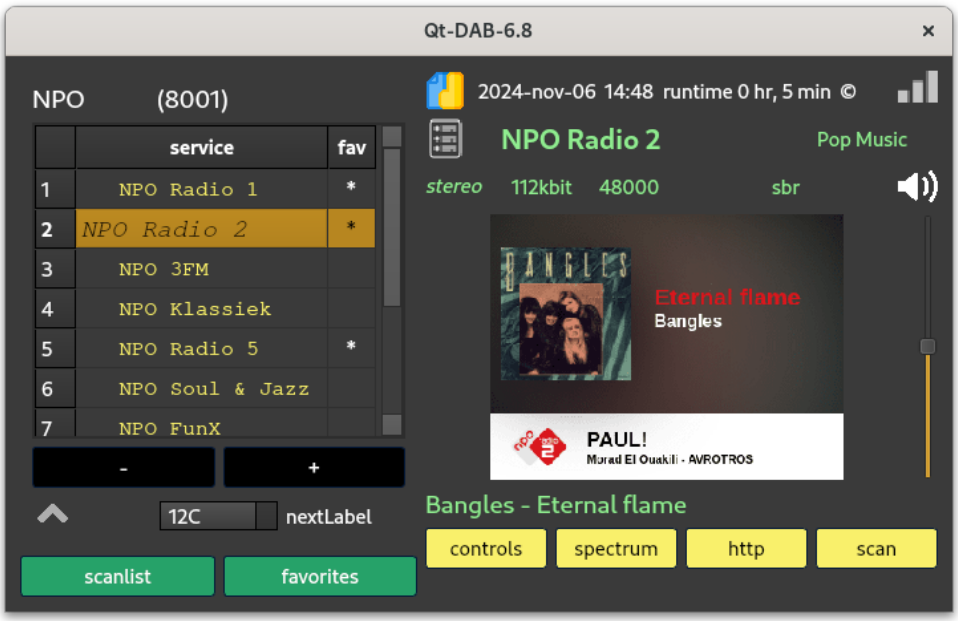
Qt-DAB 6.8*

Jan van Katwijk, Lazy Chair Computing
The Netherlands

November 10, 2024



Contents

1 Qt-DAB-6.8

1.1 Introduction

Qt-DAB is an advanced program for decoding terrestrial DAB and DAB+ transmissions. The program decodes a DAB stream, input is by an *SDR* device or from a file. Qt-DAB is (or can be) configured with a number of popular SDR devices for input, such as SDRplay RSP's, AIRspy, Lime, Hackrf, Adalm Pluto, and obviously, the cheap RTLSDR devices (including the recent V4 version), the dabsticks. Furthermore Qt-DAB can get its input from a file, or from the outside world, using *rtl_tcp*, *soapy* or a *spyserver*.

Qt-DAB is developed on Linux (x64) computers. It will also run on RPI 4 and up running some Linux variant, and is *cross compiled* for Windows (32 and 64 bit).

The focus in the development always is - next to actual decoding the signal of course - visualisation of the signal in its raw and its processed form. This is reflected in the structure of the GUI. That GUI consists of 4 larger (and a few smaller) widgets. While the main widget is always visible, the other three are visible under user control:

- the *main* widget, the main widget - always visible - contains - next to ontrols for selecting channels and services - the controls for the visibility of the other widgets;
- the *configuration and control* widget, contains a large amount of controls for different settings in the program;
- the *spectrum scope widget*, containing a *tabwidget* giving the choice of one of 6 different views on the signal;
- the *technical details widget*, showing the technical details of the audio of the selected service.

The structure of this guide is simple, in section 2 the GUI and GUI widgets are described, In section 3 some functions, related to scanning and showing the map, are discussed. in section 4 *Scheduling* of service selections is discussed, in section 5 the supported devces for the Qt-DAB program are briefly discussed, and in section 6 some notes on building an executable are presented.

The appendix contains a description of how to add a device to the list of supported devices. An overview of Qt-DAB with the 4 main widgets, the DX display and the widget for the device support, is visible is given in figure 1.

1.2 Hardware Requirements

Qt-DAB runs pretty well on modern PC's. One does need a reasonable amount of computing power, though. On an RPI the program can be compiled, on an RPI 4 and up the program will run smoothly. Obviously on an Arduino it will not.

1.3 Precompiled versions

Linux x64 For Linux on the x64 PC, a so-called *AppImage* is available. Such an AppImage is itself a small closed filesystem with all programs and libraries for running Qt-DAB aboard. However, the AppImage does NOT contain support libraries for the various configured devices.

