Qt-DAB Users Guide

An Open Source DAB Decoder

Jan van Katwijk Lazy Chair Computing
The Netherlands
J.vanKatwijk@gmail.com

April 15, 2020

Contents

1 Introduction

Qt-DAB is a program for decoding terrestrial DAB transmissions. The program is implemented in C++, with extensive use of Qt for its graphical appearance. Furthermore, it uses a number of existing open source libraries.

Qt-DAB is designed to run on both Windows and Linux (x64) computers as well as on RPI 2 and up. For *Windows* an *installer* is available, this installer will install the executable together with the required libraries. For *Linux (x64)* a so-called *appImage* is available, a kind of container, an executable file that contains next to the executable program the libraries needed to run. These precompiled versions can be found in the releases section of the repository for Qt-DAB (https://github.com/JvanKatwijk/qt-dab/releases).

For RPI's, however, one has to create an executable, no preconfigured, precompiled executable is available. This document contains a pretty detailed description on how to build such an executable.

The sourcetree for Qt-DAB contains a subdirectory *dab-mini* with sources, with configuration files and with a description on how to create an executable version with a minimal interface. This *dabMini* version is described is section 7.

Since no major changes to the Qt-DAB sources are expected, it was considered time to do some user documentation.

The structure of this guide is simple, in section 2 the GUI and GUI widgets are discussed, in section 3 command line parameters and the settings in the ini file, are discussed, in section 4 the configuration issues and options are discussed and the process of configuring and building is described.

In section 5 a brief overview is given on the DAB decoding process and the role of the various parts of the software. In section 6 the device interface as used in Qt-DAB is discussed and an explanation is given how to add a device to the configuration. Finally in section 7, a brief description is given of *dabMini*, a decoder version built on the same set of sources but with a minimal interface.

2 The GUI and GUI elements

When playing around with DAB I am ususally interested in various properties of the signal, and I want to be in control. The GUI of Qt-DAB reflects that, there is an abundant amount of buttons, selectors and displays.

To keep things manageable, the GUI is built up as a central widget, a widget that is shown always, together with a number of other widgets that might - or might not - be made visible, depending on user's settings.

While the figure on the first page shows the GUI with all widgets, figure 1 shows the central widget, the one with (most of) the controls.

This main widget of the GUI (figure 1) can be thought to consist of three elements:

Figure 1: Qt-DAB: the main widget of the GUI

- the left part, handling control for channel and service;

- the top right part displaying information;

- the bottom right part, the various controls.

2.1 Control for channel and service

Central in the left part of the GUI is the list of services, these are the services detected in the currently selected channel. *Selecting* a service is by moving the curson to the name of a service, and clicking with the *left* mouse button.

Below the list of services (see figure 2) there is (from top to bottom)

- the combobox for the *presets*. A preset can be *added* to this list by clicking with the *right* mouse button on the name of the selected service in the service list[1]. *Removing* an element from the list is by putting the cursor on the name of the service in the list of presets, and pressing the *shift* and *delete* button on the keyboard simultaneously.

- a *previous* (−) and a *next* (+) service button. With these button one can easily scan through the list of services.

---

[1]Clicking with the right mouse button on the name of a service that is *not* the selected one, will cause a small widget to be shown with some information on the service pointed to
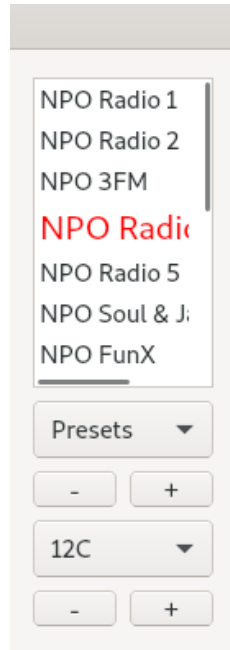
Figure 2: Qt-DAB, channel and service selection

- the combobox for *channel selection*. While DAB transmissions are in Band III, configuration provides options to select channels in the *L Band* or channels in a user defined band.

- a *previous* (−) and a *next* (+) channel button, making it easy to scan through the channels in the selected band.

Note that the software will "remember" which channel was selected, and which service was selected. On program start up, these values will be taken as start value.

2.2 Displaying information

Some general information is displayed in the top half of the right side of the GUI, see figure 3. The top line gives three elements

- the *run time*, the amount of time the program is running;

- the *current time*, this time is taken from the time encoding in the transmission. When playing a recording, the time found in the recording is shown rather than the current time of listening;

- the *copyright symbol*. Touching this with the cursor will reveal (a.o) the time and date the executable was built.

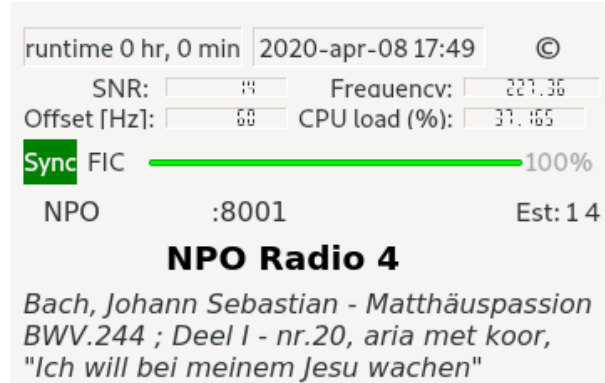Below this line, there are boxes with labels:

Figure 3: Qt-DAB, system wide information

- SNR, the measured signal/noise ratio. SNR is computed by comparing the signal strength in the null period of the DAB frame vs the average strength during transmission of the datablocks in the DAB frame;

- Frequency, the frequency, in MHz, of the selected channel;

- Offset, the frequency correction to be applied to the signal;

- CPU load, the overall CPU load, i.e. not only for running the program.

Below these - system related - pieces, there is a line with

- the *sync* flag, if *green*, time synchronization is OK;

- a *progressbar*, indicating the quality of decoding of the data in the FIC (Fast Information Channel). Since the FIC is "easier" to decode than most of the other data, a value less than 100 percent here usually indicates a poor reception.

The remainder here is devoted to describing the content of the reception, the name of the ensemble is displayed together with its ID. The name of the selected service is shown and below that name, the additional text, i.e. the *dynamic label* is shown.

