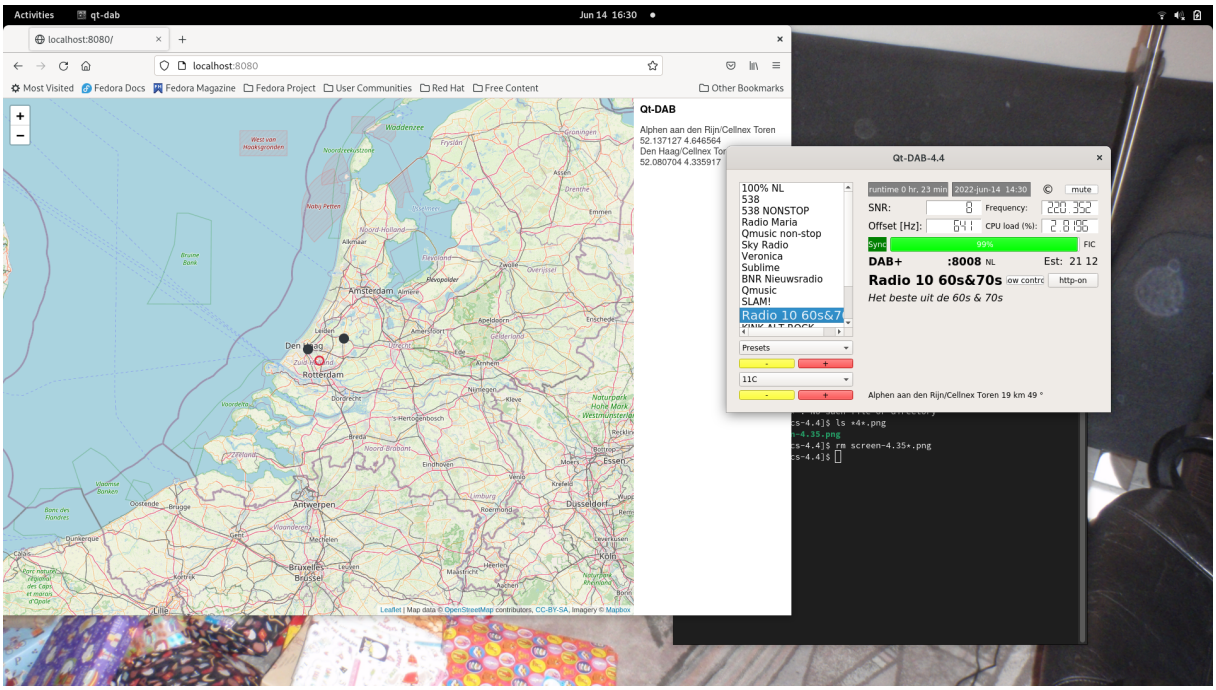
# Qt-DAB 4.4*

*User's guide for version 4.4*

Jan van Katwijk, Lazy Chair Computing
The Netherlands

June 14, 2022




*© both the software and this document is with J.vanKatwijk, Lazy Chair Computing. While the software is available under a GNU GPL V2, the manual is not. No parts of this document may be reproduced without explicit written permission of the author. I can be reached at J.vanKatwijk at gmail dot com

# Contents

# 1 Introduction

Qt-DAB is an advanced program for decoding terrestrial DAB (DAB+) transmissions. Qt-DAB, a program with a GUI, is designed to run on both Linux (x64) computers, on RPI 2 and up running Linux, and is cross compiled for Windows

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.

For *Windows*, an *installer* is available that will install the executable together with the required libraries.

These precompiled versions can be found in the releases section of the repository for Qt-DAB (https://github.com/JvanKatwijk/qt-dab/releases).

For creating an executable on an RPI 2 or higher or any other Linux system, section 5 of this report contains a pretty detailed description with scripts for Debian and Ubuntu.

Qt-DAB is implemented in C++, with extensive use of the Qt framework for its graphical appearance. It also uses a number of existing open source libraries, such as fftw, libsndfile, libsamplerate, libusb, and Qt-DAB is itself open source, available under the Gnu GPL V2.

The sourcetree for Qt-DAB contains - obviously - sources to generate an executable for Qt-DAB. It actually contains next to a subdirectory for the main version *dab-maxi* a version called *dab-mini*.

- *dab-maxi* contains sources specific to the Qt-DAB program, the configuration files (i.e. a ".pro" file and a "CMakeLists.txt" file) and the files needed for having an appImage created for Qt-DAB.
- *dab-mini* contains sources, with configuration files and with a description on how to create an executable version with a minimal interface, i.e. dabMini. Apart from the user interface, dabMini is built on the same set of sources as Qy-DAB. *dabMini* is described in section 7 and that section includes a description of how to build an executable.

The structure of this guide is simple, in section 2 the GUI and GUI widgets for the Qt-DAB program are discussed, in section 3 (a few) command line parameters and user provided settings in the ini file (configuration settings) for the Qt-DAB program are discussed, in section 4 the supported devices and their control widgets for the Qt-DAB program are briefly discussed.

In section 5, a description is given on how to configure and build an executable for Qt-DAB. In section 6 the *device interface* as used in Qt-DAB is discussed and an explanation is given how to interface another device to the system configuration (note that the device interfaces for *dabMini* is - slightly - different).

As said, in section 7, a brief description is given of the *dabMini* program.

## 1.1 Related software

Based on the Qt-DAB sources a number of related programs is (being) developed. Of course *dabMini* is one of them, some others are mentioned below, each of these has a separate repository on Github.

**dab-cmdline** is set up as a library, with a number of command line based *example* programs. The command line in these examples is simple, a channel, a servicename and some gain settings are passed as parameter, e.g.

```
dab-airspy -M 1 -B "BAND III" -C 12C -P "Radio 4" -G 80 -A default
```

Mode and Band are by default *Mode 1* resp. *Band III*, see the README in https://github.com/JvanKatwijk/dab-cmdline for details.

Example 2, the most common one, can be configured to provide support for any of RTLSDR based sticks, SDRplay devices, HackRF and Lime devices, Adalm Pluto devices and file input.

**terminal-DAB-xxx** is a program to run a DAB decoder, without using a complex GUI and accompanying libraries as Qt. As can be seen on the picture in figure 1, available services are listed on the command terminal (using the Curses library). Indicating a service in the list for selection is by using the up or down keys. The *next* or *previous* channels can be selected using the plus resp. minus keys. To keep things simple, support for the device is compiled in.

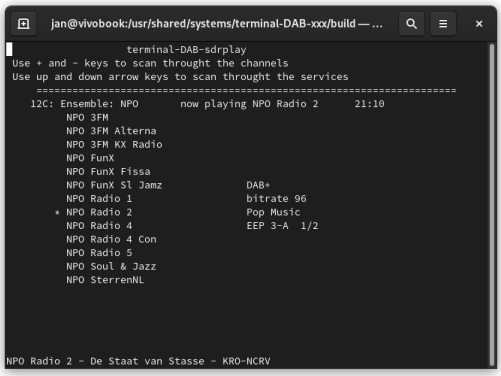



Figure 1: Qt-DAB: the terminal-DAB window

As configuration option, slides, passed as Program Associated Data (PAD), can be made visible in a separate widget. See *https://github.com/JvanKatwijk/terminal-DAB-xxx*.

**duoreceiver** Since both the FM broadcast band as BAND III, where the DAB transmissions are, are covered by common SDR hardware, it seems obvious to have a receiver covering both the FM and DAB transmissions.

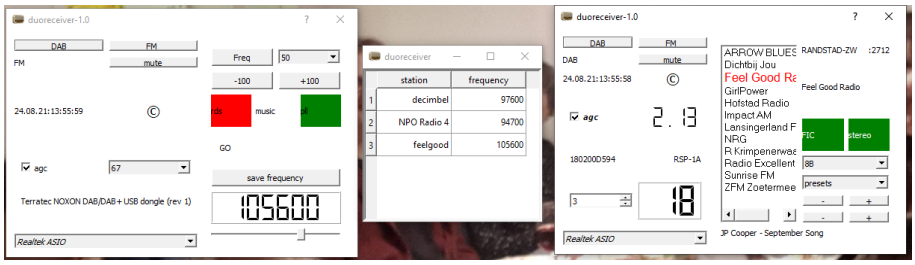



Figure 2: Qt-DAB: the duoreceiver

*Duoreceiver* is such a receiver, basically a combination of dabMini, and the FM software, see figure 2). Figure 2 shows instances of both the DAB selection and the FM selection in a single screen.

