FURCH software, detailed user guide.

Version 2.011

1. Table

Detailed guide, introduction.

FURCH is a software to display and analyse texts. If you want informations on a word, you will point and click the mouse on that word; The software is able to search dictionaries for that word. If it finds something, the information will be displayed. FURCH is a graphical application, it uses windows and more generally windgets, which are kinds of window gadgets we are now used to see in graphic applications and user interfaces. FURCH 2.011 is built on GTK2 and is developed on Linux. However, the software has been ported on Win 98 SE, Win 2000 and Win XP.

The FURCH software has currently two configuration files. The first one is " furch.conf ".and the second one is " gtkb.rc ". The second file is more a GTK configuration file than a FURCH file, however. Both files must be put into the same directory than the executable and they are used to tailor the sizes, colors and other features of the windows and buttons. They are also used to tell the software what text files to read and what dictionaries are available. Both files will be analyzed below. We must also give details on the two dictionary norms in use with FURCH at present time. Some informations on the installation of the FURCH software on Linux and on Windows OSs have also been added.

The text files.

The FURCH software can only analyse iso-latin1 files, that is to say ordinary uncoded text. Each file is considered to be a book which may be composed of many pages and even, sometimes, a single page. These files may contain special tags to indicate the separations into pages and also page headers and footers if they are present. However, this is not necessary and if there is no tag, the software will do its very best to display the file into readable pages and paragraphs. Still, the original file will not be changed. You can display texts generated by a word processor such as MS-Word or OpenOffice which contain very long lines as well as texts coming from a more simple word editor such as Notepad or Vim. These later texts have lines of a more reasonable length. We will see now the case of files containing tagged text.

Tagged text files.

The file may contain tags which are indications to the software on how to format the text. In its present state, the software understand only 8 tags. These tags are lines beginning by : .NPO , .NHO , .NTO , .NFO or .NP , .NH , .NT , .NF

Text files with strict format.

You may want to display on your computer pages of a book in such a way that what you have on your screen is as close to the original as possible. Then, you must use

the tags .NPO , .NHO , .NTO , .NFO. The letter « O » is standing here for « Original », « .NPO » means New Page Original Format, « .NHO » means New Header Original Format, « .NTO » means New Text Original and « .NFO » has been put here for New Footer Original Format. The two orders « .NPO » and « .NHO » induce a jump to the next page. So, when there is a header on the page, you will only need the order « .NHO » and you can omit « .NPO ». With these orders the length of your lines will be the same on your computer screen than on your file.

Text files with page tags.

You may want to display on your computer text files already cut into pages but in such a way that the lines on your screen will have at most a determined number of characters. For that, you must use the tags .NP , .NH , .NT ,.NF. The order « .NP » means New Page, « .NH » means New Header, « .NT » means New Text and « .NF » has been put here for New Footer . The two orders « .NP » and « .NH » induce a jump to the next page. So, when there is a header on the page, you will only need the order « .NH » and you can omit « .NP ». With these orders the length of your lines will be close to the length indicated on your « furch.conf » file. With this format you must indicate the beginning or the end of a paragraph by a blanc line.

The “ furch.conf “ file.

The “ furch.conf “ file is in fact a script file which uses its own special language. Don't panic, everything is explained by comments in the file. A comment is a line beginning by the sign “ # “, these lines are simply ignored by the software. The script language is composed of orders such as : **Books, Which_dico, Highlight** etc. Generally, each order has attributes whose list is inserted between the signs “ **:>** “ and “ **<:** “.

The order : Books.

Let us begin by the order **Books**. This order is used to tell the software what files it must analyze. Each file is considered to be a book which may be composed of many pages or perhaps a single one. The file must contain only iso latin1 text. When the text is not formatted, the software will display it at best. In any case, each displayed page will have less than 100 lines and 1000 words in them. You must also tell to the software where it can find these files, so you must give it the absolute or relative path of each file. As you may have a collection of books, the software understands notations like file5->15, telling him that it must consider file5, file6, ..., file14, file15. Though it is not necessary to give them the extension “ .txt “, each file must be a text file. Here is an example extracted from a “ furch.conf “ file :

```
#
#   list of books to read ( 1 file is a book which can contain many pages)
#   you can use the -> sign to simplify your inputs
#   example : qr_4->6.txt means : qr_4.txt qr_5.txt qr_6.txt
```

```
#
Books :>
#  essai_h.txt
   ./gour.txt
   ./kentskrid-1.txt
   ./kentskrid.txt
   ./sarmoniou1-quere.txt
   ./Bremaik-09-juillet-2007.txt
   ./ALES/ales-1->14.txt
  <:
#
```

On the above example the file " essai_h.txt " will be ignored. The first book to be analyzed will be " gour.txt ". It is in fact a single page and is considered as a one page book. It is located into the same directory than the FURCH executable. Then we have a few other books. There is also in the software directory another directory whose name is "ALES" wich contains fourteen files "ales-1.txt" to "ales-14.txt". They are in fact single page files which were used with the FURCH previous versions. They will be considered as single page books and displayed as usual.

Remark 1 : If a file doesn't exist or if you have given a bad location, the file will be ignored.

The order : Which_dico.

One of the most important features of the FURCH software is that it can handle multi dictionaries. We must tell it which dictionaries to use and where to find them. This is done like for the text files, but this time the order is " Which_dico ". Here is one example :

```
Which_dico :>
   ./dico1->3_quere
    ./roll1_kltg
    ./splf1_ex
   <:
```

On this example we are asking for the dictionaries " dico1_quere ", " dico2_quere ", " dico3_quere ", " roll1_kltg " and " splf1_ex ". They are located into the same directory than the FURCH executable. The " dico_quere " and " roll1_kltg " dictionaries are full norm dictionaries (regular norm dictionaries). The " splf1_ex "

dictionary is a simplified norm one. Nothing distinguish them in the “Which_dico” order.

Remark 1 : You can use the notation “ **->** “.

Remark 2 : The “ dico3_quere “ dictionary has not been written yet. It will simply be ignored.

Remark 3 : The order of the “ Which_dico “ list is important. It is in fact your priority list for the dictionaries. When you look for a word, the dictionaries where this word can be find are classed according to this list. Among these dictionaries, the content of the one with the best rank on the priority list will be displayed. You can however reach all the informations in all the dictionaries where your word was found. This has already been explained into the software presentation.

The order : Priority_search_dico.

```
Priority_search_dico :>  i  <:
```

You must replace **i** by 0 or 1. If you put 0 then the order of priority for your dictionaries is the order you have implicitly defined with your “ **Which_dico** “ list. If you replace **i** by 1 , the priority is weight, that is to say the dictionary with the more information will be at the first place. This priority may vary from word to word, the same dictionary doesn't have always more information on each word. Here also the information present in any the dictionary may be reached, of course.

The order : Order_dico.

Example :

```
Order_dico :>
        F         B         E
<:
```

You can choose the display order of the dictionaries into the dictionary window. For example, I have on all the figures chosen to display the Breton part in the middle, with the French to its left and the English to its right. This is easier for the debugging of the dictionaries. You can modify that choice by changing the arguments of the order “ **Order_dico** “. For example you can put the Breton first, then the English and at last the French by giving the argument :

```
   :>     B         E         F      <:
```

At the present time, there are only 3 languages in the FURCH software and these languages must be Breton, French and English. This is a serious limitation. I hope this will be improved in a future release.

The order : Level_dico.

For example :

```
Level_dico :>     3     3     3     <:
```

As it was already said the regular norm for the dictionaries has display levels. You may select the maximum display level if you want. The order "Level_dico" allows you to adjust the volume of details written for each of the 3 languages on the dictionary window.

Level -1 will simply suppress the output for the given language.
Level 0 gives only the basics. The word, its transcription in KLTG for the Breton part, the grammatical nature of the word.
For the French and English parts you have the translation of the Breton word.
Level 1 explains the word meaning.
Level 2 gives more explanations, generally grammatical ones.
Level 3 exists sometimes and it is often more chattering than useful things.