The two numbers preceded by "Est:" give - if shown - an indication of the transmitter being received. Since DAB is transmitted using a Single Frequency Network, one might receive data from more than one transmitter. Each transmitter encodes a unique identification in the transmitted signal, the Transmitter Identification Information (TII), consisting of two numbers, one for the network, one for the specific transmitter in that network.

2.3 Control elements

The controls are grouped in the lower right half of the GUI, see figure 4. The control contains 12 push buttons and 2 comboboxes, briefly discussed in the order from left to right, top to bottom.
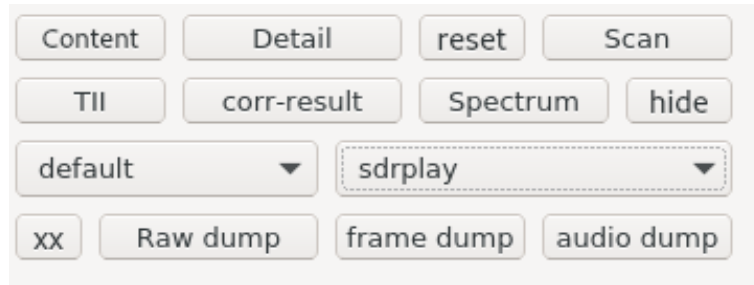
Figure 4: Qt-DAB: control elements

Content button Touching the button labeled *Content* will instruct the software to write a description of the content of the current ensemble to a file. First, a menu will appear with which the file can be selected. The file is written in ASCII and is readable by e.g. LibreOffice Calc or similar programs.

Detail button Touching the button labeled *Detail* will instruct the software to display detailed data on the selected service on a separate widget. Touching the button again will hide the widget.
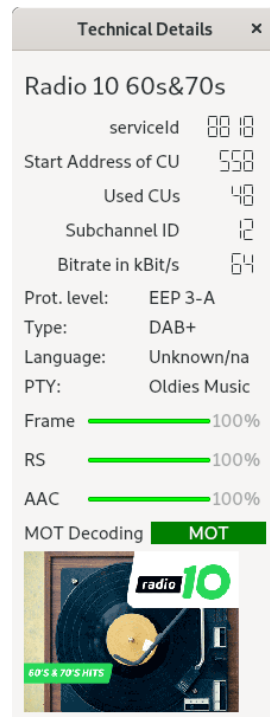
Figure 5: Service details

The widget - figure 5 - shows the name and the identification of the service, it shows where the data of the service is located in the input stream, it shows the *protection* of

the data against errors, whether it is a DAB+ or a DAB transmission, and - if available - it shows the type of the service.

For DAB+ services three progress bars are shown, in case all three are 100 percent, decoding is 100 percent. If less, then there are some issues that could not be resolved. (the top one shows the successrate of DAB+ frames passing a first test, the middle one the successrate of the Reed-Solomon error recovery on the frames passing the first test, and the bottom one tells the successrate of the AAC decoding).

Finally, if the service carries a picture as label, it will be displayed.

Reset button   Touching the button labeled *reset* will, as the name suggests, instruct the software to do a reset on the selected channel, i.e. synchronization will be done again and a fresh list of services is built up.
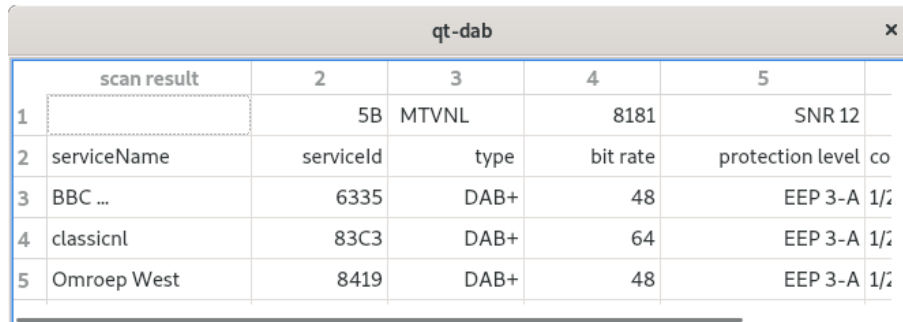
Scan button   Touching the button labeled *Scan* will instruct the software to perform a single scan over the channels in the currently selected band (default Band III) and show the results, see figure 6(i.e. the names of the ensembles found, names of services and some technical data on the services).

	scan result	2	3	4	5	
1		5B	MTVNL	8181	SNR 12	
2	serviceName	serviceId	type	bit rate	protection level	co
3	BBC ...	6335	DAB+	48	EEP 3-A	1/2
4	classicnl	83C3	DAB+	64	EEP 3-A	1/2
5	Omroep West	8419	DAB+	48	EEP 3-A	1/2

Figure 6: Fragment of the scan output

TII button   Touching the button labeled *TII* will instruct the software to show a widget (figure 7) with the spectrum of the null period between DAB frames. The TII data (Transmitter Identification Information) is extracted from the spectrum of the null periods. On touching the button again the widget will disappear.

corr-result button   Touching the button labeled *corr-result* will instruct the software to show a separate widget, making the *correlation result* for time synchronization visible. As mentioned earlier, DAB is transmitted in a Single Frequency Network and a receiver may receive data from more than one transmitter. The signal from the transmitter with the largest signal value is the one used for demodulation and decoding.

The X-axis indicates the sample numbers. The picture, figure 8, shows that there are two peaks in the displayed region, on around sample 470 and one near 500. The
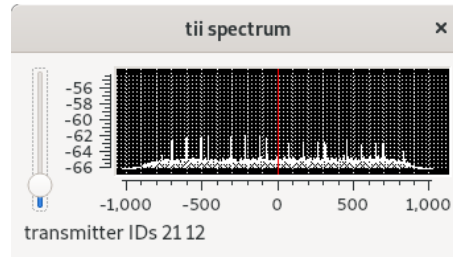
Figure 7: TII spectrum

latter is slightly stronger. Given that the samplerate is 2048000, one can conclude that the strongest signal arrives app 15 microseconds after the other one.

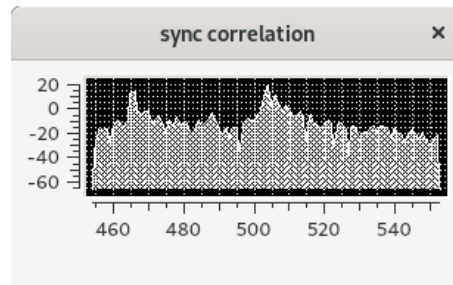Touching the button again will cause the widget to disappear.

