Cross tool interoperability for translation edit operations

In order to support multiple tools from different vendors to cooperate in modifying the target of segments and perform segmentation operations on XLIFF files we need to define the schema and processing expectations (allowed / required transforms and operations) to reduce ambiguity and constrain the allowed operations. It is also desirable to place as few limits as possible on a tool working on files created by that tool.

Definitions

Tool – Throughout the document the term tool is used often with a prefix (initial, final or intermediate). This refers to a specific software, or suite of software from one vendor designed to interoperate with each other. It is expected that a tool may assume different roles depending on what file it is processing. Therefore it is desirable that the file contain some indication of the identity of the initial tool.

Initial tool – The tool used to convert a native document into XLIFF. It may optionally perform any number of transformations and edit operations on the file. The only constraint on the output of the Initial tool is that the file is valid XLIFF. The initial tool may leverage any amount of proprietary knowledge to aid its processing.

Final tool – The tool used to convert the XLIFF file back to a native document. This will in most cases also be the initial tool, but could be different. Proprietary knowledge about the initial tool will most likely be required to act as the final tool.

Intermediate tool: Any tool not classified as initial or final is intermediate. These tools are assumed to have no knowledge about the process used to convert between the native format and XLIFF. They should also not need to depend on knowledge about the native format. These tools must produce valid XLIFF and must comply with the processing expectations described here.

Schema support

Tool identification: A mechanism to store the identity of the initial tool is necessary in the XLIFF document. If merging multiple XLIFF documents into one, like with multiple <file> elements in 1.2 it might be desirable to place the tool identity on the sub document level.

Native content identification: Independent and consistent identification of native content. For example as provided by the 'nid' attribute on all inline tags in the 2.0 draft or by placing the native code as escaped PCDATA inside the non-spanning inline tags. Spanning tags should have two attributes to identify the starting code and ending code. This is used to support flattening as described below.

Clone-ability: An attribute or flag that indicate if an inline code can be cloned (ie. duplicated). If an inline code can be cloned a new inline element is created with all attributes and child nodes the same as the cloned element. This includes content in third party namespaces. The value of the 'id' attribute is not cloned, a new 'id' is assigned to the clone. The 'clone' attribute in XLIFF 1.2 fills this purpose for some inline tags. When cloning a starting or ending code (ie. <sc> and <ec>) it is important to also create a clone of the corresponding opposite code to keep the document valid.

Removability: An attribute or flag indicating if an inline code present in source, or already present in target, can be removed from target or if it must be placed in target when the target is created.

Reorder-ability: An attribute or flag indicating that the inline code should not be re-ordered with respect to other inline codes in the segment that have this flag set. This also means that the tags nesting cannot be changed if its parent tag has this flag set. A tag not being re-orderable also implies that it is non-clone-able and non-removable. This rule comes from the fact that allowing removal and addition of a code implicitly allow reordering.

Flattening: There should be a defined way to convert a spanning inline tag into a pair of start and end tags. This is necessary to overcome the restriction in XLIFF 1.2 about segmentation of material with non-clone-able spanning inline tags. To facilitate safe flattening without information loss and to also allow un-flattening **a property indicating that the code has XML element behavior** (no overlap, nesting) is needed. All spanning codes (<pc>) by their nature have this property but the other paired codes (<sc> and <ec>) does not necessarily follow the rules of XML. So if a <pc> is converted to a <sc>,<ec> pair without such a flag this property is lost. Similarly if a <sc>,<ec> pair (without such a property) is converted into a <pc> additional restrictions on code placement are added that did not exist before conversion. When flattening the native code of the <sc> tag corresponds to the <pc> tags starting code and the <ec> tags native code corresponds to the <pc> tags ending code.

Segment Processing

These proposed processing expectations and the schema design are only designed to provide interoperability in the case where the Initial and Final tools are the same tool. Providing an interoperable way to convert native documents or data into XLIFF with one tool and back to the native format with another is outside the scope of this specification.

Initial tool

The initial tool is allowed to modify the XLIFF file as it sees fit. It may leverage proprietary information about the processing or native format and by doing so break the clone-ability and removability constraints. It is also allowed to place any mix of inline codes in both source and target. The codes may be completely different or the same. No restrictions apply to the initial tool.

1) Intermediate tool – no target

An intermediate tool follows a set of rules regarding inline codes when adding a target to a segment in an XLIFF file.

a. MUST put ALL non-removable inline codes in the target.
b. MAY put ANY removable tag into the target.
c. MAY put additional copies of clone-able tags in target.
d. MUST preserve the same order in target as there is in source for any non-re-orderable codes.
e. MAY split the segment into two segments
    a. When splitting MUST flatten any <pc> tag spanning the two segments in to a <sc> + <ec> pair. The flag indicating XML semantics for the code MUST be set on this pair.
f. MAY merge the segment with the following segment
g. MAY flatten ANY <pc> code into a <sc> + <ec> pair. The flag indicating XML semantics for the code MUST be set on this pair.
h. MAY unflatten ANY <sc> + <ec> pair with the XML semantic flag set into a <pc> span.

2) Intermediate tool – existing target

When working with a segment with content already in the target node the tool may choose between three behaviors.

- Preserve target unchanged
- Modify existing target
    a. MUST preserve ALL non-removable inline codes in target regardless if they exist in source.
    b. MAY remove ANY removable inline codes in target
    c. MAY put additional copies of clone-able tags in target, this includes any preexisting codes in source or target.
    d. MUST preserve any non-re-orderable codes in target
    e. MUST NOT add any non-re-orderable codes to target
    f. MUST NOT Split the segment into two segments.
    g. MAY merge the segment with the following segment
    h. MAY flatten ANY <pc> code into a <sc> + <ec> pair. The flag indicating XML semantics for the code MUST be set on this pair.
    i. MAY unflatten ANY <sc> + <ec> pair with the XML semantic flag set into a <pc> span.
- Delete the whole target node and start over as if working with no target.

The reason to have separate rules for this case is to allow the initial tool to deliberately break the rules on re-ordering, removability and clone-ability and still allow other tools to safely process and modify the file. The reason to not allow splitting of segments with content in the target node is because there is no guarantee that the content in the two nodes are linguistically in the same order, allowing that operation would pose a risk to the integrity of the content.

Final tool

The final tool is assumed to have the same level of processing and native format knowledge as the initial tool and follow the same rules.