MBROLA

March 12th, 2000

"It would be a considerable invention indeed, that of a machine able to mimic speech, with its sounds and articulations. I think it is not impossible." Leonhard Euler (1761)

by Vincent Pagel and Thierry Dutoit

Contents

1 MBROLA Sources General condition of use

The source code of MBROLA may only be used to produce the object code sold by your company. It is confidential and should remain safely locked, as well as its documentation.

2 A brief description of MBROLA

MBROLA v3.01 is a speech synthesizer based on the concatenation of diphones. One synthesis channel takes a list of phonemes as input, together with prosodic information (duration of phonemes and a piecewise linear description of pitch), and produces speech samples on 16 bits (linear), at the sampling frequency of the diphone database. It is therefore NOT a Text-To-Speech synthesizer, since it does not accept raw text as input.

It is distributed as a ZIP file whose name respect the format *"mbrXXXX.zip"* where *XXXX* represent the version number (e.g. *"mbr3.01e.zip"*).

It may be compiled in 3 modes depending on which stream drives the process:

- Driven by the input phonetic file: it is compiled as a standalone program named "synth" which outputs audio in a file or a pipe. This mode is a good choice under Unix platforms for end-user applications. In the following we call this mode "***standalone mode***".
- Driven by the audio output: compiled as a library, which outputs audio data into buffers of the size, requested by the main program. This allows you to easily include MBROLA inside your TTS application without temporary file mechanisms. In the following we call this mode "***library mode***".
- Same as above, included in a DLL for Windows95-98/NT (which is of course the preferred mode for Windows platforms). In the following we call this mode "***DLL mode***".

While using *library* or *DLL* mode, we now differentiate one channel and multi channel mbrola. In the first mode, one database is associated to one and only one synthesis channel, which generally fits for end-user applications. In the second mode, one can run many synthesis channel instantiations with one or more Database instances and many phonetic input streams. This second solution is adapted to multi channel telecom TTS applications.

In all those compilation modes MBROLA requires a language/voice database to run properly. For your internal use (i.e. non-commercial) you can test the voices made available on the MBROLA project homepage:

http://tcts.fpms.ac.be/synthesis

Refer to your contract to check your rights for commercial exploitation of the different Diphone Databases.

3 Distribution

Since release 3.01, Mbrola has been transformed into pure ANSI/C code, and object like programming with a strong encapsulation of data (strong because we have respected the fences we put!). One file in the distribution is generally equivalent to one object (pointer on struct). You can find an exhaustive description in the programmer's section 6.

This distribution of MBROLA contains the following files:

Makefile: Unix makefile for Gnu Make (gmake command)

DOCUMENTATION/Programmer/documentation302: this document

DOCUMENTATION/Programmer/HISTORY.txt: history of revisions

DOCUMENTATION/User/readme.txt: standalone version manual

Database: handling of different database formats

Database/database.c: functions to read diphones in the speech database

Database/database.h

Database/database_bacon.c: functions to read compressed diphone databases

Database/database_bacon.h

Database/database_old.c: functions to read diphone databases older than 2.06

Database/database_old.h

Database/diphone_info.c: description of the diphone structures

Database/diphone_info.h

Database/hash_tab.c: hash table of DiphoneInfo (access to the diphone database)

Database/hash_tab.h

Database/little_big.c: handles the little and big endian numeric conversions

Database/little_big.h

Database/rename_list.c: list of phoneme pairs (used for renaming and cloning)

Database/rename_list.h

Parser: functions to read phonemes in the input stream

Parser/fifo.c: First In First Out with chars

Parser/fifo.h

Parser/input.h: define abstract input stream

Parser/input_fifo.c: **i**nstantiation of input.h with Fifo

Parser/input_fifo.h

Parser/input_file.c: instantiation of input.h with File

Parser/input_file.h

Parser/parser.h: define abstract phoneme parser

Parser/parser_input.c: instantiation of parser.h with Input

Parser/parser_input.h

Parser/phonbuff.c: handle a phoneme buffer for pitch interpolation

Parser/phonbuff.h

Parser/phone.c: phoneme type

Parser/phone.h

Engine: Mbrola synthesis engine

Engine/diphone.c: diphone with info for synthesis

Engine/diphone.h

Engine/mbrola.c: mbrola algorithm (Ola, Smoothing...)

Engine/mbrola.h

Misc: Miscellaneous functions basically unrelated to synthesis

Misc/audio.c: audio output and audio file header (au, wav, aiff, raw)

Misc/audio.h

Misc/common.c: useful little functions (uppercase, swab...)

