Modifiers 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.

4.7.2.4.2 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 formatting.

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

```
<unit id="1">
  <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>
  <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="n1" dataRefEnd="n2">Arrêt d'urgence
        </pc>) pour interrompre le compte à rebours. </target>
  </segment>
</unit>
```

4.7.2.4.3 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 Translation Memory, or better candidates from a Machine Translation 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">
  <originalData>
    <data id="d1">&lt;ui></data>
    <data id="d2">&lt;/ui></data>
  </originalData>
  <segment>
    <source>Click <pc id="1" dataRefStart="d1" dataRefEnd="d2">
        Next</pc></source>
  </segment>
</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).

4.7.2.5 Removing Codes

When processing 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:

4.7.2.5.1 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">
  <originalData>
    <data id="d1">&lt;i></data>
    <data id="d2">&lt;/i></data>
  </originalData>
  <segment>
    <source>I read <pc id="1" dataRefStart="d1" dataRefEnd="d2">Little
        House on the Prairie</pc> to my children.</source>
    <target>         </target>
  </segment>
</unit>
```

4.7.2.5.2 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 equivalent 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 Translation Memory, or better candidates from a Machine Translation 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 Translation Memory.

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

4.7.2.6 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 4.7.2.4.1, "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**
>
> Note that often those properties are related and appear together. For example, the code in the first 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">
  <originalData>
    <data id="d1">%s</data>
  </originalData>
  <segment>
    <source>Can't open '<ph id="1" dataRef="d1" canCopy="no"
        canDelete="no"/>'.</source>
  </segment>
</unit>
<unit id="2">
  <originalData>
    <data id="d1">%s</data>
    <data id="d2">%d</data>
  </originalData>
  <segment>
    <source>Number of <ph id="1" dataRef="d1" canCopy="no"
        canDelete="no" canReorder="firstNo"/>: <ph id="2" dataRef="d2"
        canCopy="no" canDelete="no" canReorder="no"/>. </source>
  </segment>
</unit>
```

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

Constraints

- When the attribute `canReorder` is set to `no` or `firstNo`, the attributes `canCopy` and `canDelete` MUST also be set to `no`.

- Inline codes re-ordering within a source or target content MAY be limited by defining non-reorderable sequences. Such sequence is made of a first inline code with the attribute `canReorder` set to `firstNo` and zero or more following codes with `canReorder` set to `no`.

- A non-reorderable sequence of codes MUST NOT start with a code with the attribute `canReorder` set to `No` and zero or more following codes with `canReorder` set to `no`

> **Note**
>
> A non-reorderable sequence made of a single code with `canReorder` set to `firstNo` are allowed just for extraction convenience and are equivalent to a code with the attribute `canReorder` set to `yes`.

Processing Requirements

- Extractors SHOULD set the `canDelete`, `canCopy` and `canReorder` attributes for the codes that need to be treated differently than with the default settings.
- Modifiers MUST NOT change the number and order of the inline codes making up a non-reorderable sequence.
- Modifiers MAY move a whole non-reorderable sequence before or after another non-reorderable sequence.
- When a non-reorderable sequence is made of a single non-reorderable code, Modifiers MAY remove the `canReorder` attribute of that code or change its value to `yes`.
- Modifiers MUST NOT delete inline codes that have their attribute `canDelete` set to `no`.
- Modifiers MUST NOT replicate inline codes that have their attribute `canCopy` set to `no`.

> **Note**
>
> Conformance of codes to Editing Hints Processing Requirements within translations can only be checked on existing `<target>` elements, i.e. non-conformance is not reported on `<segment>` or `<ignorable>` elements without `<target>` children.

4.7.3 Annotations

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

Annotations MAY be created by an Extractor that generated the initial XLIFF document, or by any other Modifier or Enricher later in the process. For example, after an Extractor creates the document, an Enricher can annotate the source content with terminological information.

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

4.7.3.1 Type of Annotations

There are several pre-defined types of annotation and definition of custom types is also allowed.

