Localization of AOO proposal for new workflow

21 October 2012

Contents

Introduction

This document is a proposal for a new l10n workflow, with the open issues from the previous document and the comment on the mailing integrated.

The document describes an end state, of course it will be developed and committed in phases.

Quite some work has gone into designing the workflow with a few manual step as possible, at the same time it has not been “wish what you want”, because it has been high priority to only change parts that really needed changing.

There have been a discussion on file formats (.po versus .xliff), in order not to confuse the themes this document is written in a way that both file formats can be used, only the last chapter discusses advantages/disadvantages of the 2 formats.

Once the community have decided (and changed) the document, I will undertake the development, and hope a volunteer committer will help with review, guidance and commit in SVN. At the end of the document I have included a proposal for a project plan.

In order not to “if” statements all over the document, it is written as if it was decided and implemented, please do not get offended by this it is simply easier, and I am very well aware that there is a lengthy decision process after the release of the document.

Thanks to all those persons who contributed to enable me to write 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.

L10n is in more popular terms called “Localization of AOO”, or in very simple terms just “translation of AOO”. L10N defines the workflow which makes AOO available in local languages.

Localization is from the perspective of the involved people a multi-step process that involves a variety of tools and procedures. There different main types of people involved have quite different and to some extent conflicting views and requirements, therefore the workflow is a real “best of all worlds” approach.

It has been an objective to make as much as possible of the workflow automated, but there still remain a few manual steps.

The process is in short:

- Developers add messages to source files, which are automatically extracted,
- Language files are manually committed in SVN, and available to translators,
- Translation work, changes are stored in SVN,
- When building a language specific AOO it automatically uses the SVN content.

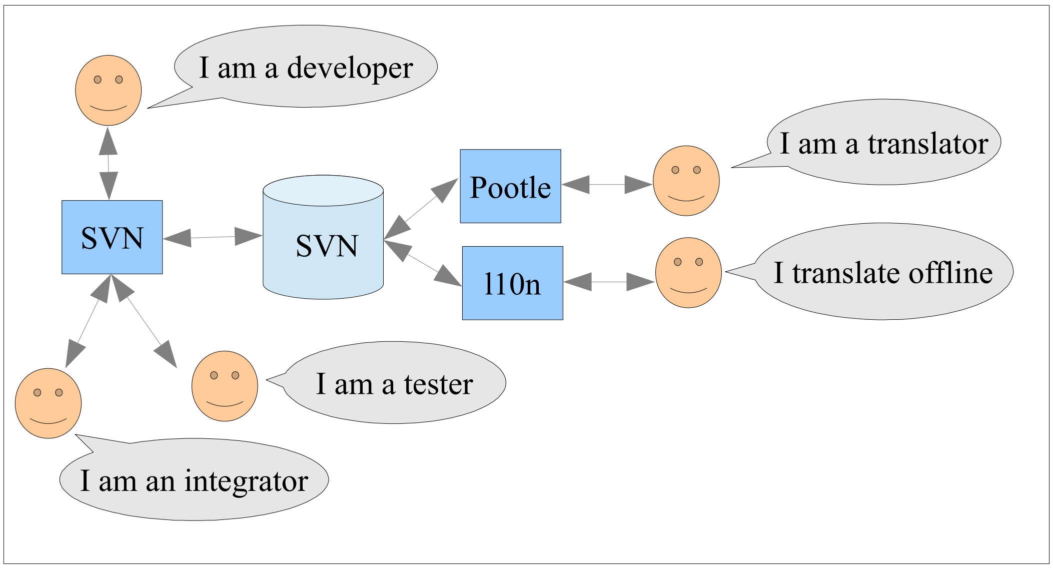
If you are looking for information about how to contribute translations then this page gives an overview.

The document has 8 parts:

- role definition,
- non-technical workflow overview,
- simplified data flow,
- detailed technical walk through of workflow,
- File formats used,
- Tools used,
- Temporary: Discussion on .po versus .xliff
- Temporary: Project plan

Actors and Systems

The l10n process can and should be viewed with respect to 4 different categories of people who access the process through different tools, but one common repository.



