Localization of AOO

14 October 2012

Contents

Introduction..3
Overview..4
Actors and Systems..5
Developers..5
Translators..6
Integrators..6
Testers...6
System: SVN..6
System: pootle server..6
L10n workflow high altitude view......................................7
Content Creation...8
Upload pootle server...8
Translation..9
Translation online ("committer")...............................9
Translation offline (non "committer")........................9
Merge SVN..10
Update pootle server...10
Language build..10
Simplified data flow..11
L10n workflow technical view..12
Content Creation...12
Upload pootle server...12
Extraction from sources (generate new sdf file).........12
Merge with pootle server database.............................14
Translation..15
Translation online ("committer")...............................15
Translation offline (non "committer")........................15
Merge SVN..16
Update pootle server...16
Language build..17
File Formats..18
Tools..19
Open issues..20
Workflow is not a designed approach.............................20
Proposal..20
Tools are writing in multiple languages...........................20
Proposal..21
Use of sdf file...21
Proposal..21
Separate projects for UI and help...................................22
Proposal..22

Build process is highly manual and error prone..............22
Proposal..23
Automatic update of pootle server 23
Proposal..23
Content control..23
Proposal..23

Introduction

This document is based on and extents Localization_for_developers. The document is work in progress showing the result of a detailed technical analysis of the current process (version 3.4.1) . As such this document should be seen as a replacement of Localization_for_developers.

The l10n process only concerns itself about localizing defined supported languages. Adding a new language is a i18 process. This document is further restricted to the ongoing translation process and closely related build process. In case of external happenings, like e.g. Germany changing rules of spelling, it should be covered with i18 procedures.

The document will hopefully spark a discussion so it can be updated with other views from the ooo-L10 mail list.

It is important to understand the current process before we start discussing detailed changes, so this is the main purpose of this page. Once all the open issues at the end of document have been discussed as solutions agreed upon, a new document will be made describing the process as it should be in the near future.

Thanks to all those persons who contributed to Localization_for_developers that has been a great starting point for this document.

Overview

Localization, often abbreviated as l10n, defines the process to make a software package available in local languages, different to the language of the developer.

Localization is from the perspective of the involved person a multi-step process that involves a variety of tools and procedures. Most importantly the 4 main categories of involved persons have quite different and to some extent conflicting views and requirement, therefore the process should be a real “best of all worlds” approach.

The current process is more or less purely developer oriented, contains a lot of different tools and depends a lot on the responsibility of the involved people. It seems to be a process that has grown out of necessity more than a planned road.

Most of the tools used as well as the central data format (SDF) are specific to AOO and not used anywhere else even though both source (c++, resource, UI files) and target (po files) are standard file formats.

Only a part of the workflow are integrated in the build system. Much of it requires manual steps to be taken. Some of the tools involved are not part of the OpenOffice SVN and, due to a hard disk crash of the old pootle server, are lost.

Translations are done with the help of a pootle server. The localization work flow can very short be seen as:

- extraction messages from source files.
- uploading message to the pootle server
- translating messages on the pootle server
- downloading messages from the pootle server
- merging messages into source files

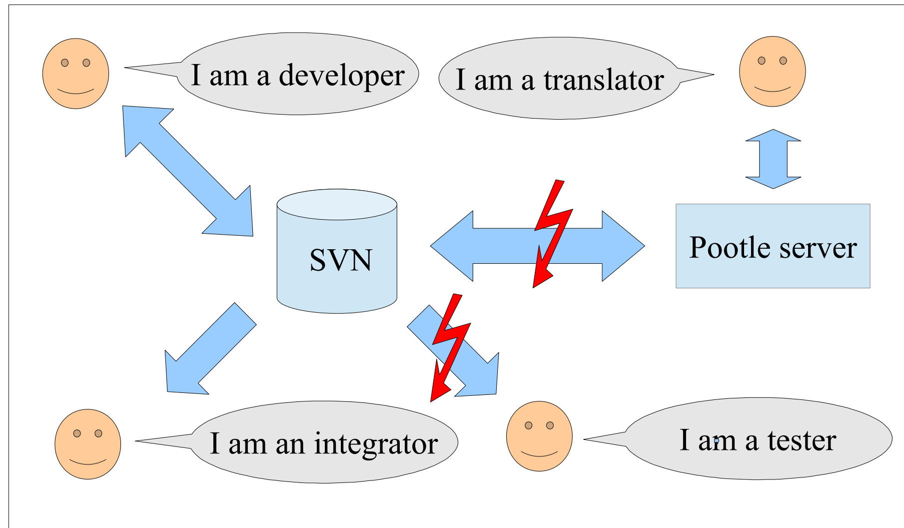
If you are looking for information about how to contribute translations then this page gives an (outdated) overview.

