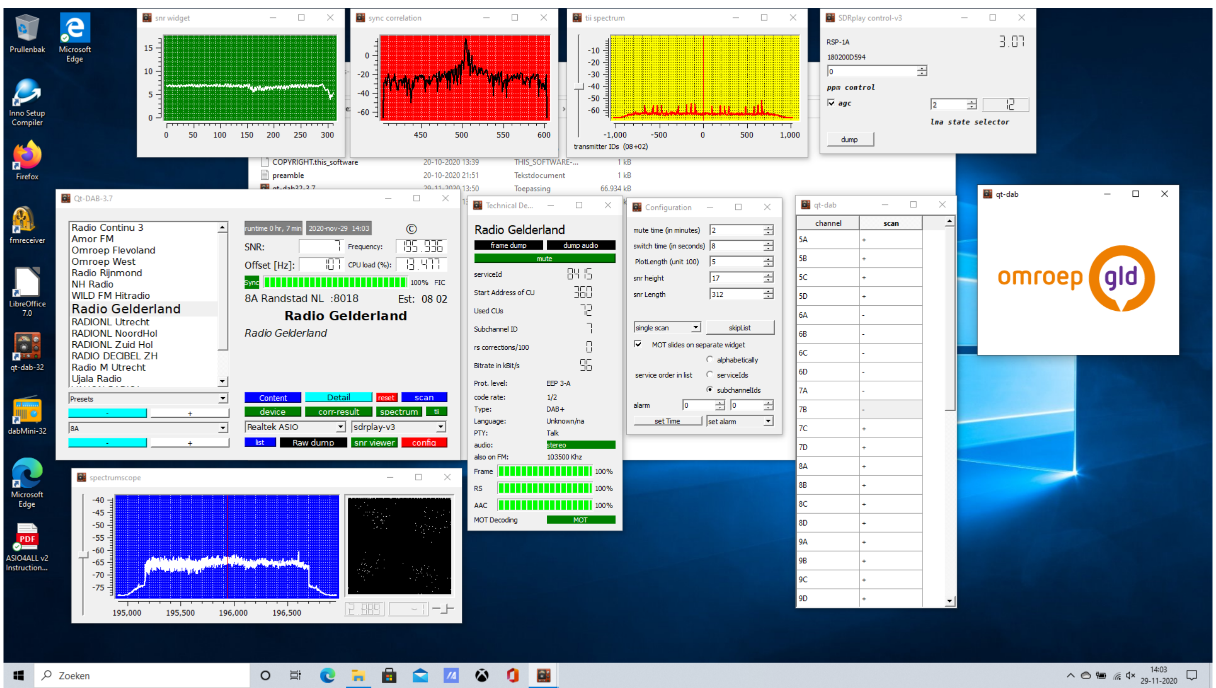
# Qt-DAB 3.7*

*User's guide for version 3.7*

Jan van Katwijk, Lazy Chair Computing
The Netherlands

November 30, 2020



# Contents

# 1 Introduction

Qt-DAB is an advanced program for decoding terrestrial 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[1]

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, see section 5. That section contains a pretty detailed description on how to build an executable on Linux system, with scripts for Debian and Ubuntu.

Qt-DAB is implemented in C++, with extensive use of Qt for its graphical appearance. Furthermore, it uses a number of existing open source libraries, such as fftw, libsndfile, libsamplerate, libusb, and Qt-DAB is itself free software, available under an open source license.

The sourcetree for Qt-DAB contains - obviously - sources to generate an executable for Qt-DAB. It actually contains subdirectories for *three* decoder versions (next to a number of shared subdirectories), *dab-maxi*, *dab-mini* and *dab-2*.

- *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 when uploaded to git (through Travis).

- *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.

- *dab-2* contains sources for an experimental version, a version with - roughly - the same functionality as Qt-DAB, however with a completely different front end architecture. It is experimental, meaning that from time to time the dab-2 specific parts are not compatible with the shared sources and there is no official support.

The *dabMini* version is described in section 7, that section includes a description of how to build an executable. The dab-2 program is used by me for experimenting, it is not further described here.

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 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.

[1] *Disclaimer: While Windows is most likely a marvellous operating system, I develop the software under Linux, and cross compile it for Windows. It turns out that in some cases, in some situations, the software - running under Windows - shows erroneous, or at least different, behaviour not found when running under Linux. Developing under Linux is easy: when something goes wrong (it happens), it is fairly easy to detect the culprit and take appropriate actions, for Windows this is (completely) different. So, while I will continue to produce - from time to time - a Windows installer for Qt-DAB and for dabMini, no garantee about their functioning under Windows is given.*

In section 5, a description is given on how to configure and build an executable for Qt-DAB. First the configuration parameters are briefly discussed, a description is given of which libraries have to be installed on a Linux system, and what to do with either cmake or qmake.

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* and *dab-2* are - slightly - different).

As said, in section 7, a brief description is given of the *dabMini* program, a decoder version built on the same set of sources but with a minimal interface.

