COOL TOOL -- FEB '94

Margin: Preparing applications for international use will be a requirement for all but the most parochial of developers in the next decade. The INTL Toolkit is a highly automated method to enable your apps for multi-lingual support today.

Cool Tool: The INTL Toolkit

You might be tempted to breeze on past this article because you think it doesn't apply to you. Most likely, you would be wrong to think so. I never gave the international market a second thought when I introduced The Ultimate FoxPro Reference 15 months ago. Today, TUFR is being used in 17 countries.

Your customers are going to be asking for help to internationalize their applications. FoxPro is already the best tool for data manipulation, with the extensive support for code pages and collating sequences. The INTL Toolkit now provides similarly extensive functionality for translating your menus and screens into multiple languages.

What INTL Does

INTL provides transparent translation of all text strings in the menus and screens of an application. Furthermore, it gives you the choice to either permit the user to select the language to operate in at run-time, or to supply language-specific versions of your application.

INTL uses a table that contains all of the text strings in your system. These text strings include menu pads, hard-coded text strings, WAIT WINDOW prompts, prompts for objects like radio buttons and check boxes, window titles, and so on. Multiple columns in this table contain the translated versions of each string for each language that you choose. When you run your application, it will pull the variables from the appropriate column based on the contents of a global variable.

You can provide a menu choice for the user that changes the value of this global variable, and thus drive the immediate translation of each text string "on-the-fly." A single application can thus be sent out to be used with multiple languages without needing multiple versions of the compiled application.

What INTL is made up of

INTL consists of a main program, several supporting programs, and a default string translation table. The main programs, INTL.PRG and INTLMENU.PRG, are never called by you - rather, GENSCRNX and GENMENUX call them as part of their process to modify your MPRs and SPRs. The modifications make calls to five supporting programs:

I.PRG is the "translate" function used by your application. For example, a window definitions that would ordinarily look like this:

```
IF NOT WEXIST("_qlu11jieb")
 DEFINE WINDOW _qlu11jieb ;
  FROM INT((SROW()-25)/2),INT((SCOL()-77)/2) ;
  TO INT((SROW()-25)/2)+24,INT((SCOL()-77)/2)+76 ;
  TITLE "The Ultimate FoxPro Reference"
ENDIF
```

becomes this:

```
IF NOT WEXIST("_qlu11jieb")
 DEFINE WINDOW _qlu11jieb ;
  FROM INT((SROW()-25)/2),INT((SCOL()-77)/2) ;
  TO INT((SROW()-25)/2)+24,INT((SCOL()-77)/2)+76 ;
  TITLE I("The Ultimate FoxPro Reference")
ENDIF
```

Note that the TITLE string has been "encapsulated" with a call to the I() function. When the screen containing this window

definition is run, FoxPro will look for the string "The Ultimate FoxPro Reference" in the translation table and return the translated version for whatever language is selected.

NOHOT.PRG is a UDF that strips out hot key assignments from a string. NOHOT() is used whenever a push button prompt, menu pad, or other string with hot key specifiers needs to be referenced minus those specifiers. NOHOT() is used extensively in generated screen code.

WPADR.PRG, WPADC.PRG, WPADL.PRG are Windows replacements for the PADR(), PADC() and PADL() functions. Generated screen programs use these functions to properly adjust the placement of text strings based the length of the translated string at runtime.

STRINGS.DBF comes with columns for English, French and German text strings. (You'll have to provide the translated strings.) You can easily add other languages by adding a field to the table for each language to be used.

A step-by-step guide to using INTL

Now that we have the players, let's run through the action. Let's suppose that you have an application built with the Power Tools (Project Manager, Menu Builder and Screen Builder). The screens have titles, text strings, radio buttons, check boxes, push buttons and popups.

1. Modify STRINGS.DBF so that it contains columns for each of the languages for which the app will be enabled. Enter the translated strings for each original string. (No, INTL does not do the actual translation of English words to another language.) STRINGS will then look similar to this:

Coriginal	Cfrench	Cgerman
One	Un	Eins
Two	Deux	Zwei
Three	Trois	Drei

2. Put GENSCRNX/GENMENUX in your FoxPro home directory.

3. Add the following lines to your CONFIG.FP/W file (adjust the path as appropriate, and reboot FoxPro after doing so):

```
_GENMENU = "G:\FPD\GENMENUX.FXP"
_GENSCRN = "G:\FPD\GENSCRNX.FXP"
_MNXDRV2 = "INTLMENU.PRG"
_SCXDRV5 = "INTL.PRG"
_INTLTIMING = "RUN"
```

4. Put copies of INTL.PRG, INTLMENU.PRG, I.PRG, NOHOT.PRG and WPAD?.PRG into the project's directory (I put them in my COMMON directory and set my FoxPro PATH to COMMON.)

5. Put STRINGS.DBF/FPT/CDX into your source code directory.

6. Rebuild the project using Rebuild All. Note that due to the new statements in your CONFIG.FP, GENMENUX and GENSCRNX are called, and they in turn execute the INTL drivers.

7. At this point, you should look at the .SPR generated. You'll see that each text string has been surrounded by the function "I" and use the original text string as a parameter.

8. You can also automatically translate text strings in your code via a similar function call. You should go through your PRGs and screen snippets looking for WAIT and similar commands. For instance:

WAIT WINDOW "Customer not found" NOWAIT

would become:

WAIT WINDOW I("Customer not found") NOWAIT

Better yet, you can use MSGSVC.PRG, which is provided with INTL, to handle *all* your message display needs. I'll discuss MSGSVC() in more detail later in this article.