You must give a level of output for each language. So, you must give 3 numbers, but in what order ? Of course, in the order you have already defined with " **Order_dico** ".You don't have necessarily the same level of output for each language. You may for example choose to display the Breton in full extent and nothing for the other languages, you will then write :

```
Order_dico :>     B         E         F      <:
Level_dico :>     3     -1      -1     <:
```

Remark : The display level is irrelevant for a simplified norm dictionary. It will simply be ignored.

The order : Highlight.

When you are displaying a HTML document or a text with a word processor, it is often useful to highlight some particular words or multi word expressions. It would be nice for us to be able to do the same thing into our book part. The order " **Highlight** " gives us this possibility. However words vary, a noun may be at the singular or plural, a verb is conjugated. FURCH will do its best to take into account all these variations. I am aware that at present time FURCH does not do a perfect job. Be patient.
How to use the order " **Highlight** " ?

Let us take one example, you want to highlight into your texts the word “ pleg “ (fold), the verb “ kozeal “ (to talk) and the expression with two words in it : “ en em “ (the reflexive pronoun). You will thus write :

```
Highlight :>
     pleg
     kozeal
     koze
     en em
          <:
```

You must put only one word or one expression on a line otherwise the software will take what you have written as a compound expression and it will try to recognize the complete expression into your texts.
With what is written above, the software will recognize the word “ pleg “, the plural “ plegoù “ and the word with a mutation “ daou bleg “ (two folds). However, the past participle “ pleget “ (folded, bended) and conjugations like “ plegomp “ ([we] bend, [we] fold) will also be recognized. This is because “ pleg “ is also the stem of the verb “ plegañ “ (to fold, to bend). You may not want to have the conjugations of the verb however and you may avoid that by specifying that the highlighted “ pleg “ must be a noun. This is done by writing “(ag)” on the same line : “ pleg (ag) “. The (ag) means “ anv gourel “ (masculine nouns). All the available orders like (ag) may be found, for example, at the beginning of the “ dico1_quere “ dictionary. The capability to precise the grammatical value of the words has however been poorly tested.
Let us continue with our list of highlighted words. We want the verb “ kozeal “ and all its conjugations. For that, we must not only give “ kozeal “, but also its stem “ koze “ otherwise a word like “ kozeomp “ ([we] talk) will not be recognized.
The expression “ en em “ will be recognized everywhere without difficulty because it does not vary. At present time, the software is not able to recognize expressions with variable parts (verbs conjugated, different pronouns etc ...)

The order : Width_pix_dico.

We must define the size of the different windows presented in the first part of this tutorial. The GTK2 software does allow adjustments by a click and drag process, so we can adjust the book and dictionary windows. As we have already seen, the dictionary window is divided into 3 sub-windows, one for each language, French, Breton, English (in the order you have already chosen by **`Order_dico`**). You must choose the size of each sub-window. For example :

```
Width_pix_dico :>
 260     260     260   <:
```

If, as above, the sum of the sub-window sizes is superior to the size of the total dictionary window, then a horizontal ruler is automatically created.
Remark : It is not necessary to have 3 equal numbers for the parameters of the order " **Width_pix_dico** "

The order : Color_i_dico. (with 0 ≤ i ≤ 4)

As you have seen in the presentation guide, 5 background colors have been used when displaying the lines of the dictionaries. A white background is for an ordinary line. A very light green background is used each time we have a new main paragraph, that is to say each time we have a new word or a new grammatical nature for a given word. A very light blue background is used as a less important separation. By insertion of white lines, the green and blue lines are at the same horizontal level in all the columns. With that trick, it is easier to go from one language to the next. A very light pink background line is used for a mouse clickable line. These lines are used to navigate into the database. The corresponding pink lines may not be at the same level for the different languages however. The last colour, yellow, is used when you have words defined as secondary entries in the dictionaries. To distinguish them from the other words, their first line is in yellow. As they can be anywhere inside the displayed article, the lines where they are found are also coloured into yellow. Usually these default colors are easily distinguishable, however you may want to adjust them to your own perception of the colors. In one occasion I have also been obliged to darken these backgrounds on a portable computer with a very low contrast screen. The " Color_0_dico ", " Color_1_dico ", " Color_2_dico ", " Color_3_dico " and " Color_4_dico " orders can be used to adjust the background color of the dictionary lines.
Though I was using the words white, very light blue, green, pink or yellow, you can give to these lines any background color you want by adjustment of the red, green and blue components of the color. These three numbers can be given by their hexadecimal values (from 0x0000 to 0xffff) or simply by their ordinary decimal values (from 0 to 65535). For the hexadecimal notation, the " 0x " is simply an indication to the software to take the number as hexadecimal. Below, you have an example of the use of these five orders for the definition of the colors to the default values white, light blue, light green, light pink and yellow.

```
#       Color_0_dico = white  (for an ordinary line)
#       Color_1_dico = very light blue  (first line of a
#                      paragraph)
#       Color_2_dico = very light green  (first line of a
#                      new entry word,
```

```
#                       first line for a new meaning of the
#                       word)
#       Color_3_dico = very light pink  (for a
#                       mouse clickable line )
#       Color_4_dico = yellow for words defined as secondary
entries.
#
#

#                     RED       GREEN    BLUE
Color_0_dico :>   0xffff    0xffff    0xffff  <:
Color_1_dico :>   0xdddd    0xdddd    0xffff  <:
Color_2_dico :>   0xdddd    0xffff    0xdddd  <:
Color_3_dico :>   0xffff    0xdddd    0xdddd  <:
Color_4_dico :>   0xffff    0xffff    0x5555  <:
#
#
```

The order : Font_dico.

To say the truth, the definition of the fonts is a very murky area, at least for me. What I will say will be however sufficient for you to choose a font for the dictionary and the book windows. The font for the dictionary window is chosen in the " furch.conf " file by the order " Font_dico ". For the book window, the fonts (we may have more than one font) are determined into the " gtkb.rc " file. This will be explained later.
We must distinguish two different cases, for Linux and for a MS Windows operating system.
For Linux. For Linux, you must choose a font which is into your system. For that you have two commands :" xlsfonts " and " xfontsel ".
In a console window, the command " **xlsfonts > file_name** " will list all the fonts available into your Linux system. Their names will be written into the file " file_name ". You can read this file with any text editor such as vi, vim, gvim, emacs, etc. The command " **xfontsel** " starts a small program that is able to display the fonts. Most fonts on Linux are not scalable, that is to say, you cannot make their size vary in a continuous fashion. You have only a few discrete sizes available, for example, on my computer for the font " courier " I have the following sizes :

8,10,11,12,14,17 **->** 26,34

So, sizes 13, 15, 16, 27, 28 .... does not exist. If you ask for one of the lacking sizes, the program will automatically revert to its default font without any warning. This

may be rather confusing because you will not have a continuous behavior of your fonts appearance.
Fonts are also “ variable width “ or “ fixed width “ ones. The most current fixed width fonts and sizes are :

```
# * lucidatypewriter  8,10,11,12,14,17 -> 26,34
# * courier           8,10,11,12,14,17 -> 26,34
# * fixed             12 -> 16,18,24
```

If you want to select the first font with size 17, you will write the line :

```
Font_dico :>   Lucida bald 17   <:
```

Into Linux you have also variable width fonts. Some very current ones are :

```
# *
# * helvetica         8,10,11,12,14,17,18,20,24,25,34
# * lucida            8,10,11,12,14,17,18,19,20,24,25,26,34
#
#Example : Font_dico :>   Helvetica bald 18   <:
```

In order to use the helvetica font you can insert the following line into the “ furch.conf “ file :

```
#
Font_dico :>   Helvetica bald 20   <:
```

You can try : medium, normal, bold, italic.

For Windows. At present time, there is no difference between Linux and Windows for the FURCH software.

The orders : Horiz_pix_total, Verti_pix_total.