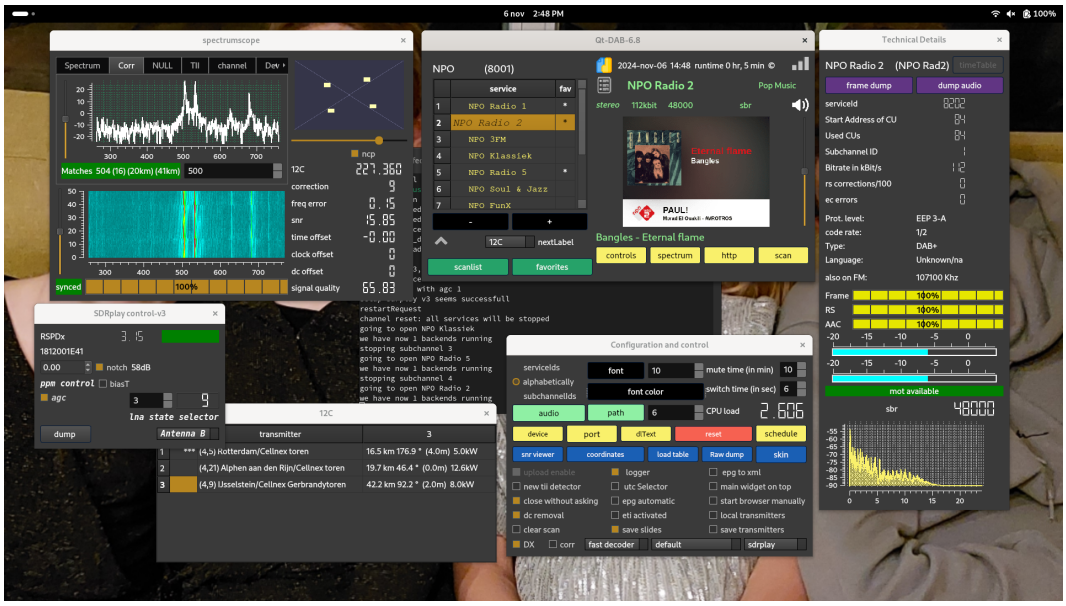


Figure 1: Qt-DAB and the 4 widgets

Windows For Windows no less than 3 installers are available, two for 32 bit versions, and one for a 64 bit version. The *difference* between the 32 versions is the support for RT2832 based (so-called) DAB sticks. The V3 version uses the regular RTLSDR library, the other version the library optimized for V4 versions of the stick (that, according to reportings is rather deaf when using with old version sticks).

Other systems For other systems no precompiled version is available. Of course, since the sources for Qt-DAB are available, one might compile an executable. Building an executable from the sources is not trivial, that is why a separate section is devoted to the build process.

2 The GUI: the widgets and the control

2.1 Introduction

When starting the software for the first time, there is -obviously- no device selected yet. Qt-DAB then shows two widgets, the *main* widget, and the so-called *configuration and control* widget. The latter shows at the right side of the bottom line a selector for configured devices[1]. Once a device is selected and could be opened, the processing starts.

2.2 The widgets

The main widget shows the relevant data for the user's selection of a channel and a service. It displays slide(s) carried in the selected service (or some default slides), and dynamic label text. Furthermore, it provides buttons for controlling the visibility of the *configuration and*

[1]Be aware that Qt-DAB does NOT provide the device libraries.

Figure 2: main widget

control widget, the *spectrum widget* and the *technical details* widget, and for operations like scanning, scheduling etc.

The *left side* of the widget is dedicated to channel and service selection. It shows on top the name of the ensemble that is being received (touching that name with the mouse will display a separate widget with the contents of the ensemble). The major part of the left side is dedicated to *ensemble display*, in this view it shows services from the favorites list and below that list a few controls. A choice can be made between the *ensemble view*, showing the list of services in the current ensemble, and the *favorites view*, showing the services from the favorites list..

Selecting a service in either view is by clicking with the mouse on the service name. The semantics of clicking with the mouse on the field *right* of the service name depend on the view mode: in ensemble view the service will be added to or removed from the favorites, in the favorites view the service will be removed from the favorites list.

With the channel selector, in the picture the small combobox labelled "12C", a channel may be selected, channels are labelled "5A" to "13F". With the "up" and "down" arrow, the selected channel number is incremented resp. decremented.

