LetsMT!

Platform for Online Sharing of Training Data and Building User Tailored MT
www.letsmt.eu/

Project no. 250456

Deliverable D2.1
Specification of data formats

Version No. v1.1

May 25, 2011

Document Information

Deliverable number:	D2.1
Deliverable title:	Specification of data formats
Due date of deliverable according to DoW:	31/08/2010
Actual submission date of deliverable:	31/08/2010
Main Author(s):	Jörg Tiedemann, Per Weijnitz
Participants:	Uppsala
Reviewer	Tilde, Moravia
Workpackage:	WP2
Workpackage title:	SMT resource repository and data processing facilities
Workpackage leader:	Uppsala
Dissemination Level:	PU
Version:	v1.1
Keywords:	data formats

Version History

Version	Date	Status	Name of Author (Partner)	Contributions	Description/ Approval Level
v0.1	July 17, 2010	initial draft	Uppsala		ready for reviewing
v0.2	August 17, 2010	final draft	Uppsala		
v1.0	August 30, 2010	final version	Uppsala		approved by all partners
v1.1	May 23, 2011	revised version	Uppsala		

EXECUTIVE SUMMARY

This is a description of data formats that will be used internally in the LetsMT! data resource repository and data formats that will be allowed when uploading parallel and monolingual data to the LetsMT! platform. The report includes a discussion of issues that have to be handled for various formats and points to solutions and tools that will be used in the project.

Contents

List of Figures

List of Tables

1 Task Description

One of the key functions of the LetsMT! platform is to provide the possibility to train domain specific SMT models tailored towards specific needs of its users. For this appropriate data resources are required. LetsMT! is based on data sharing and user collaboration. The scope of workpackage two is to develop facilities to store, process and manage resources coming from LetsMT! users. The purpose of this report is to specify data formats that will be used internally when storing parallel and monolingual corpus data in the repository and to list data formats that will be supported for the user contributions. The main focus here is to balance robustness and flexibility, meaning that supported upload formats have to be handled in a stable and robust way. Each additional formats usually increases complexity of the validation and conversion processes and, therefore, may reduce overall robustness of the system. Restricting the system to a few well supported formats will lead to user satisfaction more than a brittle system with a lot of possibilities would do. Adding support for new formats can always be done later on after carefully testing the integration of upload possibilities and after checking possible interferences with existing features.

2 Internal Storage Formats

In this section we will describe the internal data formats that we will use to store training data in the shared repository. We will use a unified format for all textual data in the repository which makes it easy to select subsets of the entire collection for training tailored SMT engines. All incoming data will be converted to this format and possibly enriched using pre-processing tools. We will also store the raw uploads to ensure data recovery and to make it possible to adjust conversion and pre-processing if necessary.

2.1 Corpus Data

All corpus data, regardless whether it is part of a parallel corpus or part of a monolingual corpus, will be stored in a simple XML format.[1] XML is a well-established and widely used markup language for the representation of structured information. XML comes with fixed standard specifications [1], naively supports Unicode [9], and the main advantage is that a lot of tools (parsers, transformation tools, query

[1] XML markup increases the size of textual data significantly. In the case of very large monolingual corpora it might be necessary to skip this additional markup and to support plain text formats as well. However, the disadvantage with this is that data must be handled in different ways according to the storage format and that a possible support for factors must be implemented in a different way.

tools, etc) are available for processing XML data efficiently. Nowadays XML can be considered to be the standard for sharing and exchanging structured information and, therefore, using XML in LetsMT! seems to be a natural choice.

For our purposes we decided to apply a simple self-contained XML annotation without restricting ourselves to any existing XML schema or document type definition. The main reason for this is to keep our repository flexible enough to be extended (for example with additional linguistic markup) without being bound to certain specifications defined for other purposes. For LetsMT! it is not necessary to follow given standard schema's as we are not providing data for download and sharing for other purposes than training SMT engines within our platform.

Basically we will only require some basic markup:

- `<s id="...">...</s>` sentence boundaries with a unique ID within the document (not necessarily unique within the entire document collection) We use a rather loose definition of sentence here – basically sentences refer to the units to be used as the text fragments to be translated (or aligned) which may be list items, table cells, translation units from translation memories, etc

- `<w>...</w>` word/token boundaries; additional attributes (in the sense of XML tag attributes) can be added to store arbitrary features of that word that can possibly be used as a factor in SMT training

Additional markup may also be included in XML documents which might be useful for some tasks, for example, domain/style-specific sentence alignment that makes use of formatting information. We intentionally want to keep the schema open to allow extensions and adjustments without the need to check possible interferences with other collections. For SMT, we basically require to be able to extract a tokenized and sentence aligned parallel corpus which is possible through the markup mentioned above. Allowing arbitrary attributes in the word boundary tags also allows us to add optional annotation that can be used in SMT training. A typical example would be to store lemmas or root forms in such attributes. Consider, for example, the following sample from the Europarl corpus that has been marked up in this way (see figure 1).

In the sample from figure 1 we can see that there are four attributes for each token marked with `<w>` tags. Two of them contain part-of-speech tags from two different taggers ('tnt' for the TnT tagger and 'tree' for the TreeTagger) and one contains the lemma (also produced by the TreeTagger). It is now possible to use any combination of these attributes and the actual word to create the input for the training procedure (supporting factored models or not). There is also additional

```
<s id="5">
 <chunk type="NP" id="c-1">
  <w tree="NN" tnt="NNP" lem="madam" id="w5.1">Madam</w>
  <w tree="NP" tnt="NNP" lem="President" id="w5.2">President</w>
 </chunk>
 <w tree="," tnt="," lem="," id="w5.3">,</w>
 <chunk type="PP" id="c-3">
  <w tree="IN" tnt="IN" lem="on" id="w5.4">on</w>
 </chunk>
 <chunk type="NP" id="c-4">
  <w tree="DT" tnt="DT" lem="a" id="w5.5">a</w>
  <w tree="NN" tnt="NN" lem="point" id="w5.6">point</w>
 </chunk>
 <chunk type="SBAR" id="c-5">
  <w tree="IN" tnt="IN" lem="of" id="w5.7">of</w>
 </chunk>
 <w tree="NN" tnt="NN" lem="order" id="w5.8">order</w>
 <w tree="SENT" tnt="." lem="." id="w5.9">.</w>
</s>
```

Figure 1: Corpus data after pre-processing.