As was said earlier, version 2.011 of the FURCH software has adjustable windows. However, the default size of the main window is set in the furch.conf file by the orders Horiz_pix_total and Verti_pix_total.
For example :

```
#
  Horiz_pix_total :>   1200  <:
  Verti_pix_total :>   850  <:
#
```

Of course, you must choose the fonts into your book window in agreement with the size you have imposed to your window. If the lines are too long or if you have too many line you will not be able to display your whole page but horizontal and vertical lift rulers will automatically appear and they will allow you to position at will the page you are displaying.
The fonts which are used into the book window are determined into the " gtkb.rc " configuration file.

The order : Width_pix_book.

Once you have defined the the size of the main window, you still need to tell the software what part to attribute to the book window and to the dictionary window. This can be done by giving the width in pixels of the book window. The difference between the width of the main window and the width of the book window will automatically be attributed to the dictionary window.
The default width of the book window can be modified by the order : "Width_pix_book" :

```
#
#  Number of pixels for the width of the window
# displaying the book part
#        example : Horiz_pix_total/2
#
    Width_pix_book :> 580 <:
#
```

As it was said into the presentation guide, what is given in the furch.conf file is only default sizes. You can adjust these sizes by dragging the corners and sides of the windows. If you quit the application by pressing on the "Quit" button, your settings will be registered into the file "furch_last_state". This file is a simple text file, you can open it, there you will find lines like the following :

```
1566   934    Size of main window width height
688    Width of book window
```

These two lines are very explicit, they give you the width in pixels of the main window (here 1566), the height (here 934) and the width of the book window (here 688) when you were leaving the application last time. These values can be very useful because it is enough for you to adjust your windows once, then to quit the application by pressing the "Quit" button, take these values in the "furch_last_state" file and modify accordingly the "furch.conf" file. Then, even if you quit the application by destroying the main window, you will have good default values for your window sizes.

The orders : Nb_char_line, Nb_line_by_page.

These orders are used to give default values for the number of characters by line and

the number of lines by page. These values will be used whenever possible. Of course, they cannot be used when we have a book in the original format so, they are overlooked with the tags ".NPO , .NHO , .NTO , .NFO". When we have a formated text with tags ".NP , .NH , .NT , .NF", the number of characters by line will be taken into account. When we have an unformatted text, both default values will be used.

Here is an example for the use of these orders :

```
#
#    Nb_char_by_line = Number of characters by line :
# This parameter is used to display texte
# whenever possible
      Nb_char_line  :>  53  <:
#
#    Nb_line_by_page = Number of lines by page :
#  This parameter is used to display texte
#  whenever possible
      Nb_line_by_page  :>  32  <:
```

Remark : It would have been better to let the software determine the number of characters by line and the number of lines by page but the programming for that purpose would have been difficult. Something of that kind as already been done for the dictionary windows, however.

The “ gtkb.rc “ file.

The “ gtkb.rc “ file is in fact a Gtk configuration file, so we have to use the Gtk orders.

Into the book window, the words are labeled buttons. The font color may be black for ordinary words, blue for expressions or red for not recognized words. There is also a green font color which is not used any more but which has been retained for compatibility with FURCH version 1. Each word, in fact each button, is clickable and its background color changes when the mouse cursor is on it. This is to indicate that it is ready to be clicked. We can also choose to permanently highlight a word to have it easily distinguishable in the page. This is done by giving a light green background to the word. All these features are possible by the definition of buttons with different styles. For example, for the red family, we will have :

```
style "button_red"
style "button_red_up"
style "button_red_hlight"
style "button_red_hlight_up"
#
```

And we have the same 4 styles for the blue, green and black (noir) families. For each button we must also define the font. It is simpler to take the same font everywhere but this is not mandatory. You may choose, in the book part, a size of letters larger than the one you have in your dictionary window. For each style the background (bg) and the foreground (fg) colors will also be defined. The colors are given by their red, green and blue components. Each component is a number whose range is from 0.0 to 1.0. For example, we have for the style “button_red” :

```
#
style "button_red"
{
   bg[NORMAL]= {0.8 ,0.8 ,1.0  }
   fg[NORMAL]= {1.0 ,0.0 ,0.0  }
     font_name = "Helvetica Bald 17"
}
#
```

The black (noir) button is special as it is used, not only for words on the page, but also for the command buttons on the lower part of the book page. This is why it has, not only a NORMAL state but also PRELIGHT, ACTIVE and INSENSITIVE states.

The names such as "button_red" are only references for software coding, you can of course take any color you want and you can choose blue letters for your red button if you prefer.The same is true for the other button families such as the family of blue

buttons.

You can have all your writings in black if you prefer not to distinguish expressions from the other words. A difference of color between the single words and the compound expression is not necessary any more in FURCH version 2 because, if you have your mouse pointer on one of the words forming the expression, this one will be entirely highlighted. That was not true in version 1.

The dictionary norms.

An important problem for FURCH is to display the information into the dictionary window in the most possible pleasant and agreeable way. As we have seen, we can have, in the present software release, up to 3 columns, one for each recognized language. The user is free to set up for each column its width in pixels and the number of characters will be determined by the software. You have also to divide your text into paragraphs. This situation is very similar to the one you can encounter when you are displaying a web document. For the web, the solution was to devise the HTML language and write softwares for that language. We will take exactly the same approach, devise languages (which I am calling dictionary norms) and conceive softwares to display the dictionaries written with these languages. These languages are like HTML, languages with flags, but they are much more simple because the problem not so complex and also because we cannot expect that the general user will learn still another difficult language. These are the reasons why the " regular norm " was devised for writing dictionaries. The dictionaries are in plain text with flags and there is no alphabetic ordering of the entries. The flags indicate the languages, the separations between paragraphs, the reading level, etc. However it soon became evident that even if it is not too difficult to write a regular norm dictionary from scratch, it is still a lot of work to transform an existing list of words into a dictionary readable by the FURCH software. It seems, speaking casually with people, that many who are studying Breton have made electronically readable lists of words for their own use. It would be nice if they could use their lists with FURCH without too much work and even better, if they could allow other people to use these lists. This is why the " simplified norm " was devised. Roughly speaking the author of a simplified norm dictionary will simply have to introduce a word or expression by a " # " character at the beginning of a line, and put after the word or expression a ponctuation sign or a jump to a new line. After that, he will put the explanations up to the next " # ". With the " simplified norm ", any existing dictionary can thus become usable with a minimum of modifications. Detailed explanations on both norms will be given below but we must first speak of the accentuated or special characters.

The problem of accentuated or special characters.

Both French and Breton have accentuated or special characters. For example :

“ àñïçæùœéèë ... “ and the corresponding capital characters “ ÀÑÏÇÆÙŒÉÈË ... “.

You must be able to indicate to the software that it must use one such character but the specified character must also exist into the font you use to display texts. In fact all the characters we need for French and Breton are in any standard latin1 font except perhaps for the French “ œ “ and its corresponding capital character “ Œ “. Often you are able to select such characters on your keyboard by striking a single key or a combination of keys. This depends from your OS settings and from the word processor you use. All these accentuated letters will be recognized by FURCH and displayed correctly into the book window or the dictionary window. However, depending of your software, you may not be able to obtain some characters by striking a combination of keys. Then, you will use key sequences beginning by the escape character “ \ “ and followed by an accent such as “ ' “ and a letter such as “ u “. The software will understand and write “ ù “. You can mix both notations if you want. So the name “ Éloïse Salaün “ can also be written :

“ Élo\:ise Sala\:un “ , “ \'Eloïse Salaün “ , “ \'Elo\:ise Sala\:un “ or any other combination you may imagine.

Remark. Here I have supposed that you are following the modern trend and that you accentuate also the capital letters.

You often cannot display correctly the special letter “ œ “. It is however better to put it into your dictionaries under the compound form “ \oe “. When fonts are written for the UTF8 norm, you will not have to correct your dictionaries.

You may also use the accentuated or special characters into the command line located at the top of the dictionary window. This, however, is not mandatory. You may as well ignore all the accents. In the software FURCH, the search for the words is made without the accents. It is then simpler not to write them when you are using the command line.