The picture illustrates the different type of people involved in the workflow, the process itself is explained later.

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

Developers

Developers construct the actual program with different languages (C++, Java, Python...) and will as part of the development embed english messages (errors, warnings …) in the source code and/or build UI containing english text. 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.

The development review process, does (as far as I know) has no check of the text (spelling, wording, consistency), that might be something to consider.

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.

It is not a requirement for the translators to know the different programming languages, because the texts are automatically made available in form of content files with the principal layout::

<original english text> = <native language text>

translator using pootle

Translators who have committer status can work with the pootle server (which uses SVN directly).

Translator using l10n

Translators who have contributor status can through the l10n.openoffice.org:

- download a single language file or all files for a language
- upload a single language file or all files for a language (requires registration)

Translators can concentrate on translating the system, without having knowledge of SVN or the workflow as such.

This procedure has extended security features, see tools chapter.

Integrators

Integrators have mainly administrative task, such as:

- committing the language files in SVN if not done by the developers,
- control the automatic nightly 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 all systems work directly on this server.

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

System: pootle server

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

The pootle server uses SVN to store the language files, so a translation is immediately available for build.

System: l10n.openoffice.org

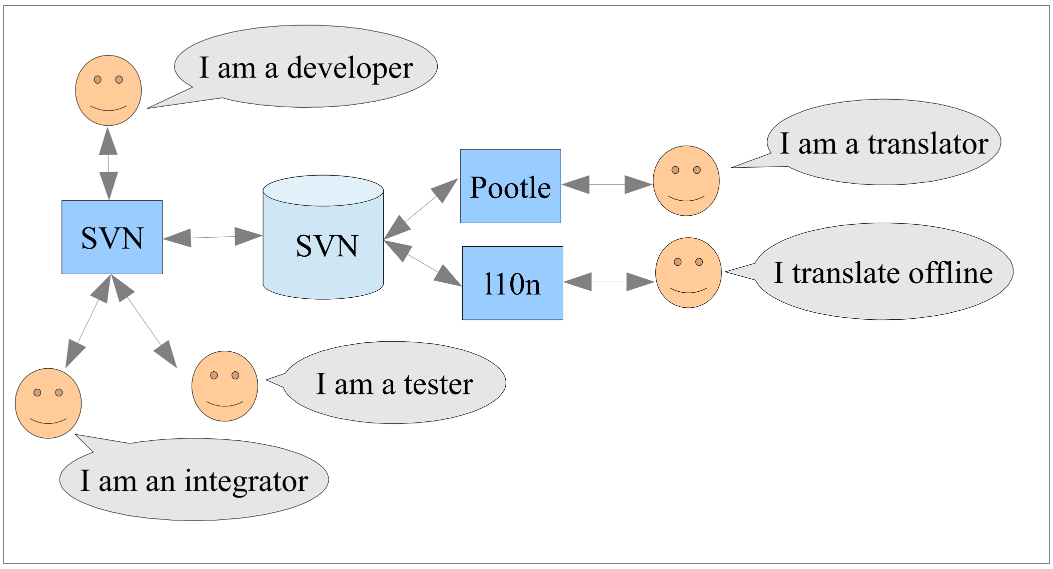
The l10n web page, offer the offline translator to download/upload files. Everybody can download files, but the upload has added security measures.

The server part of the l10n web page updates SVN.

L10n workflow high altitude view

The workflow is designed for a high degree of parallel work fully automated, basically it handles developers, translators and testers as equals.