Figure 8: Correlation result

Spectrum button Touching the button labeled *Spectrum* will instruct the software to show a separate widget, displaying the spectrum of the incoming signal, showing the constellation of the received and decoded signal and showing a measure of the quality of the signal (in general a positive value indicates a reasonable signal). The picture, figure 9 shows a reasonable though not excellent signal. Ideally the constellation shows as four dots, one in each quadrant. The more the constellation looks like a collection of clouds, the poorer the signal.

As with the other buttons, touching the button again will cause the widget to disappear.

hide button Touching the button labeled *hide* will hide (or show) the widget for the device control. The text on the button shows what the action following touching is (i.e. *hide* or *show*).

The combobox labeled *default* The combobox, labeled *default* in the picture is for selecting an audio channel. What the combobox shows depends on the computer where the program is running. In most cases *default* will do.
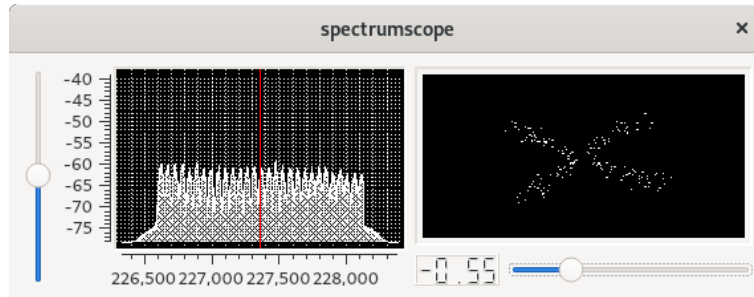
Figure 9: Spectrum of the signal

The combobox, labeled *sdrplay* The combobox labeled *sdrplay* in the picture is for selecting a device. Depending on the configuration of the software device names will show here.

xx button The button labeled *xx* instructs the software to list the elements in the *history file*. Inspired by my car radio a list is maintained of all services ever selected. Touching the *xx* button again will hide the list.

Raw dump button Touching the button labeled *Raw dump* will instruct the software to dump the raw input samples into a file. First, a menu is presented for selecting a filename. Touching the button again will stop dumping and the file will be closed. The file is in PCM format, with a rate of 2048000, 2 channels and data represented as short ints. Note that recorded files will be pretty large, per second more than 8 MByte is written.

frame dump button Touching the button labeled *frame dump* will instruct the software to write the AAC output of the selected DAB+ service to a file. First, a menu is presented for selecting a filename. As usual, touching the button again will close the file. The resulting file can be used as input to e.g. VLC for further decoding.

audio dump button Touching the button labeled *audio dump* will instruct the software to write the audio output of the currently selected service, to a file. First a menu is presented for selecting a filename. The file, a PCM formatted file, is written with samplerate 48000, 2 channels and short int (16 bits) format. Touching the button again will stop writing and close the file.

3 Command line parameters and the ini file

While the GUI provides control elements, some settings can be done via the command line and by setting values in the ".ini" file. This ".ini" file also contains settings recorded

by the software. Its default name and location is *.qt-dab.ini* and it is kept in the users home directory.

On starting Qt-DAB via the command line (a few) parameters can be passed:

- "-i filename" to use the file named as ".ini" file rather than ".qt-dab.ini" which is stored in the users home directory;

- "-P portnumber" to use the portnumber as port for *TPEG* output in the Transparent Data Channel (tdc).

- "-A filename" to use the (name, integer) pairs in the file as channel definitions rather than the channels in Band III;

- "-T" generate messages while processing on success and misses in the various decoding steps.

Furthermore, a number of settings can be done in the ".ini" file. Note that the software will store some settings on the current selections (e.g, device, channel, service) in the ".ini" file.

- dabMode: While the *default* Mode for DAB is Mode 1, Qt-DAB provides the possibility to use Mode 2 or 4 as well by setting "dabMode=X" (X in {1, 2, 4});

- dabBand: While the *default* DAB band is Band III, Qt-DAB provides the possibility to use the L Band by setting "dabBand=L_Band". Note that setting a value here overrides the band setting by using command line parameters;

- displaycolor: While the *default* setting of the background color of the various displays is *black*, setting "displaycolor=xxx" will set the background of the displays to the selected color (e.g. "white");

- gridcolor: While the *default* setting of the color of the grids and the brushes in the various displays is *white*, setting "gridcolor=xxx" will set both the gridcolor as the color of the brush to the selected color (e.g. "red");

- displaySize: While the *default* setting of the size of the X axis of the spectrum and the TII display is 1024, setting "displaySize=xxx" will set the size of the X axis to xxx, provided xxx is a power of 2;

- plotLength: While the *default* setting of the size of the segment to be seen in the correlation viewer is 1000 (i.e. the correlation is shown over the first 1000 samples of the datablock), setting "plotLength=xxx" will show the correlation result over only xxx samples (centered around the maximum correlation value);

- saveSlides: While the *default* is 1, implying that decoded slides are saved, setting "saveSlides=0" will prevent slides to be saved;

- motSlides: While the *default* is 0, implying that decoder slides are displayed on the *Technical data* widget, setting "motSlides=1" will cause slides to be shown on a separate widget.

- pictures: While the *default* path for storing slides and pictures is the directory "qt-pictures" in the /tmp directory, setting "pictures=xxx" will use the folder "xxx" for that purpose.

- epgPath: While the *default* value is the empty string, implying that files generated by the epg handler are not saved, setting "epgPath=XXX" will use the "XXX" (if not the empty string) as path to these files (assuming the path exists and the epg handler is configured in).

- filePath: While the *default* value is the empty string, implying that MOT files other than slides and epg files, are not saved, setting "filePath=XXX" will use "XXX" (if not the empty string) as path to these files (assuming the path exists).

