A Methodology for Developing SAS Applications for Use In Multiple Languages

Christopher A. Roper, Qualex Consulting Services, Inc., Fairfax, Va.
Gina M. Thomas, Qualex Consulting Services, Inc., Fairfax, Va.
Michael Gilman, Qualex Consulting Services, Inc. Fairfax, Va.

Abstract

Developing SAS® applications that will be used in several countries, each with its own native language, presents unique challenges for the SAS applications developer. This paper will present a methodology for developing a SAS application that may be used in any number of languages, without requiring a rewrite of the application code for each language translation. The application is developed once, the translation methodology is applied once, and then the application can be translated as often and to as many languages as is necessary, without having to rewrite any of the original application code.

Introduction

There is a growing demand for software applications that have the flexibility to present the application text in the native language of the end user. For example, withdrawing cash from a bank cash machine often entails pushing a button to tell the machine if you want your instructions in English or some other language. This means that the textual components of the application must be translated into several languages.

One method of translating the text of the application is for a programmer to rewrite all textual components into the new language. This approach has significant drawbacks. It forces the recoding of significant portions of the application code and thus creates an opportunity for the introduction of errors in the code. Also, updates, bug fixes, and application enhancements must be performed independently for each translated version of the application. Inevitably, bugs are introduced into the various translated versions of the application, and version control becomes very difficult. Each translation is a new version to support. Frequently, each of these versions requires its own support team. Keeping the versions synchronized adds an unnecessary layer of bureaucratic complexity to the task of version control for the application.

Methodology

This paper introduces a data driven methodology for SAS applications which accomplishes this translation task, with no ongoing modification of the application code. This methodology uses SAS data sets as translation tables to control the translation process. These tables perform two key functions. The first is to translate the textual components of all SAS frame entry widgets. This is done at build time using the build time translation table. The second function is to translate the normally hard coded text, such as error and warning messages. This is done at compile time using the compile time translation table. A data entry interface enables the translation professionals to add the translated text to these tables. This is called the translation interface.

Creating the Translation Tables

The build time translation table contains six columns. Those columns are:

- English: The English text
- Trantext: The translated text
- Maxlen: The maximum allowable length of the tranlated text
- Window: The four-level name of the frame window
- Widget: The name of the frame widget
- Method: The SCL method to apply text

Figure 1 illustrates what this translation table might look like.

Four of the build time translation table columns are generated using a custom utility that reads the catalog(s). This utility will automatically enter the values for the English, Maxlen, Window, and Widget columns. The Method column is entered by a developer. The sixth column, TRANTEXT, will be entered into the table through the translation interface by a translation professional.

The compile time translation table contains five columns. They are:

- Macro: A macro variable
- English: The English text
- Trantext: The translated text
- Maxlen: The maximum allowable length of the tranlated text
- Window: The four-level name of the frame window

Figure 2 illustrates what the compile time translation table will look like.

Figure 1: Build Time Translation Table

English	Trantext	Maxlen	Window	Widget	Method
Exit		8	LIB.CAT.SCR.FRAME	BTN1	_SET_LABEL_
Start		8	LIB.CAT.SCR.FRAME	BTN2	_SET_LABEL_

Figure 2: Compile Time Translation Table

Macro	English	Trantext	Maxlen	Window
W1BTN1	Top		8	LIB.CAT.SCR.FRAME
W1BTN2	Bottom		8	LIB.CAT.SCR.FRAME

The compile time translation table is built through an iterative process of editting each SCL source entry. Every occurrence of hard-coded text is replaced with the name of a macro variable. This macro variable is entered into the translation table along with the text it represents. In this way the macro variable maps the text in the translation table to its original location in the application source code. The window name and maximum length for the translated text are also added to the translation table at this time. And as in the build time translation table, the translated text column is left empty, to be completed by the language translation professionals.

This completes the preparation phase of the methodology. The next phase is controlled by the translation engine. It consists of three components, the translation interface, the build time process, and the compile time process.

Translation Interface

The first component of the translation engine is a simple data entry application. The translation professionals read the English text, and enter the appropriate translations, all on the same screen. Having both the English and local text on the same screen in side by side windows greatly simplifies the translation process. The translation interface interactively controls the length of the translated text by using the maximum length field from the translation table. This ensures that the text entered for the translation will fit within the defined size of the associated widget.

With the translation table finished, the application is now complete and ready to have the translation engine localize the application text. This is accomplished in two processes, the build time process and the compile time process.

Build Time Process

The build time process translates all of the textual components of the widgets at build time using Proc Build. The translation table is read to retrieve the screen name (Window), widget name (Widget), method (Method), and translated text (Trantext). Then it loops through each catalog entry and rebuilds each frame. One of the SAS build time methods is overridden. This allows the text of the widgets to be translated using the information from the build time translation table. The methods from this table assign the translation to the instance variables that contain the text for the widget. In this way the build time process can perform all of the actual translations without any interaction from a developer, thus making this process completely data driven. With this process complete, the only remaining translation left is the compile time process.

The Compile Time Process

In the compile time process, all macro variables are read from the compile time translation table. The program uses the CALL SYMPUT function to create macro variables and assign to them the translated text. This loads the macro variables into memory. Thus when the SCL entry is compiled, the macro variables resolve to the translated text. Therefore, no action by a developer is required to translate the text that was originally hard coded in the application.

Benefits Of The Methodology

The data driven methodology has several strengths. First, the code development, maintenance, and enhancements are performed by the original development team. These developers retain the ownership responsibility for the application. This ensures that problems are addressed by the person best suited to resolve them. Secondly, updates to the application code can be performed much more quickly since one team controls the application development and maintenance. Also, version control is made much simpler since there is only one version to maintain.

Language translation professionals perform the translations. They have no responsibility for application logic changes, and thus can dedicate their efforts to translation. Therefore, the time required to translate an application is greatly diminished.

With this methodology, only the localized version of the application is changed, not the original source code. Therefore, the translation process can not corrupt the application code. As quickly as the translation experts can make the translations, the application can be localized, without the need for extensive retesting and quality assurance tests. Additionally, since all the translation occurs at build time and compile time, there is no application performance degradation due to translating a window as it is presented to the end user.

Conclusion

The variety of languages in our world need not be an obstacle to creating applications to be used in several countries with disparate languages. The localization methodology described above grants the application developer the freedom to create applications for use in many languages. A corporation with offices all over the world can develop locally, and distribute globally, without fear of creating an application that will unnecessarily drain critical development resources to maintain. One application, one development team, one version to control, and any number of languages to be presented to the end users. This methodology is the tool that the SAS application developer can use to create one application for all global customers.

Author Contact

Christopher A. Roper
Qualex Consulting Services, Inc.
9219 Baily Lane,
Fairfax, Va. 22031
USA
email: QLXCAR@AOL.COM

Gina M. Thomas
Qualex Consulting Services, Inc.
9219 Bailey Lane
Fairfax, Va. 22031
USA
email: QLXGMT@AOL.COM

Michael Gilman
Qualex Consulting Services, Inc.
9219 Baily Lane
Fairfax, Va 22031
USA
email: QLXMXG@AOL.COM

Acknowledgments

SAS® is a registered Trademark of SAS Institute, Inc. in the USA and other countries.