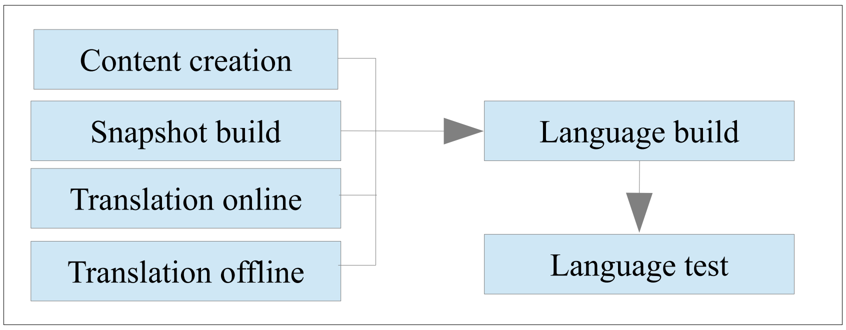
The workflow allows translators to start early on translations and thereby providing language testers with language versions parallel with the English version. In theory it should reduce the time needed between end of development and release date, and in praxis it allows early testing of language versions meaning a more stable and complete release.
Lets look at the components (using the picture from before)



The central and only repository is SVN. The SVN server is accessed by 3 programs: SVN, Pootle and l10n. It is important to note that none of these programs have local storage (except for caching etc.), so there are NO copying of data!

The common repository is the key point to a parallel workflow.

The workflow consist of the following "steps":



The 4 phases "Content creation", "Snapshot build", "Translation online" and "Translation offline" changes the language files and make the immediately available for a language build.

Of course a text cannot be translated before it is created, so there are a natural flow from top to button BUT "snapshot build", "translation online", "translation offline" and further "content creation" happens in true parallel.

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).

The texts are automatically extracted and sent to a language staging area as part of the normal build process of a local directory.

When developers build the complete AOO, all language files in all languages are automatically updated and ready for use.

When developers build the complete AOO with a language option, the language files in question are automatically split into the source files.

Sharing the language files poses no real problem, because by nature the usage is divided:

- Developers are the only ones who add lines (new text) to the language files
- If a developers remove a text, the line is marked in the language file, and the translator can remove the line.
- If a developer changes a text, it is considered a new text, and the translators must copy the translation from the old line (unused) to the new line.

Each main directories correspond to exactly one language file, so it is easy for a developer to commit the language files, in order to make them available for translation. If the developer does not commit language files it will happen as part of a snapshot build.

Snapshot build

At regular intervals (more frequently as the release date comes closer) a snapshot build is made. A snapshot build is a normal build, but the file versions are registered so it is possible to do a rebuild without using any newer files.

After the normal build, any non-committed language files will be automatically committed, in order to guarantee a consistent snapshot also in respect of languages.

With this method a snapshot build is just as complete as a release, containing both source code and language files.

Translation

There are (based on 1 file for each subdirectory):

- 8 help files,
- 47 UI/message files (this might be reduced by combining some directories)
- 1 glossary file

to be translated in 112 languages, given a total of 6.272 files to be translated. Alone the number signals the amount of effort required and the need for an efficient process.

The 8 help files, correspond to the product parts (writer, calc...) and the 47 UI files correspond to the directories in main (source tree). The tight relationship directory – language file, makes it easy for developer and translator to work together.

More importantly, when a developer make a change it is immediately available to the translator (provided the developer also commit the language files), this allows very fast turn around times in cases like bug fixes.

Additional there is one glossary file available for each language, which should be used for generic terms (e.g. Cancel) in order to secure a consistent translation. If the glossary is used the translation will automatically be controlled against the glossary during the next build.

Since the language files are stored in SVN, the translator can just as the developer backtrack changes.

Translation using pootle

Translators with committer status can work directly on the pottle server using their normal login.

Translation using l10n

The bulk of translators do not have committer status, and many are primarily interested in translation and local themes. Many want to have easy access to translation without having to concern themselves with SVN and other technical details.

L10n can be seen as a light weight passage to get the work done.

The offline translator must register a valid e-mail on l10n in order to update language files.

Once registered the offline translator can download the newest files for a language, translate these using external tools and then upload them to l10n. L10n takes care of collecting the files from SVN for download and commiting the files that have been uploaded.

L10n has no problem with the size of a zip archive.

Furthermore L10n offers a language status page, showing the status (in terms of missing lines) of each language or each file.

Language build

Developers and integrators can any time do a language build with the newest translations, there are no manual steps needed.

