Klavaro - Typing Course

Advanced User-Manual - 1.0

Felipe E. F. de Castro

November 1, 2005

Contents

List of Figures

About this document

This is not a simple user manual telling what happens when clicking at this one or that another button. If you look for something like that, then you will have to be happy with just those tips, which appear in the program. If all you want is to learn touch typing, try to use the application now and read over the instructions therein. If you have already done this and found difficulties, read the first part of this text here, about basic definitions. It describes further the program main functions.

So, what might be missing to detail? We think it would be good to put here the methods used in calculating the typing accuracy, velocity and fluidness. And also some tips about the files used by the program, showing how one may modify them in order to customize even more the course structure. Yes, you do not need to get into the C source code mess to achieve an even greater flexibility. You just have to edit some text files. But if you want to change things much more, then try to figure out the source code structure, starting by reading the last part of this document.

Thus, this manual was written chiefly for the advanced user, interested in modifying the basic characteristics of the program. If you make some nice change and you think it might be useful for most of users, you may send it for us to analize and, maybe, include it in a future official program release. If the contribution is accepted, you will surely be mentioned in our *Hall of Collaborators*:

http://klavaro.sourceforge.net/en/contrib.html

1 General definitions

1.1 Main menu

The initial window presents startup and basic configuration menus. There are 5 main startup options:

0. **Introduction**: instructions on how to position the hands.

1. **Basic course**: the file *basic_lessons.txt* defines the 43 lessons of this course. This is where the keyboard layout most influences the exercises, because the characters used in each lesson are defined by their positions in the virtual keyboard.

2. **Velocity exercises**: this exercise depends on files containing a list of words to be randomly inserted as fake sentences and paragraphs. Read about them forward: file type *.words*

3. **Fluidness exercises**: common text files are used, from which three paragraphs are randomly picked out to form each exercise. The file must contain at least three paragraphs, separated by carriage returns.

This is the unique exercise which allows (demands!) correcting errors by the backspace key. Read further forward: file type *.paragraphs*

1.2 Basic configurations

The configurations relate to the keyboard layout and the applied language.

1.2.1 Keyboard layout

The keyboard layout only has influence over the basic and adaptability exercises. For example, if you need to type with dead keys but do not want to type the accents twice, you may combine them with letters at the virtual keyboard definition. So, we might define the set
{à, ê, í, õ, ü},
instead of
{ ` , ^ , ´ , ~ , ¨ }.

1.2.2 Language

The language is always detected during the initialization, through the following environmental variables:

- `LANGUAGE`
- `LC_ALL`
- `LC_MESSAGES`
- `LANG`

The first defined variable found, at this specific order, is the one used. If the language to be used is not available in the program, it will try to use Esperanto (seldomly) or English. Anyway, it is always possible to change to one of the available languages after the program had already started.

2 Typing valuation criterions

All the three valuation criterions applied directly relate to the typing skill qualification: the greater are their values, the best is the performance estimated.

2.1 Accuracy

The accuracy is given as a function of the number of characters inserted and of the ones correctly typed. In order to make this text more organized and compact, let us use mathematical notations:

N_t is the total number of characters inserted during a lesson or exercise.

N_i is the number of characters incorrectly typed.

A_p is the accuracy reached, given by a percentual of correct hits.

The formula to calculate this is very simple:

$$A_p = \frac{N_t - N_i}{N_t} \cdot 100 \tag{1}$$

That is, an accuracy at 0% means that all the characters were incorrectly typed. And 100% means that all the characters were correctly typed. For the fluidness exercises, it is possible to get negative values, because the user can make more mistakes than the number of characters inserted. When he/she goes backward to correct these mistakes, perhaps new errors will be made: that is not pardonable, the recurrent errors must be counted too.

2.2 Velocity

The velocity is a function of the number of characters correctly typed, in a given amount of time.

t is the time the user takes to accomplish the exercise.

N_c is the number of characters correctly typed, given by $(N_t - N_i)$.

V_{cps} is the typing velocity, given as characters per second (CPS).

V_{wpm} is the typing velocity, given as words per minute (WPM).