The *scanlist* button controls the visibility of the scan list, i.e. the list of services detected at the most recent *single scan*.

As mentioned, the button - here labeled *ensemble* is used to switch views from *ensemble view* to *favorites view* and back.

Note that the *configuration and control widget* provides selectors for setting the *font*, the *font size* and the *font color* with which the service names are displayed.

Selectors If an ensemble is detected, its name shows above the servicelist ("NPO (8001)" in the picture above). *Clicking on that name* controls the visibility of the *content widget*, i.e. a widget showing the details of the services belonging to the current ensemble.

Audio services usually carry one or more slides that are displayed. In case a slide is not

(yet) found (see figure 2, a default slide is displayed (Qt-DAB has a number of default slides, a slide change for default slides will happen once a minute).

On the top line in the right half of the widget, left of the time display, one sees a small icon. When touched, a separate widget shows, displaying the directory where picture files, the tii file and the log file are shown.

When a service is selected, its name appears in bold on the right half of the widget. Left of that name ("NPO Radio 5" in the picture above) a small list-style icon controls the visibility of the so-called *technical widget*, a widget showing details of the currently selected service.

The icon to the right of the name, the *speaker*, indicates whether or not the system produces sound. Furthermore, it acts as *mute* button. If Qt_Audio is selected, below this speaker icon, a *Volume control* is visible.

Most services, when running, produce next to audio and one or more slides, text, the so-called DLS (Dynamic Label), "Diggy Dex ft. Inge van Calkar - Het Idee" in the picture above. *Clicking with the right mouse button on this text shows a (very) small menu opening the possibility of saving the text to the OS' clipboard.*

Finally, the 4 (four) buttons below this text control (from left to right)

- the visibility of the *configuration and control* widget, a widget with a huge amount of selectors for a large variety of setting of the system;
- the visibility of the *spectrum* widget, a widget dedicated to showing all kind of aspects of the incoming DAB signal;
- the *http handler*, a handler that controls showing a map with indications of the transmitters being received;
- the visibility of the *scan monitor* widget, a *separate* widget for controlling (single or multiple) scans over all or selected channels in the band.

2.3 Technical details

Most services are audio services, and it is always interesting to see what the parameters of the audio data are. The technical widget, see figure 3. shows these parameters. On top of the widget, the name of the current service is repeated, together with - between brackets - the short name - if any. Below this line, there are two buttons, both for saving audio data of the current service. The button *frame dump* - when touched - instructs the software to dump the AAC frames of the audio service into a file. Such a file can be played by e.g. VLC. The button *dump audio* - when touched - instructs the software to dump the audio output into a ".wav" file, i.e. a PCM file, with a samplerate of 48000 samples/second.

The bottom of the widget shows the spectrum of the audio output, it shows that the audio frequency goes to app. 15 to 16 KHz.

The three progress indicators above the spectrum display (for MP2 output there will be only one) show the successrate of the different steps in the transformation from decoded DAB data to audio. The steps are:

- *Frame recognition and extraction.* DAB+ audio is organized in frames, recognizing frames in the input stream requires some testing and verification;

- *Reed Solomon* decoding. The indicator tells the successrate of the Reed Solomon decoder over the last 25 frames (Note however, that the R-S decoding can fix up to 5 errors per frame);
- *AAC decoding.* The indicator tells the successrate of interpreting the AAC frames and transforming them into audio (PCM) samples.

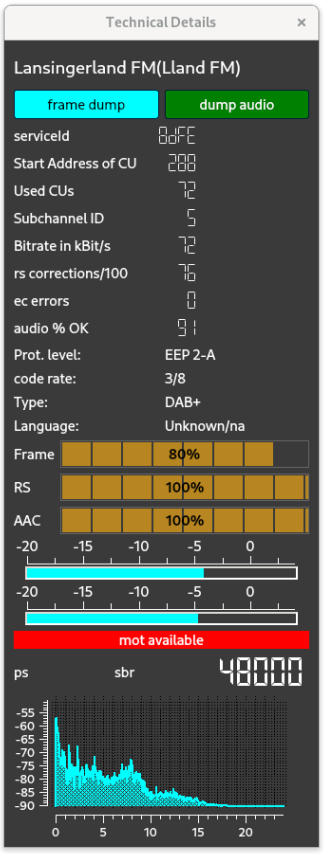


Figure 3: Technical details

Most of the other data that is displayed speaks for itself, a few less obvious numbers:

- *rs corrections* displays the number of corrections, performed by the Reed-Solomon decoder in the last 100 frames. The Reed Solomon decoder operates with frames of 120 bytes, the last 10 of them being parity bytes, and is able to correct up to 5 byte errors in a frame;
- *ec errors* displays the number of errors detected *after* the Reed Solomon decoder has repaired the errors (of course, if the parity bytes in the data contain errors, it is hard to see how an error free audio frame can be constructed);
- *audio % OK* displays the percentage of audio frames that reaches the audio output. If

the AAC decoding fails, no data is sent to the audio output (usually the soundcard). If Qt_Audio is selected, this indicator is not used;

- The *red label* with text "MOT" shows that the currently selected services does not carry MOT dat, i.e, the service described here does not contains slides and Qt-DAB will show some generic default slide. The label colors green if MOT data is detected in the current (audio) service.

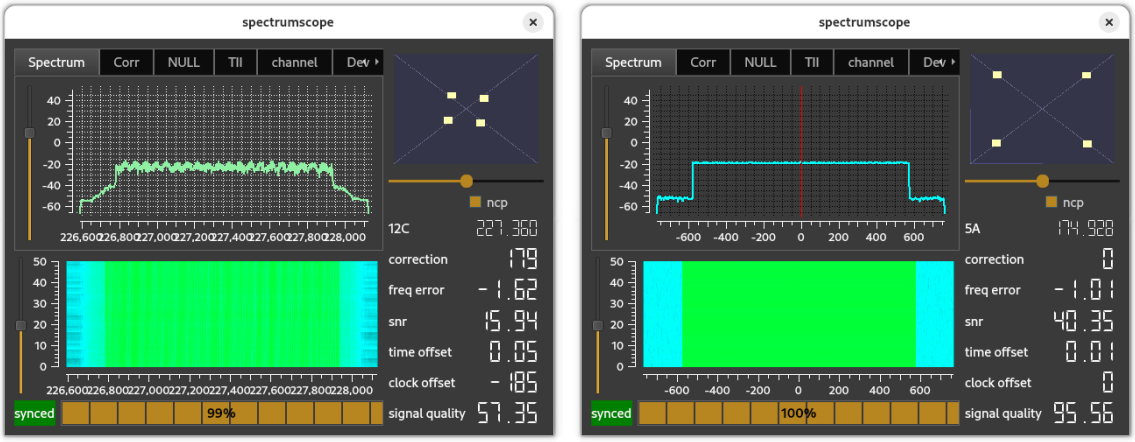


Figure 4: Spectrum scope and ideal signal

2.4 The spectrum widget

In version 6 the various scopes (displays) were put into a single widget. The "tab" defines which scope is shown. The widget further contains an IQ scope, a waterfall display and a list of quality indicators.

Figure 4 shows the spectrum of the signal. which has a width of just over 1.5 MHz. Below the IQscope, a checkbox can be set to select what will be displayed, the "dots" shown in figure 4 show the centerpoints of the constellations of the decoded signal, ideally they are fat dots on 45, 135, 225, 315 degrees. Alternatively, the whole "cloud" of dots of the constallations is shown (figure 6).

The waterfall display is directly coupled to the selected scope, it will show the progress in time of the data of the selected scope.

The first step in decoding DAB signals is *synchronization*, i.e. finding where the actual data of a DAB frame starts in the incoming sample stream. The NULL scope (see figure 5, scope right) shows the samples in the transition phase from the NULL period to the first data block.

To achieve such a synchronization a form of *correlation* is used (figure 5, scope left).

Ideally the maximum in the correlation is on (or about) sample 504 of the current segment of the incoming stream. The correlation scope here shows more than one peak, indicating that more than a single transmitter is received. The software detects here three "reasonable" peaks, it is up to the software to synchronize, given the peaks.

