FieldWorks 7 Localization

Ken Zook
August 13, 2011

Contents

1 Introduction

Provision has been made for localizing most strings in the user interface for FieldWorks 7 Language Explorer and Translation Editor. It is also possible to add localizations to any of the possibility lists.

Note: Before starting to do any localization, contact flex_localization@sil.org.

Here are some limitations with the current strategy.

- The size of dialog boxes do not change, so longer strings may cause problems.
- Older shared dialogs written in C++ cannot be localized.
- You can only localize to languages supported by the operating system.
- Each string has a single translation, so it can't be translated differently in different contexts.
- We can't localize icons and other graphics.

2 Localizing the User Interface

2.1 Overview

The localization process involves coordination between the development team and localization translators, typically located in other countries. The major disadvantage with

this approach is the translator cannot see the results of their work until they receive a new installer or set of files.

This summarizes the localization process.

1. A developer produces a POT translation file based on the current FieldWorks source files. This file is sent to the translator.
2. The poEdit program is used to convert this into a PO file, or update an existing PO file for a specific language.
3. The translator uses poEdit to add translations to this PO file.
4. The PO file is sent to a developer.
5. The PO file is merged with any existing PO file for that language.
6. The nightly build process uses the PO file to generate a localized set of DLLs and a localized strings XML file.
7. The installer includes these localized files
8. Anyone installing FieldWorks via the installer will then be able to include localized versions, if desired.

Flex and TE currently have independent UI languages.

To change the UI language in Flex, use Tools…Options, in the Interface tab choose the User interface language and the UI changes immediately.

To change the UI language in TE, use Tools…Options, pick the General tab and choose the User interface language, then restart the program.

2.2 What does a developer do?

2.2.1 Creating a POT Translation file

At any time a developer can generate a POT file to send to a localization translator. This process extracts all translatable strings from resx files and xml files under Language Explorer\Configuration. The command to generate a POT file from the fw\bin directory is

```
LocaleStrings -x -r c:\fw     (giving the path for your branch)
```

This assumes you want to extract data for all programs and that your RootCodeDir string variable in HKLM\Software\SIL\FieldWorks is set to the current fw directory. (If RootCodeDir is not set, it will use the fwroot environment variable if it is set.) If not, then you'll need to use other command line options (e.g., -r, --no-flex, --no-te, --pot). The default output file will be FieldWorks.pot in the fw\bin directory. Although the -r argument is not required, it is helpful to use if you have various branches on your machine to make sure you get the one you want. If you simply type LocaleStrings without any arguments, it will give information on the available arguments.

2.2.2 Integrating translations into the build process

When a translator returns a PO file for a given locale, it needs to be added into our Perforce repository. If this is the first Messages.xx.po file for a given locale (xx represents the locale identifier—see list at http://msdn2.microsoft.com/en-us/library/system.globalization.cultureinfo.aspx), it can simply be checked into the fw\Localizations directory. If there is already a Messages.xx.po file for this locale, you'll

need to merge the changes from the new PO file to the existing file. To do this, use the following command.

```
LocaleStrings -m oldfile.xx.po newfile.xx.po
```

This process will back up the original file (e.g., messages.fr.po-20070524120146) and then merge the information into the original file. It also generates a log file (e.g., Merge-20070524120146.log) listing any conflicts. The log file contains a summary of the changes at the end of the file.

To manually build the localized files, use the following Nant target

```
nant [release] localize
```

If you want to build release versions, be sure to include the 'release' nant target before the 'localize' target. This process creates individual output\xx directories containing translatable resource files. These are then compiled into localized dlls in output\debug\xx. It also creates a strings-xx.xml file in DistFiles\Language Explorer\Configuration for localizing the Flex XML strings.

The overnight build process creates localized version options in the installer. On an end-user machine the output\release\xx directory is stored as a FieldWorks 7\xx directory, and strings-xx.sml is stored in FieldWorks 7\Language Explorer\Configuration.

2.2.3 Upgrading a user .po file to a new version

When a translator returns a PO file for an older version and you need to build dlls for a new version and want to incorporate the new strings in the po file, use the following steps.

1. Use 2.2.1 to create a new pot file for the new version.
2. Open the user's po file with poEdit and use Catalog…Update from POT file to add in the newly created POT file.
3. Edit the updated .po file changing the Project-Id-Version to the new FieldWorks version.
4. Follow 2.2.2 to generate dlls to return to the user.
5. You can either return the newly created po file to the user (if he hasn't done any work in the meantime) or send the pot file and let him merge the new file with his.