"chunk" information that can possibly be used for other purposes or simply ignored otherwise.

Sentences in parallel corpora need to have a unique ID within the document in order to be aligned properly to the corresponding sentences in another language as we will explain in the next section. Monolingual corpora will be annotated in the same way in order to match the target factors that are used in the translation model. Sentence IDs are not so important there as they do not have to be linked to anything else. However, they can still be useful to define partial selections of some corpora if necessary without reproducing data in the repository.

2.2 Alignment Information

Each parallel corpus has to be aligned at the sentence level for the training procedures of the translation models. We will store links between sentences in external files pointing to the appropriate documents using the unique sentence IDs for identification of the aligned segments. For this we will use a simple XML format based on the XCES standard [4]. Figure 2 illustrates a sample of such an alignment file.

Basically, the alignment file lists links between segments in the source document (stored in the `fromDoc` attribute) and the target document (stored in `toDoc`). The segments are tagged with the type `targType` – sentences (`<s>`) in out case. Links are stored in the `xtargets` attribute using the unique sentence IDs in

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE cesAlign PUBLIC "-//CES//DTD XML cesAlign//EN" "">
<cesAlign version="1.0">
  <linkList>
    <linkGrp targType = "s"
             fromDoc  = "Europarl/xml/eng/ep-00-01-17.xml"
             toDoc    = "Europarl/xml/fre/ep-00-01-17.xml">
      <link xtargets="1;1" />
      <link xtargets="2;2" />
      <link xtargets="3;3 4" />
      <link xtargets="4;5" />
      <link xtargets="5;6" />
      ...
      <link xtargets="904;888" />
      <link xtargets="905;889" />
    </linkGrp>
    <linkGrp targType = "s"
             fromDoc  = "Europarl/xml/eng/ep-00-01-18.xml"
             toDoc    = "Europarl/xml/fre/ep-00-01-18.xml">
      <link xtargets="1;1" />
      <link xtargets="2;2" />
      <link xtargets="3;3" />
```

Figure 2: Sentence alignments.

source and target document separated by one semicolon (`;`). Alignments may include multiple sentences on both sides (source and target) and may be empty as well. Multiple sentence IDs (sequences of sentences) are separated by one or more space characters. To give an example taken from figure 2: sentence `3` from document `Europarl/xml/eng/ep-00-01-17.xml` is aligned to sentences `3` and `4` in document `Europarl/xml/fre/ep-00-01-17.xml`. An "empty" alignment (one-to-zero link) would, for example, look like this: `<link xtargets="21;" />`.

Several other attributes might also be useful in sentence alignment files:

certainty: The likelihood of the given link according to some automatic alignment procedure (attribute of the link-tag)

evaluate: Status of the alignment (attribute of the link-tag). In the XCES specifications this attribute has to be "all", "one" or "none". We might abuse this attribute to specify whether the link has been manually approved, rated or rejected in some way.

release: This is would be a non-XCES standard attribute for the `linkGrp` tag which we can use to specify the version or release of the corpus in the data

repository. It will be important to match the appropriate version or the corpus if we work with a revision control system and possibly different releases of the same corpus.

The file names include the path to the corpus in the data repository. In this way a sentence alignment file includes sufficient information to check out appropriate corpora from the data repositories and to extract aligned sentences from them in the format required.

There are several advantages of this way of storing alignment information in external files:

- All possible language pairs in multilingual parallel corpora (with more than two languages) can be aligned without repeating any of the contents of the documents.
- Sentence alignment can easily be adjusted without changing the original corpus annotation.
- It is easy to make selections of certain parts of a corpus or of several corpora by simply listing the links between corresponding sentences. For example, it is very easy to extract one-to-one sentence alignments only or at least to ignore empty alignments. Furthermore, a collection of parallel corpora can be assembled by simply compiling appropriate `linkGrp` sections into one alignment file.
- It is straightforward and space efficient to store various revisions of sentence alignments. This can be very useful in case we want to support online alignment using various tools or even manual inspection and modifications.

2.3 Repository Structure

The LetsMT! data repository will be based on a version-controlled file system. We will use a simple and clear file structure to store parallel and monolingual data. Each corpus identified by a unique name (parallel or monolingual) will be stored in a separate version-controlled repository. The name of the corpus will be used as the name of this repository. Repositories can be created by any user but each user will only have access to his/her own branch inside this repository that will be set up during creation time. In this way, each user can work with a copy of existing corpora through branching (of course only if permissions allow that). This is space efficient and flexible allowing users to even apply changes to their copy without breaking data integrity. For each corpus the original user uploads will be

stored in a subdirectory `raw` and the pre-processed corpus files will be stored in a subdirectory `xml`. We will create subdirectories specifying the languages using ISO 639-3 language codes [5] which will contain the actual corpus files. Meta-information about the corpus and the tools used for conversion and pre-processing will be stored in the xml-directory (possibly in connection with a "Makefile" that can be used to manage the pre-processing jobs and dependencies between certain files).

Sentence alignments will be stored for each *parallel document* in the corpus using the same base name as the two linked documents. The alignment files will be placed in appropriate subdirectories relative to the home directory of the pre-processed data using a name composed of the two language codes (for example `eng-fre` for English-French). The entire collection of sentence alignments for each language pair within one corpus will be kept in one file in the root of the pre-processed data with a name indicating the languages that are linked (for example `eng-fre.xml` for the English-French sentence alignments). Sentence alignments are symmetric and, therefore, only one alignment direction has to be stored. Here, we will use the direction with language codes in alphabetic order to be explicit. Monolingual corpora will be saved in the same format except that no alignment information has to be stored. For each corpus appropriate meta-information including domain, owner, provider, size and other statistics will be stored in a central database to make browsing the archive fast and independent of the repository contents. This information may be repeated for supporting archive maintenance procedures within the repository either in the header of XML documents or in dedicated files stored together with the corpus data. Branching will be hidden and files will be located relative to the corpus root when checking out. Figure 3 shows an example of the file structure of a typical repository containing a parallel corpus (Europarl).

2.4 Revisions & Corpus Selections

An important feature in the LetsMT! will be the flexibility of selecting data from the repository for training dedicated SMT engines. Selecting parallel corpora can be done by creating specific sentence alignment files that contain all the links necessary to specify the aligned segments to be used in training the translation model. These sentence alignment files can use exactly the same format as we use for the individual parallel corpora including appropriate `linkGrp`'s for each selected portion of the repository. A simple routine can then be used to extract the training data from the corresponding data sources in the beginning of each training procedure.

The advantage of explicitly storing selected sentence alignments is the flexibility of this format. It is possible to combine several sources and also to combine