Thus,:

$$\begin{cases} V_{cps} &= \dfrac{N_c}{t} \\ V_{wpm} &= 10 \cdot V_{cps} \end{cases} \tag{2}$$

The factor 10, in the second equation, comes from that we use an average as estimation. For each 6 characters, one word is counted in average: 5 letters

and 1 space. As one minute equals to 60 seconds, the transformation from CPS to WPM is easy to be demonstrated:

$$\mathsf{WPM} = \mathsf{CPS}\left(\frac{\text{char.}}{\text{s}}\right)\left(\frac{\text{word}}{6\text{ char.}}\right)\left(\frac{60\text{ s}}{\text{min}}\right) = 10\ \mathsf{CPS}\left(\frac{\text{word}}{\text{min}}\right)$$

It would be better to use only the velocity as CPS, which is straighter, but, for tradition reasons, we decided to keep also the WPM rating, which is very common.

2.3 Fluidness

The fluidness case is defined more by a personal taste than by following any standard. This criterion aims at valuating the concentration ability of the typist and his/her adeptness over the keyboard. It is assumed that the more the typist is used to the keyboard, the less hesitations occur between touchs; and also, the methodic application of touch typing automatically leads to a more and more constant (and healthier) rhythm.

The variable used to get to a measuring of fluidness is the time between touchs, for those characters correctly typed. Suppose the following sentence is to be typed:

```
We lived.
```

And the user has errored at letter *l* of word *lived*, changing it by *h*. He just rimarked this after he typed the letter *i*, accumulating two errors before backspacing to correct them. A possible representation for this pressed key sequence is:

W	e		h	i	<*bs*>	<*bs*>	l	i	v	e	d	.
*	*	*	!	!	!	!	*	*	*	*	*	*
t_1	t_2	t_3					t_4	t_5	t_6	t_7	t_8	t_9

<*bs*> are the correcting backspaces; the valid touchs are symbolized by asterisks and the invalid ones by exclamations. Then, symbolically, the times between touchs are those which goes throught two consecutive asterisks and the exclamations represent the introduced lags when mistakes are made. This way, the instants t_i are stamped when each correct touch occurs.

For this example, the time interval sequence is mounted this way:

$$\mathbf{T} = \{T_1 = (t_2 - t_1),\ T_2 = (t_3 - t_2),\ T_3 = (t_4 - t_3),\ \ldots,\ T_8 = (t_9 - t_8)\}$$

Generally, we have: $T_i = (t_{i+1} - t_i)$, with i varing from 1 to $(n-1)$, where n is the number of characters correctly typed. To simplify the formula definition for the fluidness calculation, some intermediate relations are put:

$$\begin{aligned} X_i &= \sqrt{\frac{1}{T_i}} \\ \overline{X} &= \frac{\sum_{i=1}^{(n-1)} X_i}{(n-1)} \\ \mathsf{SD}(X) &= \sqrt{\frac{\sum_{i=1}^{(n-1)} \left(X_i - \overline{X}\right)^2}{(n-2)}} \end{aligned}$$

And finally, the calculation for fluidness is:

$$F_p = \left[1 - \frac{\mathsf{SD}(X)}{\overline{X}}\right] \cdot 100 \tag{3}$$

One recognizes the standard deviation formula at the definition of $\mathsf{SD}(X)$, over the variable X. In an ideal case, with constant rhythm, this standard deviation becomes null and the fluidness value results 100, the maximum that can be reached. So, we bind this criterion to a percentual, the same way as done with the accuracy.

3 Used files and their formats

Some of the files used by the program are saved in a subdirectory created in the user folder:

(HOME)`/.klavaro`

Other files, as they are static, are kept in the program's data subdirectory. In GNU systems, this folder is located at:

`/usr/share/klavaro`
or:
`/usr/local/share/klavaro`

3.1 Keyboard layouts (virtual)

In order to get a basic course which is independent of (or from?) keyboard layout, it is necessary to keep information about the positions of each key in the layout installed in the computer. Given this necessity, the independence becames partial; but even so, the flexibility is very much improved, because when the program does not have knowledge about some kind of keyboard, it is still possible to insert such information. For that, the program provides a special editor that interacts with a virtual keyboard, which reflects the characters positions in a standard spatial distribution of keys, allowing to define a great diversity of layouts.