Here, you will find all the sequences presently recognized by FURCH :

“ \oe “ for “ œ “ and “ \OE “ for “ Œ “

(they will, however, be written oe and OE)

“ \ae “ for “ æ “ and “ \AE “ for “ Æ “

“ \~n “ for “ ñ “ and “ \~N “ for “ Ñ “

“ \,c “ for “ ç “ and “ \,C “ for “ Ç “

“ \`a “ for “ à “ and “ \`A “ for “ À “

“ \^a “ for “ â “ and “ \^A “ for “ Â “

“ \`e “ for “ è “ and “ \`E “ for “ È “

“ \'e “ for “ é “ and “ \'E “ for “ É “

“ \:e “ for “ ë “ and “ \:E “ for “ Ë “

“ \^e “ for “ ê “ and “ \^E “ for “ Ê “

“ \^i “ for “ î “ and “ \^I “ for “ Î “

“ \:i “ for “ ï “ and “ \:I “ for “ Ï “

“ \:u “ for “ ü “ and “ \:U “ for “ Ü “

“ \^u “ for “ û “ and “ \^U “ for “ Û “

“ \`u “ for “ ù “ and “ \`U “ for “ Ù “

“ \:o “ for “ ö “ and “ \:O “ for “ Ö “

“ \^o “ for “ ô “ and “ \^O “ for “ Ô “

Other sequences of the same kind may be added in the future, if necessary.

Remark : the acute accent may have different appearances depending on the font which is in use.

The simplified norm.

As it was already said, the simplified norm has been devised in order to facilitate reutilization of existing lexicons. I will suppose that these lexicons are in plain text, without any coding or compression. If this is not true, one will have first to transcribe the lexicon in plain text. Then, we must tell to the FURCH software that we have a simplified norm dictionary. This is done by putting the sequence of characters “ !S ” at the beginning of the file. Let us examine an example of simplified norm dictionary such as the file the “ splf1.ex “ which is distributed with FURCH. The first few lines of the file have been listed below:

```
!
!S
!# Spe1
!
!   Ceci est un exemple de dictionnaire en norme
!    simplifiée.
!   Notez : Il faut aussi mettre le radical des verbes
```

```
!           pour que FURCH puisse détecter les formes
!           conjuguées.
!   This is an example of simplified norm dictionary.
!   Note : It is necessary to have the stem of the verbs
!        as separate entries in the dictionary for FURCH
!        to recognize conjugated forms.
!
!   Dictionary splf1_ex  done January 2005
!
!
#legestr
   legestr (nom masculin)= homard, pluriel : ligistri
   . lobster
#boue. (nom masculin)= bou\'ee, pluriel : boueio\`u
   . buoy
```

We immediately remark that some lines are beginning by the character " ! ". They are comments and they are generally ignored by the software. However, the lines beginning by " !S " or by " !# " are special. The " !S " indicates that we have a simplified norm dictionary and the " !# " introduces the dictionary alias.

Dictionary aliases.

Each dictionary must receive an alias. It is a short sequence of characters which may be used in another dictionary to quote the present one. For example, here the simplified dictionary has been given the alias " Spe1 ". You must note that FURCH is case-sensitive, it differentiates upper-case letters from lower-case ones. If another author wants to quote the word " legestr " (lobster) into the " splf1.ex " dictionary, it may write for example :

```
see the word << legestr >Spe1> into the excellent
dictionary written by M ...
```

The line of this dictionary, with << legestr >Spe1> in it, will be displayed with a pink background. The reader will be able to see the explanations given for this word into the " splf1.ex " dictionary by a simple mouse click on the line. You can yourself make reference to other dictionaries in the same way. You can even quote a word into your own dictionary by just writing << something >>. Where " something " is a word or multi-word expression into your dictionary.

The choice of the alias is important. It must be short, it must be easily remembered and it must not have been already used for another dictionary.

Dictionary entries.

Each word or expression that you explain into your simplified norm dictionary is an entry for that dictionary. You must indicate that you have an entry and this is done by beginning a line with the character " # ". This character will be followed by the word or expression which itself must be followed by a punctuation or jump to next line character(s). After the dictionary entry, you can put all your explanations and translations up to the next entry, that is to say down to the next line beginning by a " # ". Your lines must not be more than 100 characters long and it is best to have much less. Limit yourself at 70 characters for example. You can use as many lines as you want, FURCH does not care. For the display into the dictionary window, FURCH cuts lines after words (no hyphenation so far) when it calculates that it doesn't have enough place for the following word. You have for example on figure 1 the output of the first 2 entries in the " splf1.ex " dictionary. Though these outputs are perfectly readable, they are not perfect because the French and English translations are not separated.

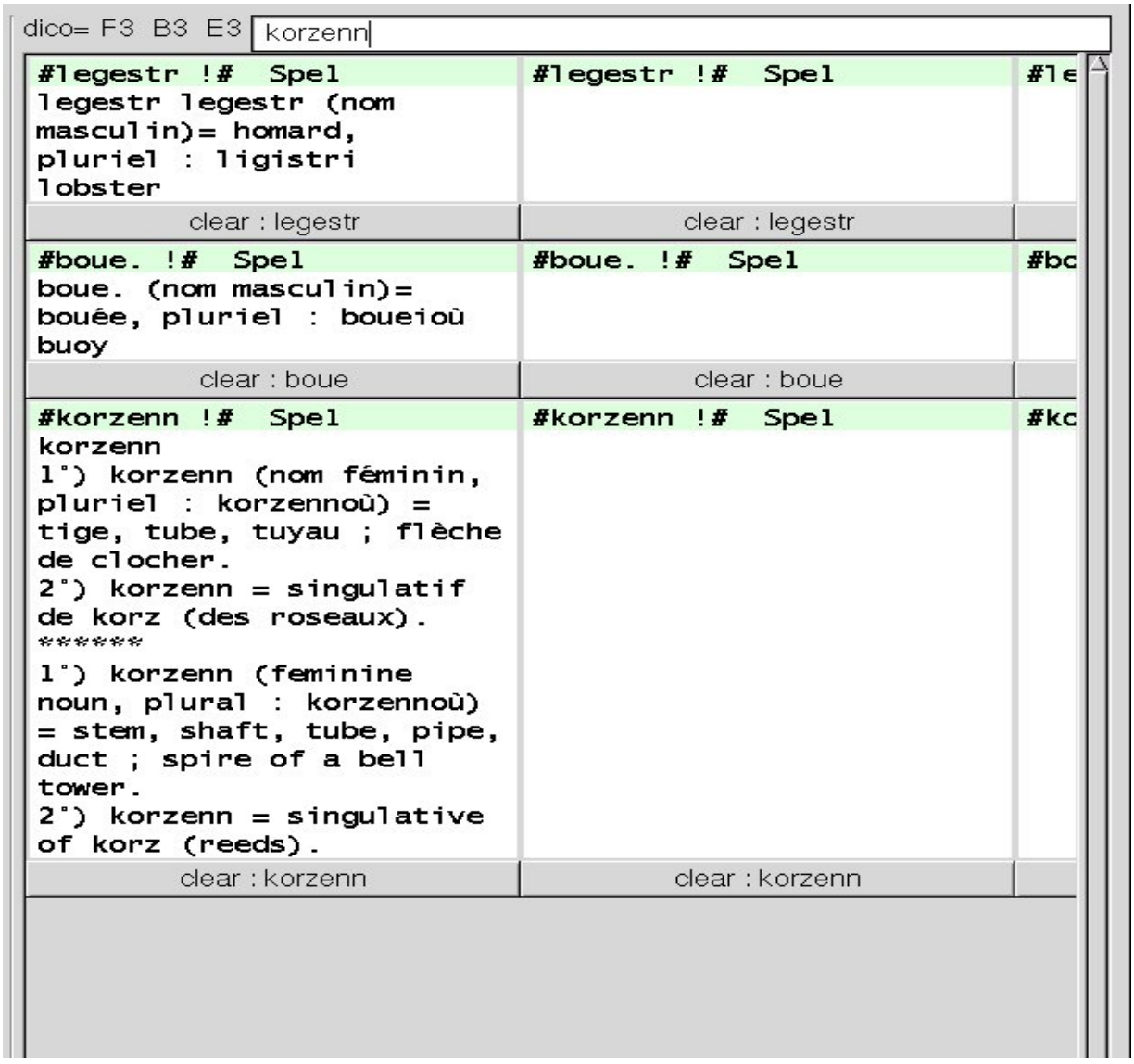

Figure 1. (splf1.jpg) *Outputs for different words of the simplified norm dictionary " splf1_ex ".*

We will see now two simple tricks which have been added to the simplified norm and which may improve your outputs. You are not obliged to use them !

