

# Libbraille: a Portable Library to Easily Access Braille Displays

Sébastien Sablé and Dominique Archambault

INSERM U483 / INOVA — Université Pierre et Marie Curie  
9, quai Saint Bernard, 75 252 Paris cedex 05, France  
`sable@users.sourceforge.net`

**Abstract.** The TiM project intends to develop and to adapt computer games for visually impaired children. In order to achieve this project a library which allows to easily access Braille displays was developed. This library provides all the functions needed to write text on the Braille display, directly raise Braille dots as well as receive the keys pressed on it. On top of that this library works with many different types of displays and is freely reusable.

## 1 Introduction

### 1.1 The TiM Project

The overall purpose of the TiM project (Tactile Interactive Multimedia computer games for visually impaired children) [2,3] is to offer computer games intended for visually impaired young children of various levels of psycho-motor development. These games are planned to be used by the children in an autonomous way, without assistance of a sighted person, like it is the case for sighted children with hundreds of titles.

TiM games will be described in a high level generic language independent of the representation or modality. Those games scripts will be interpreted by a game platform running on most operating systems.

To reach the needs of the children aimed by the TiM project, the platform will be able to render those games to all the specific devices they use, each are corresponding to a specific modality:

- tactile boards,
- Braille displays
- speech synthesisers
- customisable graphical displays (scalable font prints, adjustable colours and contrast...)

### 1.2 Support of Braille Displays

An important issue for the TiM project is to supply the support for as many existing models of Braille displays as possible, that are used currently and that

each child may have, as well as being able to add drivers easily when a new model comes on the market.

Indeed, Braille displays are generally extremely expensive (starting at 5000 Euros). It makes it very difficult for organisations to have several different models to develop drivers and for users to buy a different model.

Since there was no existing standard library to access Braille displays, it appeared that the development of a Braille library (Libbraille [1]) was crucial for the TiM project.



### 1.3 A Collaborative Approach

The high cost of Braille devices, and the fact that many manufacturers do not release publicly the protocol of their devices, explain that there was no standard API for Braille displays. A way to bypass this problem appeared to make this a collective project.

By allowing other people to use and improve the library to suit their own needs, we could share our improvements and finally correctly support a large number of devices.

That is why the code of the library was released under the *GNU Lesser General Public License* (LGPL) [4] which is used by many projects including the famous Linux. This allows people to freely copy, distribute and/or modify the library as long as they provide their enhancements under the same license. Developers who want to use the library can freely link it to their programs without any restriction.

The library was also designed so that it could be as generic as possible. This means that it should be used in any project that requires Braille functionalities.

Indeed other projects had already been done that used Braille displays. Some like *BRLTTY* [5] or *BRASS* [6], were quite advanced, however they were complete screen readers and the Braille functionalities were not independent, so that it was very hard to reuse them in other projects. The protocols used in those programs were integrated in Libbraille when the license allowed that, but with a focus on re-usability.

## 2 Features of the Library

This library is responsible for all the low level interactions with Braille displays. Text can simply be written on the display, or Braille dots can even be directly raised independently. It is also possible to get the keys that have been pressed by a user when the device has such keys.

### 2.1 Architecture

The library is organised in a modular way. A first layer provides a simple API to developers who want to access the library. This is done through a set of functions starting with the `braille_` prefix, like `braille_init`, `braille_display` or `braille_read`.

At initialisation and according to a configuration file, this first layer will load a device dependant module. This module implements the low level interaction with a given Braille terminal protocol and depends on the manufacturer and model of Braille terminal.

This module must use a lower level layer that provides a set of common portable functions to all drivers in order to communicate through the serial port, or to log some debug information.

### 2.2 Internationalisation

The TiM projects aims to access the widest possible concerned population, and in this focus, multilingual features has been integrated from the beginning of the development.

Many different Braille tables are used in different countries and even among different users of the same country! Those tables make the link between the ASCII and Braille representation of a character.

The library already supports the ability to switch between many different tables and a user can easily create his own.

### 2.3 Portability

Another important concern was that the library should not be limited to one system. So it was developed using a very portable ‘C’ code. Then the library works under all versions of MS Windows (as a dll library) but also under most Unix systems (as a shared library). It should be easily ported to other platforms if needed.

