How to build MPTK with Cmake

SUMMARY

Read this document to learn how to build the Matching Pursuit Tool Kit on Unix platforms and MAC OS X.

LAYOUT

1. Getting started

1.1. Required tools

- Cmake : at least version 2.4 [for further informations see appendix 5.1]
- gcc : tested on version at least 4.1.1
- flex: tested on version at least 2.5.4

1.2. Required packages

- Libsndfile: at least version 1.0.11 [for further informations see appendix 5.2]
- fftw: at least version 3.0.1 [for further informations see appendix 5.3]
- mptk: at least version 0.5.3

2. Basic build

Suppose you untar MPTK-0.5.3.tar.gz in directory ~/foo and you want to build MPTK binary and library files in ~/bar.

The Command list you should use will look like this:

```
bash-3.1$ mkdir ~/bar
bash-3.1$ cd ~/bar
bash-3.1$ cmake ~/foo
bash-3.1$ make
bash-3.1$ make install
```

When using the **cmake** command to generate the build system, Cmake performs a list of tests to determine the system configuration and manage the build system. If the configuration is correct then the build system is generated and written. In this case the three last lines of the console log of **cmake** command should be:

```
-- Check for working C compiler: gcc
-- Check for working C compiler: gcc - works
...
-- Configuring done
-- Generating done
-- Build files have been written to: ~/bar
```

3. Basic install

make install will install in default directory /usr/local

4. Custom build and install

If you want to change build and/or install options (for example to install MPTK in a different folder or to enable the parallel computing using multithread capacities provided by the OS).

4.1. Principle

In order to build the MPTK library and executable files, you have to generate a build system with Cmake. This build system can be parametrized using the Cmake Graphical User Interface, which is

also used to fix possible issues of the build system configuration.This will yield the following sequence of commands:

```
bash-3.1$ mkdir ~/bar
bash-3.1$ cd ~/bar
bash-3.1$ cmake ~/foo
bash-3.1$ ccmake ~/foo
```

[Using the Cmake GUI to configure build system options]

```
bash-3.1$ make
bash-3.1$ make install
```

