translation-memory-tools Documentation

Release 0.1

Softcatalá

November 15, 2014

Contents

Contents:

CHAPTER 1

Styleguide

translation-memory-tools developers try to stick to some development standards stated on the Pootle *Pootle Styleguide*.

CHAPTER 2

Technical design

2.1 Libraries

We use two data storage backends:

- Whoosh as full-text indexing and searching over the translation memories.
- SQLite as relational backend to keep track of statistics related to the translation memories built (latest fetch date, etc)

We use Mustache as template engine to generate outputs from source code to prevent mixing data and its representation.

2.2 Downloading, converting and building translation memories

builder.py launches the process to download and build the translation memories. The output is a set of PO and TMX files and a database with statistics of the files generated.

The file projects.json contains a definition of the projects for which we build the translation memories

Every project can contain a group of filesets. We support serveral fetch mecanisms

The process of building a transtlation memories works as follows:

- Every fileset is download using its downloaded mecanism (git, svn, file download, etc)
- All the files download (e.g ts, xml, etc) are converted to PO (the interal format to build the memories)
- All the segments are added a comment that indicates where the translation was imported from
- Translations memories are build using GNU Text (msgcat)
- When the process is finished we generate a TMX from the PO file

2.3 Searchable index

The file index-creation.py reads a set of PO files (translation memories) and builds a full text Whoosh index.

This index is used by the Web application.

The searchable index can be queried using JSON, using a this URL format '{0}/web_search.py?source={1}&project=tots&json=1'

2.4 Terminology extraction

The term-extract.py application run the terminology extraction module. It takes a set of PO files as source files, selects terminology and then generates a report in serveral formats (like HTML).

It uses term frequency to select the most common used terms. It uses serveral PO files to as quality measures of the output provided.

Limitations:

- It only analyzes segments of 3 words to avoid having to align terms in sentences (a complex problem)
- It does not recognizes derived forms like plurals or verbs tenses (it could be done easily using NLK)

CHAPTER 3

Deployment

3.1 Overview

A deployment consist of two artifacts:

- A set of data files generated by the execution of the tooling: translation memories, indexes for search and extracted terminology.
- The different web applications to make this data accessible to the user.

In the current configuration, it is assumed that the system building the data files and hosting the application is the same.

3.2 Getting started

There are two file locations:

- The generation location: where you generate the data files
- The deployment location: where files are deployed (usually /var/www...)

In the generation location, you need to do a git checkout of the tool in a subdirectory called "tm-git". And then, you can execute the generate*.sh scripts in the deployment directory of the git checkout. This takes a few hours and produces all the data files.

Finally, you need to execute the deploy script (deploy.sh) that creates a pre-production environment, executes the integration tests, and if they pass, then if copies all the files to the deployment location.

In git's deployment subdirectory there is a file generate-and-deploy.sh that shows how to use the different scripts to build the data files and deploy the application.