## 1.1 Related software

Based on the algorithms in Qt-DAB 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 is simple, a channel, a servicename and some gain settings are passed as parameter (https://github.com/JvanKatwijk/dab-cmdline);

**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.

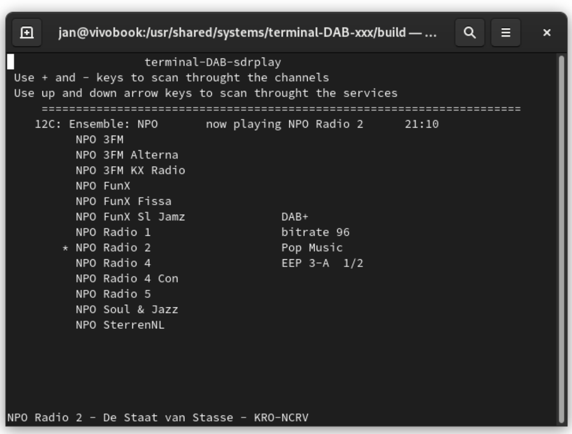



Figure 1: Qt-DAB: the terminal-DAB window

If configured, slides, passed as Program Associated Data (PAD), are shown in a separate widget. (see https://github.com/JvanKatwijk/terminal-DAB-xxx).

**dab2fm-pluto** is a program, written to exercise the transmission possibilities of the Adalm pluto device. The program is a - more or less regilar - DAB decoder, with as backend an FM modulator, feeding the transmitter side of the Pluto. The command line takes a channel, a service name, some gain setting parameters and a *transmission frequency* as parameter.

The decoder will transmit the audio of the selected service as stereo FM signal on the selected frequency. The text of the dynamic label in the transmission of the selected service is added to the FM signal as an RDS signal (https://github.com/JvanKatwijk/dab2fm-pluto).

**dabScanner and channelScanner** are programs, developed for scanning the band. *dabScanner* is a GUI driven program to continuously scan the band and record information on what is received (https://github.com/JvanKatwijk/dab-scanner)[2]. *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, a command line option exists to dump the raw input of those channels containing DAB data onto a file in the xml format (https://github.com/JvanKatwijk/channel-scanner).

# 2 The GUI and GUI elements

## 2.1 Introduction

When playing around with DAB I ususally want to be in full control, and I am (most of the time) interested in the characteristics of the signal. The GUI of Qt-DAB reflects this, there is an abundant amount of buttons, selectors and displays.

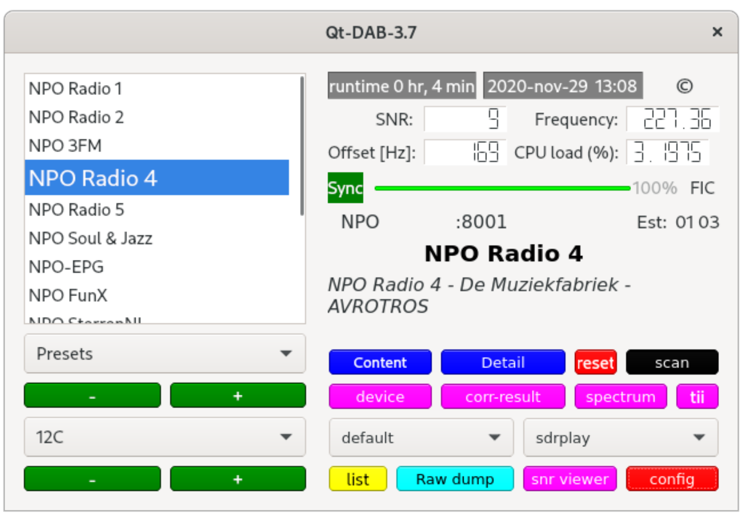



Figure 2: Qt-DAB: the main widget of the GUI

To keep things manageable, the GUI is built up as a *central* widget, a widget that is shown *permanently*[3], accompanied by a number of other widgets that might - or might not - be made visible, depending on user's settings.

[2]Note that most of the scanning can also be done using Qt-DAB

[3]closing this window will terminate the program execution

While the figure on the first page shows the GUI with all widgets, figure 2 shows the central widget, the one with (most of) the controls.

This main widget 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.

Note that the control for the selected input device is on a separate, device specific control widget.

## 2.2 Control for channel and service

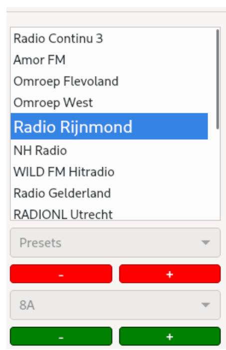



Figure 3: Qt-DAB, channel and service selection

Central in the left part of the GUI is the list of services, this list shows the services detected in the currently selected channel[4]. *Selecting* a service is by moving the cursor to the name of a service, and clicking with the *left* mouse button.

Below the list of services (see figure 3) 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[5]. 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 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.

[4]Note that the order of this list - either alfabetically, by service Id, or by the number of the subchannel - can be set in the ".ini" file and changed in the configuration window

[5]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

- 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 "remember" the selected channel and service, these values will be saved, and on program start up, these values will be taken as start value.

Note furthermore that the software will "remember" the gain settings for each channel used. On selecting a channel a next time - either explicitly or implicitly through selecting a preset service - the gain setting as it was is restored[6].

