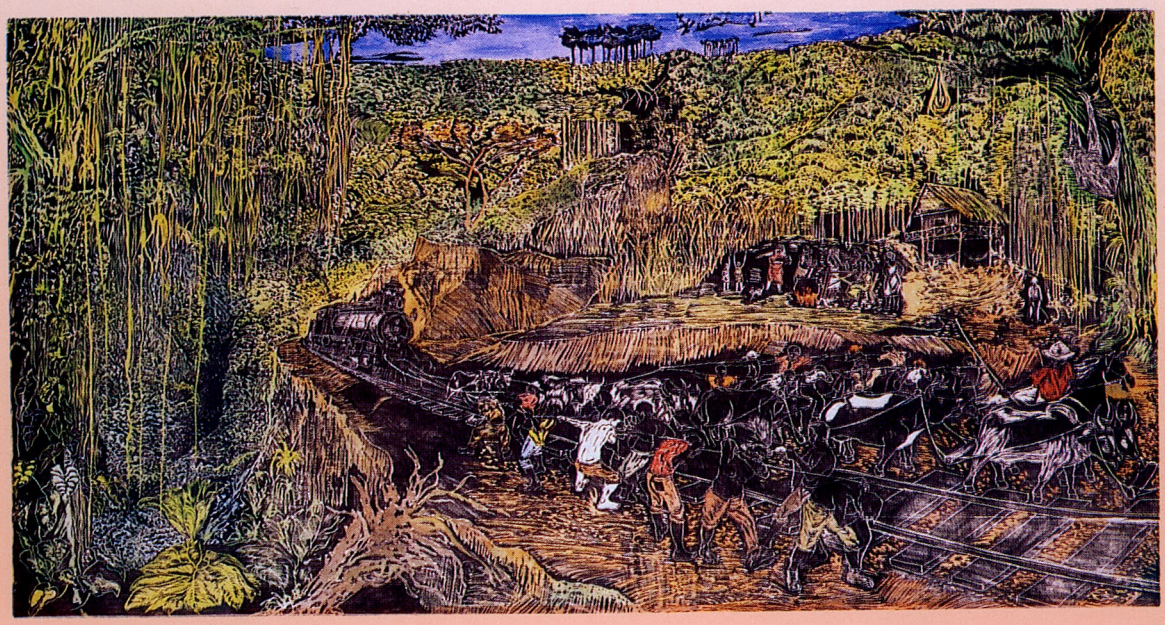
Ingeniería

Revista de la Universidad de Costa Rica
JULIO/DICIEMBRE 1994 VOLUMEN 4 Nº 2



Ingeniería 4 (2): 21-31 1994 San José Costa Rica

LNGrep: EASY AND SECURE xBASE PROGRAM TRANSLATION

Adolfo Di Mare *

RESUMEN

LNGrep es un sistema completo que puede asistir en la tarea de traducir a lenguas foráneas programas escritos en algunos de los dialectos xBase. Quien se encarga de realizar la traducción nunca necesita editar los programas fuente. Más aún, todas las versiones de cada programa residen en el mismo archivo, lo que recorta la complejidad de mantener cada una de las versiones del programa traducido.

SUMMARY

As the need to accommodate a number of (natural) languages grows, it becomes important to be able to write programs that handle multilingual input and output. LNGrep is a complete system that can assist in the translation of xBase programs into foreign languages. Whoever is in charge of the translation will not need to edit the source files. Furthermore, all the versions for each program reside in the same file, which cuts down on the complexity of maintaining each version of the translated programs.

For many years now, my brother Luis Alberto has been a gold mine of ideas for programming projects. He keeps coming up with weird requirements that I enjoy transforming into working programs.

The other day he came to my house and asked me to develop a program to translate source xBase code from English into Spanish, and viceversa. He wasn't expecting an AI program: what he needed was a way to translate every Spanish literal string in the program into its corresponding English version. A program would thus have two versions: one in the English language and the other in Spanish. Their differences would be minor, because only the string literals and other minor details would be changed in each version of program. Luis needed this program because one of his clients is an international corporation with many subsidiaries, both in the USA and in the Caribbean, and the management needed to use the same programs throughout. This can hardly be considered a special case in these days of "modern globalization".