4.7.3.1.1 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 or inherited at the `<unit>` 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>.
```

4.7.3.1.2 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:

```
<file id="f-t_a">
  <unit id="1">
    <segment>
      <source>He is my <mrk id="m1" type="term"
          ref="http://dbpedia.org/page/Doppelgänger">
          doppelgänger</mrk>. </source>
    </segment>
  </unit>
</file>
```

4.7.3.1.3 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. If and only if the `value` attribute is not present, the `ref` attribute MUST be present and contain the URI of a `<note>` element within the same enclosing `<unit>` 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">
  <notes>
    <note id="n1" appliesTo="target">Please check the translation for
        'namespace'. One also can use 'espace de nom', but I think most
        technical manuals use the English term.</note>
  </notes>
  <segment>
    <source>You use your own namespace.</source>
    <target>Vous pouvez utiliser votre propre <mrk id="m1"
        type="comment" ref="#n=n1">namespace</mrk>.</target>
  </segment>
</unit>
```

4.7.3.1.4 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>.
```

4.7.3.2 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>
```

4.7.4 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>
```

Constraints

- 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.
- Sub-flows MUST be in the same `<file>` element as the `<unit>` element from which they are referenced.

Processing Requirements

- Extractors SHOULD store each sub-flow in its own `<unit>` element.
- Extractors MAY order the `<unit>` elements of the sub-flows and the `<unit>` element, from where the sub-flows are referenced, as they see fit.

Note

Note that the static structure encoded by `<file>`, `<group>`, and `<unit>` elements is principally immutable in XLIFF Documents and hence the unit order initially set by the Extractor will be preserved throughout the roundtrip even in the special case of sub-flows.

4.7.5 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. See the definition of the `<data>` element.

Processing Requirements

- For the inline content and all non empty inline elements: The white spaces MUST be preserved if the value for `xml:space` set or inherited at the enclosing `<unit>` level is `preserve`, and they MAY be preserved if the value is `default`.

4.7.6 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 `auto`.

The `<group>` and `<unit>` elements also have the two OPTIONAL attributes `srcDir` and `trgDir`. The default value of the `srcDir` is inherited from the value of the `srcDir` attribute of the respective parent element. The default value of the `trgDir` attribute is inherited from the value of the `trgDir` attribute of the respective parent element.

The `<pc>`, `<sc>`, and isolated `<ec>` elements have an OPTIONAL attribute `dir` with a value `ltr`, `rtl`, or `auto`. The default value is inherited from the parent `<pc>` element. In case the inline element is a child of a `<source>` element, the default value is inherited from the `srcDir` value of the enclosing `<unit>` element. In case the inline element is a child of a `<target>` element, the default value is inherited from the `trgDir` value of the enclosing `<unit>` element.

Warning

While processing isolated `<ec>` elements with explicitly set directionality, please beware that unlike directionality set on the `<pc>` and `<sc>` , this method decreases the stack level as per [UAX #9].

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

Directionality of source and target text contained in the <source> and <target> elements is fully governed by [UAX #9], whereas explicit XLIFF-defined structural and directionality markup is a higher-level protocol in the sense of [UAX #9]. The XLIFF-defined value auto determines the directionality based on the first strong directional character in its scope and XLIFF-defined inline directionality markup behaves exactly as Explicit Directional Isolate Characters, see [UAX #9], http://www.unicode.org/reports/tr9/#Directional_Formatting_Characters.

Note

Note that this specification does not define explicit markup for inline directional Overrides or Embeddings; in case those are needed. Extractors and Modifiers will need to use [UAX #9] defined Directional Formatting Characters.

For instance, HTML elements <bdi> and <bdo> need both extracted as a <pc> or <sc> / <ec/> pair with the dir attribute set respectively.

All XLIFF defined inline directionality markup isolates and <sc> / <ec/> isolated spans can reach over segment (but not unit) boundaries. This needs to be taken into account when splitting or joining segments (see Segmentation Modification) that contain inline directionality markup. Albeit It is not advisable to split segments, so that corresponding inline directionality markup start and end would fall into different segments, such a situation is not too confusing. If this happens, the "watertight" BiDi box will simply span two or more segments. This is not too confusing because no XLIFF defined directionality markup is allowed on <source>, <target>, or , so all higher level protocol inheritance of directionality in such cases is from <unit> or higher.

4.7.7 Target Content Modification

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

The Extractor MAY create the initial target content as it sees fit.

The Merger is assumed to have the same level of processing and native format knowledge as the Extractor. 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 Writers modifying the target content of an XLIFF Document between the Extractor and the Merger 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.

4.7.7.1 Without an Existing Target

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

Processing Requirements

- Writers MAY leave the segment without a target.
- Modifiers MAY create a new target as follows:
  - Modifiers MAY add translation of the source text.
  - Modifiers MUST put all non-removable inline codes in the target.

- Modifiers MUST preserve the order of all the non-reorderable inline codes.
- Modifiers MAY put any removable inline code in the target.
- Modifiers MAY add inline codes.
- Modifiers MAY add or remove annotations.
- Modifiers MAY convert any `<pc>` element into a pair of `<sc>` and `<ec>` elements.
- Modifiers MAY convert, if it is possible, any pair of `<sc>` and `<ec>` elements into a `<pc>` element.

4.7.7.2 With an Existing Target

When working with a segment with content already in the target, Writers MUST choose one of the three behaviors described below:

Processing Requirements

- Writers MAY leave the existing target unchanged.
- Modifiers MAY modify the existing target as follow:
  - Modifiers MAY add or modify translatable text.
  - Writers MUST preserve all non-removable inline codes, regardless whether or not they exist in the source.
  - Writers MUST preserve any non-reorderable inline codes in the existing target.
  - Writers MUST NOT add any non-reorderable inline codes to the target.
  - Modifiers MAY remove any removable inline codes in the target.
  - Modifiers MAY add inline codes (including copying any cloneable inline codes of the existing target).
  - Modifiers MAY add or remove annotations.
  - Modifiers MAY convert any `<pc>` element into a pair of `<sc>` and `<ec>` elements.
  - Modifiers MAY convert, if it is possible, any pair of `<sc>` and `<ec>` elements into a `<pc>` element.
- Modifiers MAY delete the existing target and start over as if working without an existing target.

4.7.8 Content Comparison

This specification defines two types of content equality:

- Equality type A: Two contents are equal if their normalized forms are equal.
- Equality type B: Two contents are equal 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 [UAX #15].
- 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.

4.8 Segmentation

In the context of XLIFF, a segment is 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. Material provisions regarding segmentation can be found for instance in the Segmentation Rules eXchange standard [SRX] or [UAX #29].

4.8.1 Segments Representation

In XLIFF each segment of processed content is represented by a `<segment>` element.

A `<unit>` can comprise a single `<segment>`.

Each `<segment>` 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 `<segment>`.

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>
```