## 2.3 Displaying information

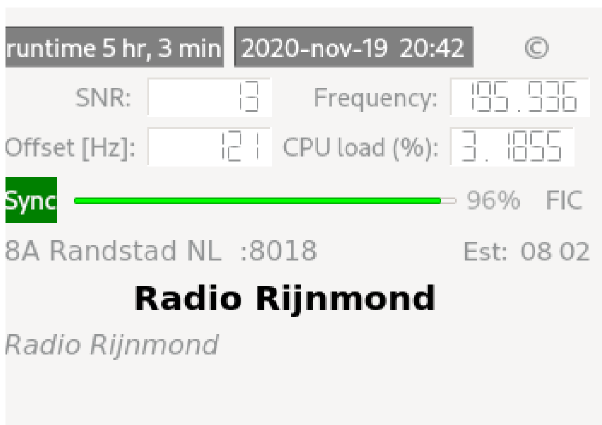



Figure 4: Qt-DAB, system wide information

Some general information is displayed in the top half of the right side of the GUI, see figure 4. The top line gives three (four) 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 current 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.

Below this line, there are boxes with labels:

[6] A setting in the ini file exists to ignore previous settings

- SNR, the measured signal/noise ratio. SNR is computed as the strength of the signal compared to the strength during transmission of the NULL period of the DAB frames;
- Frequency, the frequency, in MHz, of the selected channel;
- Offset, the frequency correction 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. If the value is lower than app 80 percent the progressbar will be shown in red.
- if an *alarm* is set, a small red field with the text "Alarm" is shown.

The remaining part of the widget 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 estimate of the transmitter identification being received. As is well known, DAB is transmitted using a *Single Frequency Network*, a network of transmitters, all transmitting the same DAB content on the same frequency, so one might (probably will) receive data from more than one of the transmitters at the same time. It is up to the software to select the strongest signal. Each transmitter in the network encodes a unique identification in the transmitted signal, the *Transmitter Identification Information* (TII), consisting of two numbers, one to identify the network, one for the specific transmitter in that network. During reception of a channel, the indication that is shown may change. Always, the TII of the strongest signal received is shown.

## 2.4 Control elements

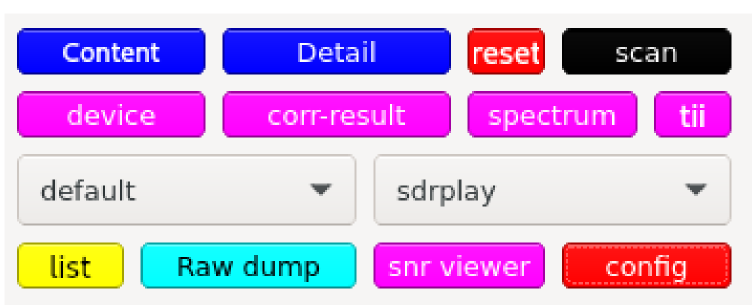



Figure 5: Qt-DAB: control elements

Most controls are grouped in the lower right half of the GUI, displayed in figure 5. The control contains 12 push buttons and 2 comboboxes, briefly discussed, in the order from left to right, top to bottom (three additional buttons related to the audio output are now located on the widget describing the service details).