Luis told me that there is a large market of organizations that serve non-english-speaking customers, and that the opportunity to earn lots of money was there for those who came up first with the right translating software. However, to perform the translation there was a special requirement: "You should not touch the source files", he told me. This is hardly an out-of- line requirement. For most companies, their programs are very important. If you are contracting a consulting firm to have some work done, it is only normal for you to take the path of lowest risk. No company likes a third party messing around with its source code; it's just too easy to delete or change a line, and wreck a whole program.

In our first conversation, Luis told me: "Adolfo, I want you to create a TEXTFILE where all the changes are done, and later applied to the source files". That seemed like a good idea: I would write a program to extract from the source files all the lines that need to be translated, then someone translates them, and they are later reapplied to the source files. I immediately sat down to produce the LNGrep System to translate xBase programs.

I took the following premises as truth. First, the lines that needed translation were those that had an xBase string in them. This still holds true in 95% of the cases, and the remaining 5% of cases are very difficult to handle with a general-purpose program. Secondly, I assumed that the translations would be made by a non- program-

* Profesor Catedrático, Escuela Escuela de Ciencias de la Computación e Informática, Universidad de Costa Rica.
email: adimare @cariari.ucr.ac.cr

mer, which meant that a TEXTFILE where the translations would take place should be used. Eventually, a programmer could take to look in the source code. Thirdly, I decided not to support TEXT-ENDTEXT blocks.

I named this approach the LNGrep System as a somewhat (un)fortunate association with the UNIX grep program, that extracts from a text file all those lines that have a common string (FIND is the downsized MS/DOS version of grep). The LNGrep programs to translate into a foreign language xBase source code are the following:

- LNGrep: Extracts into a TEXTFILE all the xBase strings from the source code.
- LNGapply: Inserts into the source code the translated lines from TEXTFILE.
- LNGswap: Changes the language of the source programs by commenting out translated lines.
- LNGclean: Removes all translation annotations from the source files to obtain the original files.

Figure 1 is a scheme of how these programs work together.

There are many ways to implement the above scheme. The one I chose keeps in every source

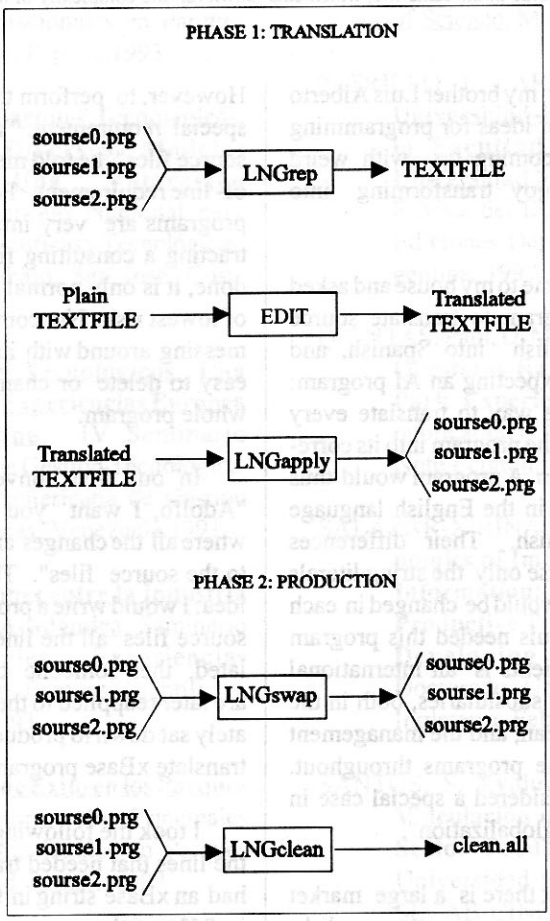


Figure No. 1

file all the program versions for each language. For this, what I did was to comment out in each source file all the translated source lines except the first one. To avoid confusing a translated line with any comment line, I also included an annotation in every translated line to identify it. Let us suppose that the original segment of code is what's shown in Figure 2 under "original source". After each string is translated and all the translations are annotated and included in the source file, this segment of code will look like what's shown under "translated source".

```
========== ORIGINAL source ==========
 xCust = "Enter Customer Code"

========== TRANSLATED source ==========
 xCust = "Enter Customer Code"  && LNG(us)
*xCust = "Código de Cliente"    && LNG(sp)
```

Figure 2

In the example of Figure 2, what happened is that the source line that had a string in need of translation got duplicated in the final source of the program. Within each group of translated lines, the first one is the English version, and the other ones are for each foreign language version.