*channelScanner* is the command line driven little brother to run a scan over a set of specified channels. If a channel contains (detectable) DAB data, a record will be written with a description of the contents of the ensemble, transmitted over that channel. Furthermore, the command line based version has as command line option to dump the raw input of the channels containing DAB data onto a file in the xml format (https://github.com/JvanKatwijk/channel-scanner).

**DAB server controlled over the web** A simple DAB server, to be controlled using an extremely simple web interface, is yet another member of the Qt-DAB family. The web interface is shown in **??**. It merely contains controls to select a channel, a service and some device settings. Sources can be found https://github.com/JvanKatwijk/webDAB.

# 2 The GUI

## 2.1 Introduction

Extensibility played a major role in the development of the GUI for Qt-DAB. Curiosity and a continuously growing of the functionality lead to the current design: a single (relatively) simple (main) widget with other widgets that can be made visible when needed.

Visibility of the additional widgets is controlled by button on the main widget. Of course, the number of buttons on the main widget is fairly large, while most of the time none of these buttons is needed. The main widget is equipped with an additional button to hide or show the button panel as shown in figure 3.

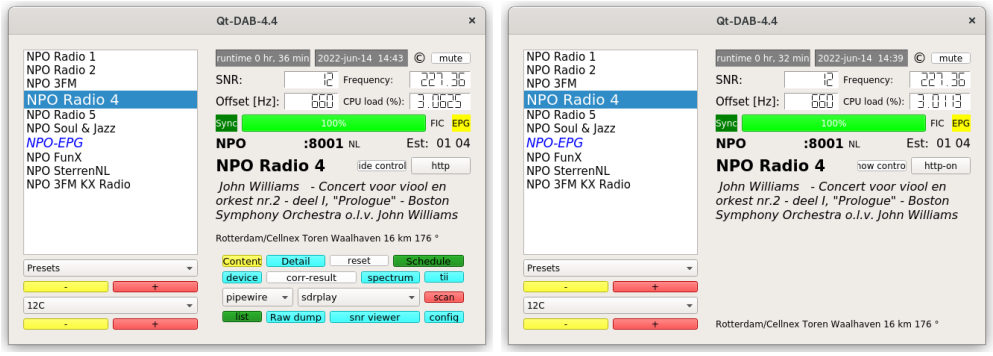



Figure 3: With and without controls

## 2.2 Main widget

The main widget is shown *permanently*[1] It can be thought to consist of three elements:

- the left part, handling control for channel and service;
- the top right part displaying information;
- the bottom right part, the various controls.

### 2.2.1 Control for channel and service

Central in the left part of the main widget is the list of services, this list shows the services detected in the currently selected channel *Selecting* a service is by moving the cursor to the name of a service, and clicking with the *left* mouse button. Clicking with the *right* mouse button on an audio service *that is not the currently selected one* will start the service in the background and feed the AAC frames into a file.

Below the list of services (see figure 4) there is (from top to bottom)

- the combobox for the *presets*. A preset can be *added* to the preset list by clicking with the *right* mouse button on the name of the selected service in the service list. Clicking with the left mouse button on the entry in the preset list instructs the software to select the *channel*, wait until the services of the channel are visible, and finally, select the service. *Removing* an element from the list is by clicking with the *right-hand mouse button* on the entry in the preset list.

[1] closing this window will terminate the program execution

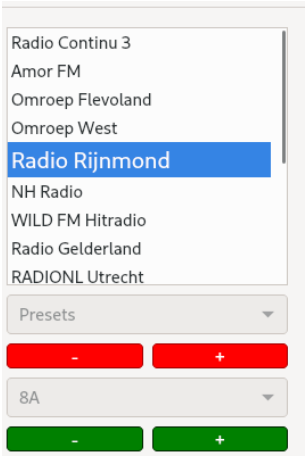



Figure 4: Qt-DAB, channel and service selection

- a *previous* (−) and a *next* (+) service button. With these button one can easily scan through the list of services.
- the combobox for *channel selection*. While (regular) DAB transmissions are in Band III, configuration provides options to select channels in the *L Band* or channels in a user defined band. The channel names are the elements in the combobox.
- a *previous* (−) and a *next* (+) channel button, making it easy to scan through the channels in the selected band.

Note that the software will record the selected channel and service, on program termination these values will be saved, and on program start up, these values will be taken as start value.

### 2.2.2 Displaying information

Some general information is displayed in the top half of the right side of the main widget, see figure 5.


Figure 5: Qt-DAB, system wide information

The top line gives four (sometimes five) elements

- when muting, the remaining *muting time* in seconds is displayed. If audio is not muted, the number display is *not* shown. The picture shows that muting is off.

- 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 actual time of listening;
- the *copyright symbol*. Touching this with the cursor when the widget is in focus, will reveal (a.o) the time and date the executable was built.
- the *mute* button, touching it will mute the audio output, for at most a time specified in the ".ini" file. The time - in minutes - can be set in the configuration widget. Touching the button while muting, will unmute the audio output.

Below this line, there are boxes with labels:

- SNR, the measured signal/noise ratio. SNR is computed as the overall strength of the signal compared to the strength during transmission of the NULL period of DAB frames;
- Frequency, the frequency, in MHz, of the selected channel;
- Offset, the frequency correction applied to the signal (in Hz). This offset is computed from the samples in the time domain;
- 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). A value less than 100 percent here usually indicates a poor reception;
- if an EPG decoder is running in the background, a yellow field with the text "EPG" is shown.

The remaining part is devoted to describing the name of the ensemble, displayed together with its ID (a hex number), if it can be deduced, the name of the country the transmission is received from, and (usuall) two number preceded by "Est". The numbers give an estimate of the so called TII codes, *Transmitter Identification Information*. The TII information - 2 numbers - is unique for each transmitter. Since Qt-DAB-4.3 the bottom line of this section - depending on the configuration - shows - if found - the name of the transmitter received, its distance to the receiver and the azimuth. Since the provider of the database has requested not to reveal the source of the database, the code loading and interpreting the database is only available as an non-open source library (see directory "library"). The code to handle tiiFiles is included in the precompiled versions for both Windows and Linux though.

Version 4.4. adds the possibility of showing the location of the transmitter on a map. The button "http" (or, if switched on "http on") is there to switch a small server "on" or "off" that sends the relevant data to port 8080. A web browser, when set to address "localhost:8080" on the local machine will show the map and a dot (or dots) on the location of the transmitter(s).

Furthermore, the section shows the name of the selected service, and the "hide" button and - if configured - the afore mentioned "http" button.

Furthermore, there is some room for the text of the *Dynamic Label*.

### 2.2.3 Various controls

The control buttons on the main widget contain for the various widgets, discussed in the following subsections, a button to show or hide the widget. The function of a few other buttons are discussed here.