- serviceOrder: While the *default* order to display the services in the list of services is alphabetically, setting "serviceOrder=1" will cause the services to be displayed based on the order of their serviceIds;

- normalScan: While the *default* way a scan is performed is as a single scan over all channels in the band, at the end displaying the result, setting "normalScan=1" will instruct the software to start scanning at the currently selected channel and stop scanning as soon as a channel is encountered with DAB data;

- history: While the *default* file for storing (and reading back) the history elements is ".qt-history-xml" in the users home directory, setting "history=xxx" will use the file here denoted as "xxx";

- switchTime: While the *default* maximum delay taken into account to select a preset value is 8000 milliseconds, setting "switchTime=xxx" will use "xxx" (if specified as number) instead;

- latency: While the *default* value for the latency, i.e. the delay in handling the audio, and determining the size of the audio buffers, is 5, setting "latency=xxx" will set the value to "xxx" (if specified as positive number);

- ipAddress: While the *default* ip address for sending datagrams to (obviously only meaningful if configured) is "127.0.0.1:, setting "ipAddress=XXX" will use "XXX" as ip address (if properly specified);

- port: While the *default* port address for sending datagrams to (obviously only meaningful if configured) is "8888", setting "port=XXX" will use "XXX" (if specified as positive number);

- threshold: While the *default* value for the threshold is 3, another value can be set by "threshold=XXX". The threshold is a value used in the time synchronization. If the maximum correlation found is at least *threshold* times the average correlation value, the maximum is considered to be OK;

- diff_length: While the *default* value for the length of the segment used to find the correct coarse frequency offset is 40, another value can be chosen by setting "diff_length=XXX"; Note that the length does have impact on the amount of computations that are needed to compute an estimate of the frequency offset.

- tii_delay: While the *default* value for the number of DAB frames that will be skipped before recomputing the TII value is 5 (basically to reduce the computational load), another value can be chosen by setting "tii_delay=XXX";

- tii_depth: While the *default* value for the tii_depth (i.e. the number of spectra used to extract the TII values) is "1", another value can be chosen by setting "tii_depth=XXX";

- echo_depth: While the *default* value for the echo_depth is 1 (i.e. the maximum amount of alternative peaks in the correlation), another value can be chosen by setting "echo_depth=XXX";

4   Supported input devices

The current version of Qt-DAB supports a variety of input devices, the SDRplay, the AIRspy, the hackrf, the limeSDR and the RT2832 based sticks. Furthermore, there is support for devices for which a *Soapy* interface is defined, and there is support for file input.

Both the *appImage* and the *Windows installer* are configured with the whole range of devices: SDRplay RSP (both the 2.13 and 3.06 library versions), the AIRspy, the hackrf, the LimeSDR and - of course - the RT2832 based dabsticks.

4.1   The SDRplay RSP

The Qt-DAB software supports all RSP's from SDRplay. Qt-DAB provides support for devices using the 2.13 SDRplay interface library, and it supports devices using the 3.06 SDRplay interface library.

As figure 10 shows, the control widgets for the two different versions resemble each other. Both have spinboxes for setting the *if gain reduction*, the *lna state* and a *ppm offset*.

An optimal value for the *ppm offset* is to be determined experimentally, the RSP II, as used here, is happy with a ppm offset 0, the oscillator error is almost zero.
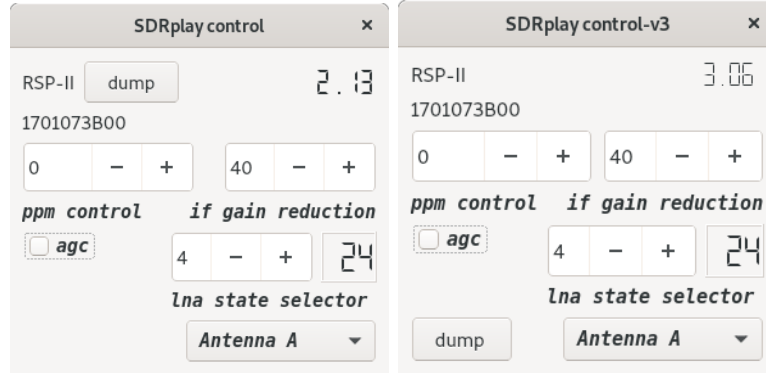
Figure 10: The two control widgets for the SDRplay

The spinbox for the *if gain reduction* is programmed to support the range of values between 20 and 59. The range of values for the *lna state* depends on the model of the RSP. The software will detect the model and fill in the range accordingly.

If the *agc* is selected, the *if gain reduction* spinbox will be hidden, its value is then irrelevant.

The RSP II has two (actually 3) slots for connecting an antenna. If an RSP II is detected, a combobox will be made visible for *antenna selection*.

A similar combobox exists for selecting a tuner in the widget for the 2.13 library controller. The SDRplay duo has two tuners. If the software detects the duo, a combobox will be made visible for selecting a tuner (Note that this feature is not tested).

Finally, both versions of the control widget contain a *dump* button. If touched, the raw input from the connected device will be stored in a so-called xml formatted file. First a menu is shown for selecting a filename. As usual, touching the button again will stop dumping and the file will be closed.

4.2   The AIRSpy

The control widget for the AIRspy (figure 11, left) contains three sliders and a push button. The sliders are to control the lna gain, the mixer gain and the vga gain.

To ease balancing the setting of the sliders, two combined settings are included in the widget, selectable by the tab *sensitivity* and *linearity*. Figure 11 right side, shows the setting at selecting the tab *sensitivity*.

Touching the button labeled *dump* instructs the software to dump the raw stream of samples into a file in the xml format (Note that while processing DAB requires the samplerate to be 2048000, that rate is not supported by the AIRspy, implying that the driver software has to do some rate conversion. The xml file though will just contain the samples on the rate before conversion).