```
Europarl
Europarl/raw/
Europarl/raw/eng
Europarl/raw/eng/ep-00-01-17.txt
Europarl/raw/eng/ep-00-01-18.txt
Europarl/raw/fre
Europarl/raw/fre/ep-00-01-17.txt
Europarl/raw/fre/ep-00-01-18.txt
Europarl/raw/ger
Europarl/raw/ger/ep-00-01-17.txt
...
Europarl/xml/
Europarl/xml/eng
Europarl/xml/eng/ep-00-01-17.xml
Europarl/xml/eng/ep-00-01-18.xml
Europarl/xml/fre
Europarl/xml/fre/ep-00-01-17.xml
Europarl/xml/fre/ep-00-01-18.xml
Europarl/xml/ger
Europarl/xml/ger/ep-00-01-17.xml
...
Europarl/xml/eng-fre.xml
Europarl/xml/eng-ger.xml
Europarl/xml/fre-ger.xml
Europarl/xml/eng-fre/ep-00-01-17.ces
Europarl/xml/eng-fre/ep-00-01-18.ces
...
Europarl/README
Europarl/xml/README
Europarl/xml/Makefile
```

Figure 3: File structure in LetsMT! repositories.

only parts of parallel corpora from the repository. The common user will probably simply concatenate complete resources but advanced users may be interested in a more fine-grained selection. For example, the platform may support selection certain criteria, such as, alignment type, maximum sentence length, rating or banning of certain segments etc. Selections in form of sentence alignment files can simply be stored in the revision controlled file system with all its advantages as described above. Hence, corpus selections can be shared, changed, old versions can be retrieved, and so on.

In order to store selections special revision-controlled repositories can be created with a unique name. In these repositories selections can be stored for each user in user-specific branches. Users can then decide again if they want to share

these selections with others or not. These corpus selection repositories might be a good place as well to store intermediate data from the training procedures that can be re-used (for example word alignments) and possibly also the final models trained on that data collection.

Selections of monolingual data for language modeling can be handled in the same way (one repository per selection with user-specific branches). However, the actual data selection will look differently and it does not make much sense to store a list of selected sentences for this purpose as this list will be extremely large in most cases. Therefore, a list of selected corpora possibly in connection with information about selection criteria would be a better choice.

3 Upload Formats

Another important part of this deliverable is the specification of upload formats which will be supported by the LetsMT! platform. The formats are classified in two categories: easy/complicated/risky and required/desired. The required formats will be prioritized from the beginning, starting with easy cases, and the desired will be included at a later stage, if possible. The risky formats are less likely to be perfectly processed. The reasons for this may vary, see comments below. The main difference between the easy and the complicated file formats is that the easy formats imply multilingual text that has already been sentence aligned. The formats classified as complicated are easy to extract text from, but require automatic sentence alignment which is a source of errors.

Regarding monolingual resources, any of the supported file formats are acceptable.

For multilingual data consisting of sets of monolingual files, a file mapping plan must be provided by the user.

Concluding from deliverable D1.1 (requirement analysis) the main demand from potential users is concerned with pre-aligned parallel data in form of translation memories in TMX format. Together with our experience with the creation of parallel corpora from various sources we will, therefore, propose the following two phases in the development of the LetsMT! platform:

1. Support for the upload of pre-aligned parallel corpora in required formats such as TMX and Moses/GIZA++ plain text format. Support of monolingual data in various formats including DOC/DOCX, PDF and possibly XML.

2. Support for the upload of parallel documents without explicit sentence alignment in a few required formats (DOC/DOCX, PDF) with explicit warnings about the risks send to the user.

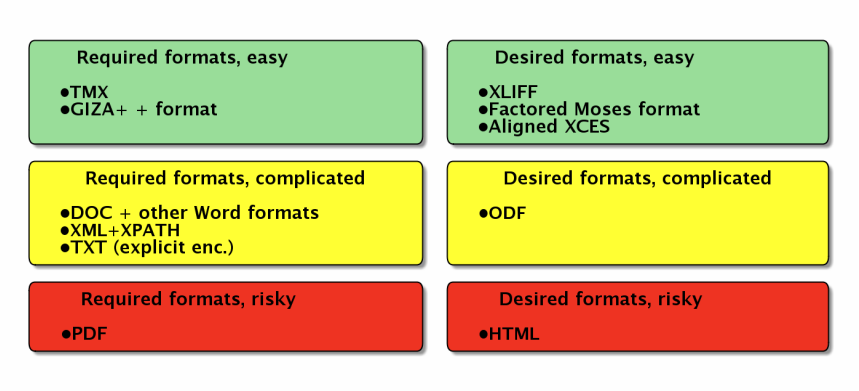

Figure 4: Classification of upload formats.

Phase 2 may include extended support for other formats for monolingual data and may also include some experimental services such as manual inspection and correction of automatic sentence alignment.

Below we will discuss some of the issues with particular data formats that have been discussed above and in the requirement analysis of deliverable D1.1.

3.1 Pre-aligned Parallel Corpora

The most important part of the data sharing facility in LetsMT! is the collection of parallel corpora that can be used for training translation models. The main contributions are expected to come from existing translation memories that have been collected in translation agencies or elsewhere. According to deliverable D1.1 the dominating exchange format is TMX (Translation Memory eXchange format). The next section will deal with this format. Other pre-aligned formats that we will support are the native plain text formats used in Giza++ and Moses, possibly even factored Moses format and maybe also XML/XCES data which is conform to the internal standards used in the LetsMT! data repository.

3.1.1 Translation Memory eXchange format (TMX)

TMX is XML-compliant and, therefore, easy to transform and to convert to the internal LetsMT! data format described in section 2. TMX specifications and official recommendations are available [7]. However, we have to be aware of vendor-specific changes and modifications that do not follow existing standards (for example in the use of language codes or encoding standards).

Basically, TMX consist of a collection of translation units (tagged with <tu> elements). Each translation unit contains a set of translation unit variants (tagged with <tuv>) for each language in which this unit has been translated to. Appropriate attributes specify language, encoding and other important meta-information. Figure 5 shows a short example (adapted from and example in [7]).