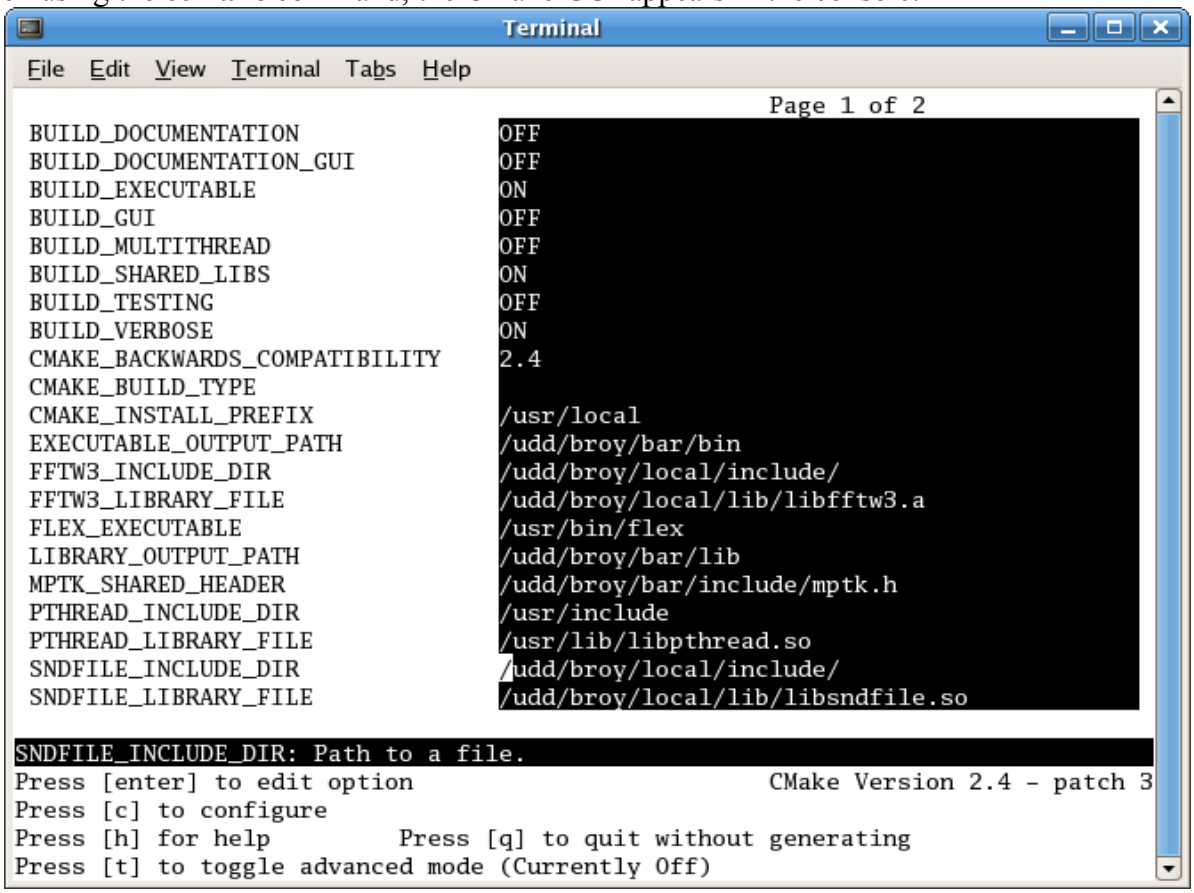
4.2.Using CCMake

a) Setting build and install option

Using the Cmake Graphical User Interface with the ccmake command allows you to:

- Specify build system options, such as whether to enable the use of multiple threads to speed up the Matching Pursuit decomposition or whether to generate the doxygen documentation.
- Set the prefix path used by the **make install** command
- Troubleshoot the various paths required to build MPTK
- Display advanced build options via the advanced mode

When using the **ccmake** command, the Cmake GUI appears in the console:



When scrolling on a value and pressing [enter], this value can be edited, the black underlaid row displays some information about the option and required path to create the build system. In the case of an option press [enter] to toggle the ON/OFF values.

After choosing options for the build and setting the required fields, press [c] to configure. The configuration of the build system is checked again by Cmake, at the end of this check if the build settings are correct, you can press [g] in order to generate the build system and quit Cmake GUI.

b) Main options

This Graphical User Interface allows the user to set the build options for MPTK, currently named BUILD_SOMETHING, if you want to:

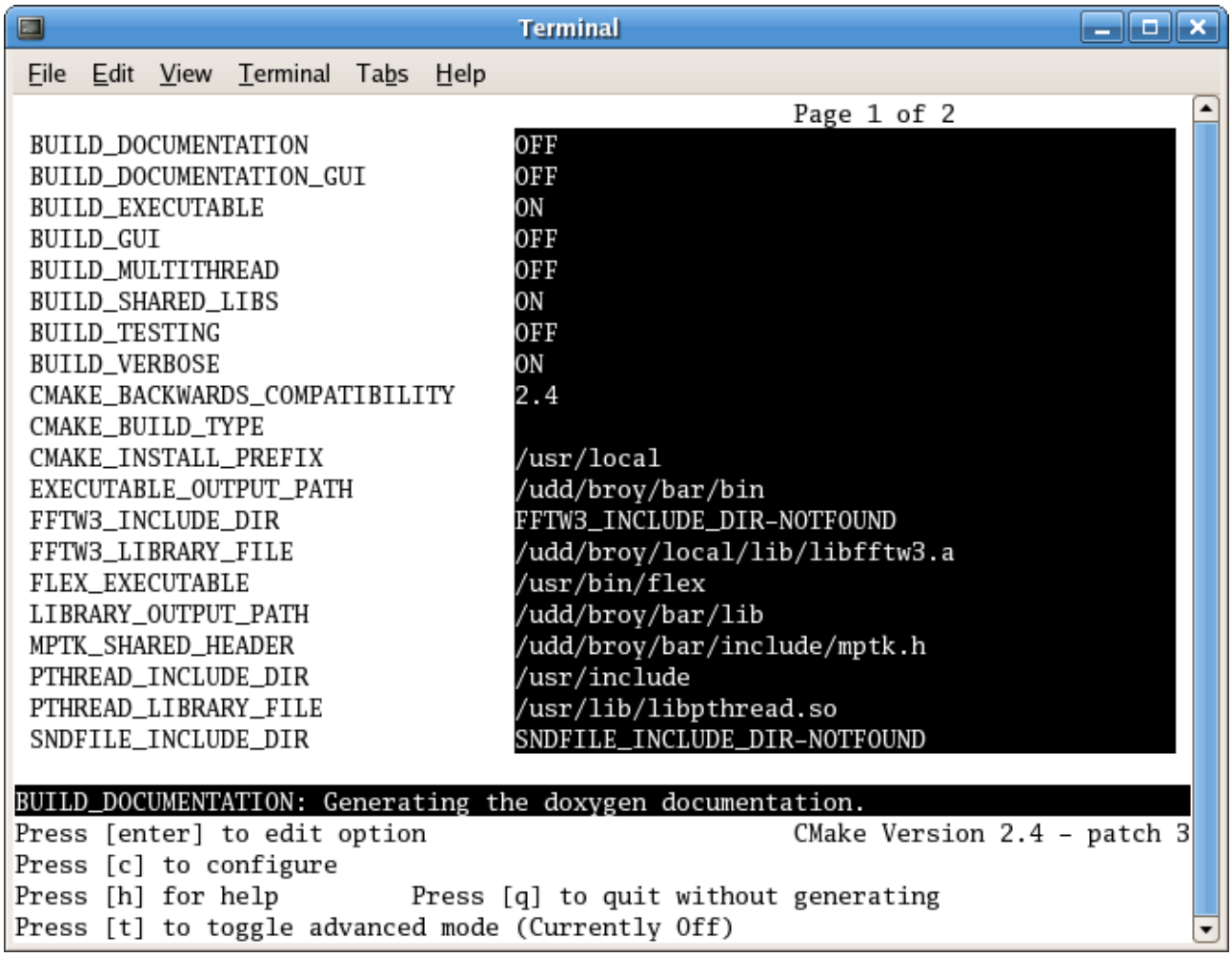
- Install MPTK executable files in a specific directory: set CMAKE_INSTALL_PREFIX with the path of this directory. (by default: /usr/local/)
- Generate the doxygen documentation for MPTK library: set BUILD_DOCUMENTATION to ON. This operation can take a lot of time, be patient. (default OFF)
- Generate the doxygen documentation for the Graphical User Interface of MPTK: set BUILD_DOCUMENTATION_GUI to ON. This operation can take a lot of time, be patient. (default OFF)
- Allow MPTK to use parrallel computing, set BUILD_MULTITHREAD to ON, this option allows to increase performance on multi-processor hardware architectures