It's easy to tell apart each group of translated lines, because the first line in a group is always a non-commented-out line, and it always has a language annotation of the form LNG(<language>), where the string <language> is chosen by the programmer. The rest of the lines in a string group are commented out lines, with a different language annotation. This scheme is quite flexible, because all the language versions of a program module can reside in a single source file:

```
 @ 14,20 say "Beautiful!"   && LNG(us)
*@ 14,20 say "¡Precioso!"   && LNG(sp)
*@ 14,20 say "¡Bellissimo!" && LNG(it)
```

The inner workings of the LNGrep system can be best explained using an example. Let's suppose that we need to translate the programs SHORT.prg and ENBOX.prg, shown in Figure 3.

The first step is to invoke LNGrep to obtain a TEXTFILE that contains all the lines that should be translated. For this, LNGrep scans the source files SHORT.prg and ENBOX.prg looking at every line that has either a quote (') or double quote ("). In any xBase language a string is enclosed in either quote character. If directory C:\PRG\SOURCE contains only these two source files, the command line invocation for LNGrep is the following:

```
C:\> cd \prg\source
C:\PRG\SOURCE> LNGrep -add *.prg LNG(us) >TEXTFILE
```

As with many other DOS programs, LNGrep accepts wildcards, as in [*.prg]. It also accepts optional command line arguments, that have the form [-/+]<option>. For example, the three command line options "-add", "+add" and "/add" are equivalent, and tell LNGrep to include a new line with an empty annotation for each of the quoted lines in the source files. TEXTFILE is the name of the file that will contain all the annotated lines. The new contents in TEXTFILE are shown in Figure 4.

```
*: SHORT.prg - Sample program
  xCustNo = " "
  @ 12,20 SAY "Customer Code" get xCustNo
  read
  @ 14,20 SAY "Hello, World!"
  CALL EnBox(xCustNo, 5)
*: EOF: SHORT.prg
```

```
* EnBox.prg
PARAMETERS row, str
* This procedure centers "str" at "row" on the screen
* and surrounds it with a double border box.
* - row < 23 is required

PRIVATE j
j=IIF("=str,'*',SUBSTR(LTRIM(RTRIM(str)),1,75))
* SET EXACT OFF ==> ('abc'='a')=.T. .AND. ('a'='abc')=.F.
IF row <= 23
   @ row-1, (80-LEN(j))/2 - 2  TO ;
      row+1, (80-LEN(j))/2 + LEN(j) + 1 DOUBLE
ENDIF
@ row, (80-LEN(j))/2 SAY j
*: EOF: ENBOX.prg
```

Figure 3

Note that every quoted line appears twice in TEXTFILE, as a result of using option "-add" when invoking LNGrep. The first line contains the empty language annotation LNG(). The second line is the source line as it appears in the source file, but it includes the default annotation, LNG(us) in this case. When options "+add" isn't specified in the command line, then LNGrep will not duplicate lines: this is very useful when running LNGrep on an already annotated source file.

```
*==> File [SHORT.PRG] && LNG(us)
[]-----------------------------------------------------------------[2:1]
 xCustNo = " " && LNG()
 xCustNo = " " && LNG(us)
[ ]----------------------------------------------------------------[3:1]
 @ 12,20 SAY "Customer Code" get xCustNo && LNG()
 @ 12,20 SAY "Customer Code" get xCustNo && LNG(us)
[ ]----------------------------------------------------------------[5:1]
 @ 14,20 SAY "Hello, World!" && LNG()
 @ 14,20 SAY "Hello, World!" && LNG(us)
[ ]----------------------------------------------------------------[0:0]
*==> File [ENBOX.PRG] && LNG(us)
[ ]----------------------------------------------------------------[8:1]
j=IIF("=str,'*',SUBSTR(LTRIM(RTRIM(str)),1,75)) && LNG()
j=IIF("=str,'*',SUBSTR(LTRIM(RTRIM(str)),1,75)) && LNG(us)
[ ]----------------------------------------------------------------[0:0]
```