```
<?xml version="1.0"?>
<!-- Example of TMX document -->
<tmx version="1.4">
 <header creationtool="XYZTool"
         creationtoolversion="1.01-023"
         datatype="PlainText"
         segtype="sentence"
         adminlang="en-us"
         srclang="EN"
         creationdate="20020101T163812Z"
         o-encoding="iso-8859-1"  >
 </header>
 <body>
  <tu tuid="0001" >
   <prop type="x-Domain">Computing</prop>
   <tuv xml:lang="EN">
    <seg>data (with a non-standard character: &#xF8FF;).</seg>
   </tuv>
   <tuv xml:lang="FR-CA">
    <prop type="Origin">MT</prop>
    <seg>donn&#xE9;es (avec un caract&#xE8;re non
         standard: &#xF8FF;).</seg>
   </tuv>
  </tu>
  <tu tuid="0002">
   <prop type="Domain">Cooking</prop>
   <tuv xml:lang="EN"><seg>menu</seg></tuv>
   <tuv xml:lang="FR-CA"><seg>menu</seg></tuv>
   <tuv xml:lang="FR-FR"><seg>menu</seg></tuv>
  </tu>
 </body>
</tmx>
```

Figure 5: Example TMX file

TMX can be interpreted as a sentence aligned parallel corpus by simply extracting linked translation unit variants and treating them as aligned sentences in various languages. Standard stream-oriented XML parsers can be used to extract these units from arbitrary large translation memories. Sentence alignment files will then include one-to-one links only that refer to these extracted text fragments. Common

parsers[2] should take care of encoding and special characters automatically. However, some robustness must be added to cope with ill-formed XML input. This should be done in an initial validation step. Here, we also have to check possible difference between various versions of TMX. Necessary restrictions regarding supported versions will be investigated in development phase of WP2.

The largest challenge with the conversion from TMX will be the treatment of "inline" markup, i.e. content markup within text fragments that is frequently been used for formatting purposes or software specific instructions.

Figure 6 shows two example TMX files with content markup (taken from [7]) to illustrate the complexity of this problem.

Original RTF with footnote:

```
Elephants{\cs16\super \chftn {\footnote \pard\plain
\s15\widctlpar \f4\fs20
{\cs16\super \chftn } An elephant is a very
large animal.}} are big.
```

TMX with content markup:

```
Elephants<ph type="fnote">{\cs16\super \chftn {\footnote \pard\plain
\s15\widctlpar \f4\fs20
{\cs16\super \chftn } <sub>An elephant is a very
large animal.</sub>}}</ph> are big.
```

Original HTML:

```
See the <A TITLE="Go to Notes"
HREF="notes.htm">Notes</A> for more details.
```

TMX with content markup:

```
See the <bpt i="1" type="link">&lt;A TITLE="<sub>Go to Notes</sub>"
HREF="notes.htm"></bpt>Notes<ept i="1">&lt;/A></ept> for more details.
```

Figure 6: Content markup in TMX files.

This kind of markup is important when translating documents that need to preserve formatting information and link structures. However, for the purpose of training it should be better ignored in order to build appropriate translation and language models.

Looking at the examples in figure 6 this would mean that the second example will be reduced to "Go to the Notes for more details" and the link from "Notes" to

[2] We will probably use one of the open source XML parsers such as expat or libxml.

"notes.htm" will be stripped off. It would be possible to handle inline markup in the translation process by treating them as units with fixed translations (for example using the XML markup allowed in the Moses decoder to create input like "`Go to the <inline translation="...">INLINE</inline> Notes <inline translation="...">INLINE</inline>`"). However, this splits the input sequence and may prevent to match important phrase translation options such as "Go to the Notes" in the translation model. Additionally, the language model will also be negatively influenced by this interruption of the sequence.

The first example in figure 6 is a bit different. Here, an additional sentence is introduced as a footnote within the main sentence. It can be argued that the contents of this footnote should be extracted as additional information for the training process. This could be an option that we might explore treating footnote as a special case in sentence segmentation. However, this can be tricky in cases where proper corresponding segments cannot be found in the target language. For example, several footnotes may be part of one source sentence but their translations are not in the same order in the target language. Matching appropriate footnotes with each other is then not immediately straightforward anymore. Therefore, extracting these special units to obtain additional training material may do more harm than any good for the estimations of translation models. Anyway, we expect that we would only loose a tiny fraction of the training material when ignoring these units as well. We, therefore, recommend to ignore them as well but we will investigate these issues further during implementation of the pre-processing tools.

Finally, we also want to mention that ignoring inline markup will make the extraction process more robust. The risk of breaking modules in the training pipeline (for example tokenization, tagging, conversion to Moses format possibly including factors) is high when including arbitrary markup.

However, there are other types of placeholder markup in some localization data. They may refer to generic units, some kind of variables, that will be replaced by the actual contents when instantiated, for example, within a piece of software. These elements should not be ignored and we will investigate ways of integrating them in the pre-processing procedures to retain them in the training data.

Another problem in the treatment of uploaded translation memories will be the handling of software-specific formatting information. TM systems use different methods for storing format information and it is impossible to keep track of all ways that are in use. For example, content markup for text in italics might be specified as follows: "`text in {\i italics}`" instead of using content markup tags such as in "`text in <bpt i="1" type="italic">{\i </bpt> italics <ept i="1">}</ept>`". In the course of implementation these issues will be addressed in the best possible way.

At last, we will look at possibilities to re-use meta-information that can be in-

ferred from the TMX documents (for example domain information, creation time, source language). However, as we cannot expect to find this kind of information in all TMX uploads we will still require meta-information by the provider of the documents.

3.1.2 Plain text Giza++/Moses format

Giza++ and Moses use a simple plain text format as input for word alignment and training of translation models. Many researchers in SMT simply work with this format when collection data as it is naively supported by the Moses toolbox. It is, therefore, desirable to support it in LetsMT! as well. Even though it will also been used in the training procedures in LetsMT! we will still convert plain text uploads to our standard internal format (see section 2).

The Giza++/Moses format basically stores a parallel corpus in two separate text files containing the actual corpus data for source and target language, respectively. Files are aligned at the sentence level in such a way that corresponding sentences (or sequences of sentences) are stored at corresponding lines in the two files. It is, therefore, necessary that both files have exactly the same number of lines (which is easy to check) and newline characters are used to mark sentence boundaries and nothing else.

One problem with plain text files is that there are many different possible character encodings that can be used. It is not always possible to detect the correct encoding automatically and, therefore, we will rely on the user's specifications when uploading these files. We will assume Unicode and UTF-8 encoding as default which has to be clear to any possible user using the platform.

Besides encoding care has to be taken about special characters. Some characters have special meanings in Moses and have to be avoided or converted in some way. For example, '|' is used to separate factors and should not appear in plain text. XML markup is sometimes used to control certain behaviors of the decoder and we might need to treat those in a special way as well. It is also possible that certain characters can break pre-processing or the SMT training pipeline, for example, some ill-formed Unicode characters.

All these issues are not directly a problem in the uploading phase but will also appear for other input (for example data derived from TMX) when converting to Moses format before training. However, special attention has to be paid to encoding issues to avoid crashes already in the initial step when converting uploads to the internal format. Hence, careful validation and robust conversion is required for plain text uploads.

3.1.3 Factored Moses format

Moses supports also factored translation models. It might be useful to even support uploads that contain various factors in the plain text Moses format. However, this leads to many additional problems. Factors have to be separated by '|' characters and the same number of factors has to be specified for each token. Space characters are not allowed between factors as they will be treated as a token delimiter. Additional '|' characters in the plain text that have to be converted and which are not used as factor delimiter are difficult to detect. Factors have to be in the same order for all tokens. Using a factored upload assuming that annotation is correct is easy. Validation and detection of possible errors is tricky. However, simple checks rejecting ill-formed corpora are already in place in the Moses toolkit and could be integrated in LetsMT! However, factored uploads are not among the requested upload types and, therefore, have a lower priority.

3.1.4 XML Localization Interchange File Format (XLIFF)

Another XML-based exchange format is XLIFF [8] which is mainly used for localization supporting the entire process of translating localizable data to various languages. It borrows some of the concepts of TMX but serves a wider purpose than just storing example translations in a database. It allows, for example, explicit annotation of non-equivalent translations and alternative translation units. The general structure of an XLIFF document is shown in figure 7 (taken from [8]).

```
<xliff version='1.2'
       xmlns='urn:oasis:names:tc:xliff:document:1.2'>
<file original='hello.txt' source-language='en' target-language='fr'
       datatype='plaintext'>
<body>
<trans-unit id='hi'>
<source>Hello world</source>
<target>Bonjour le monde</target>
<alt-trans>
<target xml:lang='es'>Hola mundo</target>
</alt-trans>
</trans-unit>
</body>
</file>
</xliff>
```

Figure 7: The general structure of XLIFF

In LetsMT! we will be interested in the source and target contents. Information

about languages is available in the file tag or explicitly in the opening tags of source and target (using xml:lang attributes in the same way as in TMX). All other types of information can simply be ignored (for example, alternative translations).

As mentioned before, XLIFF also allows to mark translations as non-equivalent, meaning that such translations are not direct translations of the source text (see figure 8).