**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 be shown

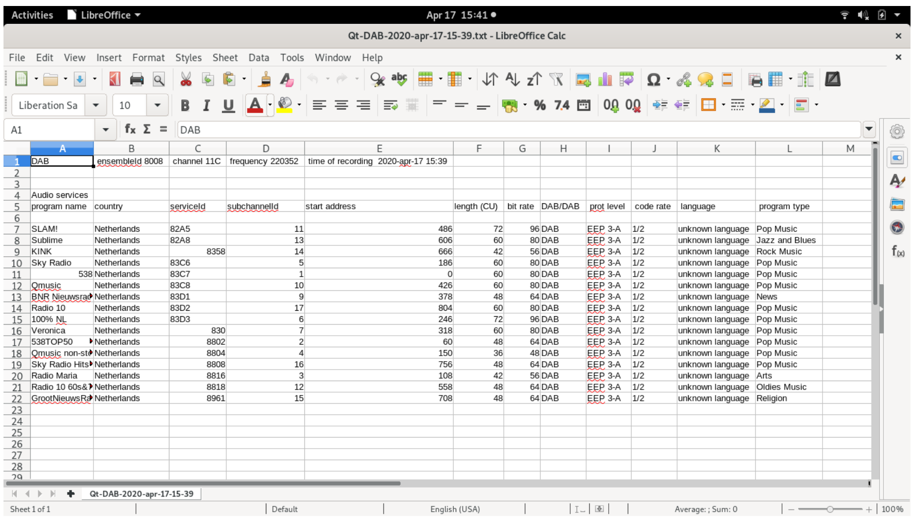

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| DAB | ensembleId 8008 | channel 11C | frequency 220352 | time of recording 2020-apr-17 15:39 | | | | | | | |
| Audio services | | | | | | | | | | | |
| program name | country | serviceId | subchannelId | start address | length (CU) | bit rate | DAB/DAB | prot level | code rate | language | program type |
| SLAM! | Netherlands | 82A5 | 11 | 486 | 72 | 96 | DAB | EEP 3-A | 1/2 | unknown language | Pop Music |
| Sublime | Netherlands | 82A8 | 13 | 606 | 60 | 80 | DAB | EEP 3-A | 1/2 | unknown language | Jazz and Blues |
| KINK | Netherlands | 8358 | 14 | 666 | 42 | 56 | DAB | EEP 3-A | 1/2 | unknown language | Rock Music |
| Sky Radio | Netherlands | 83C6 | 5 | 186 | 60 | 80 | DAB | EEP 3-A | 1/2 | unknown language | Pop Music |
| 538 | Netherlands | 83C7 | 1 | 0 | 60 | 80 | DAB | EEP 3-A | 1/2 | unknown language | Pop Music |
| Qmusic | Netherlands | 83C8 | 10 | 426 | 60 | 80 | DAB | EEP 3-A | 1/2 | unknown language | Pop Music |
| BNR Nieuwsrad | Netherlands | 83D1 | 9 | 378 | 48 | 64 | DAB | EEP 3-A | 1/2 | unknown language | News |
| Radio 10 | Netherlands | 83D2 | 17 | 804 | 60 | 80 | DAB | EEP 3-A | 1/2 | unknown language | Pop Music |
| 100% NL | Netherlands | 83D3 | 6 | 246 | 72 | 96 | DAB | EEP 3-A | 1/2 | unknown language | Pop Music |
| Veronica | Netherlands | 830 | 7 | 318 | 60 | 80 | DAB | EEP 3-A | 1/2 | unknown language | Pop Music |
| 538TOP50 | Netherlands | 8802 | 2 | 60 | 48 | 64 | DAB | EEP 3-A | 1/2 | unknown language | Pop Music |
| Qmusic non-st | Netherlands | 8804 | 4 | 150 | 36 | 48 | DAB | EEP 3-A | 1/2 | unknown language | Pop Music |
| Sky Radio Hits | Netherlands | 8808 | 16 | 756 | 48 | 64 | DAB | EEP 3-A | 1/2 | unknown language | Pop Music |
| Radio Maria | Netherlands | 8816 | 3 | 108 | 42 | 56 | DAB | EEP 3-A | 1/2 | unknown language | Arts |
| Radio 10 60s& | Netherlands | 8818 | 12 | 558 | 48 | 64 | DAB | EEP 3-A | 1/2 | unknown language | Oldies Music |
| GrootNieuwsRa | Netherlands | 8961 | 15 | 708 | 48 | 64 | DAB | EEP 3-A | 1/2 | unknown language | Religion |

Figure 6: Qt-DAB: content

with which the filename can be selected. The file is written in ASCII and is readable by e.g. LibreOffice Calc or similar programs (see figure 6).

**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.

The widgets - figure 7 - show all kinds of technical details of the selecgted service. The widget to the right has an additionaal button "timeTable". which is visible whenever the software was able to identify EPG data in the current channel[7].

The technical Data widget 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.

The widget contains three buttons:

- a *frameDump* button. Touching the button will show a menu to select a filename. The AAC frames, with the encoded audio of the selected service will be written into the file. The format is such that a program as e.g. VLC can process the data further. Writing continues until either the button is touched again or another service of channel is selected.

[7]It may take some time before the EPG data in the current channel is detected and decoded

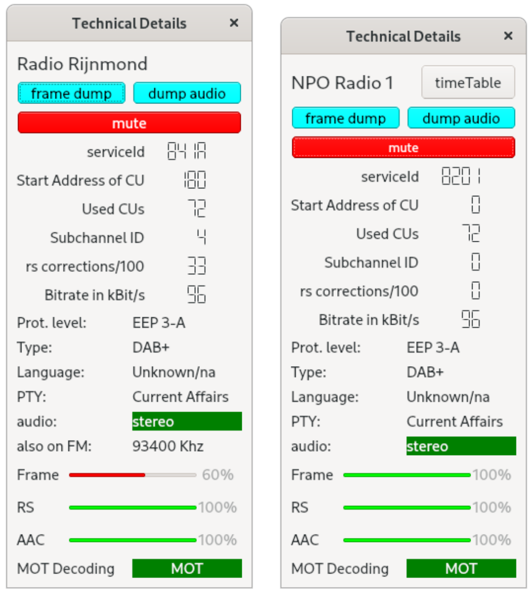



Figure 7: Service details

- an *audioDump* button. Touching the button will show a menu to select a filename. The PCM output of the selected audio service will be written into the file. Writing continues until either the button is touched again or another service of channel is selected.
- a *mute button*. Based on user suggestion, a *mute* button was added with an obvious function. Touching the mute button 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.

For both the frame dump as the audio dump button, a suggestion will be given for the filename to use, containing the service name and time of recording.

Just for my curiosity, there is a number that tells how many corrections on the incoming DAB+ frames were needed (and could be performed) by the Reed-Solomon error recovery per 100 frames. Note that the maximum amount of errors that can be corrected per DAB frame is 5, if there are more than 5 errors, that frame is errnoneous and not processed further.

A "stereo" indicator is back, now where it belongs, in the widget with the description of the service.

If the transmission of the service is also on FM, an FM frequency will be shown, as depicted on the left widget.

For DAB+ services three progress bars are shown, in case all three show a value of 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).

Below these progress indicators, a line will indicate whether or not the service carries a MOT label. If it is, the picture will be displayed. The picture will be displayed by default on a separate widget, however, a setting in the configuration widget, the setting being saved in the ".ini" file, will cause the picture to be shown on a label at the bottom the technical data widget.

**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 services, extracted from the incoming samples, is built up - if any.