Figure 4

```
*==> File [SHORT.PRG] && LNG(us)
[]-----------------------------------------------------------------[2:1]
 xCustNo = " " && LNG()
[]-----------------------------------------------------------------[3:1]
 @ 12,20 SAY "Customer Code"     get xCustNo && LNG(us)
 @ 12,20 SAY "Código del Cliente" get xCustNo && LNG(sp)
[]-----------------------------------------------------------------[5:1]
 @ 14,20 SAY "¡Hola, Mundo!" && LNG(sp)
 @ 14,20 SAY "Hello, World!" && LNG(us)
[]-----------------------------------------------------------------[0:0]
*==> File [ENBOX.PRG] && LNG(us)
[]-----------------------------------------------------------------[8:1]
j=IIF("=str,'*',SUBSTR(LTRIM(RTRIM(str)),1,75)) && LNG()
[]-----------------------------------------------------------------[0:0]
```

Figure 5

An annotation is just a comment at the end of a line that tells the LNGrep system which language the line corresponds to. Hence, all the lines that have the LNG(us) annotation are written in the English language; those annotated with LNG(sp) are written in Spanish. As many language versions as required can reside in the same source file (The program limit for LNGrep is 100 languages, which is more than anybody will ever need).

To translate the strings to another language, all the lines in TEXTFILE should be carefully edited. In this example, the result of editing TEXTFILE yields what's shown on Figure 5.

Note that every first line in a string group is edited to obtain its foreign-language translation. Also, the empty annotation LNG() is substituted by the foreign language annotation, which is LNG(sp) in this example.

Some lines that don't need translation are left alone in the string group, and their annotation is LNG().

The LNGgrep programs are smart enough to distinguish correctly each group of lines, allowing for flexible editing. For example, it doesn't matter whether the LNG(us) annotation appears as the first one in the group (as is the case in the "Customer Code" line), or in any other position ("Hello, World!").

Whoever does the translation works only in TEXTFILE, and not the source files. This is very useful because a non programmer can perform the translation, without ever handling the original source files. Also, it makes easier to check the translation, editing or printing TEXTFILE. However, it is very important that the line numbers in the line separators []--------- [nn:mmm] not be changed. These numbers are used by program LNGapply to replace lines in the source files with the translated ones from TEXTFILE.

It is valid to delete a whole group of lines from TEXTFILE altogether, but it is always necessary to avoid deleting any of

```
* : SHORT.prg - Sample program
  xCustNo = " "  && LNG()
  @ 12,20 SAY "Customer Code"      get xCustNo  && LNG(us)
* @ 12,20 SAY "Código del Cliente" get xCustNo  && LNG(sp)
  read
  @ 14,20 SAY "Hello, World!"   && LNG(us)
* @ 14,20 SAY "¡Hola, Mundo!"   && LNG(sp)
  CALL  EnBox(xCustNo, 5)
*: EOF: SHORT.prg
```

```
* EnBox.prg
PARAMETERS row, str
* This procedure centers "str" at "row" on the screen
* and surrounds it with a double border box.
* - row < 23 is required

PRIVATE j
j=IIF("=str,'*',SUBSTR(LTRIM(RTRIM(str)),1,75))  && LNG()
* SET EXACT OFF ==> ('abc'='a')=.T. .AND. ('a'='abc')=.F.
IF row <= 23
   @ row-1,  (80-LEN(j))/2 - 2   TO ;
       row+1, (80-LEN(j))/2 + LEN(j) + 1 DOUBLE
ENDIF
@ row, (80-LEN(j))/2 SAY j
*: EOF: ENBOX.prg
```

Figure 6

the *==> FILE [] lines. The source files should not be changed after TEXTFILE is produced with LNGrep and before they are changed with LNGapply.

Program LNGapply should be used to include the changes in the source file:

```
C:\> cd \prg\source
C:\PRG\SOURCE> LNGapply TEXTFILE LNG(us)
```

After applying the translated lines from TEXTFILE, the contents for SHORT.prg and ENBOX.prg will be what it's shown on Figure 6.