Misc/common.h

Misc/g711.c: G711 audio coding (ALAW and MULAW)

Misc/g711.h

Misc/incdll.h: external definitions used outside of the Mbrola package

Misc/mbralloc.c: memory allocators are here and ONLY here

Misc/mbralloc.h

Misc/vp_error.c: deals with fatal error and warnings

Misc/vp_error.h: macros for debugging purposes

Standalone: Standalone compilation front-end

Standalone/Posix

Standalone/Posix/getopt.c: provided for non-POSIX Unixes

Standalone/Posix/getopt.h

Standalone/synth.c: front-end for the compilation in the standalone mode. Main()

Standalone/synth.h

LibOneChannel: library providing one MBROLA synthesis channel

LibOneChannel/demo1.c: small demonstration program running with the library

LibOneChannel/demo1b.c: small demo showing error handling with the library

LibOneChannel/onechannel.c: library providing one mbrola channel at a time

LibOneChannel/onechannel.h

LibOneChannel/lib1.c: wrapper file to build the library lib1.c (mono channel)

LibMultiChannel: library for multi MBROLA synthesis channel for telecom

LibMultiChannel/multichannel.c: many synthesis channel from one dba

LibMultiChannel/multichannel.h

LibMultiChannel/demo2.c: demo using lib2

LibMultiChannel/lib2.c: wrapper file to build the library lib2.c (multi channel)

VisualC++: compilation projects for Microsoft Visual C++

VisualC++/DLL: Visual C++ project to build the DLL

VisualC++/DLL_USE: sample program using the DLL

VisualC++/Standalone: Visual C++ project to build a standalone binary

Bin: directory containing the output of the compilation with Make under Unix architectures.

4 Installation and Tests

4.1 On Unix

You must first unzip the distribution file mbrXXXX.zip where XXXX stand for the version number:

unzip mbrXXXX.zip

Mbrola can be compiled with the 'gmake' (gnu make) command on the following platforms:

- SUN Sparc 5/S5R4 (Solaris2.4)
- HPUX9.0 and HPUX10.0
- VAX/VMS V6.2 (V5.5-2 won't work)
- DECALPHA(AXP)/VMS 6.2
- AlphaStation 200 4/233
- AlphaStation 200 4/166
- IBM RS6000 Aix 4.12
- PC/LINUX 1.2.11
- PCPentium120/Solaris2.4

- OS/2
- BeBox
- QNX OS

Though, as Mbrola is written in standard ANSI/C, we also support POSIX compliant UNIX Platforms. Please send acknowledgment when Mbrola works on a machine/system not listed here. Before you compile anything you must define some symbols depending on the architecture you're working with:

LITTLE_ENDIAN ® 80x86 based platforms

BIG_ENDIAN ® motorola or HP based platforms

VMS ® VAX/VMS stations

DOS ® PC80x86 with Dos or Windows

SUN4 ® old Sun4 stations (not Posix compliant)

DEBUG ® Huge debugging flag

DEBUG_HASH ® Runs the database and print info about the hash table management (used to tune the memory management of the database)

According to the compilation mode you wish, you can comment or uncomment following lines of Makefile :

```
#CFLAGS += -DDEBUG
#CFLAGS += -DDEBUG_HASH
#CFLAGS += -DLITTLE_ENDIAN
CFLAGS += -DBIG_ENDIAN
```

You can add any definitions to the CFLAGS (compilation flags) variable of the Makefile, as in the following example:

optimized compilation on a Sun Station :

```
CFLAGS= -Wall -DBIG_ENDIAN -O6
```

debug mode on a VAX/VMS :

```
CFLAGS= -Wall -DLITTLE_ENDIAN -DVMS -g -DDEBUG
```

By default the compiler is set with CC = gcc ; though on many platforms cc may also work. As the hardware manufacturer generally provides cc, it is preferred when possible since the object code performance can be higher by an order of magnitude. You can type :

"make" or "make all" to generate the 'synth' binary (standalone mode).

"make clean" removes the entire object files and binaries.

"make lib1" compiles lib1.c in the library mode (one channel synth)

"make demo1" builds a demo exemplifying the use of lib1

"make lib2" compiles lib1.c in the library mode (multi channel synth)

"make demo2" builds a demo exemplifying the use of lib2

"make tags" to build Emacs popular tags (helps finding your way through the code with ESC-.). SUN Workshop uses an internal btags program for that purpose.

The intermediate object code goes into a Bin directory that is created on the occasion.

4.2 On PCs/Dos

On PC/Dos platforms, use "pkunzip synthXXXX.zip" to restore the files (don't forget to restore the embedded paths in the archive). Mbrola can be compiled with Microsoft Visual C++ (4 .0 or higher), or Borland C++ (4 .5 or higher), on the following platforms:

- PC486/DOS6 (but other PC/DOS should do, too)
- PC486/Windows 3.1
- PC486/Windows 95
- PC-Pentium/Windows 98
- PC-Pentium/Windows NT

Always check that in your project the following preprocessor directives are defined: LITTLE_ENDIAN and DOS. A project to build such a release with Visual C++ is provided under VisualC++/Standalone.