4.8.2 Segments Order

Some Agents (e.g. aligner tools) can segment content, so that the target segments are not in the same order as the source segments.

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, where N is the sum of the numbers of the `<segment>` and `<ignorable>` elements within the given enclosing `<unit>` element.

Warning

When Writers set explicit `order` on `<target>` elements, they have to check for conflicts with implicit `order`, as `<target>` elements without explicit `order` correspond to their sibling `<source>` elements. Beware that moving one `<target>` element is likely to cause a renumbering domino effect throughout the enclosing `<unit>` element.

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>
```

4.8.3 Segmentation Modification

When modifying segmentation of a `<unit>`, Modifiers MUST meet the Constraints and follow the Processing Requirements defined below:

Constraints

- Integrity of the inline codes MUST be preserved. See the section on Inline Codes and on Annotations for details.
- The entire source content of any one `<unit>` element MUST remain logically unchanged: `<segment>` elements or their data MUST NOT be moved or joined across units.

Warning

Note that when splitting or joining segments that have both source and target content it is advisable to keep the resulting segments linguistically aligned, which is likely to require human linguistic expertise and hence manual re-segmentation. If the linguistically correct alignment cannot

be guaranteed, discarding the target content and retranslating the resulting source segments is worth considering.

Processing Requirements

- When the Modifiers perform a split operation:
  - Only `<segment>` or `<ignorable>` elements that have their `canResegment` value resolved to `yes` MAY be split.
  - All new `<segment>` or `<ignorable>` elements created and their `<source>` and `<target>` children MUST have the same attribute values as the original elements they were created from, as applicable, except for the `id` attributes and, possibly, for the `order`, `state` and `subState` attributes.
  - Any new `id` attributes MUST follow the `<segment>` or `<ignorable>` `id` constraints.
  - If there was a target content in the original segment and if the `state` attribute of the original segment was not `initial`, the `state` attributes of the segments resulting from the split (and possibly their corresponding `subState` attributes) MAY be changed to reflect the fact that the target content MAY need to be verified as the new segmentation MAY have desynchronized the alignment between the source and target contents.
- When the Modifiers perform a join operation:
  - Only `<segment>` or `<ignorable>` elements that have their `canResegment` value resolved to `yes` MAY be join with other elements.
- When the Modifiers or Mergers perform a join operation:
  - Two elements (`<segment>` or `<ignorable>`) MUST NOT be joined if their `<target>` have resolved `order` values that are not consecutive.
  - The attributes of the elements to be joined (`<segment>` or `<ignorable>`) and the attributes of their `<source>` and `<target>` MUST be carried over in the resulting joined elements.
  - If attributes of elements to be joined (`<segment>` or `<ignorable>`) differ, or if the attributes of their `<source>` or `<target>` differ, the resulting joined elements MUST comply with following rules:
    - If the `state` attributes of the `<segment>` elements differ: the `state` attribute of the joined `<segment>` MUST be set to the "earliest" of the values specified in the original `<segment>` elements. The sequence of `state` values are defined in the following order: 1: `initial`, 2: `translated`, 3: `reviewed`, and 4: `final`.
    - The `subState` attribute MUST be the one associated with the `state` attribute selected to be used in the joined `<segment>`. If no `subState` attribute is associated with that `state`, the joined `<segment>` MUST NOT have a `subState`.
    - If the `xml:space` attributes differ: The `<source>` and `<target>` of the joined element MUST be set to `xml:space="preserve"`.
- When the Modifiers or Mergers perform a join or a split operation:
  - If any `<segment>` or `<ignorable>` element of the `<unit>` had a `<target>` child with an `order` attribute prior to the segmentation modification, the `<target>` child of all `<segment>` and `<ignorable>` elements in the `<unit>` MUST be examined and if necessary their `order` attributes updated to preserve the ordering of the target content prior the segmentation modification.

4.8.4 Best Practice for Mergers (Informative)