The files used to save these layout informations have *.kbd* as extensions. They are text files containing two sets of four lines. Each line represents a row of 14 keys. And each character defines its positional relation with the key which commands its insertion. For inexistent keys, a blank space is required. The first four lines define the lower-case characters, without pressing <*Shift*>. While the four last lines define the characters inserted throught the combination with the shift key. The control keys <Tab>, <Shift>, <Ctrl>, etc, are not represented at all. Despite this detailed description, the most secure way to create these files is by using the program editor, launched from the Define button at the main menu.

To know how the *.kbd* files are used on the creation of the basic course lessons, read further about its configuration file, below at section 3.6. As an example, here we show the file *qwerty_br.kbd*:

```
‘1234567890-=
qwertyuiop[]
asdfghjkl;’\
 zxcvbnm,./
~!@#$%^&*()_+
QWERTYUIOP{}
ASDFGHJKL:"|
 ZXCVBNM<>?
```

3.2 Word sets

In the velocity exercises, the words inserted come from a file with a *.words* extension, which is just a text file containing a list of words, with or without

repetition, one by line.

For each language selected at the main menu there is a default dictionary file, located at the program's data folder. But it is possible to create new dictionary files from common text ones. One just has to select the text from the Other command, at the velocity exercises window. The files created this way are saved in the user subdirectory. These files are also automatically generated when *.paragraphs* files are created, in the fluidness exercises.

3.3 Paragraph sets

In the fluidness exercises the inserted paragraphs come from the file with a *.paragraphs* extension, a text file containing a list of sentences and paragraphs.

For each language selected at the main menu there is a default paragraphs file, located at the program's data folder. But it is possible to create new paragraphs files from common text ones. One just has to select the text from the Other command, at the fluidness exercises window. The files created this way are saved in the user subdirectory and the words therein are automatically used to generate a *.words* dictionary file.

3.4 Progress statistics

At the end of each exercise, the main performance measurements hitted by the student are saved in logging files, so that they can be showed as progress (or regress :-) charts.

These files are never overwritten. All the data get accumulated at their end. Also here the user subdirectory is used to save them. If you are not happy with the simple charts provided by the program, you may use an electronic spreadsheet, for example, to better visualize the data. The C numeric format is always applied, to avoid surprises when logging with other language configurations.

3.5 Texts of the course instructions

All of the course instructions are read from text files (*.txt*), stored at the program's data subdirectory. Their name follow this model: `LL_CC_Name.txt`, where `LL` is the language code, `CC` is the country code and `Name` reminds the

role of the text in the program. The following list shows all the instruction files and what are they used for.

- `LL_CC_help.txt` - tiny ironic text to explain the redundancy of further help in such a simple program full of tips.
- `LL_CC_about.txt` - general information such as: version, authors, etc.
- `LL_CC_intro.txt` - general instructions for the introduction on a touch typing course.
- `LL_CC_basic_intro.txt` - introduction instructions for the basic course.
- `LL_CC_adapt_intro.txt` - the same thing for the adaptability exercises.
- `LL_CC_velo_intro.txt` - the same for the velocity exercises.
- `LL_CC_fluid_intro.txt` - idem for the fluidness exercises.

The only files whose names do not follow strictly this template are the original ones from English, as the language and country code used for them is just "C". If you want to modify anyone of these files, that is enough to create a copy in the user subdirectory and edit it there. The program first searches in this folder before attempting to open the default one, at the data subdirectory.