4.3 On PC/Windows

First proceed like for the PC/DOS platforms. Once synthXXXX is installed you can start building a DLL in the VisualC++\DLL directory. MbrolaDll.dsw is a Microsoft VisualC++ 5.0 project file to build a DLL. In any project you make to build a DLL with Mbrola don't forget to define the DLL, LITTLE_ENDIAN, DOS preprocessor definitions.

The Mbrola source files and a wrapper DLL interface is included in the project, it should compile smoothly. In case you have to build a new project from scratch remember that you should include only file from either LibOneChannel/ or LibMultiChannel/. Never include files from Standalone/, as this directory is only relevant for a standalone mode (see section above for an exe binary).

Several compilation modes are available, the "Win32 Bacon Static" is a good one to start with (Bacon compression scheme is included, DLL are statically linked).

In the directory VisualC++/DLL_USE , little sample programs are given that use the Mbrola DLL.

4.3.1 Black magic

There is a strange bug in Visual C++ 5.0, when you compile the project you sometime get:

```
Linking...
nafxcw.lib(dllmodul.cbj) : error LNK2005: _DllMain@12 already defined in LIBCMT.lib(dllmain.cbj)
nafxcw.lib(afxmem.cbj) : error LNK2005: "void * __cdecl operator new(unsigned int)" (??2@YAPAXI@Z) already defined in LIBCMT.lib(new.cbj)
nafxcw.lib(afxmem.cbj) : error LNK2005: "void __cdecl operator delete(void *)" (??3@YAXPAX@Z) already defined in LIBCMT.lib(delete.cbj)
nafxcw.lib(dllmodul.cbj) : warning LNK4006: _DllMain@12 already defined in LIBCMT.lib(dllmain.cbj); second definition ignored
nafxcw.lib(afxmem.cbj) : warning LNK4006: "void * __cdecl operator new(unsigned int)" (??2@YAPAXI@Z) already defined in LIBCMT.lib(new.cbj); second definition ignored
nafxcw.lib(afxmem.cbj) : warning LNK4006: "void __cdecl operator delete(void *)" (??3@YAXPAX@Z) already defined in LIBCMT.lib(delete.cbj); second definition ignored
Creating library MbrolaDl/Mbrola.lib and object MbrolaDl/Mbrola.exp
Output\Release_Static\Mbrola.dll : fatal error LNK1169: one or more multiply defined symbols found
Error executing link.exe.
Mbrola.dll - 4 error(s), 7 warning(s)
```

Solution: remove one file from the project and include it again in the list of source files, and build the project again. The problem vanishes.

4.4 Using the standalone binary

You are now ready to test the program. First try: "synth" to get an information screen about the copyright. Then, for a help screen on how to use the standalone version of the software, try :

```
synth -h
```

You get a help screen like the following:

```
> USAGE: ./synth [COMMAND LINE OPTIONS] database pho_file+ output_file
>
>A - instead of pho_file or output_file means stdin or stdout
>Extension of output_file ( raw, au, wav, aiff ) tells the wanted audio format
>
> Options can be any of the following:
> -i = display the database information if any
> -e = IGNORE fatal errors on unknown diphone
> -c CC = set COMMENT char (escape sequence in pho files)
> -F FC = set FLUSH command name
> -v VR = VOLUME ratio, float ratio applied to ouput samples
> -f FR = FREQ ratio, float ratio applied to pitch points
> -t TR = TIME ratio, float ratio applied to phone durations
> -l VF = VOICE freq, target freq for voice quality
> -R RL = Phoneme RENAME list of the form a A b B ...
> -C CL = Phoneme CLONE list of the form a A b B ...
>
> -I IF = Initialization file containing one command per line
> CLONE, RENAME, VOICE, TIME, FREQ, VOLUME, FLUSH,
> COMMENT, and IGNORE are available
```

Now in order to go further, you need to get a version of an MBROLA language/voice database from the MBROLA project homepage. Let us assume you have copied the FR1 database and referred to the accompanying fr1.txt file for its installation. Then try:

```
synth fr1/fr1 fr1/TEST/bonjour.pho bonjour.wav
```

it uses the format:

```
synth diphone_database command_file1 command_file2 ... output_file
```

and creates a sound file for the word 'bonjour' (Hello! in French)

Basically the output file is composed of signed integer numbers on 16 bits, corresponding to samples at the sampling frequency of the MBROLA voice/language database (16 kHz for the diphone database supplied by the authors of MBROLA : Fr1). MBROLA can produce different audio file formats: .au, .wav, .aiff, .aif, and .raw files depending on the ouput_file extension. If the extension is not recognized, the format is RAW (no header). We recommend .wav for Windows, and .au for Unix platforms. To display information about the phoneme set used by the database, type:

```
synth -i fr1/fr1
```

It displays the phonetic alphabet as well as copyright information about the database.