2.3 What does a translator do?

PoEdit is a free translation program you can download from the Internet and use to translate source language strings into target language strings. It provides many useful features for doing this translation including translation memory.

2.3.1 Initial setup

To prepare for translation, you need to install the poEdit program and create a catalog for the target language. These are the steps to do this.

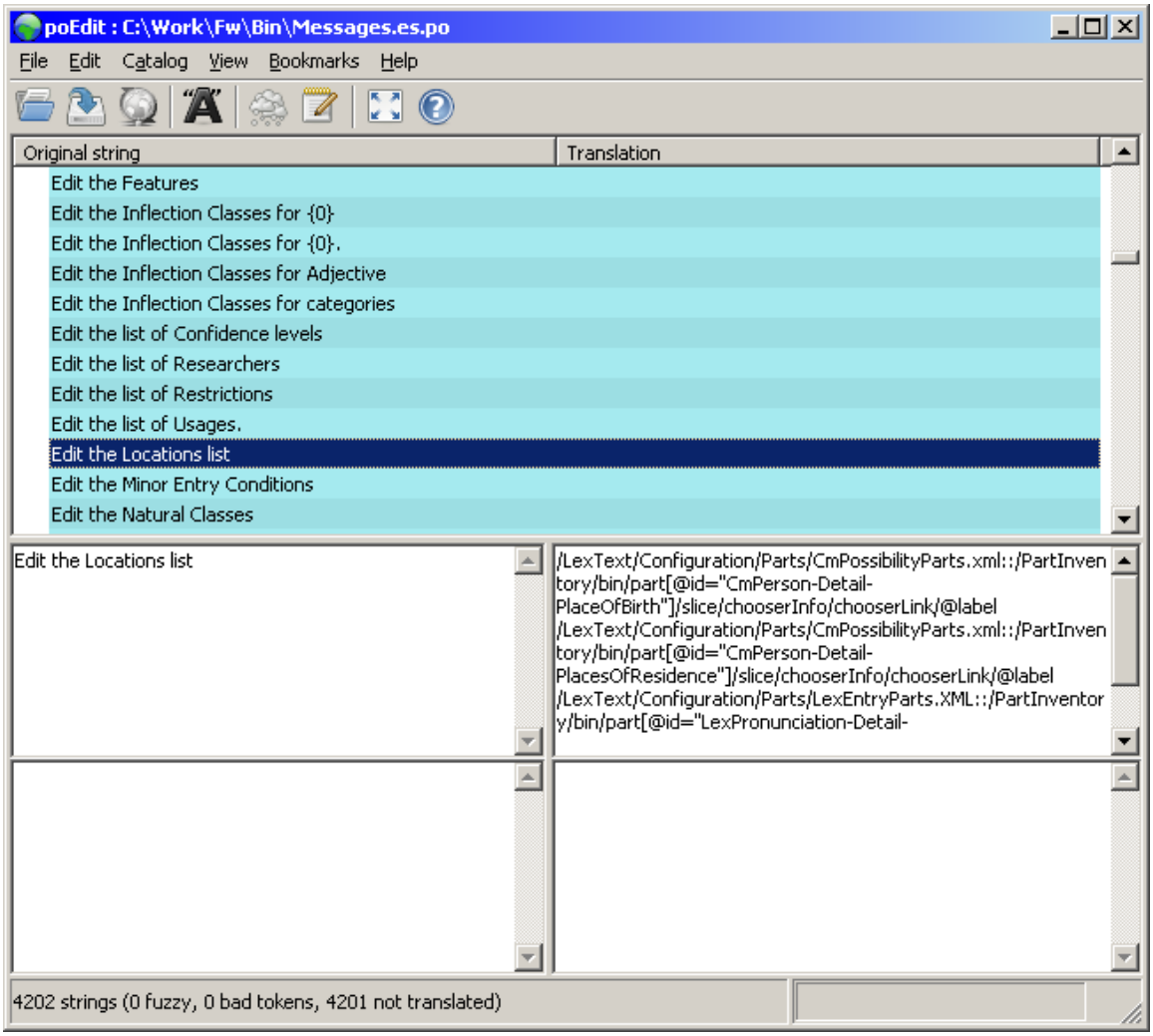
1. Download and install poEdit from http://www.poedit.net/download.php.
2. Run poEdit the first time and answer initialization information.
3. Use File…New catalogue from POT file… and select the POT file you received from the developers. When it asks for information, fill in answers similar to the following Project name and version: FieldWorks N.N

Team: your name(s)
Team's email address: your e-mail address
Language: the target language for translation
Country: the target country
Charset: utf-8
Source code charset: utf-8

This process creates a Messages.xx.po file where xx is the target system language designator you have chosen, such as ‘es’ for Spanish.