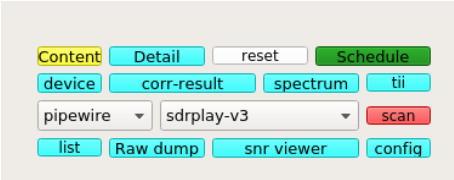



Figure 6: Qt-DAB: control buttons

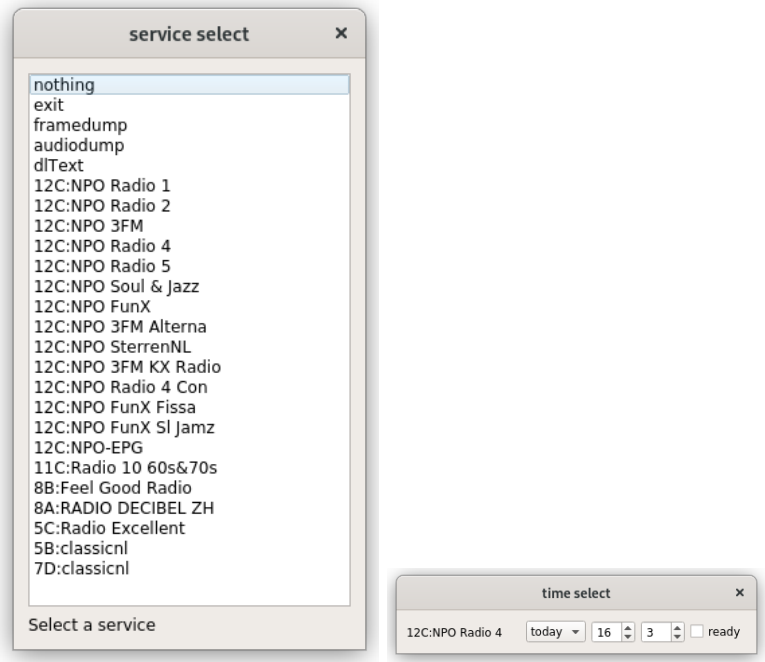



Figure 7: Qt-DAB: schedule select

**The reset button** The reset button stops and starts the current channel.

**The schedule button** Touching the schedule button will show a list of services and *actions* from which a selection can be made. The list is composed of the actions, the services in the current ensemble and the services in the preset list, see figure 7. For the actions *audiodump* and *framedump* it holds that switching to another service will stop the action. The *dlText* action is not bound to the current service.

After selecting the service a small *time select* widget will show where the time for the action can be specified to up to a week ahead.

**Audio channel select** The combobox, labeled *pipewire* in figure 6, contains the audio channel names. Of course the names of the audio channels depend on the system the program is running on.

**Device selector** The combobox, labeled *sdrplay-v3* in figure 6, contains the names of the configured devices. Note that changing the device while running the program is - at least in principle - possible.

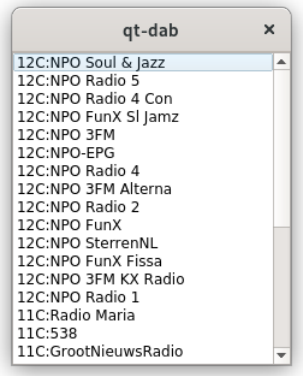



Figure 8: Qt-DAB: history list

**list** Inspired by my car radio, all services encountered are stored in a list, the history list. Selecting an entry in the list (see figure 8 is possible. Of course, when using a laptop on different locations, or with file input, not all entries are reachable.

Clicking with the *right hand* mouse button will *erase all entries in the list.*

**Raw dump** Dumping the input in 16 bits signed stereo PCM files is possible. Note that the file size is large, it will increase with 8 Mbyte per second.

## 2.3 The correlation widget

As probably known, DAB uses Single Frequency Networks (SFN) for transmission. The basic idea is to use a number of low(er) power transmitters than - as is common for AM and FM - a single high power transmitter to cover an area.

The receiver will - most likely - receive the signal from more than a single transmitter and has to synchronize with one of the signal streams to extract the data from the transmission. *Correlation* is used to synchronize, and the figure shows the result of that correlation. The receiver synchronizes - in this example - with the signal that transmits the sample on position 504, the correlation widget shows the highest peak at this position.

There are three other correlation peaks, each of which belongs to another transmitted signal. So, the picture shows that signals from at least 4 transmitters are received. Since we know that there are

2048000 samples per second, we can even compute the delay of the various signals. the peak at position 705 is 201 samples behind the sample used for synchronization: 201 samples is about 50 micro seconds behind, and since signals travel with app 300 meter per micro second, that leads to a distance of 15 km.

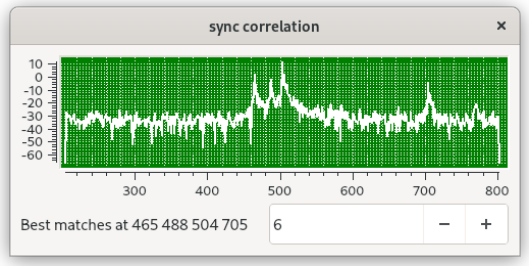



Figure 9: Qt-DAB: correlation result

The picture makes sense, the transmitter with the signal used is, from where I live, about 18 Km away, the two signals arriving early come from transmitters with a distance of app 6 and 10 Km, and the signal arriving late from a transmitter on about 35 km distance.

The spinbox at the bottom of the widget controls the length of the segment of the correlation result that is made visible.

## 2.4 The TII spectrum

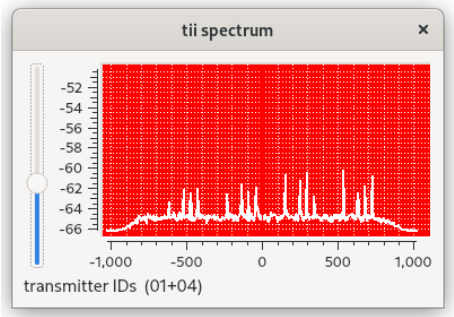



Figure 10: Qt-DAB: tii spectrum

The various transmitters in a Single Frequency Network usually carry their own identification. A DAB signal consists of frames, each starting with app 2600 low amplitude samples, followed by 76 blocks of app 2550 samples carrying the data. The so-called NULL period is not completely empty, the NULL period of each second frame contains some data, the spectrum of which shows a kind of code. The picture, figure 10 shows such code. In the picture one sees 4 times a group of 4, each group of four forms a kind of 4 out of 8 code that can be decoded.

The transmitter's name is - obviously - derived from the TII data.

## 2.5 The spectrum scope

The spectrum scope shows - as the name suggests - the spectrum of the incoming signal. Ideally, such a spectrum looks like the one in figure 11, just an almost flat line. The black field at the right hand side shows the (almost ideal) signal constellation of the decoded DAB signal. There are 4 dots, it is easy to see that a dot in each quadrant corresponds to two bits (4 possibilities).

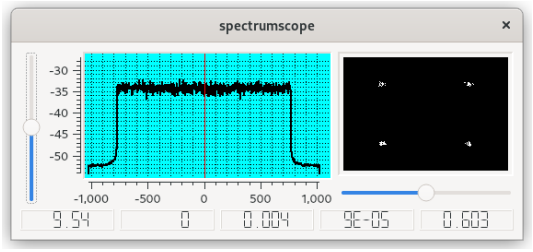



Figure 11: Qt-DAB: ideal spectrum

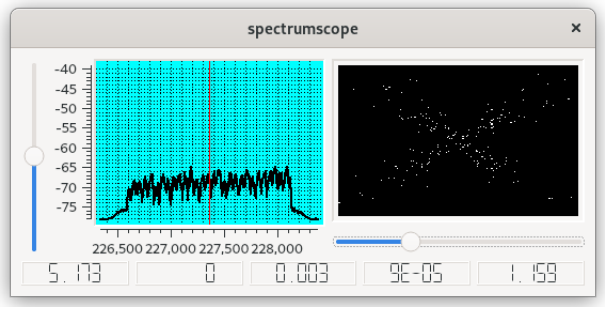



Figure 12: Qt-DAB: spectrum scope

In reality the spectrum of the signal is usually more like that from figure 12.

The bottom line of the widget shows some quality indicators of the received signal. From left to right:

- the degree to which the constellation matches the ideal constellation (10 is maximum);
- a clock offset, what is counted is the number of samples received and the difference - for 10 subsequent DAB frames - with nominal number of samples (20480000) is measured;
- the offset of the receiver clock in taking the samples;
- the error as detected in the clock of the AD converter;
- the remamining frequency offset, i.e. after correction.

## 2.6 The SNR widget

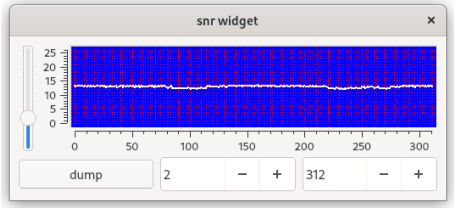



Figure 13: Qt-DAB: SNR in time

The SNR (Signal Noise Ratio) of the signal is continuously measured. The value is computed as the difference - in dB - between the overall signal amplitude and the signal amplitude in the NULL period.