c) Troubleshooting

Let us see a more complex use of the Cmake GUI with troubleshooting. Despite all the integrated tests when Cmake manages to generate the build system of MPTK, sometimes Cmake cannot determine some parameters of your configuration.When using the **cmake** command, Cmake performs a list of tests on the build system configuration in order to generate it, if there is some issue related to the configuration, the errors list is displayed at the end of the consol log. Here we have 2 includes files that Cmake cannot found:

```
-- Check for working C compiler: gcc
-- Check for working C compiler: gcc – works
...
-- Looking for C++ include sstream
-- Looking for C++ include sstream - found
CMake Error: This project requires some variables to be set,
and cmake can not find them.
Please set the following variables:
/udd/broy/foo/FFTW3_INCLUDE_DIR
/udd/broy/foo/SNDFILE_INCLUDE_DIR
```

In this case the Cmake GUI should look like this:



In the case of a path, Cmake `FFTW3_INCLUDE_DIR` and `SNDFILE_INCLUDE_DIR` fields (for example) must be set with the path to the directory containaing the include files for FFTW library and sndfile library. To set the fields values simply press [enter] while the cursor is on the revelant field, and after editing press [enter] to escape the field edition.

Important note: In the case of include files, only the directory where this include files are located is required. But in the case of library files, the entire path to the library including its name must be indicated in the corresponding field.

After choosing the options for the build and setting the required fields, press [c] to configure. The configuration of the build system is checked again by Cmake, at the end of this check if the build settings are correct, you can press [g] in order to generate the build system and quit the Cmake GUI.

d) Advanced options

You can display more information and all the configuration options by switching to the advanced mode, which is set to off by defaut and can be toggled on by pressing [t].

The list of the following options should be considered as advanced:

- BUILD_SHARED_LIBS: this option will install the MPTK library (libdsp_windows.a and

libmptk.a) in a lib directory and the include file (mptk.h) in an include directory. This directories will be created in the CMAKE_INSTALL_PREFIX path when using the **make install** command.

- BUILD_TESTING: this option allows to create tests for MPTK library and executable files using Dart server setting.
- BUILD_VERBOSE: this option displays a lot of various informations during the build.
- BUILD_EXECUTABLE: Build the executable files from MPTK (mpd, mpd_demix, mpf , mpview, mpr, mpcat) in a bin directory (default value: ON), note that this files will be install in a bin directory created within the CMAKE_INSTALL_PREFIX path when using **make install** command.

