THANKS FOR ALL THE FISH

The Web-based Babel Fish translation tool is no linguistic genius, but it can help you to understand foreign language files. In this feature we'll explain how to create a command line tool that interfaces with it over an Internet link, allowing you to translate local files in a matter of seconds.

If you click on the *Translate* link on *altavista.com* you'll find yourself faced with nothing more complex than a Web front end to *Babel Fish*, a program that understands half a dozen languages and can translate words, complete sentences and even entire Web pages. As the art of machine translation is still in its infancy even after decades of development, *Babel Fish* can't work miracles. Still, for if you only need to say "How do you do?" or "Give me a beer, please", in another language, or if you want to get the gist of a foreign-language Web site, it can be very helpful.

The *Babel Fish* Web form on *Altavista* (accessible directly via *babel.altavista.com*) allows you to type text directly into a dialog box, or to enter a Web page URL, which will then be translated for you. This is all very well, but wouldn't it be good if you were able to translate complete documents in a local file with the minimum of fuss, with no re-typing, or uploading to a Web site being required? We certainly thought so, so in this Know How feature we'll create *trans*, a Perl script that links with *Babel Fish* via the Internet and allows you to translate the contents of local files with very little effort.

As you might expect, a suitable Perl module that can be used by our script already exists; *WWW::Babelfish* by *Dan Urist*, which can be found on CPAN. As well as many other functions, *WWW::Babelfish* cleverly avoids the 1000 character limit imposed by *Bablefish* by splitting text into chunks of less than 1000 characters, then sending then individually to the Web site.

Multilingual Agent

Babelfish currently supports conversions in both directions between English and either French, German, Italian, Portuguese, Spanish or Russian.

Calling *trans* without any parameters shows which parameters are normally expected:

```
usage: trans \
    [efgpirs]2[efgpirs] file ...
    e: English
    f: French
    g: German
    i: Italian

    p: Portuguese
    r: Russian
    s: Spanish
```

In order for the *trans* script to know from and into which language it is meant to translate, the first command line parameter indicates the direction: *e2g* (*English-German*) translates from *English* to *German*, *f2e* (*French-English*) translates from *French* to *English*, for example.

The text to be translated is contained in one or more files, the names of which follow as parameters. The following call, foe example, would translate the French content of the file */tmp/french.txt* into English, and then output the result via the standard output:

```
$ trans f2e /tmp/french.txt
```

Following the old Unix tradition, it is also possible to omit the file name, in which case *trans* retrieves the data from the standard input:

```
$ echo "Der Ball ist rund" | trans g2e
The ball is round
```

In the script shown in Listing 1, you'll note that *trans* accesses the *WWW::Babelfish* module in line 3. This must have previously been downloaded and installed:

WWW::Babelfish is available on CPAN under

```
WWW-Babelfish-0.09.tar.gz
```

and requires the *libwww* bundle as well as the module *IO::String*.

As always, the installation uses the CPAN shell and

```
perl -MCPAN -eshell
cpan> install libwww
cpan> install IO::String
cpan> install WWW::Babelfish
```

he supported languages are in lines 10 and 11. The list definition operator *qw* delimits the enclosed string at the word boundaries (spaces and linefeeds) and returns a list that contains every word as an element.

In order to be able to access the complete language names (e.g. *English*) via the abbreviations (e.g. *e*) in an elegant manner later on, lines 15 to 18 build a hash table *%i2full*, which contains the abbreviations as keys and the language names as values. For this, the function *substr* takes the first letter in any language name and the function *lc* converts it into lower case.

Line 21 assembles all available abbreviations into a string *$chars* to be used later on. This string is

Listing 1: trans

```
01 #!/usr/bin/perl -w
02
03 use WWW::Babelfish;
04
05    # Dummy UserAgent
06 use constant AGENT =>
07              'Mozilla/4.73 [en] (X11; U; Linux)';
08
09    # Supported Languages
10 my @languages = qw(English French German Italian
11                    Portuguese Russian Spanish);
12
13    # Build hash that assigns language abbreviations
14    # to languages (e=>English, g=>German, ...)
15 foreach my $language (@languages) {
16    my $initial = substr($language, 0, 1);
17    $i2full{lc($initial)} = $language;
18 }
19
20    # All abbreviations in one string (efgpirs)
21 my $chars = join '', keys %i2full;
22
23    # Conversion direction from the
24    # command line (g2e, e2f, ...)
25 my $way  = shift;
26
27 usage() unless defined $way;
28
29 usage("Scheme $way not supported") unless
30  ($from, $to) = $way =~ /^([$chars])2([$chars])$/;
31
32    # Read in text to be translated
33 my $data = join '', <>;
34
35    # Contact Babelfish
36 my $babel = WWW::Babelfish->new(agent => AGENT);
37 usage("Cannot connect to Babelfish") unless
38    defined $babel;
39
40    # Perform translation
41 my $transtext = $babel->translate(
42    source      => $i2full{$from},
43    destination => $i2full{$to},
44    text        => $data
45 );
46
47 die("Error: " . $babel->error) unless
48    defined($transtext);
49
50 print $transtext, "\n";
51
52 ####################################################
53 sub usage {
54 ####################################################
55    my $msg  = shift;
56    my $prog = $0;
57
58    print "usage: $prog ",
59          "[${chars}]2[${chars}] file ...\n";
60    foreach $c (sort split //, $chars) {
61        print "  $c: $i2full{$c}\n";
62    }
63    exit(1);
64 }
```