LNGapply doesn't need any wildcard file designators because the source file names appear within TEXTFILE in every FILE line:

```
*==> File [SHORT.PRG] && LNG(us)
*==> File [ENBOX.PRG] && LNG(us)
```

LNGapply scans TEXTFILE until it finds a group of translated lines, as delimited by the line separators []---------[nn:mmm].

The numeric value "nn" is the number of the first line of the group of lines that should

be substituted by those in TEXTFILE. The line numbers "nn" appear in increasing order within TEXTFILE. The value "mmm" is the number of lines that should be removed from the source file. The first time that LNGapply is run, number "mmm" is one (1), because the source files don't have any language annotations yet. There are no line counts for TEXTFILE; only for the source files. For example, the line separator []------[7:2] means to LNGapply to substitute two [nn:2] lines in the source file, beginning at line number seven [7:mmm], by those that appear below the line separator. Any group of lines can have one or more lines.

```
*: SHORT.prg - Sample program
  xCustNo = " "  && LNG()
  @ 12,20 SAY "Código del Cliente" get xCustNo  && LNG(sp)
*@ 12,20 SAY "Customer Code"     get xCustNo  && LNG(us)
 read
  @ 14,20 SAY "¡Hola, Mundo!"  && LNG(sp)
*@ 14,20 SAY "Hello, World!"   && LNG(us)
   CALL  EnBox(xCustNo, 5)
*: EOF: SHORT.prg
```

Figure 7

As expected, there are two lines instead of one for each of the lines that have been translated. This is the result of applying the translations to the source files. The first line has the English language annotation LNG(us), and the second one is the translation of the first one. This later line is commented out, and has the Spanish annotation LNG(sp). In Figure 6 the active language version is English.

Suppose now that it is required to obtain the Spanish version of the source files. This is where program LNGswap is used:

```
C:\> cd \
C:\> LNGswap c:\prg\source\*.prg lng(sp)
```

Figure 7 shows how program LNGswap changed SHORT.prg, to make the Spanish version the current one. Note that LNGswap can work in other directories besides the current. As the

```
C:\> cd \prg\source
C:\PRG\SOURCE> fc short.bak short.prg
ASCII differences between
 c:\prg\source\SHORT.BAK and c:\prg\source\SHORT.PRG

Replace lines 3-4 in c:\tmp\SHORT.BAK
< @ 12,20 SAY "Customer Code"      get xCustNo  && LNG(us)
< *@ 12,20 SAY "Código del Cliente" get xCustNo  && LNG(sp)

with lines 3-4 from c:\tmp\SHORT.PRG
> @ 12,20 SAY "Código del Cliente"  get xCustNo  && LNG(sp)
> *@ 12,20 SAY "Customer Code"      get xCustNo  && LNG(us)

Replace lines 6-7 in c:\tmp\SHORT.BAK
< @ 14,20 SAY "Hello, World!"   && LNG(us)
< *@ 14,20 SAY "¡Hola, Mundo!"  && LNG(sp)

with lines 6-7 from c:\tmp\SHORT.PRG
> @ 14,20 SAY "¡Hola, Mundo!"  && LNG(sp)
> *@ 14,20 SAY "Hello, World!"  && LNG(us)
```

Figure 8

contents of ENBOX.prg are the same for both the English and the Spanish version, LNGswap doesn't change it.

The only difference between this version of SHORT.prg and the previous one in Figure 6, which got saved under the name SHORT.bak by LNGswap, is that in this one every annotated first line has the Spanish language annotation LNG(sp). Figure 8 is the result of running the DOS command FC [File Compare] to show the differences between SHORT.prg (the "LNG(sp)" version) and SHORT.bak (the "LNG(us)" version).

Why didn't LNGswap change file ENBOX.prg? The answer is simple. As it can be seen in Figure 6, the only annotated line in ENBOX.prg is this one:

```
t=IIF("=str,'*',SUBSTR(LTRIM(RTRIM(str)),1,75))&&LNG()
```

The annotation of this line is LNG(), the language-independent annotation. This means that no matter what the current active language is, a line with an LNG() empty annotation will not be changed by LNGswap. Neither will it ever be included in a TEXTFILE by LNGrep.

