**Internationalization (i18n) and Localization (l10n) for SAM**

Internationalization (required software infrastructure to support l10n)

1. LK error messages from the compiler and runtime engine. (support is partially complete)
2. LK built-in function descriptions and documentation
3. WEX library C++ static text strings, e.g. error messages in dialogs.
4. Number formatting routines in WEX used in plotting/dview/numeric widgets
5. DView dialogs and time representation, also in standalone app
6. SSC SDKtool user interface
7. Compute module warnings and error messages
8. SSC variable table labels and units, particularly currency
9. SAM application C++ static text strings in dialogs
10. Solarpilot messages
11. User interface page labels – need to write extractor to dump all labels/units/indexlabels into a .pot/ file for merging with C++ static string po files. Also need widget properties such as for static text labels and group box headers.
12. Code in UI callbacks that displays messages, e.g. download solar data from NSRDB
13. Startup script names of technologies and financing models, sidebar captions, etc.
14. Labels of object properties in 3D shade editor?
15. Metric labels, units, and formatting in IDE/Metrics
16. Cash flow row labels
17. Autograph labels and units
18. Loss diagram labels for PV
19. PDF reports, including in script objects and text objects.
20. Support for different sets of default files for different countries
21. Version upgrade script messages

Localization (using i18n capabilities to translate/localize the software for a region)

1. Translating all the text extracted from the various pieces above
2. Specifying currency units/etc
3. Localized default values in project files

Other questions/issues?

1. RTL support for Hebrew, Arabic, Farsi?
2. No support for vertical text layout
3. User manual?
4. Scripting manual?
5. Website? E.g. <http://sam.nrel.gov/en/> or <http://sam.nrel.gov/fr/> or <http://sam.nrel.gov/es/>
6. Announcements that are displayed on startup page?
7. Registration web service email?
8. Macros (descriptions and display labels?)

Effort

1. Huge infrastructure effort, probably close to a full FTE required to support all of the aspects listed in ‘Internationalization’ support.
2. Localization is significant effort too, but probably less than full i18n above. A full localization is probably very difficult for all technologies, but a partial one (i.e. no reports, metrics, cashflow, etc, just UI labels, or just for one technology) is perhaps not terrible.

wxWidgets i18n/l10n Getting started

Start by selecting which strings are translatable and mark them with the \_() macro.

// example : 'Hello World' is flagged as translatable

[wxMessageBox](http://docs.wxwidgets.org/trunk/group__group__funcmacro__dialog.html#ga193c64ed4802e379799cdb42de252647)( [\_](http://docs.wxwidgets.org/trunk/group__group__funcmacro__string.html#ga8a02b8875a521df57263a9e6f090f2d0)("Hello World") );

Once translatable strings are marked, you can extract the strings into a catalog using GNU gettext (you'll need to have it installed to extract the string catalog using the UNIX commands provided below, however your user will not need to have it installed since support is built inside wx).

# example sh script to generate catalog file

# it searches for strings marked with \_() in all .cpp files in directory ./src

# the generated file is called 'app\_name.pot' and is generated in ./po

CPP\_FILE\_LIST=`find ./src -name '\*.cpp' -print`

xgettext -d app\_name -s --keyword=[\_](http://docs.wxwidgets.org/trunk/group__group__funcmacro__string.html#ga8a02b8875a521df57263a9e6f090f2d0) -p ./po -o app\_name.pot $CPP\_FILE\_LIST

Catalog editors may also be able to extract strings, however the process is less automatized this way.

Translating what's in the catalog

Once you have the .pot catalog, you can edit it in your favorite catalog editor. Here [poEdit](http://poedit.sourceforge.net/) is the one used.

**Starting a translation to a new language**

* Use menu 'File > New catalog from POT file'
* Select the *.pot* catalog file
* Enter appropriate information about yourself and the language
* Translate the strings and save the file as a *.po*.

**Updating an existing translation with latest strings updates**

* Open the *.po* file in poEdit
* Choose menu 'Catalog > update from POT file'
* Select the *.pot* catalog file

Data file organization

To be translated, your program must first have a catalog name, say "myprogram". This is just an identifier for your language files, nothing more.

There will be a directory on your system for installed languages, with a subdirectory for each language, for example 'nl' for Dutch.

* Windows : these directories typically come under the application directory. So, if your program is translated into Dutch there will be a file localedir\nl\myprogram.mo.
* Linux/Unix : generally {prefix}/share/locale/[lang code]/LC\_MESSAGES/app\_name.mo
* Mac OS X : {App name}.app/Contents/Resources/[lang code].lproj/app\_name.mo
  + Note that you should create an empty folder named en.lproj (if your app is natively in English) so that the system knows this language is supported even though you have no .mo file for this language

Your installer should take care of placing the language files in the appropriate directories.