3.6 Basic course lessons

The way how are chosen the characters for each basic course lesson is specified by the file called *basic_lessons*, at the data subdirectory. Each lesson has a set of 11 lines, that must be ordered like this:

- Heading (no matter about its contents)
- Four lines specifying the normal characters
- One blank line as separator
- Four lines specifying the shifted characters
- One more blank line as separator

The specification lines are made of 0's and 1's. The 1's mean that the character corresponding to that position must be included in the lesson. Logically, the 0's mean the opposite. The number of lessons is fixed at 43, that is, the file must contain only 43 blocks of 11 lines. Follows as example the listing of the first and the last lessons of the original file.

```
Lesson 01
00000000000000
00000000000000
00010010000000
00000000000000

00000000000000
00000000000000
00000000000000
00000000000000

Lesson 43
00000000000000
11100000000000
00000000010000
11000000111100

00000000000000
11100000000000
00000000010000
11000000111100
```

Be carefull when modifying this file, because you can loose the flexibility of the basic course. If there is no interest on keeping this characteristic, then you just must respect the file format. Remember: 43 lessons!

3.7 Tips about how to use the fingers

The floating hands tips are get also from a text file, similar to the *.kbd* files. But in this case, that is enough to specify a set of only four lines, because the finger for each key is only one, independently of the combination with a shift key. *fingers_position.txt* is the file name to store this mapping between fingers and keys. It must remain in the data subdirectory.

Each digit, from 1 to 9, represents one of the hands fingers, and both thumbs are represented by only one number. The list bellow describes this codification.

1. Fourth finger (pinky), left hand
2. Third finger (ring), left hand
3. Second finger (middle), left hand
4. First finger (index), left hand
5. Thumbs, does not matter of which hand
6. First finger, right hand
7. Second finger, right hand
8. Third finger, right hand
9. Fourth finger, right hand

The file format defines a direct relation between the positions of keys in the *.kbd* files and the positions where the digits (fingers) are put. For an example, here is the listing of the original file:

```
11234466789999
12344667899900
12344667899900
11234466789900
```

4 Internal structure of the program

When reading the C source files and its headers, you must consider that four of them were generated automatically through the program Glade, version 2.6.8. The Glade file defining the interface is the *klavaro.glade*, at the source package tree root. And the derived files are:

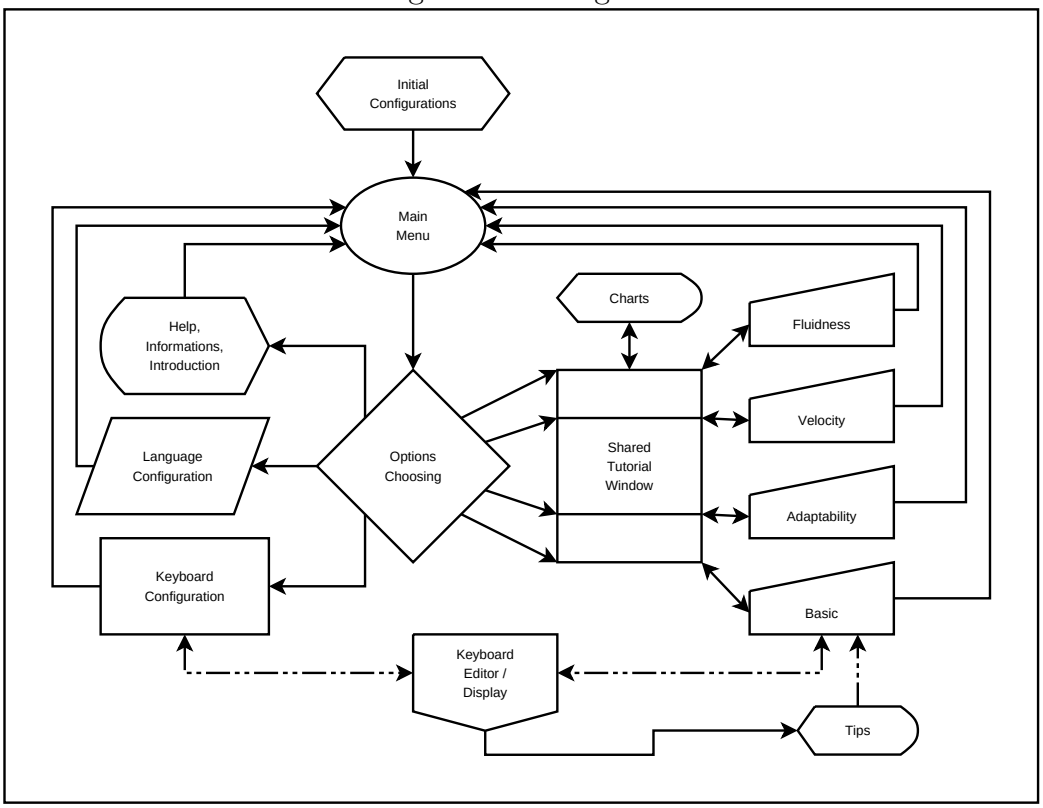
- *main.c* - no comments needed, are they?
- *support.c* / *support.h* - supports the manipulation of *pixmap* (xpm) files, internationalization, etc.
- *interface.c* / *interface.h* - functions to create all the items of the interface.