The widget shows the development of SNR over time. By default measurement is taken as an average over two DAB frames, since only one of the two carries TII information in the NULL period. The spinbox at the bottom allows choosing another value for averaging. The other spinbox, default set to 312, is for setting the length of the X-axis.

A *dump* button is there for recording the values in a file (a small application exists for inspecting such a file).

## 2.7 Technical details of the selected service

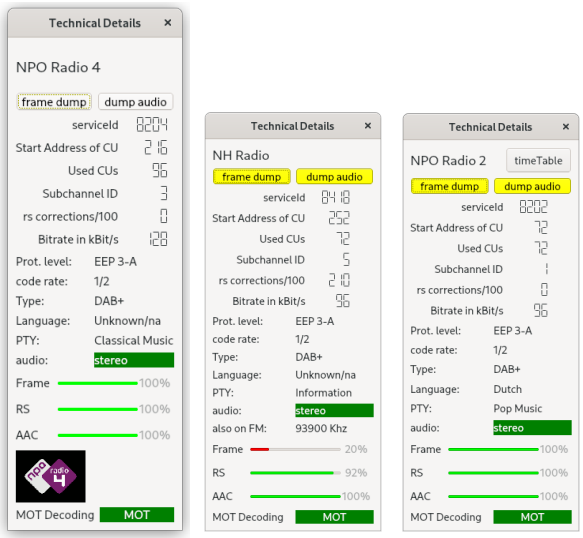



Figure 14: Qt-DAB: technical details

When selecting a service the main widget shows only the service name. Of course there are many more details. By clicking the *Details* button these details can be made visible in a separate widget, see figure 14.

The *technical details* widget contains two buttons to dump audio output into a file, one for dumping the AAC frames, the other one for dumping the PCM output (i.e. a ".wav" file).

By default the station label is displayed at the bottom of the widget, this can be changed by a setting in the configuration widget.

Some audio services can be heard on FM as well, some of those services carry as additional information the frequency in the FM band where the audio can be extracted.

In some other cases the EPG (Electronic program Guide) data can be extracted from the channel, and if that data is complete, an indication is shown at the top of the widget.

The progress indicators at the bottom give the success rates of the various steps in the transform from decoded DAB data to PCM output. For classical "DAB", using MP2 rather than He-AAC, only a single progress indicator is shown. The indicator *rs corrections/100* is somewhat special, it indicates the number of errors corrected by the Reed-Solomon module per 100 frames (per frame at most 5 byte errors can be repaired).

## 2.8 Configuration widget

Some elements in the settings of the Qt-DAB software can be changed. While changing is always possible by editing the ".qt-dab.ini" file, the configuration widget provides some possibilities:

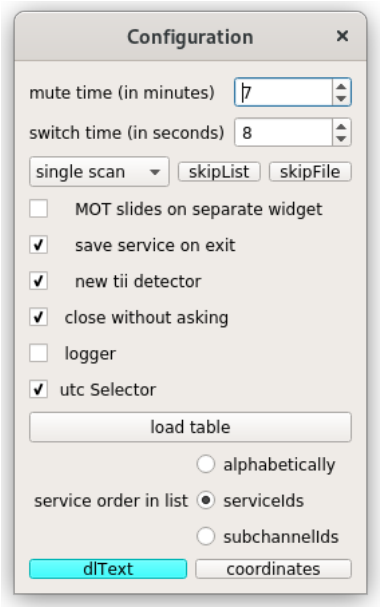



Figure 15: Qt-DAB: configuration widget

- *mute time*. The default time for muting is 2 minutes, in the spinbox labeled *mute time* this can be changed;
- *switch time*. When switching from channel to channel, it usually takes some time before the data, descrining the services in that channel, are read in and processed. When selecting a service in an other than the current channel by e.g. a preset the software will wait (at most) *switch time* seconds after setting the channel to its new value, before attempting to actually select that service;
- *scan control buttons*. Scanning, for which the main widget contains a control button is there in three tastes:
  - scan the band once;
  - scan the band once but stop as soon as a channel with a DAB signal is encountered;
  - scan the band continuously.

  With the combobox, labeled *single scan* in the picture, a selection can be made. The button labeled *skipList* - when touched - shows the skip list, a list that shows which channels are and which channels are not visited during a scan. Finally a non-local skip list, i.e. an external file, can be selected.
- *MOT selector*. When selected a station label, or slides, will be displayed on a separate widget rather than as part of the technical data widget;
- *save service*. When selected, the service, active on termination of the program will be saved and selected when starting the program again;
- *new tii detector*. When selected another algorithm is used for decoding the TII data from the spectrum of the NULL periods of DAB frames. The algorithm is slightly less conservative and will give sooner results, with an increased risk of an erroneous result;

- *close without asking*. By default the software will ask for a confirmation when closing the program. When selected, the confirmation step is skipped;
- *logger*. When selected a file selection widget wil ask for a filename for a file in which subsequently some logging information is stored. The logger is experimental and - if per country transmitter description files are installed - will tell a.o. the transmitter name, the coordinates and the distance to the receiver. However, in this version transmitter description files are only available for a few countries on request.
- *utc selector*. When selected the time on the scans and in the log file are UTC rather than local time.
- *load table*. When pressed an attempt is made to renew (or load) the database with which the TII data is mapped upon transmitters and distances. The button will be a no-op if the tii library is not available.
- *service order*. One may choose here the order in which the services in an ensemble will be displayed.
- *dlText button*. The text appearing as dyamic label texts on the main widget may be saved in a file. Touching the button will show a file selection menu, touching the button when texts are saved will stop the process. The software maintains a small cache to avoid duplicates in this file
- *coordinates*. When touched a widget will appear on which one can give the coordinates (in decimal notation) of the transmitter. The data is - obviously - used for computing distance and azimuth.

A fragment of the output of the dlText is given below

```
12C.NPO Radio 2      2022-01-10 14:47:00  Red Hot Chili Peppers - By The Way
12C.NPO Radio 2      2022-01-10 14:47:00  NPO Radio 2 - Gijs 2.4 - KRO-NCRV
12C.NPO Radio 2      2022-01-10 14:47:00  Tina Turner - Private Dancer
12C.NPO Radio 2      2022-01-10 14:51:00  Son Mieux - The Mustard Seed
12C.NPO Radio 4      2022-01-10 14:52:00   Charlie Haden   - Ellen David
12C.NPO Soul & Jazz  2022-01-10 14:53:00  Alicia Keys - Teenage love affair
```

## 2.9 Extended content description

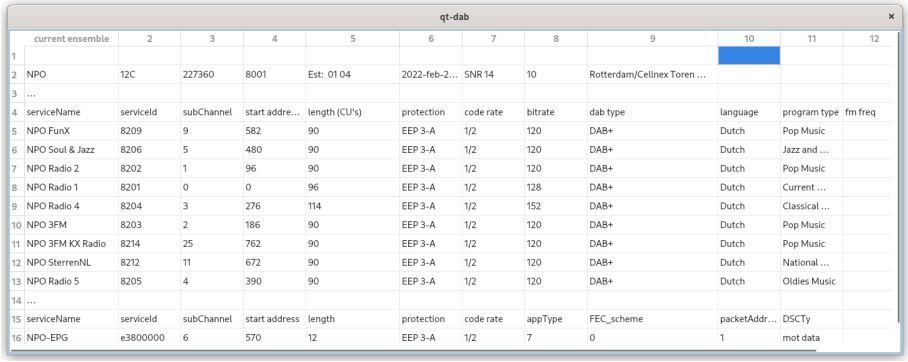



Figure 16: Qt-DAB: extended content

Sometimes one wants to see more of the services in the current ensemble. The *content widget* gives some more details as can be seen in figure 16. In Qt-DAB 4.35 the format of the data is equal to the format of the data when scanning, basically a ".csv" content with 12 columns.

Of course, one might select a service by clicking with the left mouse button on the service name in this extended overview.

*Double clicking* on any place in the widget will activate a file selection menu where one may choose a filename for a dump of the content of the widget into a ".csv" file.