You can check language codes here : <http://xml.coverpages.org/iso639a.html>

Getting it inside your app

Use code similar to this :

[wxLocale](http://docs.wxwidgets.org/trunk/classwx_locale.html)\* locale;

long language;

void initLanguageSupport()

{

language = wxLANGUAGE\_DEFAULT;

// fake functions, use proper implementation

if( userWantsAnotherLanguageThanDefault() )

language = getUsersFavoriteLanguage();

// load language if possible, fall back to english otherwise

if([wxLocale](http://docs.wxwidgets.org/trunk/classwx_locale.html)::IsAvailable(language))

{

locale = new [wxLocale](http://docs.wxwidgets.org/trunk/classwx_locale.html)( language, wxLOCALE\_CONV\_ENCODING );

#ifdef \_\_WXGTK\_\_

// add locale search paths

locale->AddCatalogLookupPathPrefix([wxT](http://docs.wxwidgets.org/trunk/group__group__funcmacro__string.html#ga437ea6ba615b75dac8603e96ec864160)("/usr"));

locale->AddCatalogLookupPathPrefix([wxT](http://docs.wxwidgets.org/trunk/group__group__funcmacro__string.html#ga437ea6ba615b75dac8603e96ec864160)("/usr/local"));

[wxStandardPaths](http://docs.wxwidgets.org/trunk/classwx_standard_paths.html)\* paths = ([wxStandardPaths](http://docs.wxwidgets.org/trunk/classwx_standard_paths.html)\*) &[wxStandardPaths](http://docs.wxwidgets.org/trunk/classwx_standard_paths.html)::Get();

[wxString](http://docs.wxwidgets.org/trunk/classwx_string.html) prefix = paths->GetInstallPrefix();

locale->AddCatalogLookupPathPrefix( prefix );

#endif

locale->AddCatalog([wxT](http://docs.wxwidgets.org/trunk/group__group__funcmacro__string.html#ga437ea6ba615b75dac8603e96ec864160)("app\_name"));

if(! locale->IsOk() )

{

std::cerr << "selected language is wrong" << std::endl;

delete locale;

locale = new [wxLocale](http://docs.wxwidgets.org/trunk/classwx_locale.html)( wxLANGUAGE\_ENGLISH );

language = wxLANGUAGE\_ENGLISH;

}

}

else

{

std::cout << "The selected language is not supported by your system."

<< "Try installing support for this language." << std::endl;

locale = new [wxLocale](http://docs.wxwidgets.org/trunk/classwx_locale.html)( wxLANGUAGE\_ENGLISH );

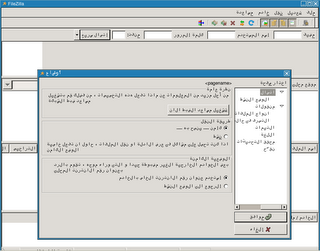
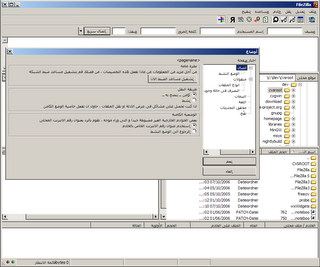
language = wxLANGUAGE\_ENGLISH;

}

}

For other languages to work, you'll need your *.po* files compiled as *.mo* files (.mo translation files are binary versions of .po files) . poEdit will do it automatically upon saving if the appropriate option is checked in the preferences.

### **Right-To-Left layout (RTL) support in wxWidgets**

Here is some information about the support for locales with right-to-left (RTL) layout out in wxWidgets. As you probably know, RTL-support is required mostly of Hebrew, Arabic and Farsi. It has been decided that wxWidgets should follow mostly the Windows API and Windows approach for displaying RTL text and layout in applications. This approach is described in a document at MSDN [here](http://www.microsoft.com/globaldev/getwr/steps/WRG_mirror.mspx). In short, all coordinates for drawing and for window positioning are mirrored horizontally which means that (apart from bitmap mirroring) little has to be done at the user level. Also, the various native Win32 controls are modified to support RTL design.  
  
This is different from what is done under GTK+, where no coordinates are mirrored. Instead, every app needs to mirror everything itself, even if the job of mirroring controls is mostly done within GTK+'s layout container automatically. As far as drawing within wxWidgets is concerned, coordinates are mirrored at the wxDC level in the GTK+ port. Some more information about GTK's RTL support can be found [here](http://www-128.ibm.com/developerworks/aix/library/au-internatlgtk/index.html).  
  
I have added two screenshots from the popular FileZilla application running in the "ar\_EG" (Egyptian variant of Arab) locale (screenshots in parts thanks to Tim Kosse). Since the tree control in the Linux screenshot is using generic wxWidgets code, this also demonstrates how drawing and scrolling is mirrored for user windows.  
  
Below is a screenshot using wxWidgets and GTK 2.4.9.  
[](http://photos1.blogger.com/blogger/4978/3977/1600/Arab_Linux.0.png)  
  
Here is roughly the same screenshot using Windows 2000.  
[](http://photos1.blogger.com/blogger/4978/3977/1600/Arab_Windows.0.png)