- *callbacks.c / callbacks.h* - set of callbacks raised from the interface controls or events.

The file that is in a greater extent modified is the *callbacks.c*, because all of the functions therein must be edited and defined after their initial automatic generation.

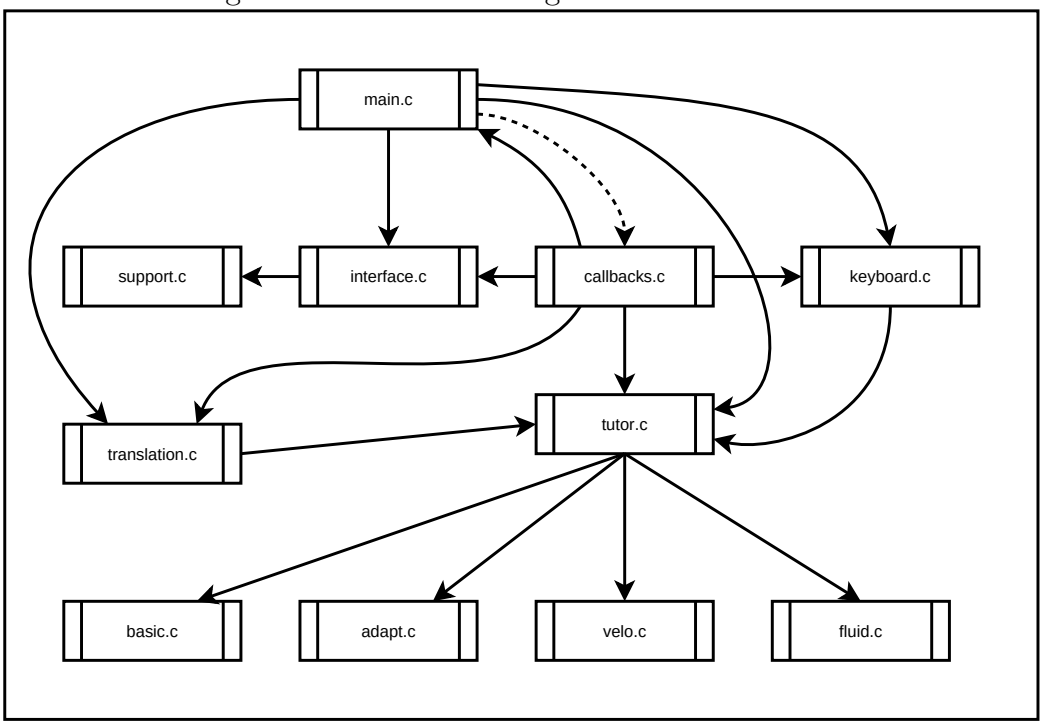
The following two figures can be taken as a starting point to understand how the program code is organized. We suggest that you first get used with the program interface, playing a lot with the application and exploring all of its features.

Figure 1: Fluxogram



A last observation to be made is that the comments in the source code are writen also in English, as this is the commonest language in the computer programming medium nowadays.

Figure 2: Relations among the source code files



As you, English native speaker, could verify reading this text, we do not master the English writing and that is why many errors may be there.

And now answer sincerely: do you think it is fair, my friend?