## 2.10 Schedule list

Qt-DAB provides a schedule facility, for up to 7 days. The schedule list (figure 17) shows the not yet fullfilled actions for the scheduler. The list shows that - next to selecting a service at a specified time - also some actions can be scheduled. While the action *nothing* does not do much, the action *exit* causes - when executed - the program to stop, the action *framedump* has the same effect as touching the framedump button on the technical details widget, but with a generated (recognizable) name. The action *dlText* - when executed - has the same effect as touching the *dltext* button, but with a generated - recognizable - filename. These files are by default written in the user's home directory.

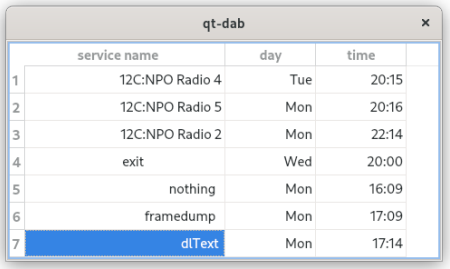



Figure 17: Qt-DAB: schedule list

If the program terminates, the schedule actions are written into a file and on program (re)start that file is read, the execution time for the actions is recomputed and a new list is created. Actions for which the execution time is then in the past are deleted.

## 2.11 Device widget

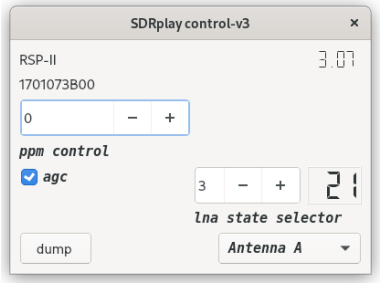



Figure 18: Qt-DAB: device widget

On program start, Qt-DAB will try to open the device that was used the last time the program ran. The main widget obviously contains a selector to select a device, the widget for device control can be made (in)visible.

In a later section in this manual the widgets for all supported devices are discussed.

## 2.12 Colors and coloring

Qt-DAB supports user defined coloring buttons and the various displays. Since it is most likely that others prefer different colors than I do, a *fixed* color scheme does not seem appropriate. A flexible

approach was chosen, one that allows the user to make color settings and changes directly from the GUI. Obviously, the color settings will be stored in the ".ini" file and used the next program invocations.

### 2.12.1 Colors that can be selected

The set of colors from which one can be selected is defined by the Qt system. The colors are represented by strings:

```
white, black, red, darkRed, green, darkGreen, blue,
darkBlue, cyan, darkCyan, magenta, darkMagenta, yellow, darkYellow,
gray, darkGray
```

### 2.12.2 Setting the colors of buttons

Since buttons with *light* colors are best visible with a *dark* font for the button text, and since buttons with *dark* colors are best visible with a *light* (white) font for the button text, both the base color of the button and the color of the text can be set. Just click with the right mouse button on a button, and twice a small menu will appear with the possible colors, the first one for the base color of the button, the second one for the text color on the button.

### 2.12.3 Setting the colors in the displays

Similar as for buttons, the colors for the displays can be set from the GUI. Click with the right mouse button on a display, and - as the picture shows - a selector will appear with a list of the supported colors. *Three* times a color has to be selected,

- the *display color*, for selecting the background color of the scope;
- the *grid color*, for selecting the color of the grid; and
- the *curve color*, for selecting the color of the line.

Setting a *brush* is possible by adding *brush=1* in the appropriate section for the widget in the ".ini" file. The color settings are kept in the ".ini" file, in sections resp *spectrumViewer*, *tiiViewer* and *correlationViewer*.

## 2.13 Putting it all together

While it most likely does not happen often, it is of course possible to show all widgets simultaneously, as is shown in figure 19.

# 3 Command line parameters and the ini file

While the GUI provides lots of control, some settings can be done via the command line or by editing 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 by default it is kept in the user's home directory.

## 3.1 Command line parameters

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

- "-i filename" to use the file *filename* as ".ini" file rather than the default one ".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), which is - obviously only meaningfull when configured.

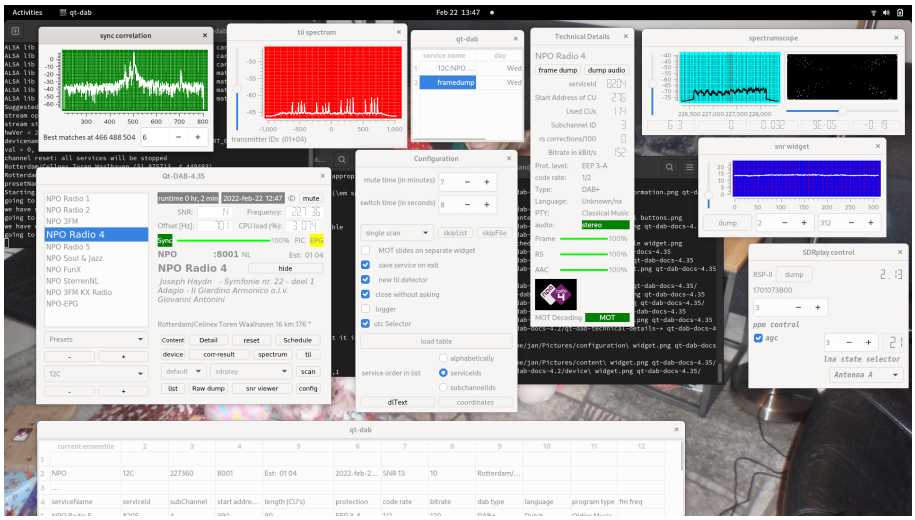


Figure 19: Qt-DAB: full screen

- "-A filename" to use the (name, integer) pairs in the file as channel definitions rather than the channels in Band IIIs. The sourcetree contains a small file as example: *testband*.
- "-F XXX" specifies a frequency on KiloHerz, works when configured for receiving and transmitting using an Adalm Pluto device;

## 3.2 Settings in the ".ini" file

Settings are stored in the ".ini" file. Note that, next to settings made by the user, the software will store *some* settings on current selections (e.g, device, channel, service) in the ".ini" file. Here we discuss the settings that cannot be set or modified from the configuration widget.

- save_gainSettings. By *default* the gain settings per channel are saved in the ".ini" file. Since these settings depend on the device, for each device section a setting "save_gainSettings=0" can be added to ignore previous values for gain setting of that channel when selecting a channel.
- dabMode: While the *default* Mode for DAB is Mode 1, Qt-DAB provides the possibility to use the obsolete Mode 2 or 4 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 obsolete L Band by setting "dabBand=L_Band". Note that passing a file with a band description as parameter overrides specifying the band in the ".ini" file.
- 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;
- saveSlides: While the *default* is 1, implying that decoded slides are saved, setting "saveSlides=0" will prevent slides to be saved;
- 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).
- 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";
- 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;
- 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";
- font and font size: The default setting for the font resp fontsize for displaying the service names in the service list is *Times* resp. 12. Since the choice of fonts is - as e.g. colors on the GUI - personal, an option is created to set font and font size. Set

  ```
  theFont=Canterell
  fontSize=14
  ```

  (or values to your likings) to change these.

Other values in the ".ini" file are set - and maintained - by the software or can be set through the configuration widget (e.g. color settings, gain settings, current device, current channel, service etc etc).

# 4 Supported input devices

Qt-DAB supports a variety of input devices, the Adalm Pluto, the AIRspy, the hackrf, the limeSDR, RT2832 based sticks and SDRplay RSP devices. Furthermore, there is support for the rtl_tcp server, for file input (raw, wav and xml), and for devices for which a *Soapy* interface library exists,

Both the *appImage* and the *Windows installer* are configured with (almost) the whole range of devices: SDRplay RSP (different versions for the 2.13 and 3.0X library versions), the Adalm Pluto, the AIRspy, the hackrf, the LimeSDR, and - of course - the RT2832 based dabsticks.