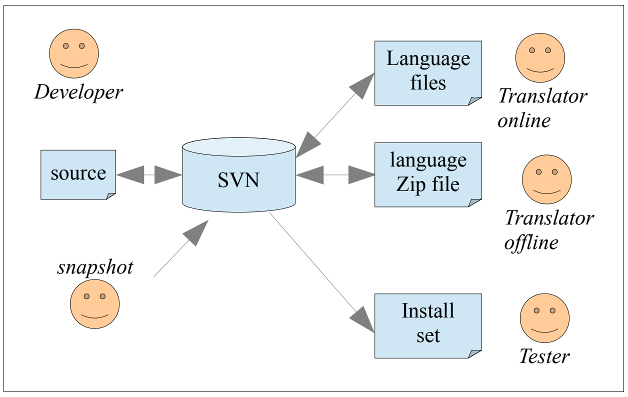
During the build the language files will be split into the parts needed by the source files.

Language test

When a language version is built, the UI test tools will also be localized and as a consequence a test suite can be executed in the local language.

Simplified data flow

As seen from the diagrams the flow is very natural and without manual steps:



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.

The extraction tool handles the following extensions:

Extensions scanned for text

Files	Extension	Desription
814	.hrc	header for resource files
98	.properties	java property files
1.040	.src	source for resource files
15	.tree	help files
53	.xcd	xml files only in postprocess
314	.xcs	xml file for java
1.365	.xcu	xml files for UI
4.543	.xhp	AOO help files
1	.xrm	xml readme file
8.243	out of 438.189	

The makefile/build contains a target **genLang**, and these extensions are globally associated with the target just like .cpp is associated with the C++ compiler. However the developer can in the local makefile overwrite the default. This makes sense in directories that contain files with text that should not be translated.

Build target **genLang** is served by the tool **generateLanguage** (source is located in main/l10ntools). **generateLanguage** extract the messages directly, only in case of Java a subprocess is started. With the standard association the tool will only run once, and not for each file, by overwriting the default it is up to the developer if the tool should run for each file or once (the tools handles multiple filenames as parameter).

If the developer decided to use other extensions, **generateLanguage** has a parameter –**useExt** that will overwrite the file extension.

Assuming the developer is in a local directory (e.g. main/l10ntools) and calls:

```
build
```

the code will be built and 1 language file will be created. The language file will be placed in a

staging area (extras/l10n/staging), think of it as an object file ready for linking.
Extraction of languages without building is done with the command:

```
cd extras/l10n
build
```

In order to generate native language files use:
In order to build a complete AOO, following command is used

```
build --all
```

This command will update all native language files.

Note: When doing a commit, please do not forget to do in extras/l10n as well, in order to make the native language files available for the translators.

Snapshot build

A snapshot build is very much like a release build, it creates an install-set.

The process of a snapshot (automated in a script) is:

```
build --all
```

Then commit any changed native language files, and then

```
build –all --with-lang
```

The snapshot build is important for both translator and tester, since it is a checkpoint where code and native language files are guaranteed to fit each other.

In order to get an assessment of the outstanding translation, run:

```
cd extras/l10n
checkLanguage
```

checkLangauge goes through all native language files, and collects (for each file and each language) the number of missing translations and the number of undetermined translations.

In order to check consistency, run:

```
cd extras/l10n
checkLanguage --consistency
```

This will check the translated files for correct glossary usage.

Translation

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

In order to help the translator and secure consistency in the product a glossary file is provided. The glossary is empty unless the local translators fill it !! The recommendation is to fill it with all terms used more than once.

Many tools can load the glossary into a translation memory and thereby automate part of the translation.

But more importantly, during the build process the translations will be checked against the glossary and inconsistencies will be logged.

Translation online

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

They need to do a commit in SVN after final editing, in order to make the translation available for language builds.

Translation offline

Many translators do not have “committer” status and uses an alternate route using the l10n.openoffice.org

Registration

In order to use this service they must first register on l10n/register with a valid mail address. Once registered they will receive a confirmation mail, which they must return.

The confirmation mail is sent to ooo-l10n@apache.org, where a person with committer status will control it manually, if ok the l10nSecurity file will be updated, and the translator can start working.