2.3.2 Using poEdit

During the translation process, you start poEdit and use File…Open to open the PO file you created.



The top pane shows the strings in the catalog. Information for the selected string is shown in the bottom panes. The left contains the source (top) and target (bottom) strings the right contains source (top) and target (bottom) notes. The source notes come from the POT file and give information on the source of the string. You can put any information you want in the target note field. To do this, click the Edit comment button in the toolbar.

You can keep track of uncertain translations by using the Fuzzy Translation toggle button on the toolbar.

Some strings contain templates with placeholders for substrings. For example 'Edit the Inflection Classes for {0}.' During operation, the program will insert a substring into the string at the {0} location (e.g., 'marker'). You can move the placeholders around to any location in the string template. Where more than one placeholder is present, you can reorder these as needed. The substring for {1} will be inserted wherever {1} occurs in the template.

Some strings are designed for menus. They have an underscore preceding the character that is used as an Alt key shortcut. For example the string '_Complex Feature' will be displayed in a menu as 'Complex Feature' and can be activated by Alt+c. You can change the location of the underscore to change the shortcut key, but you should be careful that a given letter is not underlined more than once in any menu.

All of the changes you make are stored in the PO file. When you want to see the results of your work, you'll need to send the PO file to a developer to incorporate into the next nightly build.

Note: At least for FieldWorks 5.4 and earlier, there is an 'en' string in the localization file. This string should be translated to the locale string of the language into which you are translating. For example, if you are translating into German, this should be 'de'. Any other string can cause problems, particularly with TE File…Open.

2.3.3 Merging an updated POT file

As the program changes, you may receive a new POT file from developers where new strings have been added, others have been changed, and some deleted. To continue translations, you'll need to merge the POT file into your PO file. This is done using the Catalog…Update from POT file… This will add any new strings from the POT file and delete any strings that are no longer in the POT file.

Caution: If you received a FieldWorks.pot file initially and the update is a Flex.pot file, you would lose all of the translations for TE. Before merging files, make sure your original PO file is backed up in a safe place where you can restore it if needed.

2.3.4 Translation memory

PoEdit provides a way to store translated strings in a place where common translations can be applied to any PO file. This is called translation memory. You can set up Translation Memory as follows:

1. Choose File…Preferences and go to the Translation Memory tab
2. Click Add to add the abbreviation for your target translation
3. Click 'Generate database' to update the translation memory.
4. In the 'Update translation memory' dialog, select the path(s) to search for translation files. The poEdit default is c:\Program Files\poEdit\share\locale.
5. Click Next to select files to use as the source of the translation memory.
6. Click Finish to finish building the translation memory.

You can use the Translation Memory to automatically add translations to your PO file by choosing Catalog…Automatically translate using TM. Translations added in this way are

displayed in yellow with a computer icon to the left. You can click the Fuzzy toggle to change these to normal translations.

Microsoft provides 12,400 common English words in some 50 languages in a downloadable file at: http://www.microsoft.com/globaldev/tools/MILSGlossary.mspx. This could be a valuable starting point.

2.3.5 Creating a local strings-xx.xml file

For localizing Flex, it's possible to generate strings-xx.xml directly on your machine without sending your po file to Dallas for processing. This covers most of the menus, tools, and labels in the detail view. It does not cover dialogs. To see dialog changes you'll need to send your po file to Dallas for processing on a full development system.

If you want to generate strings-xx.xml, request LocaleStrings.exe from Dallas. When you get it, copy it into your c:\Program Files\SIL\FieldWorks directory along with a copy of your messages.xx.po file, then give the following command in a command window:

```
LocaleStrings -s c:\Program Files\SIL\FieldWorks\messages.xx.po
```

You should then see a new strings-xx.xml file in c:\Program Files\SIL\FieldWorks\Language Explorer\Configuration, and after restarting Flex, you should see the changes in the UI.

