XLIFF 2.0 vs XLIFF 1.2

FEISGILTT Dublin June 2014

Yves Savourel
ENLASO Corporation
Why a version 2

• 1.2 specification had ambiguities and lacked constraints and processing requirements
• Issues to fix (e.g. `<mrk>` cannot overlap)
• New features needed
• Portions of the 1.2 specification rarely used

Despite those issues XLIFF 1.2 has been used successfully by many, in many scenarios
→ So XLIFF 2.0 should be used even more.
1.2 Issue: Specification ambiguities

2.0 remedies:

• Definitions try to be clearer
• **Constraints** and **processing requirements** are everywhere in the specification
• Provide more **examples** to illustrate the intent of the elements/attributes
• A **test suite** is available
• More and earlier **implementations**
1.2 Issue: Features creep

2.0 remedies:

• Mitigated through **modularity**
  (Descartes’ Method: break down large problems into small ones)
  → Allows to addresses smaller more specific problems one at a time: XLIFF 2 is a mosaic of “small standards”

• Plan for incremental changes of the specification. This is possible because of the modularity
  (Modifications have no or less impact on tools)
1.2 Issue: Too complex

2.0 remedies:

• Modularity again: it allows for simple to complex content:
  – Simple XLIFF = just the Core
  – Additional features added through modules

→ You implement/use only what you need in your workflow

• One way to do one thing: e.g. inline codes
2.0 vs 1.2 main differences

• 2.0 is not backward compatible with 1.x
  → Allows deep re-design
    - to break things into modules
    - different segmentation representation

• 1.2 has many features and they have not all been ported to 2.0
  → idea is to add specialized modules over time
Language pairs

• In 1.2: Each `<file>` in the document can be in a different language pair.

• In 2.0: All `<file>` in the document are in the same language pair. The `srcLang` and `trgLang` attributes are set on the `<xliff>` element.
Start of document in 1.2

xliff¹
  file+
    header?
      skl?
        phase-group*
          (glossary|reference|note|tool
count-group|prop-group|
  extension)*

body¹
  (group|trans-unit|bin-unit)*
Start of document in 2.0

```
xliff¹
  file+
    skeleton?
    (module|extension)*
  notes?
    (group|unit)+
```
The trans-unit in 1.2

trans-unit
  source\(^1\)
  seg-source?
    mrk*
  target?
    mrk*
    (context-group|count-group|
      prop-group|note|alt-trans)*)
  extension*
The `unit` in 2.0

```
unit
  (module|extension)*
notes?
originalData?
  (segment+|ignorable)*
  source
  target?
```
Segmentation in 1.2

<trans-unit id="u1">
  <source>Sentence 1. Sentence 2.</source>
  <seg-source><mrk mtype="seg" mid="1">Sentence 1.</mrk><mrk mtype="seg" mid="2">Sentence 2.</mrk></seg-source>
  <target><mrk mtype="seg" mid="1">Phrase 1.</mrk><mrk mtype="seg" mid="2">Phrase 2.</mrk></target>
</trans-unit>
Segmentation in 2.0

<unit id="u1">
  <segment id="1">
    <source>Sentence 1. </source>
    <target>Phrase 1. </target>
  </segment>
  <segment id="2">
    <source>Sentence 2. </source>
    <target>Phrase 2. </target>
  </segment>
</unit>
Segmentation in 2.0

- **New** `canResegment` attribute to allow or not to re-segment. Available on `<file>`, `<group>`, `<unit>` and `<segment>` (not available in 1.2)

- **New** `order` attribute on `<target>` to have the target in a different order than the source (not needed in 1.2)
Target state

• 1.2 has state (final, needs-adaptation, needs-l10n, needs-review-adaptation, needs-review-l10n, needs-review-translation, needs-translation, new, signed-off, translated)

• 2.0 has state (initial, translated, reviewed, final) + subState with a custom value
Target state qualifier

• In 1.2: Mix of values for target in `<trans-unit>` and `<alt-trans>` (e.g. where the translation comes from as well as why it was rejected)

• In 2.0:
  – Translation Candidates module’s `<match>` has a type and subType
  – No specific qualifier for the current translation in `<unit>` but subState is available
Inline content – Original codes

• In 1.2: Many elements (<g>, <x/>, <bx/>, <ex/>, <ph>, <bpt>, <ept>, <it>) and storing the original code is done within the segment. Also: conversion between equivalent elements (like <g> and <bpt/>/<ept/>) is not lossless.