Option -e makes Mbrola ignore wrong or missing diphone sequences (replaced by silence) which can be quite useful when debugging your TTS. Equivalent to "IGNORE" directive in the initialization file (N.B replace the obsolete ;;E=OFF , unsupported in .pho file).

4.4.1 Changing the pitch

Optional parameters let you shorten or lengthen synthetic speech and transpose it by providing optional time and frequency ratios:

```
synth -t 1.2 -f 0.8 -v 0.7 fr1/fr1 TEST/bonjour.pho bonjour.wav
```

or its equivalent in the initialization file:

```
TIME 1.2
FREQ 0.8
```

for instance, will result in a RIFF Wav file bonjour.wav 1.2 times longer than the previous one (slower rate), and containing speech in which all fundamental frequency values have been multiplied by 0.8 (sounds lower). You can also set the values of these coefficients directly in a .pho file by adding special escape sequence like :

```
;; F=0.8
;; T=1.2
```

You can change the voice characteristics with the -l parameter. If the sampling rate of your database is 16000, indicating -l 18000 allows you to shorten the vocal tract by a ratio 16/18 (children voice, or women voice depending on the voice you're working on). With -l 10000,you can lengthen the vocal tract by a ratio 18/10 (namely the voice of a Troll). The same command in an initialization file becomes "VOICE 10000".

Option **-v** gives a VolumeRatio that multiplies each output sample. In the example below, each sample is multiplied by 0.7 (the loudness goes down). Warning: setting VolumeRatio too high generates saturation.

```
synth -v 0.7 fr1/fr1 TEST/bonjour.pho bonjour.wav
```

or add the line **"VOLUME 0.7"** in an initialization file

The **-c** option lets you specify which symbol will be used as an escape sequence for comments and commands in .pho files. The default value is the semi-colon ';', but you may want to change this if your phonetic alphabet use this symbol, like in:

```
synth -c ! fr1/fr1 TEST/test1.pho test2.pho test.wav
```

equivalent to "**COMMENT !**" in an initialization file

The **-F** option lets you specify which symbol will be used to Flush the audio output. The default value is #, you may want to change the symbol like in:

```
mbrola -F FLUSH_COMMAND fr1/fr1 test.pho test.wav
```

equivalent to "**FLUSH FLUSH_COMMAND**" in the initialization file.

4.4.2 Using Pipes

A - instead of command_file or output_file means stdin or stdout. On multi-tasking machines, it is easy to run the synthesizer in real time to obtain audio output from the audio device, by using pipes.

4.4.3 Renaming and Cloning phonemes

It may happen that the language-processing module connected to MBROLA doesn't use the same phonemic alphabet as the voice used. The Renaming and Cloning mechanisms help you to quickly solve such problems (without adding extra CPU load). The only limitation about phoneme names is that they can't contain blank characters.

If, for instance, phoneme a in the mbrola voice you use is called my_a in your alphabet, and phoneme b is called my_b, then the following command solves the problem:

```
synth -R "a my_a b my_b" fr1/fr1 test.pho test.wav
```

You can give as many renaming pairs as you want. Circular definition is not a problem. E.g. "**a b b c**" will rename original *[a]* into [b] and original *[b]* into *[c]* independently *([a]* won't be renamed to *[c]*).

LIMITATION: you can't rename a phoneme into another that already exists.

The cloning mechanism does exactly the same thing, though the old phoneme still exists after renaming. This is useful if you have 2 allophones in your alphabet, but the Mbrola voice only provides one.

Imagine for instance, that you make the distinction between the voiced [r] and its unvoiced counterpart [r0] and that you are using a syllabic version [r=]. If as a first approximation using [r] for both is OK, then you may use an Mbrola voice that only provides one version of [r] by running:

```
synth -C "r r0 r r=" fr1/fr1 test.pho test.wav
```

which tells the synthesizer that [r0] and [r=] should be both synthesized as [r]. You can write a long cloning list of phoneme pairs to fit your needs.

Renaming and cloning eats CPU since the complete diphone hash table has to be rebuilt, but once the renaming or cloning has occurred there is absolutely NO RELATED PERFORMANCE DROP. So using this feature is more efficient than a pre-processor is, though a simple phoneme mapping cannot always solve incompatibilities.

Before renaming anything as #, check section 5.1.2

When one has long cloning and renaming lists, you can conveniently write them into an initialization file according to the following format:

```
RENAME a my_a
RENAME b my_b
CLONE r r0
CLONE r r=
```

The obsolete "**;; RENAME a my_a**" can't be used in .pho file anymore, but is correctly parsed in initialization files. Note to EN1 and MRPA users: the consequence of the change above is that you must change the previous call format "*mbrola en1 en1mrpa...*" into "*mbrola -I en1mrpa en1 ...*".