14
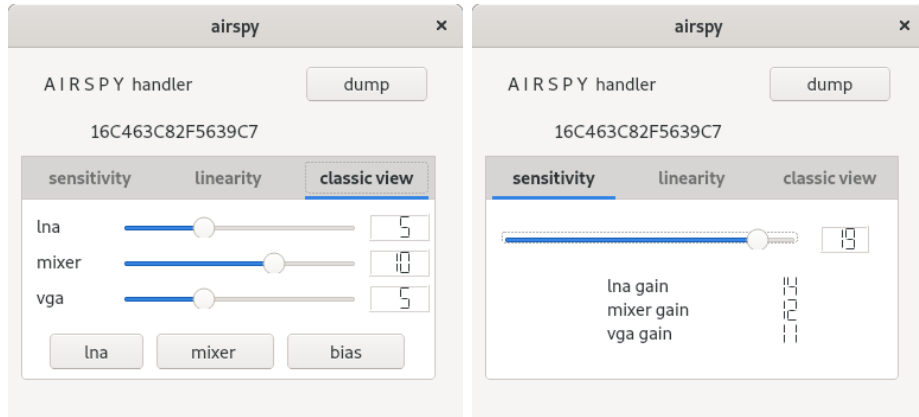
Figure 11: Widgets for AIRspy control

4.3 The hackrf
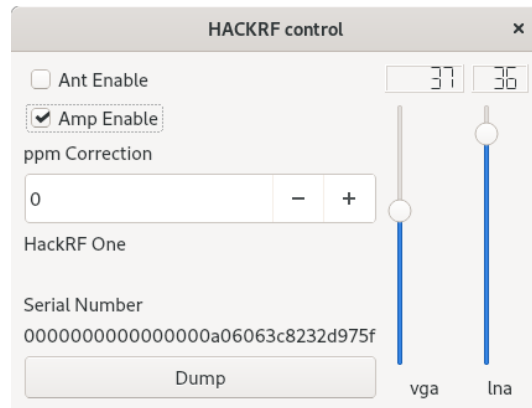
Figure 12: Widget for hackrf control

The control widget for hackrf (figure 12) shows, next to the Serial Number of the device, a few sliders, a few checkboxes, a spinbox and a push button.

- the *sliders* are there for controlling the lna and vga gain, the slider values are limited to the range of possible values;

- The *Ant Enable* checkbox is for Antenna port Power control (not used in this controller);

- The *Amp Enable* checkbox is - if enabled - for additional gain on the antenna input;

- the *ppm correction* spinbox can be set to correct the oscillator (on 227 MHz, the Qt-DAB software reports an offset of somewhat over 3 KHz);

- the *Dump* push button when pushed, starts dumping the raw input in xml file format. Touching the button again will halt the dumping and close the file.

4.4 The LimeSDR

Figure 13: Widget for lime control

On selecting the LimeSDR (if configured), a control widget for the LimeSDR appears (figure 13). The widget contains just three controls:
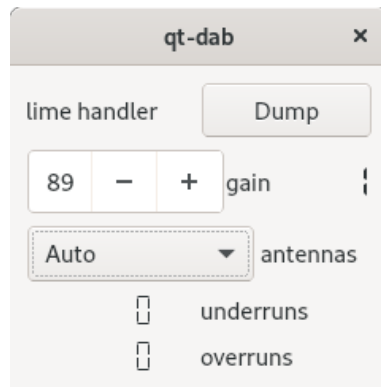
- *gain* control, with predefined values;

- *antennas*, where *Auto* is usually the best choice;

- *Dump*, if touched, the raw input from the connected device will be written to a file in the so-called xml format.

4.5 The RTLSDR stick

On selecting the dabstick (i.e. RT2832 based devices) (if configured), a control widget for the device appears (figure 14).

The widget contains just a few controls:

- a *spinbox* for setting the ppm. Note that on average the offset of the oscillator with DABsticks is (much) larger than that with devices like the SDRplay. The DAB software is able to correct frequencies to up to app 35 KHz, for some sticks the frequency error was large and correction using the ppm setting was required.

- a *combobox* for setting the gain. The support software for RT2832 based devices generates a list of allowable gain settings, these settings are stored in the combobox;

- a *combobox* for setting the autogain on or off;
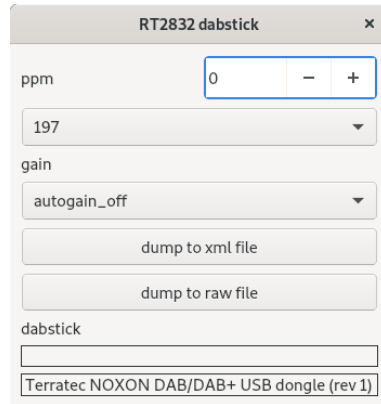
Figure 14: Widget for rtlsdr device

- a *push button* that, when touched, will instruct the software to dump the raw input in the aforementioned xml format. At first a menu appears for selecting a file. Touching the button again will stop dumping and close the file.

4.6 Support for Soapy

Soapy is a generic device interface, a kind of wrapper to provide a common interface to a whole class of devices. Qt-DAB supports Soapy, and its use is tested with the Soapy interface for the SDRplay.

Figure 15: Widget for soapy

The widget for soapy control (see figure 15) when applied to the Soapy interface for the SDRplay contains the obvious controls, similar to that of the regular control for the

17

SDRplay.

4.7   File input

Qt-DAB supports both writing raw input files and reading them back. Writing a file is initiated by either the *Raw dump* button on the main GUI, or by the *dump* button on the various device widgets. Qt-DAB differentiates between

- raw 8 bit files as generated by e.g. Osmocom software (usually files with an extension ".raw" or ".iq");

- PCM (i.e. ".wav") files, provided the data is 2 channels and with a samplerate of 2048000, generated by Qt-DAB and with an extension ".sdr";

- xml files. The xml file format was defined by Clemens Schmidt (author of QIRX) and me and aims at saving files in the original format, so to allow easy exchange between different DAB decoder implementations. In order to support proper decoding of the contents, the data in the file is preceded by a detailed description in xml, hence the name xml file format.

When selecting file input ".raw" or ".wav", a simple widget is shown (figure 16), with as indication the number of seconds the file is being played.
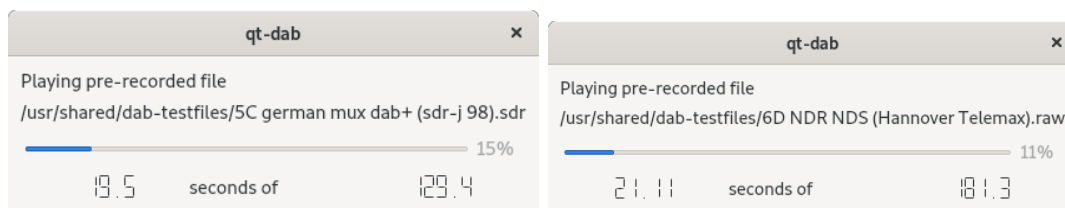
Figure 16: Widgets for file input

Since processing an xml file implies some interpretation, the widget (figure 17) for control when reading an xml file is slightly more complex. It contains - next to the progress in reading the data - a description of the contents of the file. So, the program that generated the file as well as the device used in that program are displayed, the number of bits of the samples, as well as the number of elements is displayed as is the samplerate of recording and the frequency of the recording.