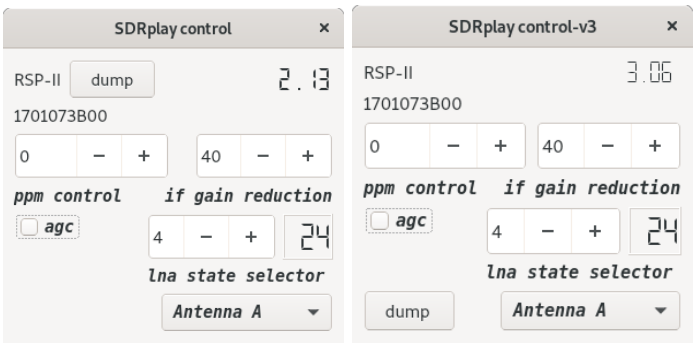



Figure 20: Qt-DAB: The two control widgets for the SDRplay

## 4.1 The SDRplay RSP

The Qt-DAB software supports all RSP's from SDRplay. Qt-DAB provides two different device handlers for the RSP's, one for devices using the 2.13 SDRplay interface library, the other one supports devices using the 3.0X SDRplay interface library.

As figure 20 shows, the control widgets for the two different versions resemble each other, their implementations differ considerably though. 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 offset is almost zero in the region of Band III.

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, I do not have a duo).

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, a suggestion for the name of the file *device name - date* is given. Touching the button again will stop dumping and the file will be closed.

If more than one connected device is detected, a widget appears on which a selection can be made which device to use.

## 4.2 The AIRSpy

The control widget for the AIRspy (figure 21, 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 21 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).

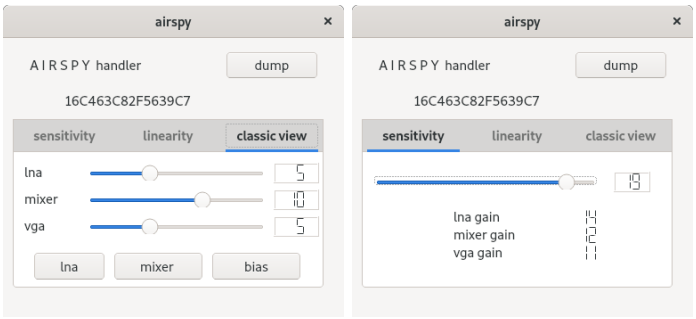



Figure 21: Qt-DAB: Widgets for AIRspy control

If more than one connected airspy is detected a widget will appear with which the device to use can be selected.

## 4.3 The hackrf

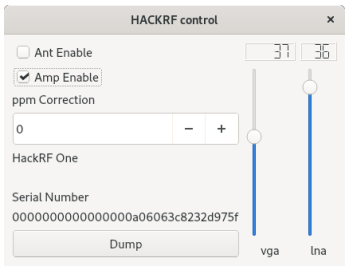



Figure 22: Qt-DAB: Widget for hackrf control

The control widget for hackrf (figure 22) 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

On selecting the LimeSDR (if configured), a control widget for the LimeSDR is shown (figure 23). The widget contains five controls:

- *gain* control, with predefined values;
- *antennas*, where *Auto* is usually the best choice;

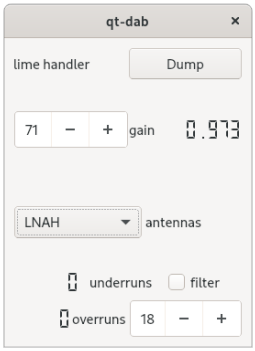



Figure 23: Qt-DAB: Widget for Lime control

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

New is the inclusion of a filter. Note that the limeSDR reads samples with a bandwidth of 204KHz, while the frequency distance between successive channels is 1712KHz. So, DX-ing in adjacent channels, where e.g. the first channel contains a strong DAB signal and the next one a weak, is difficult.

Therefore the control widget for the limeSDR has two additional controls,

- switching a software FIR filter on-off (the checkbox labeled *filter*),
- setting the size of the FIR filter (the spinbox below the checkbox).

Note that using the filter is not free, for a filter with a size of N, N * 2048000 complex additions and multiplications are performed. While on a modern PC that is not an issue, it certainly is on ARM based micros like the RPI 2.

## 4.5 The RTLSDR stick

On selecting the dabstick (i.e. RT2832 based devices) (if configured), a control widget for the device appears (figure 24).

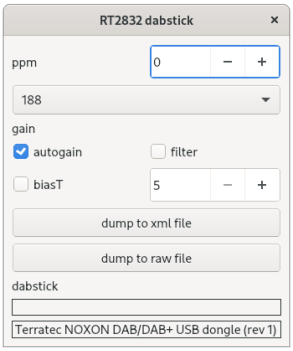



Figure 24: Qt-DAB: Widget for rtlsdr device

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;
- 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.

New is the inclusion of a *filter*. Note that the DABstick reads samples with a bandwidth of 2048 KHz for a signal with a bandwidth of app 1.536 MHz, while the frequency distance between successive channels is 1712KHz. So, DX-ing in adjacent channels, where e.g. the first channel contains a strong DAB signal and the next one a weak, is difficult.

The controller therefore contains an optional FIR filter, for which the rtlsdr control widget has two additional controls:

- switching a software filter on-off (the checkbox labeled *filter*),
- setting the size of the FIR filter (the spinbox below the checkbox).

Note that switching the filter on is not free, for a filter with a size of N, N * 2048000 complex additions and multiplications per second are performed. While on a modern PC that is not an issue, it certainly is on ARM based micros like the RPI 2.

If more than one connected RTLSDR based device is detected, a widget appears on which the device of choice can be selected.

## 4.6 The Pluto device

When selecting *pluto*, a widget (figure 25) appears with a spinbox for selecting the gain, and a checkbox for selecting the agc. If *agc* is enabled, the spinbox for the gain setting is invisible. The widget contains

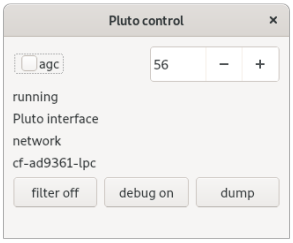



Figure 25: Qt-DAB: Widget for Adalm Pluto device

furthermore three buttons:

- the *debug control* button, when activated, instructs the software to show output on each step in the initialization process (note that the setting of the debug button will be maintained between invocations);

- the *dump* button will cause the original input - with a samplerate of 2100000 samples per second[2] - to be stored in an xml file.
- the *filter* button. The adalm pluto has as option specifying a fir-filter, to be executed within the Pluto device. This implementation of the controller for pluto will load a predefined filter onto the Pluto device which is enabled by default. With the filter button the filter can be disabled or enabled. Note that the button text indicates the action when touching, not the current state.

## 4.7 Support for Soapy


Figure 26: Qt-DAB: Widget 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.

The widget for soapy control (see figure 27) when applied to the Soapy interface for the SDRplay contains the obvious controls, similar to that of the regular control for the SDRplay.

## 4.8 rtl_tcp

*rtl_tcp* is a server for rtlsdr devices, delivering 8 bit IQ samples.

In the small widget, the ip address of the server can be given. Since the default port for the server is 1234, that port number is the one used by the client. After clicking the *connect* button, the client will look for a server and pass some parameters.

However, the port number can be set in the ".ini" file, by setting

```
rtl_tcp_port=XXX
```

where XXX is to be replaced by the portnumber of choice.

## 4.9 File input

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

- 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";

[2]The smallest samplerate that pluto gives is slightly larger than the required 2048000, 2100000 is chosen since it is easy to handle

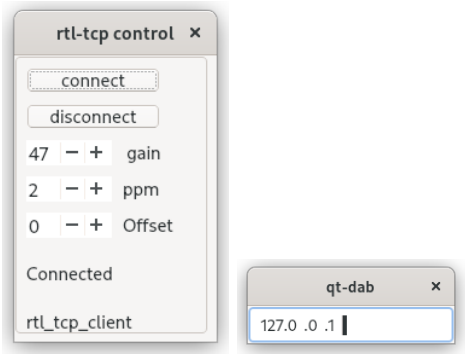



Figure 27: Qt-DAB: Widget for rtl_tcp

- 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.

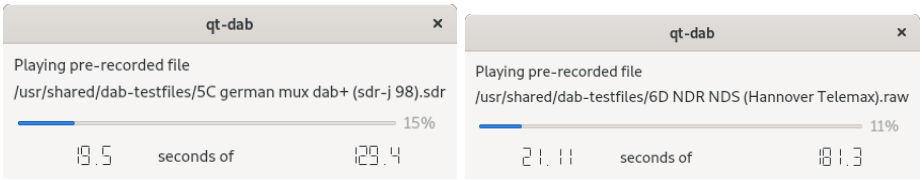



Figure 28: Qt-DAB: Widgets for file input

When selecting file input ".raw" or ".wav", a simple widget is shown (figure 28), with as indication the number of seconds the file is being played.