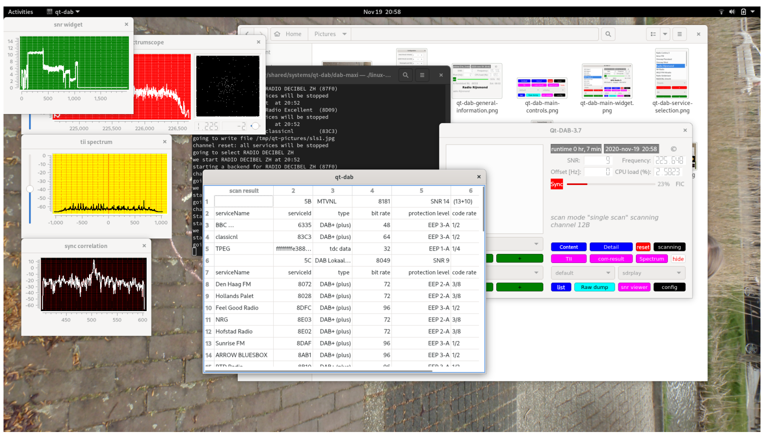


Figure 8: Fragment of the scan output

**Scan button** Touching the button labeled *Scan* will instruct the software to perform a scan. The configuration widget contains a combobox that is used to set the way the scanning occurs:

- a *single scan* will just perform - as the name suggests - a single scan over some channels in the current band. Starting at the first channel, stopping at the last one.
- a *scan to data* will start scanning with the next to the current channel and continue scanning until a channel is reached that contains DAB data.
- a *continuous scan* is like the *single scan*, but will not stop at reaching the last channel in the band but will scan the band over and over again.

Since for quite some channels it is known on beforehand that they do not contain data, a *skipTable* was implemented, a table which can be used to record channels that are to be skipped when scanning. The entry to the skipTable is also to be found on the configuration widget.

In case a full band scan is done, the results are shown, see figure 8 (i.e. the names of the ensembles found, names of services and some technical data on the services).

On touching the button, a widget is shown with the question whether or not to save the result. If saving is selected, a menu will be shown with which a filename can be selected (a suggestion for a filename, containing the date and time is given). The result will then be saved in a text file, that can be processed by e.g. LibreOffice Calc. The format of the saved data is the same as the format of the data saved when touching the content button, and the text shown is a subset of that.

**device button** Touching the button labeled *device* will hide (or show) the widget for the device control.

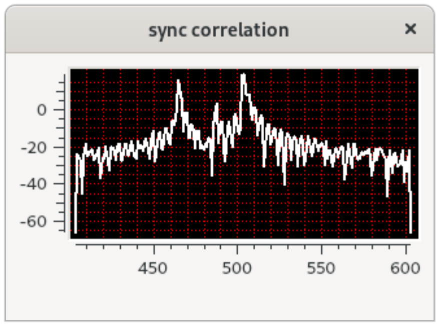



Figure 9: Correlation result

**corr-result button** Touching the button labeled *corr-result* will instruct the software to display 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 strongest signal (i.e. the highest correlation value) is the one used for demodulation and decoding.

The X-axis indicates the sample numbers, "normal" synchronization happens when the peak is on sample 504. The width of the X-axis, i.e. the amount of samples taken into account, is taken from the ".ini" file, and can be set from within the configuration widget.

The widget as depicted shows that there are (at least) three transmitters in the neighborhood. The second strongest one arrives app 40 samples before the strongest one. Since there are 2048000 samples/second, one sees that the second strongest arrives app 20 microseconds before the other.

Touching the button again will cause the widget to disappear.

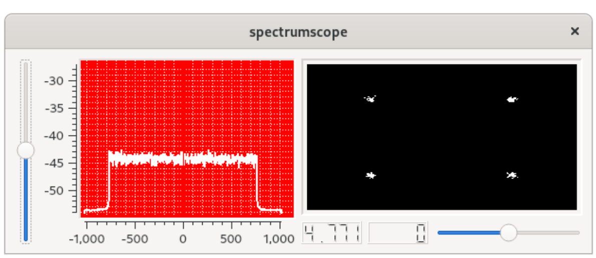



Figure 10: Almost ideal signal

**Spectrum button** Touching the button labeled *Spectrum* will instruct the software to display a separate widget, showing 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. The

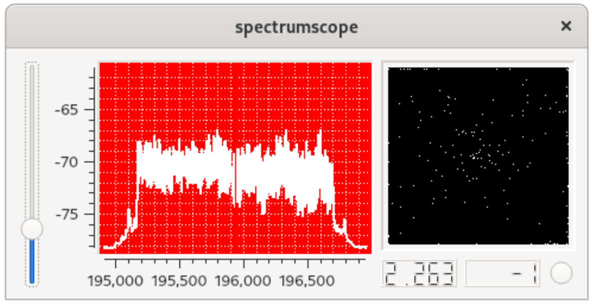



Figure 11: Spectrum of a real signal

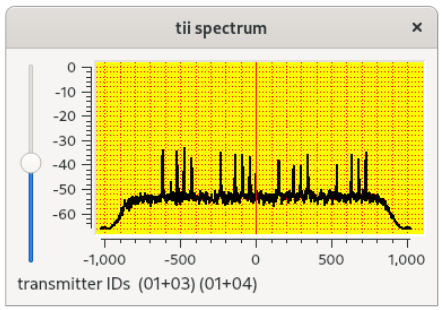



Figure 12: TII spectrum

picture, figure 10 shows the spectrum of a synthetic signal, a signal without channel disturbances. The picture in figure 11 shows a realistic, reasonable though not excellent signal. As can be seen, ideally the constellation shows as four dots, one in each quadrant, the picture 11 shows a more cloud like structure. Of course, the more the constellation looks like a single cloud, the poorer the signal.