Since a typical simple corporate implementation of XLIFF 2 is a localization tool that is at the same time an Extractor and a Merger with the full knowledge of the extraction mechanism, the community requested a non-normative best practice for merging after an XLIFF Round-trip.

First of all, it needs to be noted that Mergers are not advised to rely on their knowledge of the extraction mechanism in terms of segmentation. Modifiers are free to change segmentation during the roundtrip and even to change order of target content held in different segments of the same unit. Therefore, it can be advised as a best practice before merging to look for all segments within each unit, even and especially when the Extractor had created only one segment per unit.

When joining segments, Mergers need to observe all Processing Requirements for joining segments and joining or splitting segments

When joining segments it can happen that not all `<segment>` or `<ignorable>` elements actually have their `<target>` element children. This situation can be legal depending on a specific workflow set up. The `<target>` child within an `<ignorable>` element is always optional, but at the same can be created any time by simply copying the content of the sibling `<source>`, see Content Modification Without Target. The presence of `<target>` children can be better governed in `<segment>` elements that have the `state` attribute. The `state` attribute is strictly optional with the default `initial`, yet it is advisable for a corporate localization operation to request that their service providers progress that attribute through `translated` and `reviewed` to `final`. This attribute cannot be progressed from the `initial` state without a `<target>` child and all violations of Editing Hints will become validation errors only in the `final` state. Usage of `state` also allows for fine-tuning of a specific workflow State Machine with the dependent `subState` attribute. With the attribute `subState`, implementers can create an arbitrary number of private state machine under their prefix authorities. It is advisable to register such authority prefixes with the XLIFF TC and publish their documentation.

When Mergers need to perform the merge in a non-final state, when the presence of targets cannot be guaranteed, they are free to create preliminary targets again following the Processing Requirements for Content Modification Without Target

4.9 Extension Mechanisms

XLIFF 2.2 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.

4.9.1 Extension Points

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

- `<xliff>`
- `<file>`
- `<group>`
- `<unit>`

The following XLIFF Core elements accept custom attributes:

- `<xliff>`

- <file>
- <group>
- <unit>
- <note>
- <mrk>
- <sm>

4.9.1.1 Extensibility of XLIFF Modules

For extensibility of XLIFF Modules please refer to the relevant Module Sections.

4.9.2 Constraints

- When using identifiers, an extension MUST use either an attribute named `id` or the attribute `xml:id` to specify them.
- Extensions identifiers MUST be unique within their immediate <file>, <group> or <unit> enclosing element.
- Identifier values used in extensions MUST be of type `xs:NMTOKEN` or compatible with `xs:NMTOKEN` (e.g. `xs:NAME` and `xs:ID` are compatible).

These constraints are needed for the fragment identification mechanism.

4.9.3 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 functionality at the provided extension points.
- Mergers MUST NOT rely on custom namespace extensions, other than the ones possibly defined in <skeleton>, to create the translated version of the original document.
- Writers that do not support a given custom namespace based user extension SHOULD preserve that extension without modification.

Appendix A MIME Type for XLIFF Version 2.0 and Later Releases (Normative)

A MIME type (Multipurpose Internet Mail Extensions type) is a two-part identifier for file formats and format content transmitted on the Internet. The MIME type is the mechanism used to tell a client application the type of document being transferred from a server. It is important that servers are set up correctly so that the correct MIME type is transferred with each document.

XLIFF is registered in the IANA Media Types Registry as `application/xliff+xml`.

Appendix B XLIFF Grammar Files (Normative) (Normative)

The basic grammar and structure of XLIFF 2.2 is defined using ten (10) XML Schemas and one (1) XML catalog. The module schemas are specifically referenced from their respective modules.

1. XLIFF Core [XML Schema],

   https://docs.oasis-open.org/xliff/xliff-core/v2.2/wd01/schemas/xliff_core_2.0.xsd

2. [XML Catalog] of XLIFF Defined XML Schemas,

   https://docs.oasis-open.org/xliff/xliff-core/v2.2/wd01/schemas/catalog.xml

3. XML Schemas of XLIFF Modules are referenced from those modules.

B.1 XML Schemas Tree