It was also very important that the library could be accessed from different languages. Thanks to the “*Simplified Wrapper and Interface Generator*” [7] software, it has been possible to very easily generate bindings with others languages. Currently *Python* [8] is supported and a *Java* binding is under development.

Other languages supported by SWIG (Tcl, Perl, Ruby, Mzscheme or Guile) should be very easily added if needed.

## 2.4 Supported Displays

Here is a list of displays that are supported sorted by manufacturer :

**BAUM** Vario  
**HandyTech** Braille Wave  
**EuroBraille** Scriba  
**ONCE** Eco - Braille 20  
**MDV** MB 408S  
**Pulse Data International** BrailleNote

Some of the models are quite recent like the BrailleWave (HandyTech) or the BrailleNote (PDI), and many of them have completely different features (like the number of cells and the number of keys). The library was designed in order to facilitate design of drivers for new devices.

Other drivers for the following manufacturers are under development or have not yet been physically tested: *Alva, Blazie Engineering, EuroBraille, Tactilog, Tieman, Telesensory Systems Inc, Navigator and Papenmeier.*

## 3 Usage Overview

### 3.1 Initialisation

Interaction with the Braille library is done through functions starting with the `braille_` prefix<sup>1</sup>. Those functions are declared in the `braille.h` header.

The `braille_init` function should be called before any other function of the library. It will load the correct driver then initialise and configure the Braille display.

### 3.2 Displaying a Simple String

The simplest way to write something on the Braille display is to use `braille_display`. It must be called with a string and will display that string on the display. Simple!

```
braille_display("test started");
```

When displaying text, the Braille representation is calculated according to a Braille table which can be customised by the user.

### 3.3 Advanced dots displaying

There is a more complex function to display when a better control of what is displayed is necessary, for example when displaying something other than text. What will be displayed is a combination of text and a filter that directly manipulate dots.

---

<sup>1</sup> The source code of a complete example is available in the annexes.

Then you use `braille_filter` to modify directly some dots. The first argument of `braille_filter` is an unsigned char corresponding to which dots have to be activated.

The  $n^{th}$  dot (as shown on figure 1) can be activated by setting to 1 the  $n^{th}$  bit of the first parameter. So to activate dots 7 and 8, the value 00000011 in binary or  $2^6 + 2^7 = 252$  in decimal must be given. There is a `BRAILLE` macro that calculates this value: `BRAILLE(0, 0, 0, 0, 0, 0, 1, 1) = 252`.



**Fig. 1.** each cell is made of 8 dots numbered in the traditional way like represented on this figure

The second argument is the number of the cell which has to be modified, starting at 0. So in this example, dots 7 and 8 are set on the 24<sup>th</sup> cell of the display

```
braille_filter(BRAILLE(0, 0, 0, 0, 0, 0, 1, 1), 23)
```

Finally, `braille_render` is called. This function filters the text given by `braille_write` with the filter defined through `braille_filter` and send the data to the Braille display.

### 3.4 Typing with the Braille Keyboard

It is also possible to know which keys have been pressed on the Braille display with the `braille_read` function. This function returns a structure of type `brl_key`. This structure has an attribute named `type` concerning the type of key pressed.

If this `type` is `BRL_CURSOR`, the `code` attribute contains the number of the pressed cursor routing key starting at 0.

If the `type` is `BRL_CMD`, then a function key has been pressed on the Braille display. The `code` attribute contains a code depending on the function key. There are many codes which can be found in the `braille.h` header file.

If the `type` is `BRL_KEY`, then the user has pressed a standard ASCII code on its Braille display. The `code` attribute gives the ASCII value.

### 3.5 Stopping the Library

Finally, the `braille_close` function must always be called when closing the Braille library. It will unload the driver, free resources and close the library.

## 4 Current Status and Further Works

Libbraille is already a well working library used in different projects requiring Braille displays support.

The Free Software model appeared to be an excellent model for the development of devices drivers and many users contributed to the project.

Indeed other libraries were created following the same model for the TiM project (libspeech which can drives many speech synthesis and libboard a driver for tactile boards).

The library is far however from supporting all the existing models of Braille displays. This can only be achieved if this library is advertised enough so that owners of those devices can collaborate to improve the library...