When a line is beginning by a " . " (termination mark) or by a digit 0 → 9.

In fact, it would be more correct to state that this applies to the case where the first non space character is a termination mark or a digit. For that two cases, on the screen, we will begin a new line at that place. This may be used to separate the French from the English and to distinguish different meanings or cases. Let us see, for example, the word " korzenn " on the " splf1_ex " file, we have :

```
#korzenn
   1°) korzenn (nom féminin, pluriel : korzenno\`u) =
       tige, tube, tuyau ; flèche de clocher.
   2°) korzenn = singulatif de korz (des roseaux).
   . ******
   1°) korzenn (feminine noun, plural : korzenno\`u) =
       stem, shaft, tube, pipe, duct ;
       spire of a bell tower.
   2°) korzenn = singulative of korz (reeds).
```

With that text, the line " ****** " will separate the French and English translations, the termination mark itself is not written. Also the " 1°) " and " 2°) " will begin new lines on the screen. You can see on figure 1 the output that we have obtained.

The regular norm.

As we have seen above, with the simplified norm all the outputs are on the same column and the separation between the languages is awkward. The main drawback with the simplified norm is however that we cannot introduce grammatical informations in a form understandable by the software. In its present state, FURCH doesn't do much with grammatical informations but they are already of some use for conjugated verbs or noun plurals. The use of grammatical informations is likely to increase in future software releases and these informations are part of the regular norm. We will detail, now, how to write a regular norm dictionary. Let us take, for example, the " dico1_quere " dictionary.

The dictionary alias.

As we have seen for the simplified norm, a special commented line gives the alias chosen for the dictionary. This alias will be used for a rapid referencing of the dictionary. Here, as the alias is " Qe1 ", one of the first lines of the file will be :

!# Qe1

No indication of norm is necessary, the default is “ regular norm “ and after a few commented lines we can begin with the first entry. However, let us see first the grammatical abbreviations.

The grammatical abbreviations.

Here is a list of the abbreviations used to give grammatical indications to the software. These indications don't appear on the dictionary output window, so you may have to duplicate them in plain text for the human reader. Only the most common abbreviations are given, they are considered as more or less fixed. Other will be added if necessary.

(ag) anv gourel : masculine noun.

(agl) anv gourel lies : plural masculine noun.

(aw) anv gwregel : feminine noun.

(awl) anv gwregel lies : feminine plural noun.

(adgl) anv denel gourel lies : plural masculine person noun.

(agn) anv-gwan : adjective.

(arg) araogenn : preposition.

(agv) anv-gwan-verb : past participle (for a verb).

(es) estlamadenn : exclamation.

(gms) ger-mell strizh : definite article.

(gma) ger-mell amstrizh : indefinite article.

(gs) ger-stagañ **:** linking particle**.**

(r) rannig : particle.

(rv) rannig-verb : verb particle.

(rga) raganv : pronoun.

(rgaw) raganv gwregel : feminine pronoun.

(rgaw) raganv gourel : masculine pronoun.

(rgg) raganv gour : personal pronoun.

(rkg) rakger : prefix.

(rkv) rakverb : adverb.

(stg) stagell : conjunction.

(stgu) stagell genurzhiañ, co-ordination conjunction.

(tl) tro-lavar : expression, cast of style.

(v) verb, conjugated verbs may be indicated by vnn where nn are two digits.

Etc...

The FURCH software is using some of the above abbreviations to perform grammatical checking before suggesting a word meaning. This is particularly true for conjugated verbs.

The dictionary entry lines.

We are now ready to explain how to introduce a new dictionary entry. It would be more exact to speak of an " entry line " . It is a line (less than 100 characters long) which is beginning by a sign " # ". Below we have a few lines extracted from the dictionary " dico1_quere " :

```
#douar  *1 (ag-0)
    ?   il n'y a qu'une seule entree pour douar

    B>1 douar="douar (ag)" liester : "douaro\`u" ,
            "douareier (agl)"
```

At that place we have the entry " douar " (earth) in the dictionary " dico1_quere ". As for the simplified norm, we indicate that we have an entry by the sign " # ". We have then the word or expression up to a sign " * " which itself precedes a number which is the number of different grammatical identities for the word or expression. We have then the grammatical abbreviations. Here we have only one grammatical identity " (ag-0) " because " douar " is only a masculine noun.

Remark : The " -0 " in " (ag-0) " has no special meaning, it is only a separator for the software. Initially FURCH started as a project to make statistics on the use of words. The " -0 " was supposed to be replaced by an usage ratio. Though I still believe that this would be important for language learning, for lake of time this has been abandoned. The " -0 " and also the sign " ? ", at the beginning of a line which introduces comments between the entry line and the explanations, are all that remains of that now defunct project.

Another example of entry line is given below for the word " a " which has 5 different grammatical identities.

```
#a    *5   (arg-0)  (rv-0)  (gs-0) (v13-0) (es-0)
    ?   -1  Araogenn (eus) / Pr\'eposition (de)
    ?                           / preposition (of)
    ?   -2  Rannig-verb/ particule verbale / verbal particle
    ?   -3  ger-stagan (anv-verb + ober) /particule de liaison /
    ?                                    /linking particle
    ?   -4  verb mont amzer vreman trede gour unan
    ?               / verbe aller pres. 3i\`eme pers. du sing.
    ?                     / verb to go present third pers. sing.
    ?   -5  Estlammadenn / Exclamation / Exclamation

    B>1  a="a (arg)". Araogenn.
```

Links can be put into the entry line.

When a word is explained in a dictionary, one sometimes wants to make reference to another word in the same dictionary or to the same word into another dictionary. We have already seen for the simplified norm that we can use the notation : << expression >>.if we are making reference to an entry into the same dictionary, or to << expression >alias> if we are making reference to an entry into another dictionary given by its alias. A simple klick on the expression allows us to display the additional information on the dictionary window. The same notation can be used into the dictionary entry line, the difference is that we don't have to make a click, the additional information is considered by the dictionary author as absolutely necessary and it is always displayed.
The example below taken from " dico2_quere " shows the word " mailhurell " (swaddling-cloth) which makes reference to a synonym, the word " maillur ", into the same dictionary.

```
#mailhurell  *1   (aw-0)   << maillur >>

    B>1  mailhurell = "mailhurell (aw)". Anv gwregel,
                         liester : "mailhurello\`u (awl)".
```

Another example taken from the same dictionary " dico2_quere " is presented now. The word " koulz " (as well) was in fact already encountered but written under the form " kouls " . The complete explanations for this word are given into the dictionary " dico1_quere " whose alias is " Qe1 " and they don't need to be repeated into " dico2_quere ". It is enough to put **<< kouls >Qe1>** into the entry line to display all the information of the " dico1_quere " dictionary after what is given for " koulz " in " dico2_quere ".

```
#koulz *2 (rkv-0)  (ag-0)  << kouls >Qe1>
```

```
    B>1  koulz = "koulz (rkv) (ag)". Rakverb pe anv gourel.
 .
```

Remark : The explanations given above for the entry line references are the present norm. However, another norm, considered now as obsolete, was used into " dico1_quere ". For example the plural " deliou " (leaves) will send us to "delienn " (leaf) by the notation : " **> delienn** "

```
#deliou    *1 (awl-0)              > delienn
    B>1  deliou = "delio\`u (awl)" Liester eus < delienn >.
    F>1  delio\`u = "feuilles (nfp)" Pluriel de < delienn >.
    E>1  delio\`u = "leaves" Plural of < delienn >.

#delienn    *1 (aw-0)
    ?il n'y a qu'une seule entree pour delienn

    B>1 delienn = "delienn (aw)". Anv gwregel,
         liesterio\`u : "delienno\`u (awl)" ha "delio\`u (awl)".
```

You immediately remark the difficulties with this notation. You cannot make reference to multi-word expressions and you cannot send the reader into another dictionary. This is why this notation, though still valid, is considered as obsolete and it will be abandoned as soon as " dico1_quere " is upgraded.