```
<trans-unit id="t1">
<source>Constrained text for limited</source>
<target equiv-trans="no">Tekst angielski dla</target>
</trans-unit>
<trans-unit id="t2">
<source>display for English</source>
<target equiv-trans="no">ograniczonego pola</target>
</trans-unit>
```

Figure 8: Non-equivalence in XLIFF

Furthermore, there are possibilities to group translations across translation units and to add markup for segmentation. All these advanced techniques make it rather complex to extract appropriate units from XLIFF files. Therefore, it seems to be wise to ignore all these special cases and extract source and target content only when no other interfering markup can be found (see also the discussions on inline markup in section 3.1.1). For this standard XML parsers can be used again which will take care of encoding and special XML entities. Inline markup that is used in the same way as in TMX files can be treated in the same way as we do for TMX. According to our user study XLIFF is not on the high priority list and, therefore, it will only be added if there is growing demand on including this format. The format is well defined and, therefore, it should not require a large effort to do so.

3.1.5 LetsMT!-conform XML/XCES

The internal LetsMT! format is not a general standard that is widely used in the community. Even though we try to follow the XCES standard at least for the alignment information there will be various ways of interpreting tags and attributes (for example the information stored in the xtarget attributes). The actual corpora can also be annotated with various types of markup. However, it is possible to accept a wide range of XML annotation if, at least, sentence boundaries are marked as required for our data repository and the sentence alignments. This can easily be verified and, as usual, standard stream-oriented XML tools can be used to process documents as needed (for example, to add tokenization markup).

The XML/XCES format will be used for project internal purposes only. There is no demand in adding this format in general as we do not expect that other users will use this type of annotation for their own data collections.

3.2 Unaligned Parallel Documents

In the second phase of LetsMT! we will also support the upload of parallel documents which have not been aligned at the sentence level yet. This implies that online alignment has to be integrated into the LetsMT! platform. Adding this functionality is a rather complex task. Several issues have to be considered starting from conversion and text extraction from various document formats, sentence boundary detection, tokenization and other necessary pre-processing steps up to the actual sentence alignment. It is planned to support various upload formats but we will concentrate on robustness in order to create reliable data resources in our platform. We will, therefore, only include formats for which we see the largest demand. According to the requirement analysis in D1.1 this will be the DOC/DOCX formats used in Microsoft Office (various versions), PDF and possibly plain text format with explicit specification of character encodings.

3.2.1 Microsoft Word DOC & DOCX

The largest problem with MS Word files is that the DOC format is a proprietary format with no open standard specifications. Microsoft has the freedom to change and adjust this format as often they like and this has happened frequently in various versions of the office package. Word documents include a lot of information that is difficult to handle such as revisions, formatting, styles, embedded objects etc. However, any file format that MS Word $>=$ 2007 can open and save as structured XML documents (using OpenXML) can be parsed with standard tools. The new XML-based DOCX formats used in recent versions of Microsoft Word is well documented in contrast to the previous proprietary solutions used by Microsoft. We will support those files mentioned above trying to extract plain text from the structured information. It will be important to instruct users who upload Word documents to inspect the results of this extraction and to warn them about the complications and risks when using these kind of documents.

During the implementations restrictions with regard to versions will be investigated. We will also investigate the use of available tools such as *antiword* [10] to include experimental support for older versions of Word documents. Another option is Apache Tika (http://tika.apache.org/) which can handle various file formats, for example Microsoft Office documents. Yet another toolkit for processing MS Word files is wvWare (http://wvware.sourceforge.net/) which is used

for conversion (Word import) in various word processors such as AbiWord and KWord. AbiWord itself can also be used for converting files on the command line. Converting Word documents to text could be done with a command like this: `abiword --to=txt document.doc`. Various other formats are supported, for example, rtf, utf-8, html, latex (all formats supported by AbiWord itself). This could be a valuable option. AbiWord should be rather up-to-date with recent versions of Microsoft Office and probably supports older versions as well.

Alternatively the official Word Viewer from Microsoft (can be downloaded from http://www.microsoft.com/downloads) or a running instance of Word could be used in some kind of batch processing mode. However, it is not clear how robust such a solution would be that has to run on a separate Windows server. It is definitely not build to handle a queue of incoming conversion requests and a work-around might not be very stable.

3.2.2 Portable Document Format (PDF)

PDF in general is hard to process. It was originally a proprietary format introduced by Adobe Systems but has been officially released as an open standard in 2008 (ISO/IEC 32000-1:2008). Several versions are around ranging from 1.0 up to 1.7. PDF can be seen as a container format that may include various types of objects. PDF documents may include images (vector graphics and raster images), text (including information about fonts and encodings), interactive elements (forms), file attachments and meta-data.

For LetsMT! only text objects will be interesting. We have to ignore PDF documents that include textual information in terms of images (scanned pages for example). Even extracting running text from PDF is not simple. Multi-line sentences are at risk of being separated. Tables, frames and pictures split the text. Character encodings, especially for non-western languages are problematic. We will present a PDF extraction solution which will be lossy and may compromise the document integrity. The user will be warned. PDF content consisting of images of scanned documents will not be processed.

Our solution will be based on available software. Several PDF readers and converters are around. We have already experience with `pdftotext` which is part of the `xpdf` package [2]. The tool is quite robust and supports several text encodings such as ISO-8859-1 (Latin 1), ISO-8859-2 (Latin 2 for Eastern European Languages), ISO-8859-7 (for Greek), KOI8-R (Cyrillic) and ISO-8859-8, ISO-8859-9 (for Hebrew and Turkish). There is also an option '-layout' that can be used to maintain the physical layout of a document and in some cases this can be useful to improve segmentation tasks such as sentence boundary detection (at least from our experience). However, layout information may cause problems to

identify coherent text blocks especially in multi-column layouts. We did some initial experiments using simple post-processing for column detection and table conversion with quite some success. However, LetsMT! users who want to upload PDF documents should have the chance to adjust parameters in order to improve extraction results. Other tools that we will investigate for the extraction of plain text from PDF documents are listed below:

Multivalent: http://multivalent.sourceforge.net/
PDFBox: http://pdfbox.apache.org/
Tika: http://tika.apache.org/
pdftoxml: http://pdftoxml.sourceforge.net/
Poppler: http://poppler.freedesktop.org/ (GPL version of Xpdf)
pdftohtml: http://pdftohtml.sourceforge.net/
pdftoword: http://www.pdftoword.com/

After the extraction of text, we will proceed with standard pre-processing in order to add sentence and token boundaries which will be necessary for the conversion to LetsMT! XML and the automatic sentence alignment.

3.2.3 Plain text

The main problem with plain text input is the large variety of character encodings that can be used. Therefore, we will require explicit specifications of encodings when uploading such documents. This problem should not be under-estimated. Using the wrong settings may cause serious failures in various parts of the pipeline. We still need to include tools for validation and encoding detection even if users will specify the format. Possible problems need to be detected as early as possible. Many users will not know about these issues and probably need some guidance before uploading their resources.

After validation we will process plain text with generic or language specific tools to add appropriate markup to convert files to our internal data format. Sentence alignment will be performed after the conversion.

3.2.4 Open Document Format (ODF)

The Open Document Format is another XML-based file format for representing electronic documents in office applications. It is used in various open source and proprietary software. It is even supported by Microsoft Office 2010 and its growing popularity makes this format a good candidate to be added if the demand increases. The biggest advantage of ODF is that it is an open standard and specifications are

available [6]. However, it still requires quite some effort to handle the complex structures of possible ODF documents correctly.

3.2.5 XML & HTML

Many XML-based document formats have been mentioned already. A general support for any XML document cannot be promised as the interpretation of XML markup very much depends on the application the annotation scheme was developed for. A generic procedure in reading character data from XML files ignoring all XML elements and attributes could be possible but might not be very useful in many cases. It might be more useful together with appropriate XPath expressions (provided by the user) that point at the locations within the XML documents from which the text is to be extracted. However, this is also of lower priority in the LetsMT! project as there does not seem to be a large demand for such a functionality.

With regards to HTML there is possibly a larger need to support web documents in our platform. However, a general support of HTML is tricky. For well formed HTML documents without frames with explicitly and correctly stated character encoding, HTML processing is not that risky. The general strategy will be to extract all text nodes, no matter where in the document structure they occur. The risky parts include missing/incorrectly declared character encodings, content in frames and by other means loaded parts from secondary sources, AJAX loaded content, and more. Many unforeseen issues may appear and may break the LetsMT! platform.

Several tools are available that can be used to check and cleanup HTML and XML markup. One option is, for example, HTML Tidy which can be downloaded from `http://tidy.sourceforge.net/`. Other tools are Perl modules such as HTML::Laundry, HTML::Scrubber, HTML::Defang, HTML::StripScripts, HTML::Detoxifier, HTML::Sanitizer (all available from CPAN) or, for example, htmlcleaner available from http://htmlcleaner.sourceforge.net/. Another potential candidate for our project is the robust website parser called "Beautiful Soup" which is available from `http://www.crummy.com/software/BeautifulSoup/`. We will investigate several possibilities during the development in case we want to add HTML support in the LetsMT! platform.

3.3 Monolingual Data

Another part of the data repository will contain large monolingual corpora for language modeling. In general larger quantities of data are required and support for many file formats is necessary to obtain enough training material. The good thing is

that proper pre-processing is not as crucial as it is for parallel resources. This is due to the fact that no alignment has to be performed. For example, a few segmentation errors do not cause serious problems like they often do in parallel corpora (for example, misalignment and error propagation due to erroneous sentence boundaries). Furthermore, missing parts are not a serious problem either whereas incomplete documents are very difficult to align. Therefore, it is less risky to include a wide range of file formats already in the first phase of the project for uploading monolingual data. Basically, we will support the formats discussed above starting with the more basic and frequently used ones and then moving to more complex and less widely spread formats. Meta-data such as language, domain and origin will be provided by the user as usual. The monolingual corpora will be subjected to the same permission and sharing policy as the other repository resources.

4 Validation of Data Uploads

Very important for the robustness of the system is proper validation of the data uploaded by the user. Data validation procedures depend very much on the data format and may include several steps. In general, we need to trust the input by the users when specifying the format of the documents included in any upload. We will require explicit specifications of all necessary parameters to identify the internal structures and formats of each upload. According to those parameters, validation and conversion processes will be started. Here, we will be as strict as possible in order to avoid any polution of our repository with noisy data sets. We will use standard tools and official specifications to carry out this task.

One example is the conversion of translation memory files in TMX format. First of all, we will use validating XML parsers to check the file format (for example tools and libraries from the libxml package [3]). We will use the official DTD:s specified for the latest TMX versions to validate the markup (available from [7]). Byte-order markers (BOM) will be used to identify encodings. Validation failures will be reported through status messages in the metadata database which will be accessible from the frontend of the LetsMT platform. Documents that cannot be validated will not be converted and, therefore, never enter the resource database. Logfiles of conversion processes will be stored in the repository in order to allow inspection of the processes that cause failures. Other formats will require other procedures but may re-use similar tools. For example XLIFF uploads will also be validated using common XML parsers but with appropriate DTD:s or XML schemas. Some formats will be very difficult to validate because they allow such a variety of information encoded inside of the document. This is, for example, the case for MS Word documents, PDF files and also for plain text file formats. In

format	conversion & validation
TMX	check BOM encoding (Perl module File::BOM) XML parsing with DTD validation (libxml, tmx.dtd) standalone XML validation without DTD for older versions of TMX conversion with our own LetsMT TMX conversion module (using standard stream-oriented XML parsers) special treatment of in-line markup (mostly ignore content)