```
Core XML Schema
    |
    +---Candidates Module XML Schema
    |
    +---Glossary Module XML Schema
    |
    +---Format Style Module XML Schema
    |
    +---Metadata Module XML Schema
    |
    +---Resource Data Module XML Schema
    |
    +---Size and Length Restriction Module XML Schema
    |
    +---Validation Module XML Schema
    |
    +---ITS Module XML Schema (W3C namespace subset)
    |
    +---ITS Module XML Schema (additional attributes)
    |
    +---Plural, Gender, and Select Module
```

B.2 Support Schemas

Third party support schemas that are normatively referenced from this specification or from the machine readable artifacts that are a part of this multipart product are distributed along with the XLIFF-defined schemas in a subfolder named `informativeCopiesOf3rdPartySchemas` and further subdivided in folders according to the owner/maintainer of the schema.

Warning

Schema copies in this sub-folder are provided solely for implementers convenience and are NOT a part of the OASIS multipart product. These schemas belong to their respective owners

and their use is governed by their owners' respective IPR policies. The support schemas are organized in folders per owner/maintainer. It is the implementer's sole responsibility to ensure that their local copies of all schemas are the appropriate up to date versions.

Currently the only included third party support schema is http://www.w3.org/2001/xml.xsd [http://www.w3.org/2009/01/xml.xsd] at https://docs.oasis-open.org/xliff/xliff-core/v2.2/wd01/schemas/informativeCopiesOf3rdPartySchemas/w3c/xml.xsd in this distribution.

Appendix C Specification Change Tracking (Informative)

This is to facilitate human tracking of changes between XLIFF Versions 2.2 and 2.1.

1. Produced two versions of the specification:

   Core: simplified version that does not include optional modules.
   Extended: complete version, including all modules.

2. Changed namespace for the core module to `urn:oasis:names:tc:xliff:document:2.2`.

3. Added an optional `ref` attribute to `<note>` element.

4. Changed the type of `version` attribute to an enumeration containing 2.0, 2.1 and 2.2 as valid values.

5. Allowed an optional `<notes>` element at the start of `<xliff>`.

6. Allowed an optional <*mda:metadata*> element at the start of `<xliff>`.

7. Removed references to Schematron, NVDL and Test Suite from this specification.

8. Removed the informative Change Tracking Extension.

9. Renamed Appendix B to "XLIFF Grammar Files (Normative)".

10. Added new *Plural, Gender, and Select Module*.

11. Updated Appendix A with the official MIME type listed in IANA Media Type Registry.

12. Allowed an optional `<notes>` element in <*res:resourceItem*>.

In spite of the above mentioned changes, fixes, clarifications, and additions, the practical workings of the previous versions of the XLIFF Core have not been affected. All valid XLIFF 2.0 and 2.1 files are valid XLIFF 2.2 files. The changes introduced in version 2.2 are designed to preserve compatibility with versions 2.0 and 2.1.

NVDL and Schematron files used in previous versions of XLIFF are available at https://github.com/oasis-tcs/xliff-xliff-22/tree/master/xliff-21.

Appendix D Acknowledgements (Informative)

The following individuals have participated in the creation of this specification and are gratefully acknowledged:

- Filip, David - Huawei Technologies Co., Ltd.
- Morado Vázquez, Lucía - University of Geneva
- Nita, Mihai - Google Inc.
- Raya, Rodolfo M. - Individual
- Schnabel, Bryan - Individual
- Souto Pico, Manuel - cApStAn SA
- Umaoka, Yoshito - IBM