Below the constellation window, two numbers are shown. The *quality indicator* shows - according to some metrics - the quality of the constellation in a range from 0 .. 5. The *clock error* tells the amount of samples too many or too few in processing 10 DAB frames (a DAB frame is built up from 196608 samples with a rate of 2048000, so 10 DAB frames take slightly less than one second).

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

**TII button** Touching the button labeled *TII* will instruct the software to display a widget (figure 12) with the spectrum of the null period from the start of the DAB frames. The TII data (Transmitter Identification Information) is extracted from the spectrum of these null periods. The line at the bottom of the widget displays the (mainId, subId) combination(s) found as giving the strongest signal. On touching the button again the widget will disappear.

**The combobox labeled *default*** The combobox labeled *default* in the picture is for selecting an audio channel. What the combobox shows depends on the (sound card of the) computer where the program is running. In most cases *default* will do.

**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.

**list button** The button labeled *list* 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 *list* button again will hide the list (touching the list with the right mouse button will clear it).

**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. The menu will suggest a filename of the form "*device name-channel-date*.sdr" (date as derived from the DAB stream). Touching the button again will stop dumping and the file will be closed. The resulting 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 2048000 I/Q samples (each I/Q sample represented as two short ints, i.e. 2 * 2 bytes) are written.

**snrView button** Especially when looking at the performance of different antennas, the development of the SNR over time might be interesting (see figure 13). The snrView button makes a widget (in)visible that displays the SNR over time. The SNR here is computed by looking at the (amplitude of the) signal over the data blocks in a DAB frame and the (amplitude of the) signal in the null period of the DAB frame. A configuration option exists - the button as shown here - with which the snr values can be stored in a file. A separate utility exists for making the recording visible. (Per 2 DAB frames one computation is performed, about 312 per minute) The length of the period (as well as the height of the display) can be chosen in

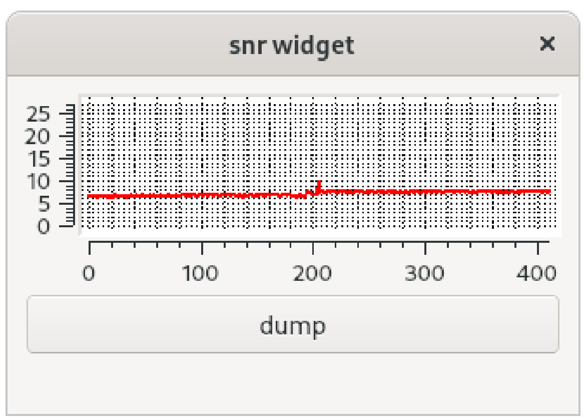
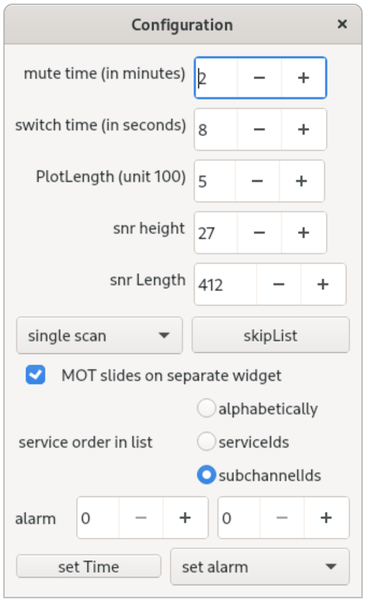



Figure 13: Development of SNR

the configuration widget. Working in the "lazy chair", I am only using a simple whip antenna, the snr shown is not that spectacular, but in directing my other antenna it was quite useful.

**config Button** Settings can be found in the ".ini" file. Many of the settings are automatically stored (e.g. the selected device, the selected channel, etc etc), some are only read by the Qt-DAB program, but can be edited by a user. Of course, editing a configuration file is not always fun, and it does not always make sense to edit the ".ini" file while the program is running. That is why I added a "config" button. The button controls the visibility of a widget with a few controls for settings in the ".ini" file. The Qt-DAB program has been adapted such that modifications to the settings are applied (almost) instantaneously.




The configuration widget supports:

- setting the *mute* time, the time - in minutes - used to mute the audio is the mute button is set to *mute*;
- setting the *switchDelay* time, the time used to detect DAB data in a channel when selected;
- setting the *plotLength* in units of 100 samples, i.e. the number of samples shown in the correlation widget;
- settings for the height and length of the snrView widget;
- selecting the *scan mode*. While the default way to scan is to make a single scan over all channels in a given band, an alternative is to scan, starting with the successor of the currently selected channel, until a channel is detected that contains DAB data. A third possibility is to have a *continuous scan*. The mode can be selected with the combobox.
- modifying the *skipTable*. When scanning a band, channels, marked with a "-" in the skipTable are ignored. Modifying the setting for a channel is by double clicking on the channel setting in the skipTable. Settings in the skipTable are maintained in the ".ini" file.
- setting the location for the mot slides, by default at the bottom of the technical data, alternatively in a separate widget.

- while not obvious to everyone, there are different ways to order the list of services. The widget contains a selector for the ordering.