4.5 Machine dependant hints for best using Mbrola

4.5.1 On MSDOS

With the standalone version, generating wav files is easier:

```
synth fr1/fr1 TEST/bonjour.pho bonjour.wav
```

Then you can play the RIFF Wav file with your favorite DOS or Windows sound utility. On OS/2 pipes may be used just like below.

4.5.2 On modern Unix systems such as Solaris or HPUX or Linux

Type:

```
synth fr1 bonjour.pho -.au | audioplay
```

where audioplay is your audio file player (* the name vary with the platform, e.g. splayer for HPUX *).

If your audioplayer has problems with sun .AU files, try with .wav or .raw. Never use .wav format when you pipe the output (mbrola can't rewind the file to write the audio size in the header). Wav format was not developed for Unix (on the contrary Au format let you specify in the header "we're on a pipe, read until end of file").

NOTE FOR LINUX: you can use the GPL rawplay program provided at

ftp://tcts.fpms.ac.be/pub/mbrola/pclinux/

4.5.3 On Sun4 (old audio interface)

Those machines are now quite old and only provide a mulaw 8Khz output. A hack is:

```
synth fr1 input.pho - | sox -t raw -sw -r 16000 - -t raw -Ub -r
8000 - > /dev/audio
```

Provided you have the public domain sox utility developed by Ircam, you should hear *'bonjour'* without the need to create intermediate files. Note that we strongly recommend that you DON'T use SOX, since its resampling method (linear interpolation) will permanently damage the sound.

Other solution: The UTILITY.ZIP file available from the MBROLA home-page provides RAW2SUN that does this conversion.

4.5.4 On VAX or AXP workstations

To make it easier for users to find MBROLA, you should add the following command to your system startup procedure:

```
$ DEFINE/SYSTEM/EXEC MBROLA_DIR disk:[dir]
```

where "disk:[dir]" is the name of the directory you created for the MBROLA_DIR files. You could also add the following command to your system login command procedure:

```
$ MBROLA :== $MBROLA_DIR:MBROLA.EXE
$ RAW2SUN :== $MBROLA_DIR:RAW2SUN.EXE
```

to use the decsound device:

```
$ MCR DECSOUND - volume 40 -play sound.au
```

See also the MBR_OLA.COM batch file in the UTILITY.ZIP file available from the MBROLA Homepage if you cannot play 16 bits sound files on your machine.

4.6 Default Parser Manual

The default parser is the parser that was provided before release 3.01. Implicitly it means that you can replace it with your own one, thanks to the setParser_MBR function. Basically the work of the parser is to return to Mbrola a phoneme with a length, and its pitch points.

We provide a default parser that allows you to give optional pitch points, the intonation curve being linearly interpolated between those points.

4.6.1 Input file format

Example of a command line :

```
synth fr1/fr1 bonjour.pho bonjour.wav
```

For example the phonetic input file bonjour.pho simply contains :

```
; Bonjour
_ 51 25 114
b 62
o~ 127 48 170.42
Z 110 53.5 116
u 211
R 150 50 91
_ 91
```

This shows the format of the input data required by MBROLA. Each line contains a phoneme name, a duration (in ms), and a series (possibly none) of pitch pattern points composed of two float numbers each: the position of the pitch pattern point within the phoneme (in % of its total duration), and the pitch value (in Hz) at this position.

Hence, the second line of bonjour.pho :

```
_ 51 25 114
```

tells the synthesizer to produce a silence of 51 ms, and to put a pitch pattern point of 114 Hz at 25% of 51 ms. Pitch pattern points define a piecewise linear pitch curve. Notice that the pitch pattern they define is continuous, since the program automatically drops pitch information when synthesizing unvoiced phones.

Blank characters or tabs separate the data on each line. Comments can optionally be introduced in command files, starting with a semi-colon '**;**'. This default can be overrun with the **-c** option of the command line.

Another special escape sequence '**;;**' allow the user to introduce commands in the middle of .pho files as described below. This escape sequence is also affected by the **-c** option.

4.6.2 Changing the Frequency Ratio or Time Ratio

A command escape sequence containing a line like **"T=x.x"** modifies the time ratio to **x.x**, the same result is obtained on the fundamental frequency by replacing T with F, like in:

```
;; T = 1.2
;;F=0.8
```

4.6.3 Flush the output stream

Note, finally, that the synthesizer outputs chunks of synthetic speech determined as sections of the piecewise linear pitch curve. Phones inside a section of this curve are synthesized in one go. The last one of each chunk, however, cannot be properly synthesized while the next phone is not known (since the program uses diphones as base speech units). When using mbrola with pipes, this may be a problem. Imagine, for instance, that mbrola is used to create a pipe-based speaking clock on a HP:

```
speaking_clock | mbrola fr1 - -.au | splayer
```

which tells the time, say, every 30 seconds. The last phone of each time announcement will only be synthesized when the next announcement starts. To bypass this problem, mbrola accepts a special command phone, which flushes the synthesis buffer : "#"

This default character can be replaced by another symbol thanks to the command:

```
;; FLUSH new_flush_symbol
```

Another important issue with piping under UNIX, is the possibility to prematurely end the audio output, if for example the user presses the stop button of your application. Since release 3.01, Mbrola handles signals.