confined to the current package using *my*, but is also available in the subfunction *usage*.

$way in line 25 uses *shift* to retrieve the first command line parameter, which indicates the direction of the translation. If no parameter is present, the user has obviously not understood the syntax of *trans*, so the function *usage* provides some operating instructions and aborts the program.

The regular expression */^([$chars])2([$chars])$/;* in line 30 interpolates with */^([efgpirs])2([efgpirs])$/;* and checks whether the direction indicator conforms to the format *x2y*, where *x* and *y* assume the value of *e*, *f*, *g*, *p*, *i*, *r* or *s*. Since the expression is used in a list context and there is a list to the left with the elements *$from* and *$to*, after a successful match this will contain the values within the brackets of the regular expression. For *g2e* this would be *g* in *$from* and *e* in *$to*. If the match fails, however, the result is an empty list which is interpreted as *false* within the Boolean context of *unless*.

Line 33 reads in the text to be translated, either from files named in the command line or from the standard input if no files names are specified. The *join* function joins the lines into a long string while obviously retaining the linefeeds.

The Babelfish Object

Line 36 creates a new *WWW:Babelfish* object and instructs it (via the *agent* parameter pair) to pass itself off as a Netscape browser using the constant specified in line 6 with Perl's *use constant*. This makes it possible to define functions which look like macros, and that are optimised by Perl in such a way that they are by no means inferior to constant scalars.

According to its documentation, *WWW:Babelfish* returns the value *undef* if anything goes wrong, which line 37 would take as a cue to abort.

Finally, line 41 sends all data to *Babel Fish* on the Web. The *translate* method receives the full (English) names for source and destination language (parameter names *source* and *destination*), as well as the text to be translated in the form of a string value for the *text* parameter.

The object sends the data to the form, interprets the returned HTML and extracts the result from it, all without any intervention from the user. The result is contained in *$transtext*. A value of *undef* indicates an error, which is captured by line 47 and displayed as a message using the *WWW::Babelfish* object's *error* method. Lastly, line 50 sends the result to the standard output.

Operating Instructions

In order to make it easy for new users to learn the operation of *trans*, the *usage* function defined from line 53 outputs a message and then brief operating instructions. Afterwards, the program is terminated using *exit(1)*.

The operating instructions are generated dynamically by *usage* from the content of the variables *$chars* and *%short*, which contain the valid abbreviations and a table listing abbreviations with their corresponding language names.

The Perl function *split* in line 60 splits strings into their components using *//* as a pattern, and returns an array which contains every character as an element. The function *sort* sequences the array of lower case letters alphabetically and the hash *%short* provides the corresponding language names.

The sequence of supported languages could also be retrieved using the *languages()* method of the *WWW::Babelfish* object, which returns an array containing all current languages. *trans* does not do this, however, as the range is relatively static.

Great Moments in Translation

We thought you'd like to see few examples of Babelfish's translation capabilities. In all cases we called *trans* with only the language direction parameter, then we entered the text to be translated via the standard input and finished with *^D* (Control+D):

```
$ trans g2e
Einen Radi und eine Mass
Bier, aber schnell!
^D


A Radi and measure beer, but fast!
```

Not too bad, but by no means perfect. Let's try English to French instead:

```
$ trans e2f
waiter, a bottle of your
finest English wine please!
^D


serveur, une bouteille de votre
vin anglais plus fin s'il vous pla%t!
```

Or English to German:

```
$ trans e2g
waiter, this beer
tastes terrible!
^D


Kellner, dieses Bier
schmeckt schrecklich!
```

So, there you go. It works. We have to warn you that more complex translations are handled less well, though. Technical information in particular tends to get a bit mangled. It can still help you understand documents that you'd normally have to send off to a translator for interpretation at great cost, however, and all with the greatest of ease. ■

Michael Schilli

works as a Web engineer for AOL/Netscape in Mountain View, California. He is the author of „GoTo Perl 5", published in 1998 by Addison-Wesley (and in 1999 as „Perl Power" for the English-speaking market). Homepage: http://perlmeister.com.