3 Localizing the Lists

3.1 Overview

At this point we have a separate approach for localizing lists stored in the project file. As of FieldWorks 7.0 you can export one or more lists in any number of languages. These exported files can be imported into any project to add the localizations to the lists in that project. This import will not create, delete, or reorder any items in the destination list, or modify any English strings, but will only supply localized translations for existing English strings.

To export the lists, in the Lists area choose File...Export and choose Translated Lists from the export dialog. This brings up another dialog that allows you to choose which lists to export, and which languages in addition to English to export, along with the output file name for the list. The available languages come from vernacular and analysis writing systems in the FieldWorks Project Properties dialog whether or not they are checked. The export will include English plus all of the checked languages whether or not they currently have content. This makes it easy for a translator to fill in any missing translations.

To import a list that was exported via the Translated Lists option, choose File...Import...Translated List Content, then select the xml file to import. This will add or replace any translated strings in the lists that are included in the xml file.

3.2 Translation file format

The xml file used for translations contains one or more lists in a format similar to the nested XML file used for template list files such as SemDom.xml. The core structure for the file is

```
<?xml version="1.0" encoding="UTF-8"?>
<Lists date="10/30/2010 1:08:58 PM">
  .... List element for each list.
</Lists>
```

The Lists element contains a date, but the date is currently unused. This is a typical list element:

```
<List owner="LexDb" field="DomainTypes" itemClass="CmPossibility">
<Name>
<AUni ws="en">Academic Domains</AUni>
<AUni ws="fr">Domaines techniques</AUni>
</Name>
<Abbreviation>
<AUni ws="en">AcDom</AUni>
<AUni ws="fr"></AUni>
</Abbreviation>
<Possibilities>
<CmPossibility>
<Name>
<AUni ws="en">anatomy</AUni>
<AUni ws="fr">anatomie</AUni>
</Name>
<Abbreviation>
<AUni ws="en">Anat</AUni>
<AUni ws="fr">Anat</AUni>
</Abbreviation>
</CmPossibility>
<CmPossibility>
<Name>
<AUni ws="en">anthropology</AUni>
<AUni ws="fr">anthropologie</AUni>
</Name>
<Abbreviation>
<AUni ws="en">Anthro</AUni>
<AUni ws="fr">Anthro</AUni>
</Abbreviation>
<SubPossibilities>
<CmPossibility>
<Name>
<AUni ws="en">cultural anthropology</AUni>
<AUni ws="fr">anthropologie culturelle</AUni>
</Name>
<Abbreviation>
<AUni ws="en">Cult anthro</AUni>
<AUni ws="fr">Anthro cult</AUni>
</Abbreviation>
</CmPossibility>
</SubPossibilities>
</CmPossibility>
</Possibilities>
</List>
```

This sample contains an Academic Domains list with three items—two top level items and one subitem (cultural anthropology). Each name and abbreviation contains an English and a French string. Every field must have an English form, otherwise the export does not export the field. Any number of additional languages can be added to the strings. Every language (other than English) that is present will be imported during the import process. The list abbreviation demonstrates a French string that is missing a translation (e.g., AUni ws="fr"></AUni>). The data in the file is all UTF-8 Unicode characters. The

List element contains attributes for owner, field, and itemClass. These are needed during the import process to locate the desired list.

These are the possible List elements that can be included in the file:

Academic Domains
<List owner="LexDb" field="DomainTypes" itemClass="CmPossibility">
Anthropology Categories
<List owner="LangProject" field="AnthroList" itemClass="CmAnthroItem">
Complex Form Types
<List owner="LexDb" field="ComplexEntryTypes" itemClass="LexEntryType">
Confidence Levels
<List owner="LangProject" field="ConfidenceLevels" itemClass="CmPossibility">
Education Levels
<List owner="LangProject" field="Education" itemClass="CmPossibility">
Genres
<List owner="LangProject" field="GenreList" itemClass="CmPossibility">
Lexical Relations
<List owner="LexDb" field="References" itemClass="LexRefType">
Locations
<List owner="LangProject" field="Locations" itemClass="CmLocation">
Morpheme Types
<List owner="LexDb" field="MorphTypes" itemClass="MoMorphType">
Notebook Record Types
<List owner="RnResearchNbk" field="RecTypes" itemClass="CmPossibility">
Categories
<List owner="LangProject" field="PartsOfSpeech" itemClass="PartOfSpeech">
People
<List owner="LangProject" field="People" itemClass="CmPerson">
Positions
<List owner="LangProject" field="Positions" itemClass="CmPossibility">
Restrictions
<List owner="LangProject" field="Restrictions" itemClass="CmPossibility">
Roles
<List owner="LangProject" field="Roles" itemClass="CmPossibility">
Semantic Domains
<List owner="LangProject" field="SemanticDomainList" itemClass="CmSemanticDomain">
Sense Types
<List owner="LexDb" field="SenseTypes" itemClass="CmPossibility">
Status
<List owner="LangProject" field="Status" itemClass="CmPossibility">
Text Chart Markers
<List owner="DsDiscourseData" field="ChartMarkers" itemClass="CmPossibility">
Text Constituent Chart Templates
<List owner="DsDiscourseData" field="ConstChartTempl" itemClass="CmPossibility">
Text Markup Tags
<List owner="LangProject" field="TextMarkupTags" itemClass="CmPossibility">
Time of Day
<List owner="LangProject" field="TimeOfDay" itemClass="CmPossibility">
Translation Types
<List owner="LangProject" field="TranslationTags" itemClass="CmPossibility">
Usages
<List owner="LexDb" field="UsageTypes" itemClass="CmPossibility">
Variant Types
<List owner="LexDb" field="VariantEntryTypes" itemClass="LexEntryType">

The semantic domain list has some additional structure in each semantic domain as illustrated here:

```
<CmSemanticDomain guid="63403699-07c1-43f3-a47c-069d6e4316e5">
<Name>
<AUni ws="en">Universe, creation</AUni>
<AUni ws="fr">L'univers physique</AUni>
</Name>
<Abbreviation>
<AUni ws="en">1</AUni>
<AUni ws="fr"></AUni>
</Abbreviation>
<Description>
<AStr ws="en">
<Run ws="en">Use this domain for general words referring to the physical universe. Some languages
may not have a single word for the universe and may have to use a phrase such as 'rain, soil, and things
of the sky' or 'sky, land, and water' or a descriptive phrase such as 'everything you can see' or
'everything that exists'.</Run>
</AStr>
<AStr ws="fr">
<Run ws="fr"></Run>
</AStr>
</Description>
<Questions>
<CmDomainQ>
<Question>
<AUni ws="en">(1) What words refer to everything we can see?</AUni>
<AUni ws="fr">Quels sont les mots qui font référence à tout ce qu'on peut voir?</AUni>
</Question>
<ExampleWords>
<AUni ws="en">universe, creation, cosmos, heaven and earth, macrocosm, everything that exists</AUni>
<AUni ws="fr">univers, ciel, terre</AUni>
</ExampleWords>
<ExampleSentences>
<AStr ws="en">
<Run ws="en">In the beginning God created &lt;the heavens and the earth&gt;.</Run>
</AStr>
<AStr ws="fr">
<Run ws="fr"></Run>
</AStr>
</ExampleSentences>
</CmDomainQ>
</Questions>
</CmSemanticDomain>
```

Semantic Domains as well as several other list types store a fixed unique identifier with each item. This is stored in the guid attribute of CmSemanticDomain. These special lists always store the same guid for each item regardless of which language project it is in. Standard lists still have guids for each item in the project file, but they are not identical across project files, so these are not listed in list exports. The lists that use fixed guids are

- Anthropology Categories
- Complex Form Types
- Morpheme Types
- Notebook Record Types
- Semantic Domains
- Translations Types
- Variant Types

In addition to the normal fields for a list item, these items have some additional fields:

- Semantic Domain items also have a sequence of questions with example words and sentences for each one as illustrated above.
- Location and Person items have an Alias field.
- Complex Form Type and Variant Type items have a ReverseAbbr.
- Lexical Relation items have a ReverseName and ReverseAbbreviation field.