• In 2.0: Fewer elements: <ph/>, <pc>, <sc/> and <ec/>. Original codes optionally stored outside the segment. Lossless conversion.
Inline content – Original codes

• In 1.2:
  <source>Line 1. <ph id="1">&lt;BR&gt;Line 2.</source>

• In 2.0:
  <originalData>
  <data id="d1">&lt;BR&gt;</data>
  </originalData>
  ...
  <source>Line 1. <ph id="1" dataRef="d1">Line 2.</source>
Inline content – Original codes

- 1.2 has only one editing hint: `clone` (and not on all inline elements)
- 2.0 has `canCopy` (equivalent to `clone`), `canOverlap`, `canDelete` and `canReorder` as well as mechanism to create new inline codes. Has also constraints and processing requirements associated with these flags.
Inline content – Sub-flows

• In 1.2: in `<sub>` element within the code content (possibly recursively)
  Or in a separate `<trans-unit>` but without interoperable link.

• In 2.0: Elements for inline codes have a `subFlows` attribute to point to another `<unit>` where the sub-flow text is located.
Codes – Text representation

• 1.2: `equiv-text` attribute

• 2.0: two attributes:
  – `equiv` provides a text equivalent (same as 1.2: provides empty string, spaces, line breaks, etc. in plain text)
  – `disp` provides a user-friendly display (e.g. to display some context for a variable)
Inline content – Annotations

• In 1.2: `<mrk mtype="seg">`
  In 2.0: `<segment>` *(structural, not annotation)*

• In 1.2: `<mrk mtype="protected">`
  In 2.0: `<mrk translate="yes|no">`

• In 1.2: `<mrk comment="text">`
  In 2.0:
  `<mrk type="comment" value="text">`
  `<mrk type="comment" ref="#n=noteId">`
Inline content – Annotations

• In 1.2: No way to have overlapping \texttt{<mrk>}
  \textit{→} Important obstacle to implement any type of annotation, for example another standard such as ITS.

• In 2.0: Use \texttt{<sm/>} and \texttt{<em/>} lossless conversion with \texttt{<mrk>...</mrk>
The `translate` attribute

- In 1.2:
  ```xml
  translate="$yes|no" on <group>, <trans-unit>
  and <bin-unit>
  <mrk mtype="protected"> (but no way to un-
  protect nested content)
  ```