The numbering of the paragraphs.

It has already been said into the first part of this documentation that one of the very important features of the FURCH software was its capability to display information under a synoptic fashion with languages separated into columns. The information is also cut into small paragraphs which are beginning at the same level. The reader can thus have a look first at the explanations in Breton, and if he has difficulties, he will have at the same level a translation into English or French. We must now see how to obtain a synoptic display. This is done by a special paragraph numbering. We already know that we must differentiate the languages, that a word can have different unrelated meanings or grammatical values, that we can have different reading depths. All the needed informations will be given by a numbering such as :

B>1[2.0.1]

which means Breton paragraph, first meaning, level 3 (non essential level) complements 0.1.

It is best to take one example. Let us have " word1 ", a word or expression written in an old text book page with some old and fancy spelling. This word can have two

different meanings corresponding to two different grammatical values. We will have into the dictionary something like the following :

```
#word1  *2   (ag-0) (rkv-0)

?   text by AlN Marsh 18 2005. Corrected by XXX May 28 2005.

    B>1   word1 = "word_kltg (ag)". Anv gourel ...
    B>1[0.0]  Some text explaining the meaning of the masculine
              noun word1 ...
    B>1[0.1]  Additional text on the meaning of word1 ...
    B>1[0.1.1] Complements on additional text ...
    B>1[0.1.2] Still other complements ...
    B>1[1.0]  Some grammar for word1 ...
    B>1[1.1]  Additional grammar ...
    B>1[2.0]  Not essential complements, chattering ...


    B>2   word1 = "word_kltg (ag)". Rakverb ...
    B>2[0.0]  Some text explaining the meaning of the
              adverb word1 ...
    B>2[0.1]  Additional text on the meaning of the adverb word1..
    B>2[0.1.1] Complements on additional text ...
    B>2[1.0]  Some grammar for the adverb word1 ...
    B>2[1.1]  Additional grammar ...
    B>2[2.0]  Not essential complements, chattering ...
    B>2[2.0.1]  Still other not essential complements.
```

The first line is the dictionary entry line. It begins by the sign " # " and we have already seen its structure. This first line can be followed by one or several comment lines beginning by a sign " ? ". These lines are not displayed on the screen. After that, we have a line such as :

```
    B>1   word1 = "word_kltg (ag)". Anv gourel ..
```

where the " B>1 " means Breton, first meaning of the word. There, you can write again the word with its spelling of origin, give the spelling in a modern orthographic system such as KLTG, university or any other system you may like and add the grammatical gender of the word.
You will then introduce paragraphs with a numbering such that " B>1[0.x.y] " , " B>1[1.x.y] " or " B>1[2.x.y] " where " x.y " are digits used to class the paragraphs. The first digit " [0...] " , " [1...] " or " [2...] " indicates the lecture level. Lecture level 1 is indicated by " [0...] ", lecture level 2 by " [1...] " and lecture level 3 is indicated by " [2...] ". Lecture level 0 is when there is no " [...] " such as in " B>1 ". This

notation is rather awkward, it is maintained for historical reasons.
The user of FURCH can choose to display the paragraphs up to some given level numbered from -1 (no display) to 3 (full display). In the dictionary file, you are not obliged to put the levels in increasing order. You can have for example :

```
B>1   word1 = "word_kltg (ag)". Anv gourel ...
B>1[0.0]  Some text explaining the meaning of the masculine
          noun word1 ...
B>1[1.0]  Some grammar for word1 ...
B>1[1.1]  Additional grammar ...
B>1[2.0]  Not essential complements, chattering ...
B>1[0.1]  Additional text on the meaning of word1 ...
B>1[0.1.1] Complements on additional text ...
B>1[0.1.2] Still other complements ...
```

The dictionary writer can put for example the grammar just after a basic definition of the word and put the complements for word definition at the end if he wants. However, the numbering " [.x.y] " must be in increasing order.
Remark : " x " and " y " start from 0 but the 0 can be omitted.
For example " B>1[0.0] " is equivalent to " B>1[0.] " or to " B>1[0.0.0] " and of course, " B>1[1.0] " is equivalent to " B>1[1.] " or to " B>1[1.0.0] "
The second meaning of " word1 ", here for example as adverb, will have a paragraph numbering such that :
" B>2 ", " B>2[0.x.y] " , " B>2[1.x.y] " or " B>2[2.x.y] " where " x.y " are, as above, digits used to class the paragraphs. In order to facilitate the separation of the word different meanings or grammatical genders, the first line of the paragraphs " B>1 ", " B>2 ", " B>3 " ... will appear on the dictionary window with a light green background. For all the other paragraphs, the first line will have a light blue background. We have already seen how to adjust these colors in the " furch.conf " file.
So far, we have only considered the case of paragraphs written in Breton. Those written in French will be numbered : " F>1 " , " F>1[0.x.y] " , " F>1[1.x.y] " , " F>1[2.x.y] " , " F>2 " ... and those written in English : " E>1 " , " E>1[0.x.y] " , " E>1[1.x.y] " , " E>1[2.x.y] " , " E>2 " ...
You are not obliged to regroup the paragraphs according to the language, you must simply have the paragraphs of each language classed in a natural fashion disregarding the other languages. Below you have a valid ordering where the Breton has been put first and then the French and English :

```
B>1   word1 = "word_kltg (ag)". Anv gourel ...
B>1[0.0]  Some text explaining the meaning of the masculine
          noun word1 ...

B>2   word1 = "word_kltg (ag)". Rakverb ...
B>2[0.0]  Some text explaining the meaning of the
```

```
          adverb word1 ...

F>1   word1 = "word_kltg (ag)". Nom masculin ...
F>1[0.0]  Some text in French which is the translation of the
          text in Breton.

F>2   word1 = "word_kltg (ag)". Adverbe ...
F>2[0.0]  Some text in French (translation of the Breton).

E>1   word1 = "word_kltg (ag)". Masculine noun ...
E>1[0.0]  Some text in English which is the translation of the
          text in Breton.

E>2   word1 = "word_kltg (ag)". Adverb ...
E>2[0.0]  Some text in English (translation of the Breton).
```

However, this is not the only possibility. You can also regroup the first meaning with the 3 languages and then have the second meaning with also its 3 languages :

```
B>1   word1 = "word_kltg (ag)". Anv gourel ...
B>1[0.0]  Some text explaining the meaning of the masculine
          noun word1 ...

F>1   word1 = "word_kltg (ag)". Nom masculin ...
F>1[0.0]  Some text in French which is the translation of the
          text in Breton.

E>1   word1 = "word_kltg (ag)". Masculine noun ...
E>1[0.0]  Some text in English which is the translation of the
          text in Breton.

B>2   word1 = "word_kltg (ag)". Rakverb ...
B>2[0.0]  Some text explaining the meaning of the
          adverb word1 ...

F>2   word1 = "word_kltg (ag)". Adverbe ...
F>2[0.0]  Some text in French (translation of the Breton).

E>2   word1 = "word_kltg (ag)". Adverb ...
E>2[0.0]  Some text in English (translation of the Breton).
```

Other mix are also possible. You will, of course, choose the configuration which is best for the writing and the correction of your article. When you have a lot of explanations for each meaning of the word, the second configuration may be better.

Special features for level 0 paragraphs.