If in the previous example the user wants to interrupt the speaking clock message, the application just needs to send the USR1 signal. You can send such a signal from the console with:

```
kill -16 mbrola_process_number
```

Once mbrola catches the signal, it reads its input stream until it gets EOF or a FLUSH command (hence, surrounding sections with flush is a good habit).

4.6.4 Limitations of MBROLA

There is no more limitation on the number of pitch points one can assign to a phoneme, or on the number of phonemes without pitch points. There is no more limitation on extra low pitch (sometime used to produce vocal fry).

Phonemes can be synthesized with a maximum duration that depends on the fundamental frequency with which they are produced. The higher the frequency, the lower the duration. For a frequency of 133 Hz, the maximum duration is 7.5 sec. For a frequency of 66.5 Hz, is 5 sec. For a frequency of 266 Hz, is 3.75 sec.

5 Programmer's Manual

First, we describe in this section the object oriented philosophy used since release 3.01.

5.1 Philosophy and architecture

Actually nothing (or nearly nothing) prevents us to program in standard C/ANSI with an object like convention which authorize:

1. "weak" encapsulation
2. Inheritance
3. Polymorphism

5.2 Encapsulation of Object's attributes

Let's exemplify the programming conventions with the char Fifo found in Parser/fifo.h. First we define a structure describing a Fifo.

```
typedef struct
{
char* charbuff;          /* circular buffer for phonetic input */
int buffer_pos;                /* Current position */
int buffer_end;                /* Last available phoneme */
int buffer_size;               /* number of chars in Phobuffer */
} Fifo;
```

To make distinction between public and private data, the convention is to never directly access the features of a Fifo out of its fifo.c implementation file. To reach this goal we exclusively access members through function-like macros.

```
#define charbuff(ff) ff->charbuff
#define buffer_pos(ff) ff->buffer_pos
#define buffer_end(ff) ff->buffer_end
#define buffer_size(ff) ff->buffer_size
```

It allows the following:

```
Fifo* my_fifo;
..
int length= buffer_size(my_fifo);
```

The programmer should not cheat to discover whether buffer_size is a function or a macro, thus encapsulating the data and making them independent of the Fifo's real implementation (modulo a complete recompiling). C is not C++ and your compiler won't be able to carry out strong type checking just as with inline functions, that's the reason why attributes don't respect the full convention below (according to our conventions we should have use the name buffer_size_Fifo()).

The methods always respect the format: functionname_ObjectName just like below and take a pointer on the object as a first argument. Methods beginning with init are always constructor, and those beginning with close are destructors:

```
Fifo* init_Fifo(int size);
/*
  * Constructor with size of the buffer
  */
void close_Fifo(Fifo* ff);
/*
  * Release the memory
  */
void reset_Fifo(Fifo* ff);
/*
  * Forget previously entered data in the circular buffer
  */
int write_Fifo(Fifo* ff, char *buffer_in);
/*
  * Write a string of phoneme in the input buffer
  * Return the number of chars actually written
  */
int readline_Fifo(Fifo* ff, char *line, int size);
/*
  * Read a line from the input stream in a circular buffer
  * Return 0 if there's nothing to read
  */
```

5.2.1 Inheritance and Polymorphism

Inheritance alone can always be simulated through the is_a_client_of relation, the most interesting case being polymorphism. Polymorphism is interesting for multiple format database handling, and live input parser definition inside of the synthesizer.

The abstract type below specifies an Input object providing the methods close, reset and readline .

```
typedef struct Input Input;
typedef int (*readline_InputFunction)(Input* in, char *line, int
size);
typedef void (*close_InputFunction)(Input* in);
typedef void (*reset_InputFunction)(Input* in);
struct Input
{
void* self;
readline_InputFunction readline_Input;
close_InputFunction close_Input;
close_InputFunction reset_Input;
};
```

This type can be derived into Input_File (the input stream is a file) or Input_Fifo (the input stream comes from a Fifo as described above). The part of the object corresponding to the features overloaded on the basic Input type is stored in the self part.

```
#include "input.h"
#include "fifo.h"
static int readline_InputFifo(Input* in, char *line, int size)
{ return( readline_Fifo((Fifo*) in->self,line,size) ); }
static void reset_InputFifo(Input* in)
{ reset_Fifo((Fifo*) in->self); }
static void close_InputFifo(Input* in)
{ MBR_free(in); }
Input* init_InputFifo(Fifo* my_fifo)
{
Input* self= (Input*) MBR_malloc( sizeof(Input) );
self->self= (void*) my_fifo;
self->readline_Input= readline_InputFifo;
self->close_Input= close_InputFifo;
self->reset_Input= reset_InputFifo;
return self;
}
```

5.2.2 Inheritance and cross-reference graph

The Database, Input and Parser objects contain deferred (=virtual) methods and thus allow polymorphism.