Touching the *cont* button will instruct the software to restart reading at the beginning of the segment in the file after reaching the end.

Figure 17: Widget for xml file input

5   Configuring and building an executable

5.1   Introduction

While for both Windows and Linux-x64 there are ready-made executables for installing resp. executing the Qt-DAB program, there are situations where one wants (or needs) to create its own version. For e.g. use of the software on an RPI one has to create an executable, for e.g. using the software with other or non-standard configured devices one has to create an executable. This section will describe the configuration options and the building process.
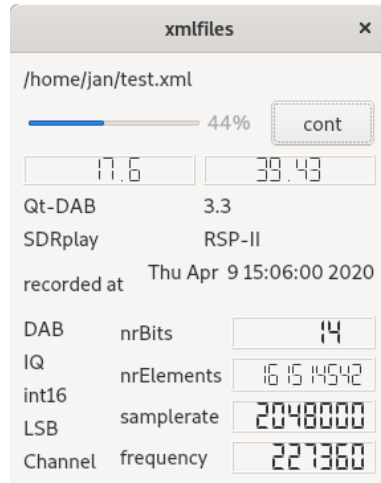
5.2   What is there to configure?

The Qt-DAB software can be built using either qmake or cmake generating a Makefile. The current *configuration file* for qmake, *qt-dab.pro*, has more options for configuring than the configuration file for use with cmake, *CMakeLists.txt*.

QMake and CMake take a different approach, while the configuration options for use with qmake requires some editing in the *qt-dab.pro* file, selecting configuration options with cmake is ususally through command line parameters.

Note that the *qt-dab.pro* file contains a section *unix* and a section *win* for Windows that contain settings specific to the OS used. The CMakeLists.txt file is only used for Linux-x64.

5.2.1   Finding the right qwt library (qt-dab.pro only)

It turns out that linking the qwt library sometimes gives problems. While in Fedora based systems, specifying linkage is as below, i.e. the *-lqwt-qt5* is the right one, in Debian

based systems the line *-lqwt* line should be chosen by commenting out the other one.

```
#correct this for the correct path to the qwt6 library on your system
#LIBS          += -lqwt
LIBS           += -lqwt-qt5
```

5.2.2 Console or not (qt-dab.pro only)

```
# CONFIG += console
  CONFIG -= console
```

While for tracing and debugging purposes it might be handy to see all the (text) output generated during execution, for normal use it is not. Including or excluding *console* in the configuration determines whether or not a console is present when executing.

5.2.3 Configurable common devices

Configuring devices is simple, for devices as mentioned above as well as for *rtl_tcp* the *qt-dab.pro* file and the *CMakeLists.txt* contain a description. File input (all versions, i.e. raw files, sdr files and xml files) is standard configured in Qt-DAB executables, changing this would imply significant changes to the sources.

Using the qt-dab.pro file For configuring devices in the *qt-dab.pro* file, comment out or uncomment the line with the devicename.

```
  CONFIG += dabstick
  CONFIG += sdrplay-v2
  CONFIG += sdrplay-v3
  CONFIG += lime
  CONFIG += airspy
  CONFIG += hackrf
  CONFIG += soapy
  CONFIG += rtl_tcp
```

Note that for *soapy*, and for *limeSDR* there is no support in generating a windows executable and *hackrf* is untested under windows.

Using the CMakeLists.txt file The CMakeLists.txt file contains support for AIRspy, SDRplay, SDRplay_V3, RTLSDR, Hackrf and LimeSDR. Including a device in the configuration is by adding "-DXXX=ON" to the command line, where XXX stands for the device name.

5.2.4 Configuring SSE

In the deconvolution of the data in the FIC blocks, use is made of deconvolution code generated by the spiral code generator. If the code is to run on an x86-64 based PC, some speed up can be obtained by using the code generated for use with SSE instructions. Of course, the compiler used in the building process has to support generating the right instructions, as fas as known, the Mingw compiler, used for generating the windows executable, does not.

The qt-dab.pro file contains in the unix section

```
#For x64 linux system uncomment SSE
#For any other system comment SSE out and uncomment NO_SSE
#CONFIG += SSE
CONFIG  += NO_SSE
```

When using cmake, pass "-DVITERBI_SSE=ON" as command line parameter.

5.2.5 Configuring audio

- When running the Qt-DAB program remotely, e.g. on an RPI near a decent antenna, one might want to have the audio output sent through an IP port (a simple listener is available).

- Maybe one wants to use the audio handler from Qt.

- The default setting is to use *portaudio* to send the PCM samples to a selected channel of the soundcard.

The *Linux* configuration for the Qt-DAB program offers in the qt-dab.pro file the possibility of configuring the audio output:

```
#if you want to listen remote, uncomment
#CONFIG        += tcp-streamer          # use for remote listening
#otherwise, if you want to use the default qt way of sound out
#CONFIG        += qt-audio
#comment both out if you just want to use the "normal" way
```

If cmake is used, pass "-DTCP_STREAMER=ON" as parameter for configuring the software for remote listening, use "-DQT_AUDIO=ON" for qt audio, or *do not specify anything* for using portaudio in the configuration.

Note that the configuration for Windows is only for "portaudio".

5.2.6 Configuring TPEG in the tdc

Handling TPEG in the tdc is only partially supported. Interpretation of the data is not part of the Qt-DAB software, however, the software can be configured to extract the TPEG frames and send these to an IP port.

In the qt-dab.pro file, we have

```
#very experimental, simple server for connecting to a tdc handler
CONFIG        += datastreamer
```

In cmake the parameter "-DDATA_STREAMER=ON" can be passed to include handling TPEG as described in Qt-DAB.

5.2.7 Configuring IP datastream (qt-dab.pro only)

IP data can be extracted from the DAB stream and send out through an IP port.

```
#to handle output of embedded an IP data stream, uncomment
CONFIG         += send_datagram
```

Note that - if not specified in the ini file - defaults are used for ip address and port.