The level 0 paragraphs are those introduced by “ B>1 “, “ B>2 “, “ B>3 “, ... , “ F>1 “, ... , “ E>1 “. The first line of these paragraphs is displayed with a light green background. Usually, I am putting on these paragraphs only grammatical informations, translations into other languages or transcriptions of the word into the KLTG orthographic system. In its present state of development, the FURCH software doesn't use these informations. However, these informations are already tagged for a future use. Let us take one example extracted from “ dico2_quere “, the word “ komz “ may be a verb, a verb stem and a word :

```
#komz  *3  (v-0) (pgv-0) (aw-0)
    B>1  komz = "komz (v)". Verb, anv-gwan-verb : "komzet (agv)".
    B>1[0.0]   ..................
    B>1[1.0]   ..................
    B>1[1.1]   ..................
    B>1[1.2]   ..................

  F>1  komz = "parler (v)". Verbe, participe pass\'e : < komzet >.
    F>1[0.0]   ..................
    F>1[1.0]   ..................
    F>1[1.1]   ..................
    F>1[1.2]   ..................

   E>1  komz = "to speak (v)". Verb, past participle : < komzet >.
    E>1[0.0]   ..................
    E>1[1.0]   ..................
    E>1[1.1]   ..................
    E>1[1.2]   ..................

    B>2  komz = "komz (pgv)". Penngef ar verb < komz >.
    F>2  komz = Radical du verbe < komz > (parler).
    E>2  komz = Stem of the verb < komz > (to speak).

    B>3  komz = "komz (aw)". Anv gwregel, liester : "komzo\`u
(awl)".
    B>3[0.0]   ..................
    B>3[0.1]   ..................
    B>3[0.2]   ..................
    B>3[0.3]   ..................

    F>3  komz = "parole (nf)". Nom f\'eminin (en breton),
              pluriel : < komzo\`u >.
    F>3[0.0]   ..................
    F>3[0.1]   ..................
    F>3[0.2]   ..................
    F>3[0.3]   ..................

    E>3  komz = "speech, spoken words". Feminine noun (in Breton),
```

```
          plural : < komzo\`u >.
E>3[0.0]   ..................
E>3[0.1]   ..................
E>3[0.2]   ..................
E>3[0.3]   ..................
```

As the word " komz " has three different grammatical values, we have for each language three level 0 paragraphs. These paragraphs have been reproduced above. You can remark that some information has been placed between cotes, for example :

```
B>1  komz = "komz (v)". Verb, anv-gwan-verb : "komzet (agv)".

E>1  komz = "to speak (v)". Verb, past participle : < komzet >.
F>1  komz = "parler (v)". Verbe, participe pass\'e : < komzet >.
```

These words, written in red above, together with the abbreviated grammatical identifications, will be used as additional entries to the database in a future version of the FURCH software. The words in French and English can also be used as entries for pseudo French-Breton and English-Breton dictionaries. This will not replace true dictionaries but will be helpful as long as these dictionaries are not written.

The regular norm tips-and-tricks.

As for the simplified norm, special tricks have been introduced to allow some control of the on-screen display. As each end-user can choose the width of each dictionary column, the authors cannot know when a new line will begin. It is however sometimes necessary to impose a jump to new line at some precise location into the text to facilitate the reading. This can be done by the introduction of a new paragraph but we have then a colored first line which may not be wanted. We have 3 other possibilities to introduce a break in the text and impose the start of a new line at that place. In the text of our dictionary we can begin a line :

1) by a termination mark : " . "
2) by a digit *0 → 9.*
3) by a sign " < "

When I am saying that a line is beginning by some character, I intend that the first character which is not a space character is this character, here a termination mark, a digit or a sign " < ".

Remark 1 : The termination mark at the beginning of a line in a dictionary text is a command character which is never displayed on the screen.

Remark 2 : I am using the sign " < " to introduce citations. It is then often convenient to make that sign start a new line to put into evidence the beginning of the citation. However this is perhaps not always what we want. We can prevent the new line jump by beginning the line by a comma " , ". That comma being a command character will not be displayed on the screen. So, to have " , < " at the beginning of a

line will prevent the jump to next line and the citation will be written in the text without separation. The comma can also be used in front of the digits with the same effect.

Secondary entries in a regular norm dictionary.

We have already seen that an entry in a regular norm dictionary was a line beginning by a character « # ». However, the writing of a dictionary is a very long process. We will have to wait for years before we have enough words in the data base for the FURCH software to become really useful. That is why, in the hope of speeding up somewhat the process, the notion of secondary entry has been introduced. A secondary entry is a word which is defined inside an article written for a primary entry. For example, in an article written for the word « priest », the words « parish priest » and « vicar » have been defined. A secondary entry is indicated in the dictionary by the notation : <" word "> and its translation by (" word "). Inside the marked segment we put the secondary entry and optionally grammatical tags like the usual (ag) (aw) (rkv)... The grammatical tags are never displayed on the screen but they may be used internally by the software. Another problem is for the use of capital letters. The software will systematically transform the first letter of a word in a secondary entry to lower case for its data base index except if we write :

<". Word ">. The point after <" indicates to the software that it must do that. This behaviour is useful for proper names. Except for the first letter, the other upper case letters remain unchanged into the data base index.

As was explained into the FURCH presentation, in order to facilitate reading, a secondary entry word is indicated by a yellow background at the start of the article and by yellow backgrounds, for the lines with that word, inside the article.

Perhaps other norms for FURCH ?

As was already said, the regular norm is the native FURCH norm and the simplified norm has only been introduced to make easy the use of already written lists of words. It was supposed that these lists were simple, without grammatical indications. Some people may want to make dictionaries with a norm simpler than the regular one but still with more possibilities than the simplified norm. They can do that and their norms may be included into FURCH. However, keep in mind that introducing a new norm in the software is a lot of work and this can only be considered for a large enough dictionary.

Now a last remark for the regular and simplified FURCH norms. Some commercial companies consider their norms as private intellectual properties and they try to obtain software patents for self-evident features. Contrarily to that trend, FURCH norms are declared “ open “ and you can use them into your own products. There is no guaranty of any kind with these norms however, except for the weak moral

promise that FURCH will do its best to remain compatible with its older norms, or that the dictionaries will be transformed to become compliant with the new norms.

Installation of the FURCH software.

Installation on Windows OSs.

FURCH is a “ GNU/Linux/Gtk/gcc “ project. However “ gcc “ and “ Gtk “ have been ported on Windows and so FURCH will also work on Windows 98 SE, 2000 and XP. At present time FURCH is using “ gcc “ and “ Gtk ” as they are implemented into “ Dev-Cpp ”.

Professional programs have automatic installation softwares and smart amateur's ones also. My knowledge is much too limited on that particular question so you will have to install FURCH by hand. However this is easy for Windows and it may avoid you some conflicts with your already installed softwares.

As for any installation of a Windows program, you need dynamic libraries. For FURCH on the XP-OS or W2000-OS, you will download the following “ dll “ :

iconv.dll
intl.dll
libatk-1.0-0.dll
libcairo-2.dll
libgdk-win32-2.0-0.dll
libgdk_pixbuf-2.0-0.dll
libglib-2.0-0.dll
libgmodule-2.0-0.dll
libgobject-2.0-0.dll
libgtk-win32-2.0-0.dll
libpango-1.0-0.dll
libpangocairo-1.0-0.dll
libpangowin32-1.0-0.dll
libpng13.dll
zlib1.dll

Now, the question is : Where to put these libraries ?

The first possibility is : Somewhere into your PATH. The variable PATH is a list of directories where the system searches for the libraries it needs. The system searches first into the first directory of its list, then into the second, etc ... The search stops to the first find. You can visualize your PATH list in opening a DOS window and typesetting " path " (not the " " please !). The system will write the list of the PATH directories separated by semicolons " ; ". You will have a " system " or " system32 " directory and perhaps a " .\ " directory at the beginning of the list. You can modify your path by editing the " autoexec.bat " file into Windows 98 or by the following actions for win2000 or winXP :

Change session to become administrator ;

Have a right click on the desktop icon and then click « properties » ;

Then choose the « Advanced » tab ;

Then click the « Environment Variables » button.

In the « Environment Variables » window you will highlight the « Path » variable in the « Systems Variables » section (the lower one). Click « edit ». You can now add directories into your PATH variable but you will be allowed to do that only if you are a member of the administrators group.