A (minor) nuisance of this scheme is that some of the lines get shifted one space to the right. For example, if the source file has the following left-justified line:

```
@ 14,20 say "Hello, World!"
```

then after annotation and translation it becomes the following:

```
 @ 14,20 say "Hello, World!"
*@ 14,20 say "¡Hola, Mundo!"  && LNG(sp)
```

An extra space at the beginning of the line is inserted to make room for the comment character '*'. One way around this is to shift every line in the source file one space to the right. However, it will be easy to find many software project managers that won't agree with this correction.

The last option I added to LNGrep is "-new". The first time LNGrep is run on a set of files, the option "-add" is used to have LNGrep duplicate each line in TEXTFILE. But when maintenance is performed on a module, it is nice to have LNGrep output to TEXTFILE only the newer lines to translate. These are lines that have xBase strings in them, but that don't have a language annotation. When option "/new" is specified in the command line, only strings that don't have language annotations are written into TEXTFILE. Hence, after the very first time, it is usual to invoke LNGrep as follows:

```
C:\> LNGrep c:\prg\source\*.prg lng(sp) -new >TEXTFILE
```

When maintaining the source programs, many times it can become quite cumbersome to examine the annotated source file. Program LNGclean can help here. LNGclean can remove all the annotated lines from a source file, leaving only those for the chosen language. To store in file "short.sp" a cleaned-up Spanish version of

```
*************
* SHORT.PRG *
*************
*: SHORT.prg - Sample program
 xCustNo = " "
 @ 12,20 SAY "Código del Cliente" get xCustNo
 read
 @ 14,20 SAY "¡Hola, Mundo!"
 CALL EnBox(xCustNo, 5)
*: EOF: SHORT.prg
```

Figure 9

SHORT.prg, the following command can be used:

```
C:\> LNGclean c:\prg\source\short.prg lng(sp) >short.sp
```

The result of this command is shown in Figure 9.

The English version of SHORT.prg can be obtained just as easily. Figure 10 shows the result of using the command:

```
C:\> LNGclean c:\prg\source\short.prg lng(us) >short.us
```

that leaves in C:\short.us the English version of SHORT.prg.

LNGclean always writes each file name before its contents, surrounded by an asterisk box. LNGclean can be used to do away with all the annotations, but a batch (.bat) file is needed for that. LNGclean does not require the source file to be in any particular language; it will always extract the version requested in the command line. LNGclean also accepts wildcards.

LNGclean can be used to compare the cleaned-up version of the program with its original source code. If they are different, then a mistake has crept in to the translation process. If not, for sure the original version of the program is intact. The DOS command FC can be used for this, and often it will help to use the "/w" option to have FC ignore whitespace:

```
C:\> cd \prg\source
C:\PRG\SOURCE> LNGclean short.prg lng(us) >short.us
C:\PRG\SOURCE> FC short.prg short.us /w
```

```
*************
* SHORT.PRG *
*************
*: SHORT.prg - Sample program
 xCustNo = " "
 @ 12,20 SAY "Customer Code"   get xCustNo
 read
 @ 14,20 SAY "Hello, World!"
  CALL EnBox(xCustNo, 5)
*: EOF: SHORT.prg
```

Figure 10

A minor amount of error handling has been implemented in the LNGrep programs. For example, LNGswap will use the current active language if it doesn't find the language annotation from the command line in a string group. Suppose that a segment of code contains the following:

```
xCust = "Enter Customer Code" && LNG(us)
```

If it is requested that LNGswap produce the French version for this segment of code, the result will be the following:

```
xCust = "Enter Customer Code" && LNG(fr)
*xCust = "Enter Customer Code" && LNG(us)
```

LNGswap used the English version in place of the French one because it found no French annotation in the string group.

All the programs in the LNGrep-system accept wildcards, where appropriate. When no language annotation is specified, then LNG(us) is used as a default.

ADVANTAGES AND DISADVANTAGES

It is not clear that the LNGrep approach is the best one to translate programs. The main shortcomings of this system are the following:

[A] The source code becomes "stained" with language annotations.

[B] Many lines appear more than once in the source code, which makes it more difficult to read and maintain.

[C] If a line number in the line separators []-----[nn:mmm] is changed or deleted in TEXTFILE, then the result can be a seriously damaged source file.

[D] LNGrep is slow.

[E] Multiple statement lines are not supported.

[F] The source code must be recompiled to obtain each language version, this is, different executables are required for each language version.