- In 2.0:
  ```xml
  translate="$yes|no" on:
  <file>, <group>, <unit> and <mrk>
  ➔ <unit id="1" translate="no"> does not
  mean there is nothing to translate in the unit. More
difficult to implement, but more powerful.
<alt-trans> proposal

- In 1.2: Candidates are in <alt-trans> with alttranstype=proposal (default)
- In 2.0: Candidates are marked up using the Translation Candidates module (<matches>)
<alt-trans> proposal

• In 1.2: <alt-trans> applies to the whole source if it doesn’t have the \texttt{mid} attribute, to a segment when it does have it.

• In 2.0: <match> applies to any span in the content. This allows candidates to match across segments, on segments, on sub-segments parts.
In 1.2: \texttt{match-quality} is the only measurement available and it is equivalent to a similarity score.

In 2.0: Several distinct values:
- \texttt{similarity} (how source candidate is similar to source)
- \texttt{matchQuality} (how “good” is the translation)
- \texttt{matchSuitability} (overall indicator, can be used to sort candidates from the same origin).
In 1.2: The `<alt-trans>` element with `alttranstype="previous-version"`, etc. allows to store some level of track changes.

In 2.0: The Change Tracking module allows to record successive versions of changes for various items.
Glossary

• In 1.2: The `<glossary>` element is just a place to store custom glossary (no specification about the format, etc.)
• In 2.0: The Glossary module offers a simple format with the basic information: source, definition (optional), translations (optional).
<bin-unit> element

• Supported through to the 2.0 Resource Data module (used along with <unit>).
• The Resource Data module can also provides context information for the translators, such as screen shots, etc.
Size and Length Restriction

• 1.2: Attributes \texttt{maxwidth}, \texttt{minwidth}, \texttt{maxbytes}, \texttt{minbytes} \textbf{and} \texttt{size-unit}

• 2.0: The Size and Length Restriction module is a full-fledged specialized module allowing for profiles, specification of the encoding to use, the normalization to perform, handling on the inline code size/length, etc.
Extensions

• Not allowed everywhere and they have constraints and processing requirements
• Simply treat them like modules you don’t implement.

The main difference between a module you don’t implement and an extension is that you MUST preserve the module and (only) SHOULD preserve the extension.
Extension points for elements

• In 1.2:
  `<alt-trans>, <bin-unit>, <group>, <header>, <tool>, <trans-unit> and <xliff>`

• In 2.0 Core:
  `<file>, <group>, <unit> and <skeleton>`
Extension points for attributes

• In 1.2:
  <alt-trans>, <bin-source>, <bin-target>, <bin-unit>, <bpt>, <bx>, <ept>, <ex>, <file>, <g>, <group>, <it>, <mrk>, <ph>, <seg-source>, <source>, <target>, <tool>, <trans-unit>, <x> and <xliff>

• In 2.0 Core:
  <xliff>, <file>, <group>, <unit>, <note>, <mrk> and <sm>
Metadata module

• Not in 1.2, but similar to the `<prop-group>` and `<prop>` elements in 1.0 (was deprecated in 1.1)
• Allows to carry basic custom information without defining your own namespace.
• Tools can offer generic edit/display for such metadata.
Inline content – Invalid characters

- No equivalent in 1.2
- Some special characters cannot be represented in XML (e.g. control characters)
- In 2.0: Use `<cp hex="HHHHH">` where HHHHH is the hexadecimal Unicode code of the character.
- Same as in LDML (Unicode’s Locale Data Markup Language)
Fragment identifiers

• No equivalent in 1.2
• Several sets of IDs in XLIFF
  → Cannot use the usual \#id notation
    (because id may be duplicated)
• More and more needed for linked data
• In 2.0: Specific fragment identifier syntax
defined for XLIFF MIME type. Syntax supports
modules and extensions.
Format Style module

• No equivalent in 1.2
• In 2.0: Aim at offering a place where to define metadata needed to output an HTML “preview” of the document. The \texttt{fs} and \texttt{subFs} attributes fill that role.
Validation module

- No equivalent in 1.2
- A “must-support” for QA tools
- Allows to check for basic presence or absence of strings or sub-strings, number of occurrences, etc.
- Options for normalization, case-sensitivity
- One aspect missing: regular expression (very difficult to standardize)
What about features not in 2.0?

- XLIFF 2 is meant to **evolve**
- TC needs to get the requirements, the proposals and the implementations for new features
- Future modules can be implemented using extensions first, then moved to a module (e.g. ITS Allowed Characters to replace the 1.2 charclass attribute).
Overall

• 2.0 is a better foundation for tools
• It can evolve incrementally with less (and even no) disruptions to existing 2.0 tools
• Somewhat more difficult to implement (many constraints and processing requirements)
  → But one can use libraries
• May take up more disk space
  → But packages are often zipped nowadays
Links

• XLIFF 2.0 Specification:
  http://docs.oasis-open.org/xliff/xliff-core/v2.0/xliff-core-v2.0.html

• XLIFF 1.2 Specification:
  http://docs.oasis-open.org/xliff/xliff-core/xliff-core.html

• TC Comment Mailing List:
  https://lists.oasis-open.org/archives/xliff-comment