5.2.8 Selecting an AAC decoder (qt-dab.pro only)

By default the *faad* library is used to decode AAC and generate the resulting PCM samples. An alternative is to use the *fdk-aac* library to decode AAC (contrary to the libfaad the fdk-aac library is able to handle newer versions of the AAC format, these newer versions are not used in DAB (DAB+)).

Selecting the library for the configuration is by commenting out or uncommenting the appropriate line in the file *qt-dab.pro* (of course, precisely one of the two should be uncommented).

```
  CONFIG    += faad
  #CONFIG   += fdk-aac
```

5.2.9 Configuring threading

Processing DAB (DAB+) requires quite some processing power. On small computers like an RPI2, performing all processing on a single CPU core overloads the core.

In order to allow smooth processing on multi core CPU's, an option is implemented to partition the workload. In order to partition processing, uncomment

```
  DEFINES += __THREADED_BACKEND
```

in the *qt-dab.pro* file.

In case cmake is used, edit the file CMakeLists.txt and comment out or uncomment the line

```
#add_definitions (-D__THREADED_BACKEND)  # uncomment for use for an RPI
```

5.2.10 Configuring EPG processing

By default MOT data with EPG data is not dealt with. The Qt-DAB sourcetree contains software from other sources that can be used to decode EPG and write the decoded data into a file in xml format.

In order to configure the software to include the epg handling part uncomment

```
CONFIG          += try-epg
```

in the *qt-dab.pro* file, or add

```
-DTRY_EPG=ON
```

to the command line when using cmake.

5.3 Preparing the build: loading libraries

5.3.1 Installing the libraries

Prior to compiling, some libraries have to be available. For Debian based systems (e.g. Ubuntu for PC and Stretch for the RPI) one can load all required libraries with the script given below.

```
sudo apt-get update
sudo apt-get install git cmake
sudo apt-get install qt5-qmake build-essential g++
sudo apt-get install pkg-config
sudo apt-get install libsndfile1-dev qt5-default
sudo apt-get install libfftw3-dev portaudio19-dev
sudo apt-get install libfaad-dev zlib1g-dev rtl-sdr
sudo apt-get install libusb-1.0-0-dev mesa-common-dev
sudo apt-get install libgl1-mesa-dev libqt5opengl5-dev
sudo apt-get install libsamplerate0-dev libqwt-qt5-dev
sudo apt-get install qtbase5-dev
```

5.3.2 Downloading of the sourcetree

Since the script also loads *git* the sourcetree for Qt-DAB (including the sources for dab-mini) can be downloaded by

```
git clone https://github.com/JvanKatwijk/qt-dab.git
```

5.3.3 Installing support for the RTLSDR stick

It is advised - when using an RT2832 based "dab" stick - to create the library for supporting the device

```
git clone git://git.osmocom.org/rtl-sdr.git
cd rtl-sdr/
mkdir build
cd build
cmake ../ -DINSTALL_UDEV_RULES=ON -DDETACH_KERNEL_DRIVER=ON
make
sudo make install
sudo ldconfig
cd ..
rm -rf build
cd ..
```

5.3.4   Installing support for the AIRspy

If one wants to use an AIRspy, a library can be created and installed by

```
wget https://github.com/airspy/host/archive/master.zip
unzip master.zip
cd airspyone_host-master
mkdir build
cd build
cmake ../ -DINSTALL_UDEV_RULES=ON
make
sudo make install
sudo ldconfig
cd ..
rm -rf build
cd ..
```

5.3.5   Installing support for SDRplay RSP

If one wants to use an RSP from SDRplay, one has to load and install the library from "www.SDRplay.com".

5.4   Finally: building an executable

5.4.1   Using cmake to build the executable

After installing the required libraries, and after editing the configuration (if required), compiling the sources and generating an executable is simple.

Using cmake, creating an executable with as devices the SDRplay, the AIRspy, and the RTLSDR based dabsticks, the following script can be used:

```
cd qt-dab
mkdir build
cd build
```

```
cmake .. -DSDRPLAY=ON -DAIRSPY=ON -DRTLSDR=ON
make
```

The CMakeLists.txt file contains instructions to install the executable in "/usr/local/bin".

```
sudo make install
```

5.4.2 Using qmake to build the executable

Assuming the file qt-dab.pro is edited, the same result can be obtained by

```
cd qt-dab
qmake
make
```

In some Linux distributions replace qmake by qmake-qt5!
The *qt-dab.pro* file contains in both the section for unix as for windows a line telling where to put the executable

```
DESTDIR        = ./linux-bin
```

By default in Linux the executable is placed in the *./linux-bin* director in the *qt-dab* directory.

6 Adding support for a device

Qt-DAB is an open source project. Anyone is invited to suggest improvements, to improve the code and to add code for e.g. yet unsupported devices.

While Qt-DAB can be configured for a variety of devices (basically the devices I have access to), there is obviously a multitude of other devices that are worthwhile to use with Qt-DAB.

The Qt-DAB software provides a simple, well-defined interface to ease interfacing a different device.

The interface is defined as

```
class deviceHandler: public QObject {
public:
        deviceHandler   (void);
virtual ~deviceHandler   (void);
virtual int32_t getVFOFrequency (void);
virtual int32_t defaultFrequency(void);
virtual bool restartReader (int32_t);
virtual void stopReader (void);
virtual int32_t getSamples (std::complex<float> *, int32_t);
virtual int32_t Samples (void);
virtual void resetBuffer (void);
```

```
virtual int16_t bitDepth (void);
virtual void show ();
virtual void hide ();
virtual bool isHidden ();
};
```

A device handler for a - yet unknown - device should implement this interface.
A description of the interface elements follows

- *getVFOFrequency* returns the current oscillator frequency in Hz;

- *defaultFrequency* returns a frequency in the range of valid frequencies;

- *restartReader* is supposed to start or restart the generation of samples from the
  device. Note that while not specified explicitly the assumed samplerate is 2048000,
  with the samples filtered with a bandwidth of 1536000 Hz. The parameter - in Hz
  - indicates the frequency to be selected. *restartReader* when already running has
  no effect.

- *stopReader* will do the opposite of restartReader, collecting samples will stop;
  stopReader when not running has no effect.

- *getSamples* is the interface to the samples. The function should provide a given
  amount of samples, the return value is the number of samples actually read.