Since processing an xml file implies some interpretation, the widget (figure 29) 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.

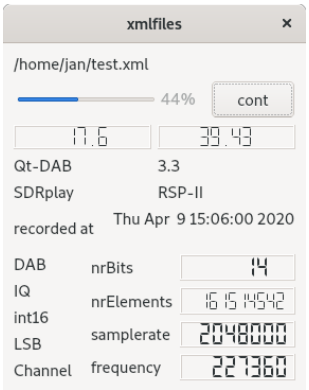



Figure 29: Qt-DAB: Widget for xml file input

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

# 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 *win32* for Windows that contain settings specific to the OS used. The CMakeLists.txt file is only used for Linux-x64.

Both the AppImage and the Windows installer are built using qmake as makefile generator.

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

It turns out that referring to and linking the qwt library sometimes gives problems.

For my Ubuntu 16 system (the one where I build the AppImage) the setting for the qwt library should be

```
#INCLUDES += /usr/include/qwt
```

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 might have to be chosen, commenting out the other one.

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

The sa

### 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 by default configured in Qt-DAB executables, changing this is possible, but implies 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 += airspy
CONFIG += dabstick
CONFIG += sdrplay-v2
CONFIG += sdrplay-v3
CONFIG += lime
CONFIG += hackrf
CONFIG += pluto
CONFIG += soapy
CONFIG += rtl_tcp
```

Note that for *soapy*, and for *limeSDR* there is no support in generating a windows executable, due to the absence of a suitable dll. Furthermore, for Windows select "pluto-2" rather than pluto.

**Using the CMakeLists.txt file** The CMakeLists.txt file contains support for AIRspy, SDRplay, SDRplay_V3, RTLSDR, Hackrf, pluto 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 the TII database

**Using qmake/make** Configuration for (not) using the database is by selecting one of

```
#CONFIG += preCompiled
CONFIG += tiiLib
#CONFIG += noTables
```

Configuring for *tiiLib* will select program items such that a *tiiLib*, if available of course, will be loaded and interpreted. *If no such library is available, Qt-DAB will function as if* noTables *is configured.*

Selecting *preCompiled* will require sources that are not part of the source tree, but are available upon request under a non-disclosure license (contact the author).

**Installing the tiiLib** The tiiLibrary is - for Linux-x64 and for RPI - available in the source tree. The library should be placed in e.g. /usr/local/lib. Note that the library itself depends on *curl*, so the curl library should have been installed as well.

**Using cmake** In the CMakeLists.txt file, by default the "tiiLib" configuration is chosen.

### 5.2.5 Configuring SSE

In the deconvolution of data, use is made of 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. If the code is to run on an RPI, it is - depending on the configuration - sometimes possible to speed up the process by using ARM specific 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

```
CONFIG += PC
#CONFIG += RPI
#CONFIG += NO_SSE
```

Selecting "CONFIG += PC" selects SSE instructions, and deselects threading of backends - after all, a standard PC has more than sufficient power to run the decoding in a single thread.

Selecting "CONFIG += RPI" selects options suitable for having the software run on an RPI. However, the precise content depends on the processor architecture and the compiler chain.

Selecting "CONFIG += NO_SSE" is for e.g. Mingw cross compiler for Windows.

When using *cmake*, pass "-DVITERBI_SSE=ON" as command line parameter for PC's.

### 5.2.6 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".

### 5.2.7 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.

### 5.2.8 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.

### 5.2.9 Selecting an AAC decoder

By default the *faad* library is used to decode AAC and generate the resulting PCM samples.

The source tree contains - in the directory *specials*, the sources for the libfaad-2.8 version. It is quite simple to create and install an appropriate library if the Linux version supports a faad liubrary that is somehow incompatible.

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
```

(see the subsection for installing the libraries).

### 5.2.10 Configuring for platforms

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
DEFINES += __MSC_THREAD__
```

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

It is recommended to use

```
CONFIG += PC
```

in the qt-dab.pro file, when targeting towards a standard x64 based PC running Linux, using this will set the SSE and the threading.

It is recommended to use

```
CONFIG += RPI
```

in the qt-dab.pro file when targeting for an RPI, the threading will be set and the NO_SSE option is set.

### 5.2.11 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: installing 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 Buster 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 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
```

*Note that on newer versions of Ubuntu, this list might change, especially w.r.t. Qt5 libraries. It seems that "qt5-default" is not available as separate library anymore, it just can be removed then.*

If *libfaad* is the selected aac decoder, install

```
sudo apt-get install libfaad-dev
```

If *fdk-aac* is the selected aac decoder, install

```
sudo apt-get install libfdk-aac-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 from the repository by

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

The command will create a directory *qt-dab* with subdirectories for dab-mini, dab-maxi and the unsupported dab-2.

### 5.3.3 Installing support for the Adalm Pluto

The Pluto device uses the *iio* protocol. Support for *Pluto* is by including

```
sudo apt-get install libiio-dev
```

and - to allow access for orinary users over the USB - ensure that the user name is member of the pugdev group, and create a file "53-adi-plutosdr-usb.rules" is in the "/etc/udev/rules" directory.

```
#allow "plugdev" group read/write access to ADI PlutoSDR devices
# DFU Device
SUBSYSTEM=="usb", ATTRS{idVendor}=="0456", ATTRS{idProduct}=="b674",
MODE="0664", GROUP="plugdev"
SUBSYSTEM=="usb", ATTRS{idVendor}=="2fa2", ATTRS{idProduct}=="5a32",
MODE="0664", GROUP="plugdev"
# SDR Device
SUBSYSTEM=="usb", ATTRS{idVendor}=="0456", ATTRS{idProduct}=="b673",
MODE="0664", GROUP="plugdev"
SUBSYSTEM=="usb", ATTRS{idVendor}=="2fa2", ATTRS{idProduct}=="5a02",
MODE="0664", GROUP="plugdev"
# tell the ModemManager (part of the NetworkManager suite) that
# the device is not a modem,
# and don't send AT commands to it
SUBSYSTEM=="usb", ATTRS{idVendor}=="0456", ATTRS{idProduct}=="b673",
ENV{ID_MM_DEVICE_IGNORE}="1"
SUBSYSTEM=="usb", ATTRS{idVendor}=="2fa2", ATTRS{idProduct}=="5a02",
ENV{ID_MM_DEVICE_IGNORE}="1"
```

### 5.3.4 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 ..
```

### 5.3.5 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 ..
```

### 5.3.6 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".

### 5.3.7 Making the installed libraries visible

The installation of these device handlers will install libraries in the

```
/usr/local/lib
```

directory. Note that the path to this directory is NOT standard included in the search paths for the Linux loader. To add this path to the searchpaths for the Linux loader, create a file

```
/etc/ld.so.conf.d/local.conf
```

with as content

```
/usr/local/lib
```

The change will be effective after executing a "sudo ldconfig" command.

The installation of these device handlers will furthermore install some files in the

```
/etc/udev/rules.d
```

directory. These files will ensure that a non-root user has access to the connected device(s).

Note that in order for the change to be effective, the *udev* subsystem has to be restarted. The easiest way is just to reboot the system.

## 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/dab-maxi
mkdir build
cd build
cmake .. -DSDRPLAY=ON -DPLUTO=ON -DAIRSPY=ON -DRTLSDR=ON ... -DRTL_TCP=ON
make
```

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

```
sudo make install
cd ..
cd ..
```

### 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/dab-maxi
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.

## 5.5 Configuring for pluto-rxtx

Qt-DAB can be configured to use the Adalm Pluto as both receiving device and as transmitting device, but in Linux only/ Configuring is easy,

```
CONFIG         += pluto-rxtx
#CONFIG         += pluto
```

is all that is needed, i.e. unselect regular pluto and select pluto-rxtx.

When configured, and when the device *pluto-rxtx* is selected, the audio of the selected service, augmented with the text of the dynamic label encoded as RDS signal.