One addition to version 3.6. was an ”alarm”. The reason is simple: When listening to a (music) service, I usually forget to switch to my favorite news service in time, so I want an automatic change to a selected service at a given time.

Operation is simple: touching the ”set time” button will display the current time (hours and minutes). The alarm can be set by touching the ”set alarm” field in the combobox, which will make a list of sservices visible from which one service can be selected.

Then, the alarm starts, a small label will be visible on the main widget, and at the selected time the selected services will be started. Of course, in case the service is in a channel different from the one active at that time, a channel switch will be performed.

## 2.5 Colors and coloring

Starting with version 3.5 the GUI of Qt-DAB supports a form of coloring of the buttons and the various displays. Since it is most likely that others prefer different colors than I do, a *fixed* color scheme was not appropriate. The first approach was to ask the user to edit the ”.ini” file for changing the setting of a color, but this did not seem user friendly. A more 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.

### 2.5.1 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
```

### 2.5.2 Setting the colors of a button

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.

### 2.5.3 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


Figure 14: Qt-DAB: EPG timetable data

- 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*.

### 2.6 EPG Handling

An experimental version of an EPG Handler (Electronica Program Guide) is implemented in the software. The EPG Handler can be made part of the configuration. When included, the software will look for an EPG service in the currently selected channel and run a decoder for it in the background. Whenever time table data can be identified and decoded, it will be attached to the description of the service involved.

Selecting a service for which time table data is available will cause the software to show an additional button on the technical data widget, with which time table data can be made visible.

*Note, however, that the software is experimental and - at least here in the Netherlands, the times mentioned are one or two hours off.*

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

## 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. Note that the color settings are discussed in section 8, 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 obsolote 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 obsolete L Band by setting "dabBand=L_Band". Note that setting a value here overrides the band setting by using command line parameters;
- 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";

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 SDRplay, the AIRspy, the hackrf, the limeSDR and RT2832 based sticks. 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 3.06/7 library versions), the Adalm Pluto, 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 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 3.06 and 3.07 SDRplay interface library.

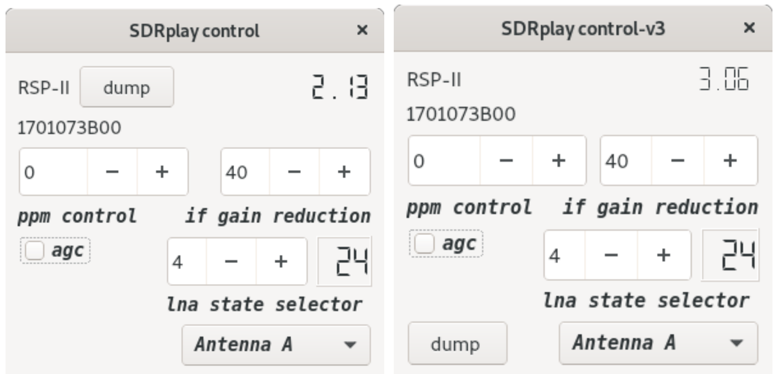



Figure 15: The two control widgets for the SDRplay

As figure 15 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.

## 4.2 The AIRSpy

The control widget for the AIRspy (figure 16, 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 16 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).

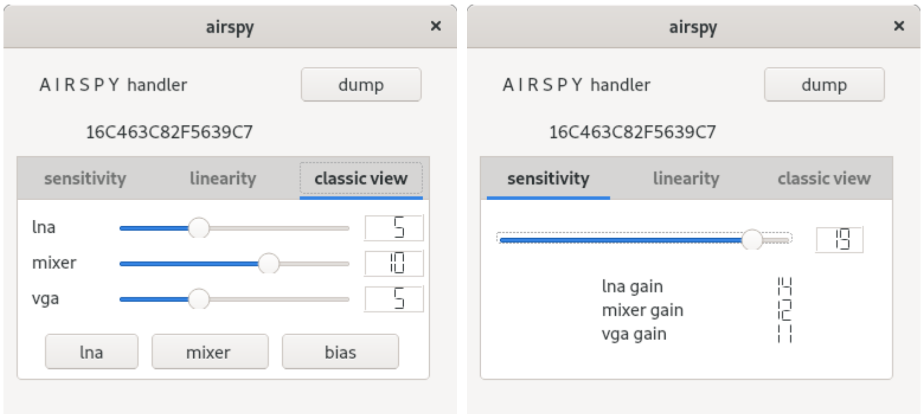



Figure 16: Widgets for AIRspy control

## 4.3 The hackrf

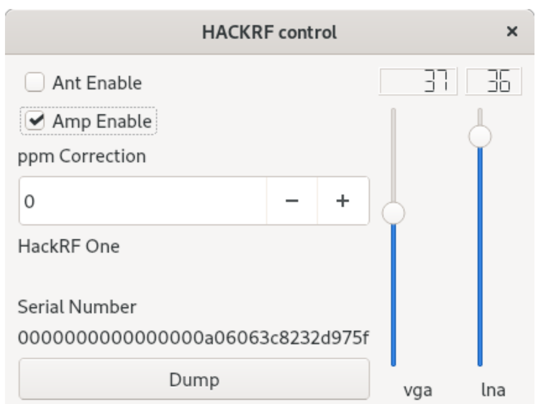



Figure 17: Widget for hackrf control

The control widget for hackrf (figure 17) 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

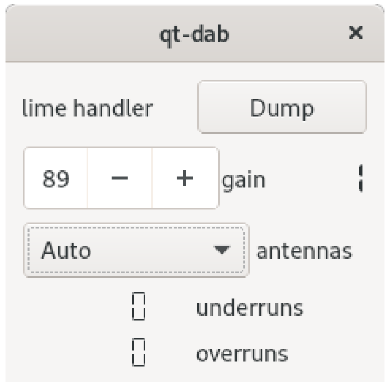



Figure 18: Widget for lime control

On selecting the LimeSDR (if configured), a control widget for the LimeSDR is shown (figure 18). 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 19).

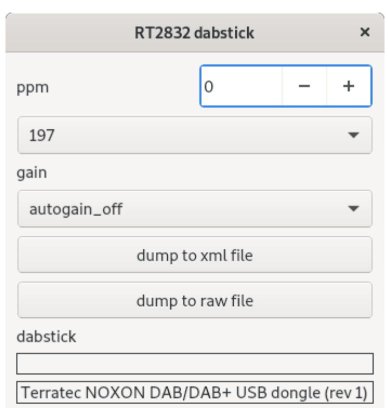



Figure 19: Widget for rtlsdr device

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.

## 4.6 The Pluto device

When selecting *pluto*, a widget (figure 20) 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 furthermore three buttons:

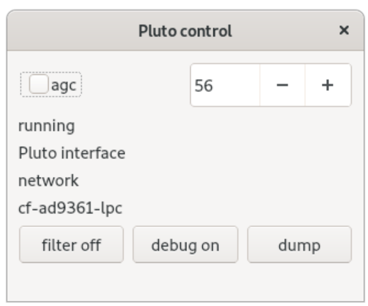



Figure 20: Widget for pluto device

- 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[8] - 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.

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

### 4.7 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 21: Widget for soapy

The widget for soapy control (see figure 21) 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 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";
- 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 22), with as indication the number of seconds the file is being played.

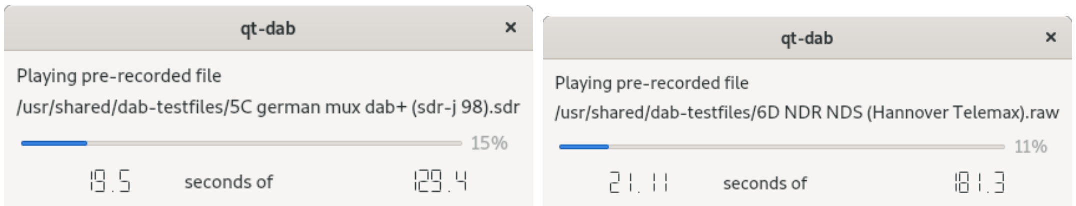



Figure 22: Widgets for file input

Since processing an xml file implies some interpretation, the widget (figure 23) 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.

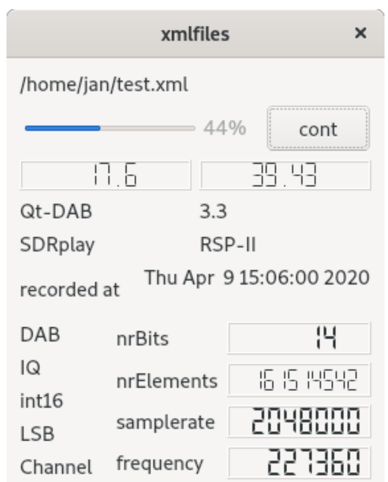



Figure 23: Widget for xml file input

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 *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 by default configured in Qt-DAB executables, changing this is possible, but implies 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 += 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.

**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 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.

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.

### 5.2.5 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

By default the *faad* library is used to decode AAC and generate the resulting PCM samples. *It turns out that both Ubuntu 20 and Fedora 32 install - by default - the libfaad-2.9 which is not compatible with the DAB+ output.*

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.

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).

#### 5.2.9 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.

#### 5.2.10 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 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
```