The document has 5 parts:

- a relative non-technical overview of the process,
- a detailed technical overview of the process
- a detailed technical data flow/storage view
- a detailed technical view of the tools used with parameters etc.
- an open issues list.

Actors and Systems

The l10n process can and should be viewed with respect to 4 different categories of people who access the process through 2 different systems. The translator consider pootle server to be repository whereas the others consider SVN the main repository.



Note: this view only relates to the l10n procedure, the picture for the whole project is a lot more complex.

The red lighting indicates that the pootle server only works indirectly on the SVN server.

The red lightning indicates that data is being copied:

- to/from pootle server, which requires manual intervention during the build process
- to tester which is quite normal, since a tester normally get an install-set.

Developers

Developers construct the actual program, using dedicated development tools.

Developers will as part of the development process embed messages (errors, warnings …) in the source code and/or build UI. The embedded texts are defined to be in English but the source code are in different programming languages, making extraction a challenge.

Developers are fluent in their language (C++, java, python etc.) but for sure not in all the native languages supported by AOO therefore localization is needed.

Developers uses solely SVN as their repository.

Translators

Translators add texts in the local native language, relating (translating) to the original message. In a release there is a 1-n relation between the original message and the supported languages, where n is the number of supported languages.

Translators does in principle not need to have programming knowledge because in essence they are presented with a list of texts extracted from the source and delivers the translated text back.

Translators work solely with the pootle server which today has no direct connection to SVN but work in parallel with SVN and are updated manually with regular intervals.

Integrators

Integrators initiate and control the build process.

Integrators does in principle not need to have programming or translation knowledge, because they are basically doing administrative tasks.

Testers

Testers check the total system and do a quality assurance of the behaviour.

Testers need a deep knowledge of the behaviour of the system, but deep technical knowledge is not needed.

Today testing seems to be very limited and not formalized in respect of the l10n process.

System: SVN

The sub version server is the actual repository and idealy all systems should work directly on this server.

All source files, documents etc. are stored in SVN.

System: pootle server

The pootle server provides an environment for translators to work in.

Today the pootle server contains all the translations and are updated from SVN and are as a consequence not synchronized and without version control (during the translation process). Furthermore many translators work offline without any control.

L10n workflow high altitude view

The workflow seen from the outside is quite simple, but still some of the shortcommings should be very obvious.

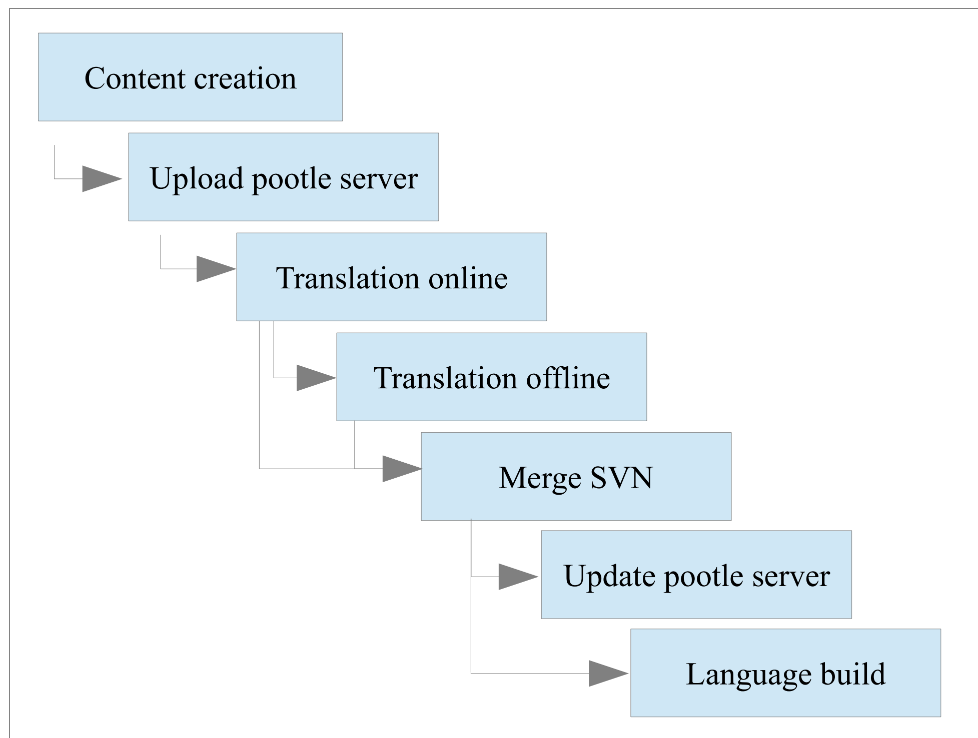
The workflow is designed as a waterfall, but one of the good norwegian ones where water is pumped back up at night time. Idealy for each release each section is done only once (waterfall), but in real life two things happen (norwegian night pumping):