6 Application Programming Interface

The explanations given in the previous section are particularly useful to the user who wants to design ad-hoc parsers. Though one can keep on working with the default parser.

6.1 One channel mode

You can build a demo by running "**make demo1**" under Unix, or simply build the library with "make lib1". With Windows and Visual C++ the DLL project builds an equivalent of lib1, and numerous examples are provided in the DLL_USE directory. The complete one channel mode interface is given section 7.24. Let's exemplify the use below:

First, initialize the engine with a diphone database. All the functions in the API return an error code. A negative value means there was a flaw during the process, in case of error, an explicit error message can be obtained from **lastErrorStr_MBR**().

```
err_code= init_MBR("h:/mbrola/database/fr1" );
if (err_code<0)
    handle_error();
```

If the default parser is plugged, one can use the regular syntax in write_MBR to send phonemes to the engine:

```
if ( ( write_MBR("_ 51 \n b 62 \n") < 0) ||
( write_MBR("o~ 127 50 170 \n Z 110\n") <0) ||
( WriteSpeechFile(output)<0) ||
( write_MBR("u 211 100 200\n R 150 \n_ 9\n#\n") < 0) ||
( WriteSpeechFile(output)<0) )
      handle_error();
close_MBR();
```

Each time one calls **init_MBR**(), one should call a pending **close_MBR**() to release allocated memory. Once **close_MBR**() is called, one can call init_MBR() for a brand new database. If one wish to work with the same database but forget previously entered phonemes, then use **reset_MBR**().

Let's describe how **WriteSpeechFile** works:

```
int WriteSpeechFile(FILE *output)
{
int i;
while ( (i=readtype_MBR(buffer, 16000, LIN16)) == 16000)
  fwrite(buffer, 2, i, output);
if (i>0)
{ /* write last chunk */
  fwrite(buffer,size,i,output);
  return 0;
}
else
  return i; /* return an error code */
}
```

It reads sample buffers from the engine until it can't get any more (**readtype_MBR** returns 0), or an error occurs. Readtype can return 0 for two reasons: either a flush has been encountered, either we don't have enough data in the default parser, as it needs a look ahead to interpolate pitch values. This is the case after **write_MBR**("o~ 127 50 170 \n Z 110\n"), synthesis on the /Z/ can't be carried out until we get the pitch point on "u 211 100 200". This way asynchronous read/write operations are allowed.

The small error handling function simply does:

```
void handle_error()
{
  char err[255];
  lastErrorStr_MBR(err,sizeof(err));
  printf("Code %i\n%s\n", lastError_MBR(), err);
  exit(-1);
}
```

At any time, one can use the get_* and set_* functions to modify internal parameters of the synthesizer.

Important note about the vocal tract length capabilities: one can modify the size of the speaker's throat with **setFreq_MBR**. The lower this frequency, the deeper the voice. This very simple method takes advantage of the playback sampling rate to shift the formants up and down, just like when changing the speed of a tape player. Thus, to be effective, any call to **setFreq_MBR** must be accompanied with a call to the audio hardware setting the requested playback sample rate. Otherwise the speed and pitch will sound odd.

6.1.1 Multi channel mode

One can build a demo by running "**make demo2**" under Unix, or simply build the library with "**make lib2**". The complete multi channel mode interface is given section 7.25.

It looks strangely close to the one channel mode, except that one passes a pointer to a synthesizer structure for every function. Another point is that it doesn't hide any more the parser's details to the user. Thus if one wants to use the default parser, one has to effectively build it.