Complaint [A] is, to a certain extent, solved with the use of program LNGclean. The following commands can do the trick:

```
C:\> LNGswap  c:\prg\source\*.prg lng(us)
C:\> LNGclean c:\prg\source\*.prg lng(xx) >clean.all
```

The first line makes the current language LNG(us). The second one leaves in file "clean.all"

all cleaned-up sources for language "xx", but as there are no LNG(xx) annotations anywhere, then the one used is the English version that was just made current by LNGswap. Some further work is necessary to manually split "clean.all" to obtain all the original sources.

Is there a way to "clean-up" a source file to work on it, edit it, and then reapply the translations? The answer is no.

Maintenance must be done on the "stained" source file. This is why LNGclean is a solution, to a "certain extent".

```
@echo off
:: BatLNG.bat ==> Intermediate step to LNGclean many files

echo The original files will be copied to directory BAK
if    (%2)==() goto help
if not (%3)==() goto help
if not exist bak\null md bak
LNGclean %1 %2 >E:\RamDisk\t.bak
copy    %1 bak
copy    E:\RamDisk\t.bak  %1
del     E:\RamDisk\t.bak
goto end

:help
echo USAGE BatLNG lngfile[.ext] LNG(language)
:end
```

```
C:> for %a in (*.prg) do call BatLNG %a lng(us)
```

Figure 11

A smart programmer could use programs like DIFF3, capable of finding the difference between two files to modify a third file according to these differences, to work on a copy the source file, edit it, and then make the changes back in the original file:

```
C:\> cd \prg\source
C:\PRG\SOURCE> LNGclean short.prg LNG(us) >tmp.prg
C:\PRG\SOURCE> edit tmp.prg
C:\PRG\SOURCE> compile SC00 tmp.prg
C:\PRG\SOURCE> edit tmp.prg
C:\PRG\SOURCE> diff3 tmp.prg short.prg ...
C:\PRG\SOURCE> copy tmp.prg short.prg
C:\PRG\SOURCE> LNGrep /new short.prg >TEXTFILE
```

This requires, of course, a "most careful" programmer (I was never myself confident on the capabilities of DIFF3).

Figure 11 is the BatLNG.bat file that can be used to clean-up a bunch of files. BatLNG must be invoked from the command prompt using the DOS FOR command, as shown on the lower part of same Figure. BatLNG applies LNGclean to a file, and leaves a backup of the original source in the BAK directory. Even though LNGclean accepts wildcards, it binds everything into a single file.

I minimize both complaints [A] and [B] arguing that most translated literal strings appear in @ GET SAY commands, and those are never difficult to understand.

Complaint [C] is difficult to overcome. It would require heavy machinery to maintain the source code synchronized with the translations. Even though the LNGrep programs have some error detection built into them, any (dumb) operator can create havoc. Those who edit the translation TEXTFILE should be advised to think ahead when global replacements are performed, or when huge blocks of data are deleted.

It is particularly important not to make a numeric global string translation, as in subst(1==>2,noask), because many line separators will fit the pattern. For example, []---[1,1] becomes []--[2,2]! Also, it isn't wise to move around blocks of code. Probably it is easier to require every translator to be a programmer.

Complaint [D] can be overcome with money. Just get a faster computer. RAM is not an issue. It

is cheaper to re program the function LNGtools.Get_Line(), but then a program with a very long line could be damaged. Here "long" means "a line with more than 250 characters".

Complaint [E] cannot be overcome, because xBase languages don't allow comments within statements. For example, the following code will not compile:

```
ALFA =;
     "radio boo-boo" +;    && LNG(sp)
*    "radio gaga"   +;     && LNG(us)
     ALFA + ALFA
```

This is a problem in the language implementation. The only solution I have found is to use intermediate variables:

```
 temp = "radio boo-boo"   && LNG(sp)
*temp = "radio gaga"      && LNG(us)
 ALFA = temp + ALFA + ALFA
```

I know that this isn't good enough, because only a programmer can change the original source code into something that LNGrep can process. Should you find a better trick, please let me know.