- Some sections happens in parallel (e.g. Translators start working with early code)
- Some sections are repeated due to problems found in later sections

This is quite normal and normally not a real problem provided the process is automated and has a number of quality gates.

However the current process there is only a single automated quality gate which are pure technical (solving: “Can the product be built without errors?”) the rest is left to us humans.

The workflow only concentrates on the l10n process which is only a subset of the total lifecycle process.



The model shows at least one problem, the parallelism of “Translation online” and “Translation offline”. To put it a bit on edge, this works because there are no alternatives and because there are few volunteers.

Content Creation

Developers construct/develop new functionality or correct bugs/issues using different tools and programming languages. During the programming they may insert texts in the source files, this is done very differently depending on programming language and type of application (UI or error/information messages).

All text are written in English according to the programming guidelines, however there are no review process to secure the quality of the text or consistency with the rest of the product.

Note: A developer can insert the text directly in the source file or in a resource file, for the program both ways work, however only a limited number of file extension types are today scanned for texts, so in worst case some texts are never translated.

Upload pootle server

The source files are stored in SVN. In general the content of SVN is floating since it contains the absolute last updates, with the consequence that a total build very often will fail. To circumvent this problem a snapshot is made from time to time, guaranteeing a successful build but the package might not function correctly.

The snapshots can be used for a manually started extraction to the pootle server.

The extraction program loop over all files in SVN

- building one big sdf file
- the sdf file are then split into multiple template files
- the template files are merged with the existing po files in the pootle server
- pootle server database contain one set of po files for each language

The purpose is to decouple the development process from the translation process. The purpose is achieved, but the route is highly manual and error prone.

If life was ideal, translation would only take place when development is completed, but typically translation takes place at several stages of the development process for several reasons:

- A release consist of changes to multiple function group (e.g. draw, write and calc), and these developments are finished at different point in times. Whenever a development of a group is finished this group can be translated and thus the decoupling will be repeated.
- Translation often takes place while testing is ongoing, any bug fixing must lead to a new decoupling, and since there are no version control of the translated parts it can only be controlled manually if there are changes.
- There are currently no short-cuts to fast translate a bug fix that involves a known text change

Note: This part of the process is highly manual and very error prone, since it involves coordinating the effort of a high number of people

Translation

Translation takes place on an offline copy consisting of multiple po files. These po files are generated each time, so any additional information the translators would like to keep (e.g. comments) are lost.

At the moment there are 276 different files to translate for each language. In order to split the work UI and Help are separated, there are

- 20 help files (but they are big!)
- 256 UI/message files (typically an average of 20lines)

Having that many files to translate makes it more likely to get content inconsistency (same term is translated differently).

Since the files are solely generated from the sources, there are no glossary file available, making it very difficult for new volunteers to help. Furthermore there are no control of how accelerators are used.

The online and offline translation process are handled quite differently.

Note: Today there are no version control and as such no computer controlled review and as a consequence the content quality varies.

Translation online ("committer")

The po files are stored in pootle server database and thereby available to translators with through the HTML interface.

Due to the lack of version control, team work must be controlled carefully

Once a translation is complete, the translator(s) must manually inform the integrator that the set is ready for merge.