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*.

### 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.

# 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

```
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 ();
virtual QString deviceName ();
private:
QFrame *myFrame;
};
```

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 should have 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 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). Note that if the widget for *newDevice* is *myFrame* from the parent class *deviceHandler*, the *default* implementation for these function does not to be reimplemented.

## 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     += ./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 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
#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
#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
```

is added.

### 6.3 Linking or loading of device libraries

The approach taken in the implementations of the different device handlers is to *load* the required functions for the device library on instantiation of the class. This allows execution of an executable 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 24: dabMini

As picture 24 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 buttons, there is the *audio channel* selector, set by default on *default*.

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

**Presets** Presets are implemented as in Qt-DAB, i.e. touching a *selected* service in the service list with the right mouse button will add the "channel:name" pair describing the service to the preset list. *Selecting* a preset service is by touching the service in the service list with the *left* mouse button. *Removing* a service from the preset 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.

**Stereo indicator** Based on some user requests, a stereo indicator was re-introduced.

**Mute** Based on a user request, a *mute* button was added, the button - when touched - will mute the audio output for a number of seconds. Touching the button when muting is on, will unmute the sound. Default value is 10 seconds, the value can be changed by setting a value "muteDelay=xxx" in the ini file, xxx indicates the number of seconds.

Touching the mute button when sound is muted will end muting.

## 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; 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.