- *Samples* tells the amount of samples available for reading. If the Qt-DAB software
  needs samples, the function Samples is continuously called (with the delay between
  the calls) until the required amount is available, after which *getSamples* is called.

- *resetBuffer* will clear all buffers. The function is called on change of channel.

- *bitDepth* tells the number of bits of the samples. The value is used to scale the Y
  axis in the various scopes and to scale the input values when dumping the input.

- The GUI contains a button to hide (or show) the control widget for the device.
  The implementation of the control for the device will implement - provided the
  control has a widget - functions to *show* and to *hide* the widget, and *isHidden*, to
  tell the status (visible or not).

Having an implementation for controlling the new device, the Qt-DAB software has
to know about the device handler, which requires a change to the configuration file (here
we take qt-dab.pro) and the file radio.cpp, the main controller of the GUI.

Modification to the qt-dab.pro file Driver software for a new device, here called *newDevice*, should be a class *newDevice*, derived from the class *deviceHandler*.

It is assumed that the header is in a file *new-device.h*, the implementation in a file *new-device.cpp*, both stored in a directory *new-device*.

A name of the new device e.g. *newDevice* will be added to the list of devices, i.e.

```
   CONFIG += AIRSPY
...
   CONFIG += newDevice
```

Next, somewhere in the qt-dab.pro file a section describing XXX should be added, with as label the same name as used in the added line with CONFIG.

```
newDevice {
   DEFINES          += HAVE_NEWDEVICE
   INCLUDEPATH      += ./devices/new-device
   HEADERS          += ./devices/new-device/new-device.h \
                          .. add further includes to development files, if any
   SOURCES          += ./devices/new-device/new-device.cpp \
                          .. add further implementation files, if any
   FORMS            += ./devices/new-device/newdevice-widget.ui
   LIBS             += .. add here libraries to be included
}
```

Modifications to radio.cpp The file "radio.cpp" needs to be modified in three places

- In the list of includes add

  ```
  #ifdef HAVE_NEWDEVICE
  #include new-device.h
  #endif
  ```

- The names of selectable devices are stored in a combobox.

  ```
  #ifdef HAVE_AIRSPY
  deviceSelector -> addItem ("airspy");
  #endif
  ....
  #ifdef HAVE_NEWDEVICE
  deviceSelector -> addItem ("newDevice");
  #endif
  ```

- If selected, the class implementing the device handler should be instantiated,

```
#ifdef HAVE_AIRSPY
if (s == "airspy") {
   try {
       inputDevice = new airspyHandler ....
....
#endif
#ifdef  HAVE_NEWDEVICE__
        if (s == "newDevice") {
           try {
               inputDevice       = new newDevice (..parameters..);
               showButtons ();
           }
           catch (int e) {
               QMessageBox::warning (this, tr ("Warning"),
                                        tr ("newDevice not found\n"));
               return nullptr;
           }
        }
        else
#endif
```

7   dabMini

7.1   Why a dabMini

I often run DAB decoder(s) on an RPI2 or 3. Since these RPIs are headless, control
is from my laptop. Sometimes I find the GUI of Qt-DAB too large, especially when
my only concern is listening to the audio. In that case I do not need any of the push
buttons and the comboboxes on the main GUI widget. While using *dabRadio* for that
purpose (or *qml-dab*) for some time, I realised that most of the corrections and changes
as applied to the sources - quite many - for Qt-DAB were not applied to the sources of
these programs.

So, in order to maintain consistency of sources between Qt-DAB and a version with
a small GUI I designed and implemented *dabMini* using the Qt-DAB sources. To ensure
maintaining the consistency, a subdirectory was made in the Qt-DAB sources containing
the (few) files special for use with this dabMini. Interesting is that - next to changes to
device handlers to accomodate for the demise of the device control widgets - only 2 files
needed to be changed.

7.2   The GUI

As picture 18 shows, the GUI is minimal. The *device control* is at the top right. De-
pending on the selected device, one or two spinboxes (usually some lna setting and some

Figure 18: dabMini

other gain setting) are shown together with a checkbox for the agc. *dabMini* will - on program start up - look for any of the configured devices being connected, and take the first one encountered.

To the right of the service list, a *channel selector* is available, with a < (previous) and a > (next) button for easy scanning though the channels, and, below these, a < (previous) and > (next) button for easy scanning though the services in the service list.

The bottom of the GUI contains two comboboxes, the one labeled *Presets* is - as the name suggests - for the presets, the other one for selecting an audio channel on the soundcard.

7.3   Building an executable

As an example, loading libraries and building an executable of the program on an RPI (running Buster) is described here.

7.3.1   Installing the libraries

For e.g. the RPI running Buster, the following lines will install all required libraries

```
sudo apt-get update
sudo apt-get install git cmake
sudo apt-get install qt5-qmake build-essential g++
sudo apt-get install pkg-config
sudo apt-get install libsndfile1-dev qt5-default
sudo apt-get install libfftw3-dev portaudio19-dev
sudo apt-get install libfaad-dev zlib1g-dev rtl-sdr
sudo apt-get install libusb-1.0-0-dev mesa-common-dev
sudo apt-get install libgl1-mesa-dev libqt5opengl5-dev
sudo apt-get install libsamplerate0-dev
sudo apt-get install qtbase5-dev
```

Note that on other platforms libraries might be named in another way.

Assuming the only device that needs support is an RT2832 based stick, execute the lines from the following script

```
git clone git://git.osmocom.org/rtl-sdr.git
cd rtl-sdr/
mkdir build
cd build
cmake ../ -DINSTALL_UDEV_RULES=ON -DDETACH_KERNEL_DRIVER=ON
make
sudo make install
sudo ldconfig
cd ..
rm -rf build
```

7.3.2 Download the sourcetree for Qt-DAB

Dowload the sourcetree for Qt-DAB

```
git clone https://github.com/JvanKatwijk/qt-dab.git
```

7.3.3 Generate an executable

The settings in the file *CMakeLists.txt* are such that no changes are needed, just execute the lines from the following script (the "make" will take app 10 minutes on an RPI 3) to build and install an executable.

```
cd qt-dab
cd dab-mini
mkdir build
cd build
cmake .. -DRTLSDR=ON
make
sudo make install
```

This will install the executable *dabMini-1.0* in */usr/local/bin*.