XLIFF	check BOM encoding (Perl module File::BOM) XML parsing with XML Schema validation (libxml, xliff.xsd) standalone XML validation as backoff conversion with our own LetsMT XLIFF conversion module (using standard stream-oriented XML parsers) ignore 'alternative translations' and other markup
Text	check BOM encoding (Perl module File::BOM) possibly verify language & encoding (using language identifiers like `textcat`) general validation is difficult, need to trust user input
PDF	validation using `pdfinfo` (part of the xpdf package) conversion to text using `pdftotext` (also part of xpdf) typographic ligature normalizer (dedicated LetsMT module) conversion to LetsMT format via the *Text* module validation & conversion of PDF in general is difficult and error prone; users need to be warned
DOC	validation & conversion to text with Apache Tika (`http://tika.apache.org/`) conversion to LetsMT format via the *Text* module validation & conversion of MS Word documents is difficult due to the various versions and secrets of this format; users need to be warned
Moses	validation and conversion using the *Text* module special module to convert alignments and reserved characters
tar/zip/gzip	validate & unpack using standard GNU/Unix tools validation & conversion using format specific modules

Table 1: Preliminary list of and remarks on data format specific validation and conversion tools.

those cases, it is impossible for automatic processes to judge whether the content is fully acceptable or not. This is also the case for the outcome of automatic sentence alignment processes. Some internally implemented heuristics will help to reduce the noise and to flag possible errors in the repository. A preliminary list of tools that we will use for importing data is given in table 1.

The general procedures for any data import will be the same for every data format supported. We will first run validation tools on all documents. We proceed with conversion and data normalization if and only if the validation step succeeds. The status of each import will be reported in specific metadata fields that can be read by the web frontend. It is also important to mention that validation and conversion processes will be run as off-line processes that will not block the system otherwise.

The web frontend has to take care of the task to inform the user about validation and conversion errors. Note, that documents that could not be validated or converted correctly do not enter the repository and, therefore, do not pollute the data collection. Important is also that users are aware of possible problems and conversion limitations. This is another task for the interface developed for interacting with LetsMT users. In many cases it will be difficult to automatically detect errors especially if we allow uploads of arbitrary languages. Here, again, the web frontend has to enable possibilities to inspect conversion results (at least some snapshots) in order to give users the opportunity to improve or remove noisy content.

5 Risk Analysis

In this document we discuss our plans for the data sharing facilities in LetsMT! The formats mentioned and described will guide us in the development of the platform. There will be the need to adjust these goals due to unforeseen problems and changes but also according to new developments and opportunities that have been discovered in the course of implementation. We tried to outline risks already in the introduction and discussed the need for a priority list especially with respect to upload formats. The most important task for the data sharing facilities is to provide a robust and fast interface to the data repository. We, therefore, propose to concentrate on a few well supported formats first before adding further formats to the platform. In this way users will not be disappointed as often by broken and buggy processes and we do not have to count with heavy maintenance tasks when running the system.

Another important point is that users have to be aware of possible problems when uploading data in various formats. It is necessary to warn them about possible

risks and to give them the opportunity to inspect conversion results and to influence the upload and pre-processing steps. It is crucial to include validation tools for all formats supported and to check all aspects of a user upload in order to identify possible problems and to give feedback in case of detected errors.

Another big risk is the possibility of attacks and damage caused by hackers. Even though we require user authentication we will not be free of such attacks and we have to be aware of risks that certain file formats bear when we allow arbitrary uploads. This has to be carefully investigated during implementation and testing.

6 Conclusions

The success of LetsMT! crucially depends on the data sharing facilities that we will build during the project. In this deliverable we specify the internal data formats and the structure of the data repository. We will use a unified format for corpus data and sentence alignment which will be organized in a version-controlled file system.

Data uploads will be supported in a variety of formats. The implementation will be done in two phases:

Phase 1: Support for uploading pre-aligned parallel corpora in standard formats (TMX, Moses format) and for uploading monolingual data in popular formats (DOC, PDF, plain text). Experimental support for other formats for uploads of monolingual data. This will enable us to analyze the risks in including them in phase 2 even for parallel data uploads.

Phase 2: Support for uploading unaligned parallel documents in a few popular formats (DOC, PDF, plain text). Online sentence alignment and possibilities to inspect intermediate results (after conversion, pre-processing, alignment) and possibilities to influence these processing steps (adjusting parameters, selecting alternative tools that have been integrated in LetsMT!).
Start: M10

Data sharing facilities will be refined until the end of the project. Additional pre-processing tools will be included and the robustness of the system will be improved. Support of additional formats will also be added as time permits.

References

[1] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, François Yergeau, and John Cowan. *Extensible Markup Language (XML) 1.1*. Textuality and Netscape, Microsoft, W3C & Sun Microsystems, Inc., 2006. http://www.w3.org/standards/xml/ and http://www.w3.org/TR/xml11.

[2] foolabs. Xpdf - a toolkit for viewing and processing pdf documents. http://www.foolabs.com/xpdf/.

[3] Gnome. libxml – the xml c parser and toolkit of gnome. http://www.xmlsoft.org/, 2011.

[4] Nancy Ide, Keith Suderman, and Laurent Romary. *XCES – Corpus Encoding Standard for XML*. Department of Computer Science, Vassar College, Poughkeepsie NY, USA & Equipe Langue et Dialogue LORIA/CNRS, Vandoeuvre-lès-Nancy FRANCE, 2008. http://www.xces.org/.

[5] *ISO 639-3:2007 – Codes for the representation of names of languages – Part 3: Alpha-3 code for comprehensive coverage of languages*, 2007. http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=39534, http://www.sil.org/iso639-3/.

[6] OASIS. *OpenDocument v1.0 specification*. http://www.oasis-open.org/committees/download.php/12572/OpenDocument-v1.0-os.pdf.

[7] Yves Savoure. *TMX 1.4b Specification – OSCAR Recommendation*. The Localisation Industry Standards Association, 2005. http://www.lisa.org/tmx/tmx.htm.

[8] Yves Savourel, John Reid, Tony Jewtushenko, and Rodolfo M. Raya. *XLIFF Version 1.2 – OASIS Standard*. Organization for the Advancement of Structured Information Standards (OASIS), 2008. http://docs.oasis-open.org/xliff/xliff-core/xliff-core.html.

[9] Unicode, Inc. *The Unicode Standard: A Technical Introduction*, 2010. http://www.unicode.org/standard/principles.html.

[10] Adri van Os. On MS Word document readers (antiword), 2008. http://www.winfield.demon.nl/.