Translation offline (non "committer")

The integrator will manually extract the po files from the pootle server and send the files to the translators without "committer" status. The copy is not under version control or otherwise controlled.

Once the translation is complete the the translator must send the files back to the integrator.

There are no computer control with which translations are outstanding, which are in manual review and which are completed, this is currently controlled by the integrator.

Note: Neither bugzilla nor the mailing list allows these big attachments, so it must be sent to a private mail address or posted on a private web page.

Merge SVN

The integrator must manually decide that all offline translations are back and all online translators have finished (translation review is left to the single translator team).

At a point in time decided by the integrator to start the merge, which consist of several manual steps:

- synchronize po files with content of the pootle server database
- add the offline translated files
- convert po files to sdf file (one pr language)
- store sdf file in SVN.

This part of the process does not allow for glossary files, because the converters would have no source parts to relate the glossary to.

Update pootle server

Now it is time to synchronize the pootle server, to make sure then content is identical with SVN.

Based on the new sdf file (one pr language) the following actions are taken:

- Convert sdf til template file
- update templates in pootle server

Language build

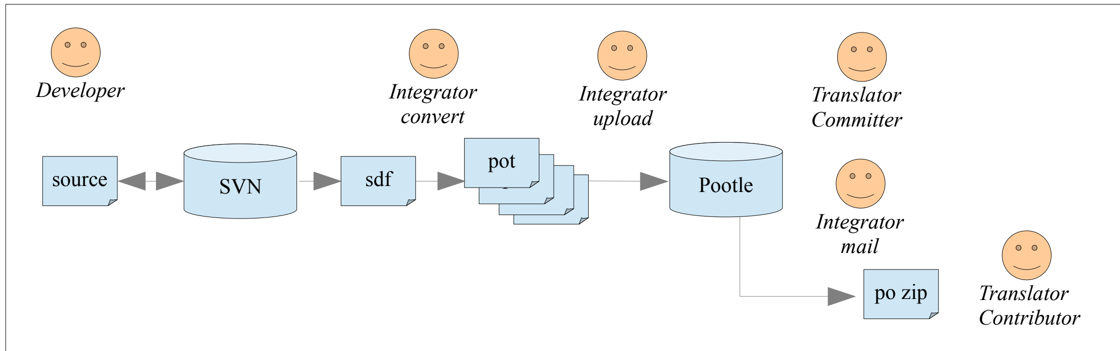
Finally a test release can be built, and the testers can control the final result.

It should be noted that there currently no formal testing of the native language versions.

Simplified data flow

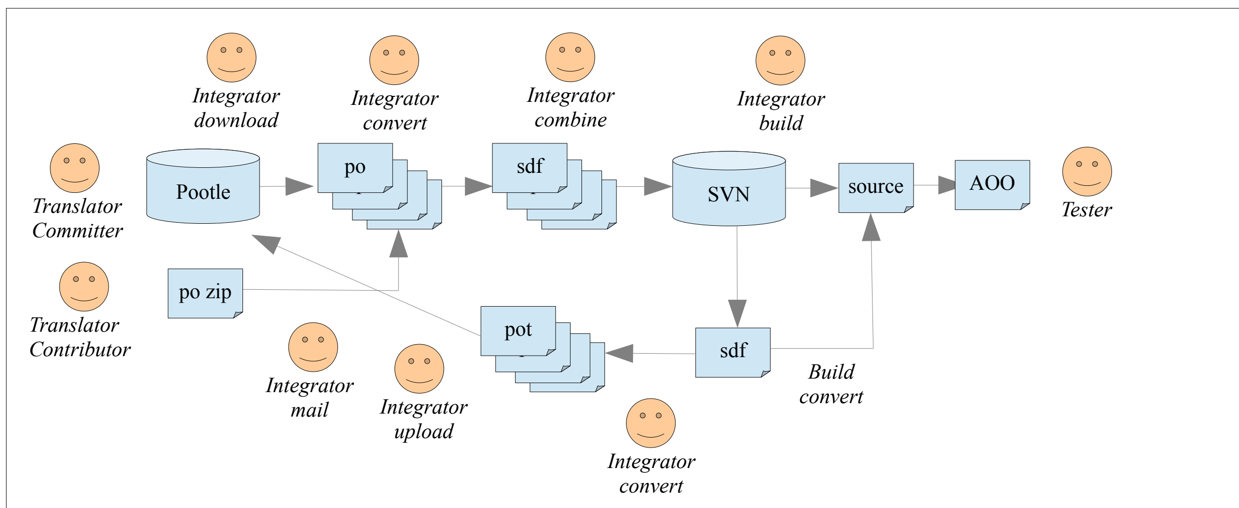
The current data flow is pretty complex, and it seems more like a “invented as needed” structure.