The following enhancements are also planned to be included in libbraille:

- Support for more models of displays: it depends on the collaboration of people with those models or of manufacturers
- Java wrapper and in general any language if possible: the aim is to make of libbraille some “universal” Braille library usable by anyone
- Development of a virtual driver: with a virtual Braille keyboard displayed on the screen. This will allow developers to create Braille enabled programs, without owning an expensive Braille terminal for test purpose
- Development of a simple configuration front-end: at this time, the focus has mostly been on the developer aspect of the library. It is planned to improve the user friendliness by adding a simple configuration back-end
- Ability to configure keys layout: this is a medium term objective that should be very useful for users. It will allow users to customise the layout of keys on the Braille keyboard at runtime, depending on the software they wish to use

More information can be found at: <http://libbraille.org>

## Acknowledgements

The TiM project is funded by the European Commission<sup>2</sup>, under the IST (Information Society Technologies) Programme 2000 (FP5/IST/Systems and Services for the Citizen/Persons with special needs) under the reference IST-2000-25298.

## References

1. Libbraille – <http://libbraille.org>
2. Archambault, D. and al. (2000). TiM : Tactile Interactive Multimedia computer games for visually impaired children. European Commission, Information Society Technologies IST-2000-25298.  
<http://www.snv.jussieu.fr/inova/tim/>

---

<sup>2</sup> The contents of this paper is the sole responsibility of the authors an in no way represents the views of the European Commission or its services.

3. D. Archambault, D. Burger, and S. Sablé, “The TiM Project: Tactile Interactive Multimedia computer games for blind and visually impaired children”, in *Assistive Technology – Added Value to the Quality of Life, Proceedings of the AAATE’01 Conference, Ljubljana, Slovenia* (Črt Marinček, C. Bühler, H. Knops, and R. Andrich, eds.), (Amsterdam, The Netherlands), pp. 359–363, IOS Press, Sept. 2001.
4. The GNU Lesser General Public License – <http://www.gnu.org/copyleft/lesser.html>
5. The BRLTTY Project – <http://www.cam.org/nico/brlty>
6. Braille and speech server – <http://www.butenuth.onlinehome.de/blinux/>
7. Simplified Wrapper and Interface Generator – <http://www.swig.org>
8. Python, an interpreted, interactive, object-oriented programming language – <http://python.org>

## Annexes

### Usage Example in C

```
#include <braille.h>

int main()
{
    // Initialising the library
    if(!braille_init()) {
        fprintf(stderr, "Error initialising libbraille\n");
        return 1;
    }

    // Displaying a Simple String
    braille_display("test started");

    // Advanced dots displaying
    char *hello_msg = "more complex test";
    braille_write(hello_msg, strlen(hello_msg));
    // Raising dot 7 and 8 on characters 1 and 24 of the Braille display
    braille_filter(BRAILLE(0, 0, 0, 0, 0, 0, 1, 1), 0);
    braille_filter(BRAILLE(0, 0, 0, 0, 0, 0, 1, 1), 23);
    braille_render();

    // Typing with the Braille Keyboard
    while(1) {
        brl_key key;
        key = braille_read();
        switch(key.type) {
            case BRL_NONE:
                break;
            case BRL_CURSOR:
                printf("cursor: %d\n", key.code);
        }
    }
}
```

```

        break;
    case BRL_CMD:
        switch(key.code) {
            case BRLK_FORWARD:
                printf("reading further right on the display\n");
                break;
            ...
            default:
                break;
        }
        break;
    case BRL_KEY:
        printf("braille: %o, code: %d, char: %c\n",
            key.braille, key.code, key.code);
        break;
    default:
        printf("unknown type %d\n", key.type);
    }
    usleep(100);
}

// Stopping the Library
braille_close();
exit(0);
}

```

### Usage Example in Python

```

$ python
Python 2.1.3 (#1, Apr 11 2002, 00:19:11)
>>> from libbraille import *
>>> braille_init()
libbraille 0.9.0
Processing file: /usr/local/etc/libbraille.conf
Braille device: /dev/ttyS0
Braille driver: libbrailleno (Fake)
Dot Translation Table: french.tbl
Braille display: 1 row of 40 cells.
1
>>> braille_display("hello world!")

```