Transmission frequency is set default to 110 MHz, but can be set as command line parameter using the "-F XXX" flag, where XXX stands for the frequency **expressed in KHz**.

*Note that this only applies to configuring for Linux and using qt-dab.pro with qmake for configuring*

# 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 the devices I have access to, there is obviously a multitude of other devices that are worthwhile to be used with Qt-DAB.

## 6.1 The Qt-DAB device interface

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

The interface is defined as a class, where the actual device handler inherit from.

```
class deviceHandler {
public:
         deviceHandler  ();
virtual ~deviceHandler ();
virtual bool restartReader (int32_t);
virtual void stopReader ();
virtual int32_t getVFOFrequency ();
virtual int32_t getSamples (std::complex<float> *, int32_t);
virtual int32_t Samples ();
virtual void resetBuffer ();
virtual int16_t bitDepth ();
virtual void show ();
virtual void hide ();
virtual bool isHidden ();
virtual QString deviceName ();
private:
int32_t lastFrequency;
};
```

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

- *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 should have no effect.
- *stopReader* will do the opposite of restartReader, collecting samples will stop; *stopReader* when *not* running should have no effect.
- *getVFOFrequency* returns the current oscillator frequency in Hz;
- *getSamples* is the interface to the samples. The function should provide a given amount of samples, the return value is, however, 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 a 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.
- *deviceName* returns a name for the device. This function is used in the definition of a proposed filename for *dump*s.
- 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).

## 6.2 What is needed for another device

Having an implementation for controlling the new device, the Qt-DAB software has to know about the device handler. This requires adapting 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 implement 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     += ./qt-devices/new-device
   HEADERS         += ./qt-devices/new-device/new-device.h \
                      .. add further includes to development files, if any
   SOURCES         += ./qt-devices/new-device/new-device.cpp \
                      .. add further implementation files, if any
   FORMS           += ./qt-devices/new-device/newdevice-widget.ui
   LIBS            += .. add here libraries to be included
}
```

**Modifications to radio.cpp** The file "radio.cpp" needs to be adapted 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. So, in the neighborhood of

```
#ifdef HAVE_AIRSPY
deviceSelector -> addItem ("airspy");
#endif
#end{verbatim}
}
the text
{]footnotesize
\begin{verbatim}
#ifdef HAVE_NEWDEVICE
deviceSelector -> addItem ("newDevice");
#endif
```

  is added.

- If selected, the class implementing the device handler should be instantiated, so, in the direct environment of

```
#ifdef HAVE_AIRSPY
if (s == "airspy") {
   try {
      inputDevice = new airspyHandler ....
....
#endif
```

  the code for allocating a device handler is added

```
#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
```

## 6.3 Linking or loading of device libraries

The approach taken in the implementation of the different device handlers is to *load* the required functions for the device library on instantiation of the class. This allows execution of Qt-DAB even on systems where some device libraries are not installed.

The different existing drivers can be used as example if there is a need to implement the dynamic loading feature. Obviously, if an executable is generated for a target system that does have the library for the device installed, there is no need to dynamically load the functions of that library.

# 7 dabMini

## 7.1 Why a dabMini

I often run a DAB decoder(s) on an RPI2 or 3. Since these RPIs are headless, control (and often the sound) is from my laptop. Sometimes I find the GUI of Qt-DAB too large, especially when my only concern is to listen to the audio. In that case I do not need any of the push buttons and the comboboxes on the main GUI widget, nor the additional widgets.

While I was using *dabRadio* for that purpose (or sometimes *qml-dab*), I realised that most of the corrections and changes that were applied to the sources - quite many - of 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* by using the Qt-DAB sources. To ensure 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

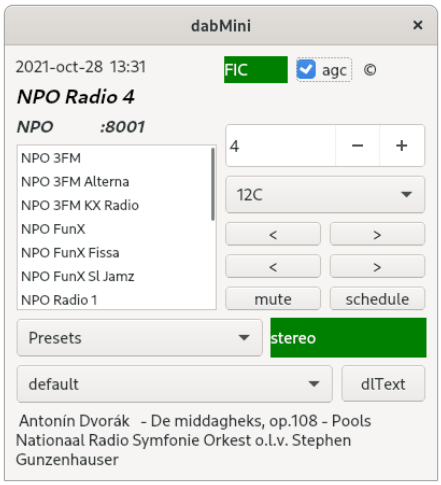



Figure 30: Qt-DAB: dabMini

As picture 30 shows, the GUI is minimal. The *device control* is at the top right. Depending on the selected device, one or two spinboxes (usually some LNA setting and some other gain (reduction) 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 use 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.

Below these selectors one find the *mute* and the schedule buttons. Other than with Qt-DAB touching the *mute* button the duration of muting cannor be controlled from the GUI, its duration is set in the ini file. The *schedule* button - when touched - asks for input to the scheduler - similar as for Qt-DAB. The *audiodump* and *framedump* commands are omitted.

The *Presets* behave as the one in Qt-DAB. The selector labeled *default* is the selector for the audio channel. Note that - Linux obly - both *pulse* and *pipewire* can be configured such that audio is sent

over bluetooth.

The *dlText* button controls - as with Qt-DAB - writing the text of the dynamic label to a file.

The bottom of the GUI contains the so-called *dynamic Label*, a large comboboxes labeled *Presets*, a stereo indicator and a button labeled *mute*.

## 7.3 dabMini on Windows

While it is certainly possible to download the sources and build an executable for windows, the *releases* section of the Qt-DAB repository (https://github.com/JvanKatwijk/qt-dab/releases) contains an installer for dabMini though.

## 7.4 dabMini on x64 Linux

An appImage for dabMini, configured with the whole range of devices, is available on the Qt-DAB repository.

## 7.5 Building an executable on Linux and RPI

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

### 7.5.1 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
cd ..
```

Assuming support for Pluto is wanted, then install

```
sudo apt-get install libiio-dev
```

(see section 5.3.6 for some comments on making the device visible).

### 7.5.2 Download the sourcetree for Qt-DAB

Dowload the sourcetree for Qt-DAB from the repository

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

### 7.5.3 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 (with the selected device(s)) (the "make" will take app 10 minutes on an RPI 3) to build and install an executable. As an example, constructing and installing an executable of *dabMini-2.0*, configured for Dabsticks, Pluto and the 2.13 support library for the SDRplay RSP, we need

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

# 8 Acknowledgements

Qt-DAB and derived programs are written and maintained by me. The software is provided *as is*, and made available under the Gnu GPL V2 license.

Many people contributed (and contribute) by providing feedback, suggestions and code fragments, in particular:

- Andreas Mikula, for continuous feedback, testing and suggestions;
- Stefan Pöschel, for providing code for and giving suggestions to handling the AAC code;
- Stuart Langland, for its comments, suggestions and code contributions;
- probonopd, for its contribution with creating appImages;
- Przemyslaw Wegrzyn, for contributing code for handling charsets;
- Paul Howard-Beard, for his enthousiastic experiments with new features, comments and his suggestion to add features like a mute button, the per channel gain settings, and alarm;
- Michael Lass, for showing me the use of the Gcc address sanitizer, pointing out some (actually too many) address violations discovered by the sanitizer and giving suggestions and advice for the repair; and
- Herman Wijnants, for his continuous enthousiastic feedback on and suggestions for the Windows version of Qt-DAB.

Furthermore I am grateful

- to SDRplay ltd (Andy Carpenter), for providing me the possibility to use the Ia and II versions of the SDRplay RSP devices, all wonderful devices;
- to Benjamin Vernoux, for making an AIRSPY device available;
- to Great Scott Gadgets, for making an HACKRF device available;

- to Jan Willem Michels, for making a LimeSDR device available, and
- to Olaf Czogalla, for donating an RT2832 based stick after having lively discussions on TPEG.
- to Robin Getz (Analog Devices), for making an Adalm Pluto available, a device with lots of possibilities, still to discover.

Qt-DAB is developed as hobby program in spare time. Being retired I do have (some) spare time and programming Qt-DAB (and my other programs) is just hobby. Contributions are always welcome, especially contributions in the form of feedback and additions and corrections to the code, but obviously also in the form of equipment that can be used.

If you consider a financial contribution, my suggestion is to support the red cross or your local radio amateur club instead.