The first part shows the text flow from developer to translator:



The second part shows the text flow from translator to tester:

As seen from the diagrams there are many manual steps, and many different temporary files only needed to come from a to b.



L10n workflow technical view

This chapter is identical to L10n workflow high altitude view but seen from a technical view showing actual commands, names of files and directories as well details of the tool behaviour.

Content Creation

Developers write text that needs to be localized. In principle the texts can be kept in files with any extension since most compilers are quite large in that respect. However the programming guidelines should secure that only defined extensions are used.

It is worth to note that the most common files (.cxx, .hxx, .cpp, .hpp, .py) are NOT scanned.

Note: If a developer for some good reason decides to use a file with a non-standard suffix, it will NOT be searched for messages.

Upload pootle server

The upload process is the very complicated and totally manual.

The outcome of the process in general it makes a snapshot copy of the texts in SVN and makes it available on the pootle server and as zip files to contributor translators.

After the texts is extracted and until they are merged back they are NOT in any source control, nor is parallel development controlled.

Extraction from sources (generate new sdf file)

Before starting this process, all sources needs to be checked out (read-only). In order to ensure that the source is complete it is good practice to do a “build –all” first.

The process is started with:

```
cd main
localize -e -l en-US -f en-US.sdf
```

This is a perl script that will call

```
solver/350/<platform>/bin/localize_sl<.exe>
```

which is the actual executable. Sources for this executable is found in l10tools/source.

localize_sl loop across the entire tree looking for files with a known extension. As seen in the table below the number of relevant files are small compared to the total number of files.

Extensions scanned for text			
Files	**Extension**	**Tool**	**Desription**
814	.hrc	transex3	header for resource files
98	.properties	jpropex	java property files
1040	.src	transex3	source for resource files
15	.tree	xhtex	help files
0	.ulf	ulfex	?
53	.xcd	cfgex	xml files only in postprocess
314	.xcs	cfgex	xml file for java
1365	.xcu	cfex	xml files for UI
0	.xgf	xmlex	?
4543	.xhp	helpex	AOO help files
0	.xrb	xmlex	?
1	.xrm	xrmex	xml readme file
0	.xtx	xtxex	?
0	.xxl	xmlex	?
8243	Files to be scanned, total number of files is 438189		

The tools are all separate executables meaning that for each file to be scanned a separate process with the corresponding tool is started, especially in MS-Windows this leads to prolonged duration.

The results of the single scans is contained in a single sdf file, which are then passed to the next phase.

The resulting sdf file is generated in directory containing main (normally trunk).

The resulting foo.sdf.main has at the moment:

- 12994113 bytes
- 72492 lines
- 45341 lines of the 72492 originate from the helpcontent2 module
- 27151 lines of the 72492 originate from UI and simple messages

At the moment localize runs with errors on Windows: jpropex, a shell script that calls a java program does not run. Linux is OK.

Note: On Linux or MacOS you have to use a full qualified path to the output file. Otherwise you won't get an output file and also no error. The tooling seems to be very error-prone.

Merge with pootle server database

The sdf file created by localize is transformed/converted into template pot files using

```
oo2po -P en-US.sdf templates
```

This set of pot files in the directory **templates** should now be updated on the pootle server. Copy the complete *templates* directory in the *po* directory of the pootle server in the related project directory.
Assuming our project id is *aoo34* and the pootle server is under */var/www/Pootle*:

All help files are located in a single module so it easy to distinguish between UI and help. First move the help files (in order not to copy them into the UI directory):

```
cp -r templates/helpcontent2 \
      /var/www/Pootle/po/aoo34help/templates/helpcontent2
rm -rf helpcontent2
```

Then copy the UI files:

```
cp -r templates /var/www/Pootle/po/aoo34/templates
```

Update all existing languages to be aligned with the new templates.

```
cd /var/www/Pootle
./manage.py update_from_templates –project=aoo34
./manage.py update_from_templates –project=aoo34help
```

or

```
./manage.py update_from_templates --project=aoo34 –language=de
./manage.py update_from_templates --project=aoo34help –language=de
```

to update a specific language.