5. Fixing FFTW used plan with wisdom in FFTW configuration

MPTK currently relies on FFTW to compute Fast Fourier Transforms. The principle of FFTW is to adaptively select the fastest running codelet to perform a given FFT, which is good for performance. However, this can lead to non deterministic behaviour of MPTK since two consecutive runs of the same command (e.g. 'mpd') with the same options on the same data can yield slightly different results. To avoid this issue we have implemented a mechanism which allows MPTK to fix which FFT codelet is used for a specific type of FFT computation. To do this, you just need to create and export a environment variable called MPTK_FFTW_WISDOM and set this environnement variable with the path of the file you want to use in order to save the FFTW configuration. MPTK will create a « wisdom » file, that will be reloaded each time you use MPTK. It means that the codelets use to compute FFTW plan will be the same between two successive decompositions with the same parameters.

For bash or sh or ash:

```
export MPTK_FFTW_WISDOM=/udd/broy/local/fftw_wisdom_file
```

If you want to fix permanently the used FFTW plan, just set this environment variable when the console is launched, using for that the /etc/profile, ~/.profile or .bashrc file, according to your system configuration.

For tcsh ou csh :

```
bash-3.1$ setenv MPTK_FFTW_WISDOM=/udd/broy/local/fftw_wisdom_file
```

If you want to fix permanently the used FFTW plan, just set this environment variable when the console is launched, using for that the /etc/csh.cshrc file, according to your system configuration.

6. Appendix

6.1. Getting Cmake

Cmake, is a cross-platform, open-source make system. Cmake is used to control the software compilation process using simple platform and compiler independent configuration files. Cmake generates native makefiles and workspaces that can be used in the compiler environment of your choice. Cmake is quite sophisticated: it is possible to support complex environments requiring system configuration, pre-processor generation, code generation, and template instantiation. Please

go here to learn more about Cmake.

There are pre-compiled binaries available on the Download page for some UNIX platforms. The tarballs will extract a README and another tarball that may be extracted into the desired install location.
There are several possible approaches for building Cmake from a source tree:

- If there is no existing Cmake installation, a bootstrap script is provided:

```
./bootstrap
make
make install
```

(Note: the make install step is optional, Cmake will run from the build directory.)

- An existing Cmake installation can be used to build a new version:

```
cmake .
make
make install
```

(Note: the make install step is optional, cmake will run from the build directory.)

- On UNIX, if you are not using the GNU C++ compiler, you need to tell the bootstrap which compiler you want to use. This is done by setting the environment variable CXX before running configure. For example on the SGI with the 7.3X compiler, you build like this:

```
(setenv CXX CC; setenv CC cc; ./bootstrap)
make
make install
```

- For more options with bootstrap, run ./bootstrap --help

6.2.Getting libsndfile

Libsndfile is a C library for reading and writing files containing sampled sound (such as MS Windows WAV and the Apple/SGI AIFF format) through one standard library interface. It is released in source code format under the Gnu Lesser General Public License.

Where to download: http://www.mega-nerd.com/libsndfile/or in the « released files » section of the MPTK Gforge: http://mptk.gforge.inria.fr/

Install this package on your computer using the instructions given on this web site will be easy an will yield the following sequence of commands after unzipping the package:

```
./configure [options]
make
make install
```

6.3.Getting fftw

FFTW is a C subroutine library for computing the discrete Fourier transform (DFT) in one or more dimensions, of arbitrary input size, and of both real and complex data (as well as of even/odd data, i.e. the discrete cosine/sine transforms or DCT/DST).

Where to download: http://www.fftw.org/ or in the « released files » section of the MPTK Gforge: http://mptk.gforge.inria.fr/

Install this package on your computer using the instructions given on this web site will be easy an will yield the following sequence of commands after unzipping the package:

```
./configure [options]
make
make install
```

6.4.Help, contact and forums

If you need help with the software:

1. check if a more recent **release** fixes your problem;
2. if not, use the **Help forum** to ask questions.

Other **Forums** are available for open discussions about the Matching Pursuit algorithm and its MPTK implementation.

Some articles exposing scientific results related to MPTK are available in PDF format through the **Released Files** page.

If you want specific informations, or if you want to communicate privately with us, please write to matching.pursuit@irisa.fr.

Request for help sent to this address won't be answered. Please use the **Help forum** instead.

Thank you for your interest in The Matching Pursuit ToolKit !