Download

The translator downloads the files from l10n/download. It is possible to either download single files or all files for a language. As part of the download a marker is set in l10nOffline file with the e-mail of the translator.

The marker allows tracking of who is active translating.

Upload

Once the translation is finished or when a part is finished the translator uploads the files using l10n/upload.

The files are controlled for:

- Syntax validity
- Consistency against the glossary
- Download check, match of e-mail and download is max. 4 days old

If correct, the files are committed in SVN (done internally on the server side, using **l10nUpdateSVN** script).

In case the files have changed in SVN during the translation and SVN cannot merge the changes automatically, the translator is requested to download a new version of that file and manually insert the translation.

When all files are updated in SVN, the translator received a confirmation, and the translation is immediately ready for language build.

Status

The translator can any time see the outstanding translations on l10n/status. The page provided a list of all languages, coloured with green meaning all translations are done. Each language can be opened to a page where the single files can be viewed. Using the page eliminates the need to walk through all files downloaded in search of something to translate.

Language build

To generate an install-set forone or more specific languages, use:

```
build –all –with-lang="..."
```

To generate an install-set for all languages, use:

```
build –all –with-lang
```

The “--all” will make a build in extras/l10n which will first update all language files based on the extractions (in staging area) from the source and then secondly process all resource 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] = "...";
```

generateLangauge 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.)

Language test

Currently there are no formal testing of language version. It is however a future plan to have the UI tools updated with language files for automated testing.

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
.xcd	xml files only in postprocess
.xcs	xml file for java
.xcu	xml files for UI
.xhp	AOO help files
.xliff	a format with the same usage of .po, but it has more functionalities and is standardized.
.xrm	xml readme file

Tools

A small set of tools 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 show the tools used in the workflow:

tool	description
build	Standard build tool
checkLanguage	Used to check syntax and consistency in language files
generateLanguage	Integrated in build to extract text and generate resource files
jpropex	Called from checkLanguage to translate .properties files
l10n/download	Web page and php script to download language files from SVN
l10n/status	Web page and php script to show status of translation process
l10n/upload	Web page and php script to upload and commit language files in SVN
l10nUpdateSVN	Script to commit language files in SVN

L10nUpdateSVN

Using a script to update/commit in SVN is of course something that needs to be done with care. HOWEVER the script is limited to native language files, which can be regenerated.

Behind the scenes the script is either using the user of a committer or an extra user “l10n”.

Temporary: Discussion on .po versus .xliff

There seems to have been big discussions on file format, the above described workflow DOES NOT depend on the file format.

However there are advantages and short commingles of both formats:

.po

This is a very simple format, even though it is not recommend to edit with vi/notepad. It is widely used.

Advantages

- We have it today, and the offline translators have tools like poEdit installed

Disadvantages

- It provided no standard facility to store originating file names, this must be done in a comment.
- It provided no facility for status of the translation (like: not-translated, not-reviewed...)

.xliff

This is a simple XML format, and can be edited with a standard xml editor even though it is not recommended. It is used more or less solely for translation.

Advantages

- It can store originating file names as an XML tag.
- It has a status tag of the translation (like: not-translated, not-reviewed...)
- It is supported by pootle server

Disadvantages

- Offline translators need to get used to a new tool

Temporary: Project plan

After we have discussed this document in detail, it is time to get it done. Assuming there are no changes (which does not seem likely), I will propose the following milestones

- Make a branch in SVN based on a stable snapshot
- develop **generateLangauge**
- update build/makefiles
- Test build process against old process (.po files should be identical in content)
- Merge branch back to main branch.

This is the part needed for the developers, at this point there are no change for translators (but the pootle server update is easier).

Parallel with the first part:

- update l10n.openoffice.org, by moving all content to an archive folder and build it up with all valid content. Use mainly pointer to information stored elsewhere, do not copy.
- Make l10n/download l10n/register l10n/upload l10n/status, but do not activate.

And as the last part:

- change pootle server to use SVN
- active l10n pages