Probably it is also possible to specify both projects with *--project=aoo34, aoo34help* and a list of languages with *--language=de,fr,es,...* (not tested it yet)

Translation

Translation takes place, either directly via the pootle server's html frontend or via an offline editor.

Translation online ("committer")

Translators with status as "committer" can work directly on the pootle server.

However they have no glossary available, so it is highly possible that the same term is translated differently in different modules and it happens for sure over time as different people work on the translation.

The changes are done directly in the po files, there are NO version control, and NO review control.

The separation of help content from UI content has many advantages but one huge disadvantage, there are no control that e.g. menu names are identical in help as in the UI.

Translation offline (non "committer")

Many translators do not have "committer" status and can therefore not use the online pootle server.

The normal procedure is that a "committer" generates a zip file with all the files, mails the location to the translator.

The translator uses an offline tool like poedit.

However they have no glossary available, so it is highly possible that the same term is translated differently in different modules and it happens for sure over time as different people work on the translation.

Once the translation is complete the translator send the files back to the "committer" that updates the po files behind the back of the pootle server.

There are no special quality checks in place to secure that the content of the translation are consistent with earlier translations.

If you update *po* files for an existing language (translated external) you should update the stores with (after having copied to po files)

```
./manage.py update_translation_projects --project=aoo34,aoo34help
./manage.py update_stores --project=aoo34,aoo34help –language=de
```

Merge SVN

Once the integrator decides that all parts are translated and quality controlled it is time to get the texts back into SVN.

First step is to resync the database into the **po** files because otherwise the made changes are only in the database. For example sync the UI strings for *de* back into the *po* files.

```
./manage.py sync_stores --project=aoo34 -language=<lang>
./manage.py sync_stores --project=aoo34help -language=<lang>
```

The next step is create a new **sdf** based on this updated **po** files.

```
po2oo -l de -t en-US.sdf --keeptimestamp --skipsource \
       <lang> new_<lang>.sdf
cp new_<lang>.sdf extras/l18n/source/<lang>/localize.sdf
```

This command used the template en-US.sdf and created a new **sdf** file containing the new *de* translations. If you skip the parameter *skipsource* the *en-US* source translations are also included in the *sdf* file. Can be useful for some verification.

There is a utility gsicheck to check the files syntactically, this is however currently not in use.

Note: this step has to be repeated for each language.

Update pootle server

In order to update the pootle server with the newest templates, we repeat earlier steps:

```
oo2po -P en-US.sdf templates
```

Let assume we are currently in some temp directory and have existing po files in **aoo34/es/...** and have new templates in **aoo34/templates/...** then we can create a new set of po files with

```
pot2po -t aoo34/es aoo34/templates es_new
```

This command will merge the existing translations found in **aoo34/es** and merge them with the new templates and stores the new po files in *es_new*. This new po files can be copied in the Pootle project directory **<pootle_install_dir>/po/aoo34/es**. The database have to be synchronized with the new po files.

```
./manage.py update_translation_projects --project=aoo34,aoo34help
./manage.py update_stores --project=aoo34,aoo34help -language=de
```

Language build

Use the normal command:

```
build –with-lang="..."
```

When the office is built with configure switch --with-lang="..." then extras/l10n is built and the localize.sdf files are rearranged. In l10n they are grouped according to language. Now they are grouped according to module (and directory.) The sdf files in extras/l10/<platform>/misc/sdf are zipped into one archive per module and delivered into main/solver/340/<platform>/sdf/<module>.zip and then forgotten (at least for the processing of src files.)

Resource files (src files) are processed when the other modules are built. The original src files contain strings only for en_US in lines that look like

```
Text [en_US] = "...";
```

transex3 adds the missing languages by adding lines like

```
Text [de] = "...";
```

By default all (available) languages are added not just the ones given to configure's --with-lang switch. The augmented src files are placed in <module>/<platform>/misc/... These are then aggregated into some srs files in <module>/<platform>/srs/. In a (or several) following step(s) the srs files are aggregated into res files, one for each language.