9. STRINGS.DBF now contains records for every string in the application's menus and screens. You'll need to go into this table and provide translated versions for each string.

10. Run the application. All strings will appear in English (or whatever you configured as your native language when you set up INTL).

11. Initialize the variable _INTLLANG outside the app or in the main program to be equal to the language to translate to.

_INTLLANG = "German"

12. Running the app again will display all text strings translated.

How INTL works

Consider how you would design a multi-lingual application without the aid of the INTL Toolkit. Every screen and menu would have to be duplicated and modified for each language you wanted to use--a maintenance nightmare no matter how you went about it.

You've already seen how INTL solves this problem. The solution lies with that new tool you've been reading about: GENSCRNX. As you've seen in these pages over the past few months, GENSCRNX is a pre- and post-processor for FoxPro's native GENSCRNX. It takes your original .SCX/.SCT files, modifies a copy of them, feeds those new files through GENSCRN, and then (optionally), modifies the resulting .SPR.

The beauty of GENSCRNX is that it contains "hooks" for other developers to extend GENSCRNX's native functionality. INTL is such an extension for GENSCRNX - it seeks out every text string in your menus and screens and surrounds them with I(), the INTL translation function.

A menu generator preprocessor, GENMENUX, which does for menus what GENSCRNX does for screens, is also included with the INTL Toolkit. GENMENUX was originally designed for INTL by Steven Black and has been made more powerful and generic by a fellow Canadian, Andrew Ross MacNeill. Like GENSCRNX, it is in the public domain.

Using GENSCRNX and GENMENUX drivers to encapsulate strings with a localizing function is, on the surface, a very simple solution. However, there are many little quirks in the implementation that INTL goes to great lengths to solve. Centering or justifying displayed strings and adjusting the relationships of objects as they change size in different languages is an amazingly complex task; INTL generates the necessary code and it works very well.

Other goodies

The whole point of this column is to bring tools to your attention that will cause you to sit back and exclaim, "This is SO COOL!" The INTL Toolkit also comes with a couple of other tools that have caused me to do just that.

First, it forces you to poke around (at least a little bit) with GENSCRNX and GENMENUX. This is a benefit in and of itself.

Second, MSGSVC() is a function that provides a single, run-time clearinghouse for all messaging in FoxPro. It serves as a platform to launch all messages and dialogs. While created for INTL, any developer will benefit from making a single pass through your source code and replacing existing message display commands with calls to MSGSVC(). MSGSVC() can create self-adjusting modal alert dialogs with either "canned" or custom push buttons, as well as issue the various forms of the WAIT command. A separate table, MSGSVC.DBF, defines not only the original and translated messages, but the

message type and a host of other information such as whether or not the bell is rung, the bitmap or icon to display along with the message, and so forth. In other words, most aspects of displaying messages to the user become table-driven. MSGSVC is useful even in applications that don't need to be localized for different languages.

Third, the INTL manual contains more information in one place about international issues and FoxPro than you will probably find anywhere else in the FoxPro world. A 40+ page section on international issues covers character sets, code pages, character translation, collation sequences, and culturally correct sorting. Another 40 pages guides you through concepts, problems, and solutions for enabling applications for multi-lingual use. (Both chapters drew heavily on material that first appeared in *Inside FoxPro 2.5 for Windows* by Bob Grommes, published by New Riders Publishing; Steve was a co-author of that book.)

Steven Black has been developing multi-lingual applications and devising better ways to do it since 1986, and it shows. He's worked with the United Nations in Geneva and External Affairs in Canada. Anyone who attended Les Pinter's presentation on developing global applications at the 1993 DevCon heard a terrific tribute to Steve's work: Les, who has advanced his own enabling techniques for a couple of years, saw the beta of Steve's INTL Toolkit about a month before the conference, and was so impressed he rewrote his presentation to feature Steve's methodologies rather than his own.

Finally, the source code itself is a treasure. I get to see a lot of source, and Steve's is among the best I've seen. Rare indeed is the developer who won't be able to learn from Steve's techniques - or get a chuckle from his wry commenting.

Minor Quibbles

It's hard to find fault with INTL, but there is one area where INTL will not help you: reports. Unlike the Screen Builder and Menu Builder, the Report Writer doesn't generate program code, so there is no place for preprocessors like GENSCRNX to "hook in". What would really move the INTL Toolkit from "awesome" to "beyond wonderful" would be a routine that could be called at runtime as a substitute for REPORT FORM which would make a temporary copy of the report layout, process text literals in it (primarily titles, column headers, and footers) with I(), and then execute REPORT FORM against the temporary report table. As with screens, it would not be trivial to make this work without placing limitations on the developer's visual design flexibility. But, if it could be made to work, it would be hard to imagine what anyone could do to improve on the INTL Toolkit.

Thoughtful readers might wonder if the overhead of calling I() throughout an application results in a significant performance hit. Performance of run-time localized applications is suprisingly robust. Often, the difference is not even noticable. Obviously, the extra I/O and the use of an additional work area for the strings table could be problematic for minimal configurations, such as those running standard FoxPro for DOS in limited memory. In such situations, you can select generate-time localization instead of run-time localization, in order to get better performance. Generate-time localization avoids the run-time lookup of the translated string. The potential downside (depending on your requirements) is that you will have to distribute a separate version of your application for each language.

Where to find INTL

See the demo of the INTL Toolkit on the Companion Disk for a trial run. Note that the installation and use of the demo is slightly different from the actual product - be sure to read the README file for specifics. The full product is available for $395 (US) from Steven Black Consulting, 2 Bay Street, Kingston Ontario, K7K 6T7, Canada. Voice: 613-542-4393, CIS: 76200,2110.