3.3 What does a translator do?

A translator would typically edit the XML file discussed above. Each string (French in this example) that is missing a translation, or needs a translation corrected should be edited in Notepad or ZEdit (after using Options…ReOpen As…Unicode (UTF-8)).

```
<List owner="LexDb" field="DomainTypes" itemClass="CmPossibility">
<Name>
<AUni ws="en">Academic Domains</AUni>
<AUni ws="fr"></AUni>
</Name>
<Abbreviation>
<AUni ws="en">AcDom</AUni>
<AUni ws="fr"></AUni>
</Abbreviation>
<Description>
<AStr ws="en">
<Run ws="en">This list holds information about Academic Domains</Run>
</AStr>
<AStr ws="fr">
<Run ws="fr"></Run>
</AStr><Possibilities>
<CmPossibility>
<Name>
<AUni ws="en">anatomy</AUni>
<AUni ws="fr"></AUni>
</Name>
<Abbreviation>
<AUni ws="en">Anat</AUni>
<AUni ws="fr"></AUni>
</Abbreviation>
</CmPossibility>
</Possibilities>
</List>
```

The Name, Abbreviation, and Description at the top of each list are properties of the list, not an item in the list. The translated name will show up in the Lists area when the Flex interface is switched to that language. Also, the name and description will show up in the Tools...Configure...List dialog. (The list abbreviation is currently unused in the user interface.) List items will show up whenever a language is enabled in the Analysis Writing Systems section of the FieldWorks Project Properties dialog. A translator should never alter an English string since these need to match the English strings that are already in the project file.

It would also be possible to do the translations in some other format (e.g., SFM) as long as the results are converted into the XML form described above before importing into Flex.

3.4 Import considerations

During an import into Flex, the list is located using the extra attributes in the List element. All existing items in the list are then retrieved from the project file. The program then matches each item from the project file with the same item from the import file. When items use guids, the items are matched via the guid. Otherwise the items are matched via the English name. When a matching item is found, each of the translated strings from the import file are added to the matching item from the project file, replacing any translated strings that were already there. The English strings are never altered. This process will also not add, remove, or reorder any items in the project file. If an item in the project file is not matched with an item from the import file using this process, that item will not be changed by the import process. This matching process ignores list hierarchy, so unless guids are used in the matching process, if there are identical names in a hierarchical list, the process will not work correctly.

Since semantic domains have a series of questions, the import process must match questions for each item between the project file and the import file. Since there are no fixed guids stored with questions, the only way it can match questions is using the English Question string. Again, if a question cannot be matched in this way, it will not be changed.

Since the English string is used for matching in all cases where there are no fixed guids, this is why the translator should never alter the English string in the translation file. Actually, if English strings are altered for anything other than an item name or question, it will have no effect on the import process since these additional English strings are not involved in matching items or questions.

Caution: If a blank element is present in the import file, an existing translation for that language will be deleted from the project. For example, if the file contains

```
<CmPossibility>
<Name>
<AUni ws="en">anatomy</AUni>
<AUni ws="fr"></AUni>
</Name>
</CmPossibility>
</Possibilities>
```

and there is currently a French translation for ‘anatomy’ in the project, that translation will be deleted. However, since there isn’t a German element in the file, an existing German translation would not be deleted.

3.5 Automatic loading for new project

When a new project is created, the first time it is opened in Flex, the program will automatically look for any localized list files in c:\Program Files\SIL\FieldWorks\Templates. It looks for file names matching LocalizedLists-*.xml. If any are found, they will each be loaded automatically just as though the user chose File...Import...Translated List Content on each file. The format of the LocalizedLists file must be identical to that exported by File...Export...Translated Lists(s). Normally the file should end with the language code (e.g., LocalizedLists-de.xml for German), and the contents of the file should only have English and the specified language code. Note, although the localizations are added during project initialization, they will not be visible to the user until they add and check

the localization language to their analysis writing systems. The LocalizedLists file may contain any number of the lists that can be exported from Flex using File...Export...Translated List(s). The file may contain any number of writing systems as well. Writing systems will be created in the project if they are not currently in the project. Any translations in the file will overwrite corresponding translations currently in the database.

This feature allows a localization package to add the binary program localization files along with this custom LocalizedLists file, thus enabling the program to work in a given locale. It also means we do not need to add localizations to NewLangProj.xml, SemDom.xml, and POS.xml since these localizations can now come in with a LocalizedLists file.

Existing databases can also be updated to what's in the LocalizedLists file by loading it manually through File...Import...Translated List Content.

This process does mean you should not leave any LocalizedLists files in your Templates directory unintentionally or you'll probably get unwanted translations in new projects that are created. This is mainly a concern for developers or support personnel.