The resulting res files are delivered to main/solver and become part of the installation sets. Multi-language versions contain res files for more than one language.

At runtime the ResMgr class from the tools module is responsible to use the resource files of the currently selected language whenever a string is requested (as is the case for e.g. all button texts and in general for all text visible in the GUI.)

File Formats

Quite a number of different file formats are involved in the localization process. The following list is not complete and may be inaccurate:

Extension	Desription
.hrc	header for resource files
.properties	java property files
.po	contains the translated strings from a .pot file. Used on the pootle server.
.pot	created by gettext from source files. Contains strings that need translation. Not used by OpenOffice except as part of the pootle server update.
.res	created by transex3 from .srs files.
.sdf	used to store localized/localizable strings and their origins. Comparable to .po files.
.src	source for resource files Most strings used in the GUI are defined in .src files.
.srs	Made by rsc (which calls rscpp and rsc2) from multiple src files with *all* language strings included.
.tree	help files
.ulf	?
.xcd	xml files only in postprocess
.xcs	xml file for java
.xcu	xml files for UI
.xgf	?
.xhp	AOO help files
.xliff	a format with the same usage of .po, but it has more functionalities and is standardized.
.xrb	?
.xrm	xml readme file
.xtx	?
.xxl	?

Tools

A large number of tools, implemented in a variety of languages (C++, Java, Perl, Python, sh) are involved in the localization process. They mostly extract strings from source files and merge the translated strings back in, or transform between different data formats.

The following list is not (yet) complete and may (still) be inaccurate:

tool	description
build	Standard build tool
cfgex	Called from localize_sl to translate .xcd .xcs .xcu files
gsicheck	Tool to do a syntax check on sdf files
helpex	Called from localize_sl to translate .xhp files
localize	Perl script to control localize_sl
localize_sl	Program that scan all sources for text strings
manage.py	A Python script to manage the pootle server database
oo2po	Standard program used to convert sdf files to po files
po2oo	Standard program used to convert po files to sdf files
rsc	Resource compiler
rscpp	Resource compiler
Rsc2	Resource converter
jpropex	Called from localize_sl to translate .properties files
ulfex	Called from localize_sl to extract strings from .ulf files. NOT USED
xhtex	Called from localize_sl to translate .htex files
xmlex	Called from localize to extract strings from .xrb .xxl .xgf files. NOT USED
xrmex	Called from localize_sl to translate .xrm files

Open issues

The current localization workflow as outlined above has several drawbacks and plenty of room for improvement.

The drawbacks as well as other ideas to make the l10n process robust and stable have been collected below. These issues should be discussed either through the wiki or through the mailing list.

When there is a proposed solution to all issues, that the community in general agree to, this document will be converted into the proposed structure with a list of to-dos.

The list of issues is not prioritized.

Workflow is not a designed approach

The current workflow is probably created as needed and as a consequence it has big portions of "left-over" from

- the original openOffice (not localized)
- the SUN era
- the ongoing integration of openOffice in the Apache environment
- the l10n process is merely a "must" and not as interesting to work on as other parts
- The localization workflow is convoluted and hard to understand
- Much tooling is involved outside the build process.
- Some of this tooling seems to be lost after a disk crash of the old OpenOffice pootle server

This results in a manual process that is undocumented and known only to a select few.

Proposal

Once we agree on all issues a design paper on a proposed structure will be make available and be basis for discussion.

Tools are writing in multiple languages

The tools involved are written in a variety of languages: C++, Java, Perl, and Python. This is not bad in itself. For example it makes sense to parse Java property files with Java code. But there is also C++ code for iterating over the tree of source files that uses hard coded lists of other executables and scripts for processing individual files. That leads to many processes to be created and destroyed, something that is notoriously slow on Windows.

Some of the tools are not used anymore. For example there are no .xtx, .xrb, .xxl, .xgf, or .xcd files. Therefore the xbtxex and xmlex tools can be dropped. (May have already happened for xmlex) Others are used but do not run (like the jpropex tool). And then there is our own preprocessor for

handling resource files, which might be replaceable by the standard C/C++ preprocessor (which parses the included hrc files anyway since they are included in C++ code.)

On Linux or MacOS you have to use a full qualified path to the output file. Otherwise you won't get an output file and also no error. The tooling seems to be very error-prone. A lot of space for improvements.