The following code build 3 independent default phoneme parsers:

```
/* Input Fifo with a buffer of 100 chars */
fifo1= init_Fifo(100);
fifo2= init_Fifo(100);
fifo3= init_Fifo(100);

/* Input stream of the synthesizer */
input1= init_InputFifo(fifo1);
input2= init_InputFifo(fifo2);
input3= init_InputFifo(fifo3);
/* Plug the fifos on the default parsers */
parser1= init_ParserInput(input1,"_",120.0,";",1.0,1.0);
```

```
parser2= init_ParserInput(input2,"_",120.0,";",1.0,1.0);
parser3= init_ParserInput(input3,"_",120.0,";",1.0,1.0);
```

To use one's own parser, see the next section. Once this is done, as many databases as synthesis channels must be opened (let's say 3 channels in this example).

```
Database* main_dba= init_DatabaseMBR2(argv[1],NULL,NULL);
if (!main_dba)
  handle_error(True);
```

Of course opening 3 or more times the same database would spoil a lot of memory since many internal structures could be shared. Instead of using **init_DatabaseMBR2** one can clone an already opened database:

```
Database* clone_dba1= copyconstructor_DatabaseMBR2(main_dba);
Database* clone_dba2= copyconstructor_DatabaseMBR2(main_dba);
Database* clone_dba3= copyconstructor_DatabaseMBR2(main_dba);
```

Cloned database just behave like regular Database, i.e. their destructor must be called before leaving. Once we have a Parser input and a Database, we can open a synthesis channel:

```
Mbrola* channel1= init_MBR2(clone_dba1,parser1);
Mbrola* channel2= init_MBR2(clone_dba2,parser2);
Mbrola* channel3= init_MBR2(clone_dba3,parser3);
```

In this particular example, one can write phonemes in the parser, and read samples from the synthesis engine with instructions such as:

```
write_Fifo(fifo1,"_ 51 \n b 62 \n o~ 100\n Z 120")
while ((i=readtype_MBR2(channel1, buffer, 16000, LIN16))==16000)
  fwrite(buffer,size,i,output);
```

Of course the call to write_Fifo is completely dependent of the fact that this example uses the default phoneme parser. In this particular case, the polymorphic object Parser, which was passed to the constructor of channel, reads its input data from Fifo1.

6.1.2 Designing and plugging your own parser

The user can write his own implementation of a Parser, as long as it follows the definition of Parser/parser.h. The file parser_simple.c below gives an example of a parser that reads phonetic inputs with the format: Phoneme Duration Pitch_At_0% Pitch_At_100%.

In practice this example does not take into account that the Engine synthesize diphones. As the word states, a diphone is made of two phonemes, thus one must know both parts of the diphones to utter it. Thus each phoneme file being used with parser_simple must end with two silences: the first one reveal 1st half of the last phoneme, and the second one reveal the second half (a complete example is provided in VisualC++/DLL_USE/mbrola/parser_simple.cpp). Many people forget to include the second silence as the result sounds correct without. Though, the total length of the synthetic message won't agree with the requested one.

```
    /*
    * FPMs-TCTS SOFTWARE LIBRARY
    *
    * File: parser_simple.c
    * Purpose: parse a simple "pho file" (demonstration of the mbrola
DLL)
    * Instanciation of parser.h
    *
    * Author: Vincent Pagel
    * Email : mbrola@tcts.fpms.ac.be
    *
    * Copyright (c) 1995-2018 Faculte Polytechnique de Mons (TCTS
lab)
    *
    * 18/09/98 : Created
    */

    #include <stdio.h>
    #include "mbrola.h"
    #include "parser_simple.h"

    static void reset_ParserSimple(Parser* parse)
    {
    /* nothing to do */
    fseek( (File*) parse->self,0,SEEK_SET);
    }

    static StatePhone nextphone_ParserSimple(Parser* parse, LP-
PHONE* ph)
    {
      char phoneme[255]; /* phoneme name */
      float length; /* length in milliseconds */
      float pitch0; /* pitch at 0% */
      float pitch100; /* pitch at 100% */

     if ( fscanf( (FILE*)parse->self," %s %f %f %f ",phoneme,&length,&pitch0,&pitch100
) ==4 )
      {
          *ph= init_Phone(phoneme,length);
           appendf0_Phone(*ph, 0.0 , pitch0);
           appendf0_Phone(*ph, 100.0, pitch100);
           return PHO_OK;
      }
    else
      {
            return PHO_EOF;
```

```
  }
}

static void close_ParserSimple(Parser* parse)
        /* Destructor */
{
  fclose( (FILE*) parse->self);
  free(parse);
}

Parser* init_ParserSimple(char* input_name)
/*
  * Constructor of the parser. Parse a text file of the form
 * PHONEME LENGTH PITCH_AT_BEGINNING PITCH_AT_END
  */
{
  FILE* input;
  Parser* parse;

  /* open the text file */
  input=fopen(input_name,"rt");

  if (!input)
        return NULL;

  parse= (Parser*) MBR_alloc( sizeof( struct Parser) );
  parse->reset_Parser= reset_ParserSimple;
  parse->close_Parser= close_ParserSimple;
  parse->nextphone_Parser= nextphone_ParserSimple;
  parse->self= (void*) input;
  return(parse);
}
```

6.2 Mbrola architecture

In following chapters the exported functions and variables of all the source files in the project are described. After the file descriptions, a symbol index is provided to allow fast localization of any function, variable or define.