Complaint [F] can be overcome with a completely different approach. One way is to use a Resource Compiler to maintain a database with strings that get read at run time; another is to write a program than changes the strings stored in the object file (I believe both approaches have been implemented, at least for the C language). People have come up with other variations on theses ideas, but the LNGrep approach seems cleaner, and far easier to implement. The LNGrep system should be good enough for most small programming shops. Small is beautiful.

The advantages of this approach are the following:

[A] There is only one source code. With Source Control Systems, such as Tlib or PVCS, it is possible to maintain several versions of a program. However, the fewer files a system has the easier it is to manipulate, and this is the approach supported by LNGrep. To obtain a different language version you just run LNGswap on the source code and recompile.

[B] The original source code is not manipulated by the person that does the translation. Often those who can translate neatly cannot manipulate directly the source files. The LNGrep systems allows the division of labor that reduces the cost of the translation.

[C] It's easy to know what has been translated. Just run LNGrep without option "+new" on the original source file to get a complete TEXTFILE that contains all the translated lines. An alternative is to use GREP to extract every line that has an annotation:

```
C:\PRG\SOURCE> GREP -n+ -i LNG( *.prg
```

(Note the absence of the closing parenthesis ")").

[D] Lines that need no translation can be annotated with LNG(). This prevents many errors that otherwise could creep in when manipulating a lot of data.

[E] LNGrep is free!

There are more advantages to the LNGrep system than disadvantages, which means that you should start using it right away. Should you find a bug, just contact me and I will try to fix the problem.

As usual, the use of the LNGrep system is at your own risk!

IMPLEMENTATION

I implemented this programs using the Borland Turbo Pascal programming language. I chose Turbo because it's a language that I have mastered through the years. For my programming courses at the University I use Turbo. Turbo was natural for me to use, but other languages are just as valid for a project like this.

The LNGrep system consists of four programs. The source code for each program is in a file that has the same name as the program:

- LNGapply.pas <==> LNGapply.exe
- LNGrep.pas <==> LNGrep.exe
- LNGswap.pas <==> LNGswap.exe
- LNGclean.pas <==> LNGclean.exe

All the programs share a library of tools that I developed for this project called LNGtools.pas. The important components of LNGtools are the following:

<=> TYPE Tokenized_T, used to parse all input lines that contain an xBase string.

<=> OBJECT TString_Group, used to store all the input lines that belong to the same string group. In every program, the variable SG contains all the lines in the current string group.

<=> String handling routines, like RTrim(STRING):STRING, are needed to joggle input lines.

<=> Tokenize() is as procedure that takes apart input lines and figures out whether they have an LNG() annotation, and whether or not they are a comment line.

<=> Command_Line() is the procedure used to parse the command line argument. It relies a lot on System.ParamStr().

Each of the programs are pretty straight forward. There is a main loop where each program line gets read, until the end of file is reached. For each line, it is decided whether it belongs to the current string group, in which case it gets added to it, or if it belongs to a new group, then the old one is processed out and a new one is begun. Some care must be taken because there are several cases to be handled, but there is nothing special in that code.

I have tried to document fairly well the code, just in case you need to understand it. The only part where I was forced to use OOP is when I implemented OBJECT TString_Group, because using regular procedural programming there would had forced me to invent many unnecessary names. All the programs can be compiled with Turbo Pascal v5.5, or a later version.

CONCLUSION

As economies become more dependent on each other, the language barrier becomes more of a problem. Systems like LNGrep can help to overcome this problem, and increase productivity throughout.

ACKNOWLEDGMENTS

My brother Luis Di Mare came up with the idea of LNGrep. Later, my friends Enrique Bermúdez and Joseph Bannister provided valuable criticism and editorial suggestions. Lastly, this document was written using Eric Meyer's superb editor VDE.com, which he distributes as shareware. For most purposes, it is better than the commercially available mega-monsters.

BIBLIOGRAPHY

[1] Castillo, JoAn: "ANSI Standards Committee makes progress"; Data Based Advisor; July 1993; pg-76.

[2] Petzold, Charles: "Unicode, Wide Characters, and C"; PC Magazine; Vol 12 #3; Nov-9, 1993; pg 369.

[3] Plauger, P. J.: "The header <locale.h>"; The C Users Journal; vol 9 #3; March 1991; pg 7.

[4] Plauger, P. J.: "The Standard C Library"; Prentice Hall; 1991.