A second possibility, and probably the safest one, is to put your dlls into the directory where your executable is. Generally Windows-OS goes first to the directory of the executable in its search for the dlls.

You can also to introduce the characters " **.\;** " at the beginning of the PATH. So, the system will be obliged to search first into the directory where your executable is before any other location. You will then put all the libraries into the directory where you have your " furch.exe " executable. However, on Linux, experts are signaling a mild security problem with that solution.

Once you have installed the libraries, you will create a new directory, for example " Q2-11 ", in that directory you will put the executable " furch.exe " and the other text files needed by the software. You need :

dibenn1 : a list of recognized word endings

furch.conf : the FURCH configuration file

gtkb.rc : the configuration file for gtk.

You need also the dictionaries which are quoted into the " furch.conf " file. At the time of the software's version 2.011, you need the dictionaries :

dico1_quere : the first volume of the " Quéré " dictionary.

dico2_quere : the second volume of the “ Quéré “ dictionary.

roll1_kltg : a lexicon for words written in KLTG.

roll2_kltg : the second volume of words written in KLTG.

yezh1_kltg : a lexicon for irregular verbs, etc..., in KLTG.

splf1_ex : an example of dictionary for the simplified norm.

Then, you need the text files which will be displayed in the book window. Each file is considered to be a book. These files must be latin1 text. They can be formatted with tags like .NP, .NPO... or be not formatted. So you can display texts generated by softwares such that OpenOffice, MS-Words, etc...

One example of formatted text is : sarmoniou1.txt which you can download at the FURCH or GUTENBERG web sites.

The source code has also been included into the package proposed for download. You don't need these files, except if you want to make a compilation yourself with the “ Dev-Cpp “ software. You must be aware that the source code is using version GTK.2.8.12 which is not the most recent one for Windows.

Installation on Linux.

Linux is a free and marvelous software, however it could not be considered as fully operational for the common end-user. Linux has three main drawbacks :

1. Compatibility of Linux with the hardware is poor. Constructors don't bother to develop drivers for Linux as the number of installed systems remains low. This number remains low in the general public because of the two following reasons :
2. It is often necessary to recompile the kernel for the introduction of a new driver. This action is dangerous as you can definitely spoil your system. For the configuration of your new kernel you will have to choose between thousand of esoteric options. There exist ways around all these difficulties (save your old kernel, start from a Knoppix kernel, etc). However someone who is not a professional is likely to have a hard time with kernel set-up.
3. The introduction of a any new software into your system is virtually impossible. The only exception is when someone has already packaged the software for your distribution and moreover for the exact version number which is installed on your computer. There are some efforts to improve that situation (for example the « klik » procedure for Knoppix or the « autopackage » software) but widespread use of such tools has yet to come.

What has been said above was true a few years ago and the situation has improved a lot with distributions like Ubuntu but you still have to install pre-compiled packages specific to your distribution.

For the FURCH software, we will try to turn around the difficulties and give

indications so anyone who is not a specialist will have a reasonable amount of chances to successfully install the software.

The FURCH software is very simple, it only makes use of standard C functions and of the GTK2 library. It uses also only standard GTK functions. As gcc and GTK are in every distribution, you will only have to install these two libraries (the complete versions also called development versions). Both libraries will be compatible because they are from the same distribution. You will also need the " make " package which is in all the distributions. Once you have installed these three packages and often they are already there by default, you will make the compilation :

make -f makegtkb2

With that line the order " make " uses the Makefile " makegtkb2 " which you can download together with the two header files and the source files of the software :

gtkb2_0.h
gtkb2_ftexte.h
gtkb2.c
gtkb2_1.c
gtkb2_2.c
gtkb2_3.c
gtkb2_4.c
gtkb2_5.c
gtkb2_6.c
gtkb2_7.c
gtkb2_8.c
gtkb2_9.c

The order make will compile, link the files and generate the executable " furch.x ". It will also launch the execution of the software.

As for Windows, you need, in the same directory where you have " furch.x ", the 3 following files :

dibenn1 : a list of recognized word endings
furch.conf : the FURCH configuration file
gtkb.rc : the configuration file for gtk.

You need also the dictionaries which are quoted into the " furch.conf " file and of course the files corresponding to the pages you want to read.

Remark : Linux and Dos-Windows text files are different. Each line on a Dos-Windows file ends with CR/LF (Ascii 13 + Ascii 10) and each line into a Linux file ends with a single LF. However, FURCH-Windows accepts Linux text files and FURCH-Linux accepts Dos text files. So, to save place and to simplify the software distribution, the text , data, configuration and dictionaries files are only distributed under their Dos-Windows form. They will be accepted by both versions of FURCH.

If you have any doubt, you can convert (under Linux) the Dos-Windows files into Linux files by the utilities " dos2unix " or " fromdos " and the reverse can also be done by the utilities " unix2dos " and " todos ". On Windows you can convert a Linux text file by using NotePad, editing and saving back the text file. The CR character will be automatically added at the end of each line by NotePad. The number of lines that NotePad can handle is severely limited however and you may want to use other equivalent free text editors for that kind of conversion. Word2003 can also do the job.

The FURCH software and Mac OSs

The present version of FURCH for Linux is based on GTK2.x. This version of GTK has been ported on Mac OS X. So, it should be possible to use the Linux version of FURCH and compile it on a Mac. I don't have , however, such a machine at my disposal, so someone else using Mac OS X will have to do the job. I will be happy to add the executable on the FURCH web page or to add a link toward another web page having this executable.

Conclusion.

FURCH is a free project, its code is public and it is released under the terms of the GPL license. You can improve this code and use parts of it into your own applications. Your modifications must be published and released under the terms of this same license, however. For the literary part of the project, texts and dictionaries, each author may do as he wants. Some texts are in the public domain, others may be copyrighted. The dictionary " dico_quere " is copyrighted, but free for any non commercial use. This, in fact, is to let me free to extract and expand parts of it. It is also to avoid any anarchic corrections and modifications by other authors. People can send corrections, if they are accepted the name of the correctors will be added to this dictionary in order to recognized their work. If these corrections were not accepted (an unlikely situation), the authors would still have the possibility to publish their own dictionaries with the same words and the complements and corrections that were not accepted. I would certainly be happy to add these new dictionaries to the FURCH database. In a more general way, I strongly encourage everybody to write dictionaries, in the orthographic system of their choice, with perhaps only a few words in each dictionary. Everything is useful. Let me take one example. Here are a few words of the scythe family :

1. “ gwignet “ : a small sickle to cut grass, to fish sand-eels, for the druids to pick mistletoe...
2. “ falzig “ : a sickle.
3. “ fosilhon “ : a heavy sickle to cut wood, to prune trees.
4. “ falz “ : a scythe.

All these words are used in the region around Tréguier. Are they understood elsewhere ? Are some other words in use in other regions ? These four words may constitute a dictionary and you can have as many such dictionaries than you have districts in Brittany. What we can do with that ? Well, trace philogenic trees with appropriate softwares now widely in use for genetic studies. With that kind of methods, we would be able to recognize the transmissions, evolutions and transformations of the words and because words travel with people, we would be able to trace the commercial routes in Brittany. Imagine also that Cornish language be still alive, by comparing the local names of fishes or other marine animals on each side of the channel, we would be able to trace the origin of the populations in Brittany, to differentiate Cornish and Welsh origins, etc. Time is running fast against traditional Breton, we must quickly note these humble words. They are the memory of our Nation. Modern electronic communication tools allow us to save and distribute them with almost no expense. Let us use these new possibilities. Many people have already recorded talks, tales, songs etc. We can now disseminate these gatherings. FURCH software is one tool in that grand undertaking. Take your share in that project, improve the present documentation, write your own dictionaries, send corrections to other authors, note words, record stories on magnetic tapes or CDs, take pictures, scan old books. Everything is precious and can now be published, distributed free of charge. We must not let the chance pass by !

Alphonse Nandert, December 15, 2007

Written with OpenOffice 2.2