At the moment localize runs with errors on Windows: jpropex, a shell script that calls a java program does not run. Linux is OK.

Streamline the number and implementation of the tools used for extraction and merging of localizable strings. Use the right language for each task.

Proposal

Rewrite localize_sl, include the conversion programs (more efficiently).

Use gcc preprocessor instead of our own.

Use of sdf file

AOO uses its own non-standard file format (SDF) for handling localized strings. In order to use a pootle server for the actual translation, all .sdf files have to be transformed into .po files and, after translation, back into .sdf files. It should be also taken into consideration a future migration to xliff format for translation handout.

Proposal

The .sdf files are merely intermediary files between the source files and the po files, and should be eliminated.

The choice of .po or .xliff is not so easy:

1) source <-> .po and .pot files

> The advantage of this approach is that all translators knows .po
>
> The very big disadvantage is that the format has no standard way of storing extra information. We need to store the relative path of the originating source file (as in .sdf) in order to be able to split the information.

2) Source ↔ .xliff

> The advantage of this approach is that we can store extra information as needed, furthermore there are xliff editors out there including pootle server. It would also eliminate the need for template files.
>
> The disadvantage is that it is a new format, and offline translators would need to change editor.

Personally I would prefer .xliff since it makes programming a lot easier, but I think we need to listen carefully to the translators.

Separate projects for UI and help

We should create 2 separate projects: one for **UI** and one for **Help**. And we should keep it separated between versions because there will be probably some overlap with potential conflicts. Maybe an approach of keeping two versions in pootle to give translators the chance to work on translation after a release. And to allow future development toward the next release in parallel.

For example something like:

Apache OpenOffice 3.4 UI (aoo34)
Apache OpenOffice 3.4 Help (aoo34help)
Apache OpenOffice 4.0 UI (aoo40)
Apache OpenOffice 4.0 UI (aoo40help)

note: there are already 2 projects (a0034 and a0034help)

At the moment there are 276 different files to translate. Having that many files to translate makes it more likely that the same term is translated differently and currently there are no glossary list available.

Proposal

The process makes 2 files (.xliff or .po) for each language:

- localize_ui.<xx>
- localize_help.<xx>
- glossary.<xx> this file is not generated but maintained by the translators

These 3 files are delivered to the pootle server, translated and sent back for storage in SVN.

These files are handled as other files in respect to versions and releases.

Build process is highly manual and error prone

Total workflow should be automated.

A developer can insert the text directly in the source file or in a resource file, for the program both ways work, however only a limited number of file extension types are today scanned for texts, so in worst case some texts are never translated.

Integrate the string extraction into the build process. Most of the files that can contain localizable strings are already part of the build system, mostly for the merge process. For example there are make-rules for transforming and merging rsc files into .srs and then into .res files. Add rules for the string extraction. This would allow developers to count new strings and the buildbot could extract the new strings and upload them to the pootle server.

Proposal

Add a new target in the makefiles (l10n_gen). Developers can then assign which files belong to this target.

Localize_sl should be rewritten so it can run in multiple makefiles (no directory scanning). Localize_sl will generate a snippet file that will be stored in a staging area (l10n/stating) and as last step in the "build –all" process, l10n will be "built", that is the snippets will be used to update the single language files. With this process the language files will always be "ready" for use in the build process.

However the pootle server still need to be manually updated.

Automatic update of pootle server

Translators need versioning possibilities

Offline translation needs to be controlled (delivery etc).

At the moment there are no computerized control over when a translation is ready for merge, nor can a translation be given a status like e.g. "ready for review".

pootle server can use SVN directly, and thereby offer version control, however at the moment this is not used.

Proposal

Make a new subproject in main called l10n, this project contains the language files (basically extras today), but also .mk file for generation.

The pootle server works direct on SVN. With this philosophy translators are seen as just another breed of developer (bot work with languages) and we have all the advantages of a version system when working on larger translations.

Content control

PO->SDF There are currently no control of the content quality (it is possible to make a translation, where all translated text are "not-translated" and it will pass.

PO->SDF There are no check, that changed text are changed in the translation.

Proposal

Write a new tool that controls the tranlated part (based on the idea from poConsistency) and integrate in the "build –all" process.