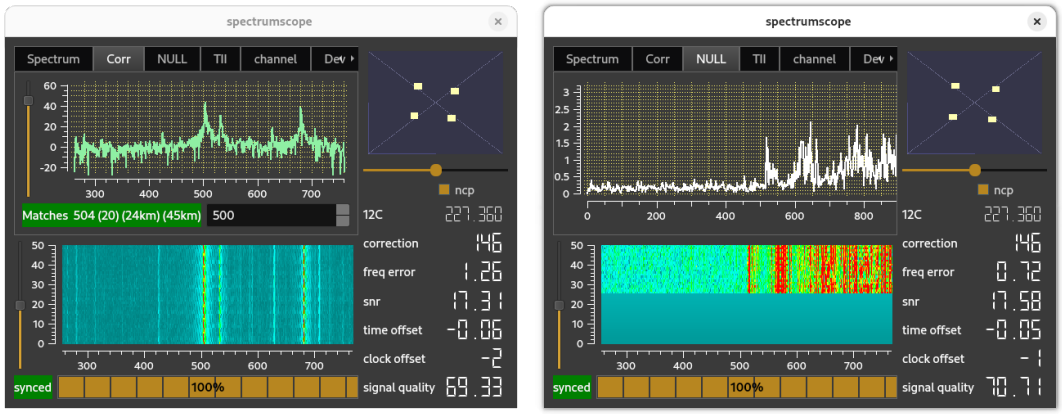


Figure 5: Correlation scope and NULL signal

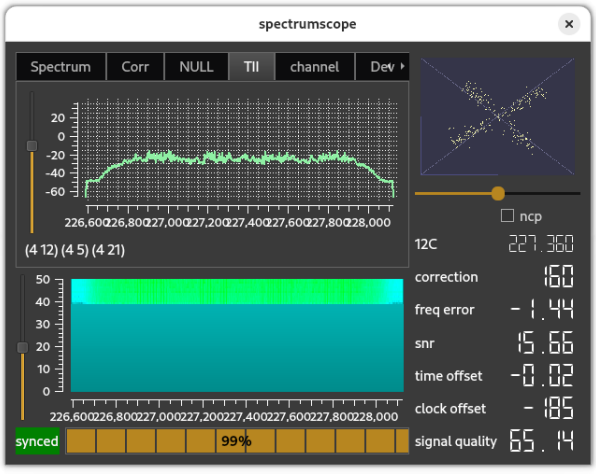


Figure 6: TII scope

If the current transmitter is "known", i.e. we know its name and distance, the distance and the estimated distances of other peaks to the receiver site are shown. In the picture here, the transmitter transmitting the strongest signal has a distance of 16 km, the others, 19 km resp. 40 km to the receiving location

As known, most transmitters send some identification data, encoded in the NULL period of the DAB frames. This data, Transmitter Identification Information (TII) consists of two numbers, a main Id and a sub Id, unique for each transmitter in a given country. The TII scope shows the spectrum of the NULL period, that is where the TII data is to be found. In this case, the software was able to detect that the TII data of the strongest signal is (04 05) and signals from other transmitters in the network were deciphered as (4 21) and (4 12). The pair (4, 5) is transmitted by a nearby transmitter in Rotterdam.

A 5-th scope, a so-called channel scope (see figure 7) shows the channel response.

The first data block of a DAB frame has predefined data, so an estimate of the "channel" is made by looking at the transformation of a selected set of carriers between transmitter and receiver. The green coloured line in the picture shows the ampliyude of the channel,, the red line the phase offset[2].

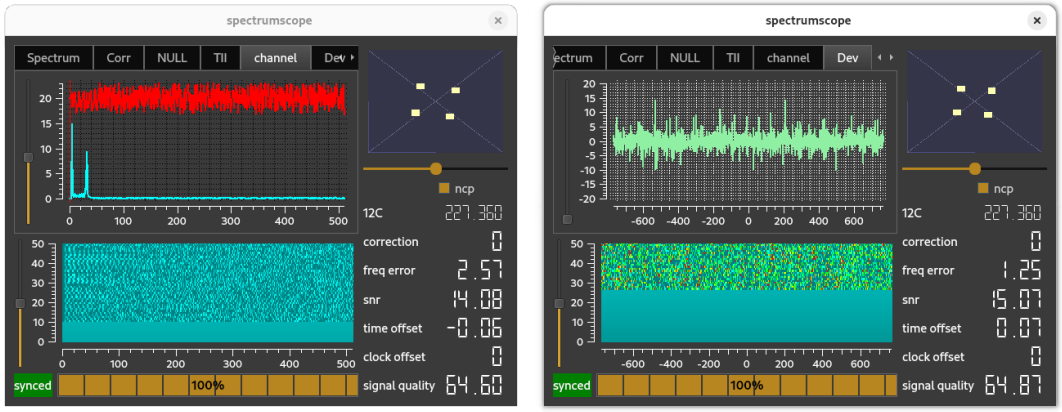


Figure 7: channel scope and deviations scope

The *deviation scope* shows the offsets, expressed in Hz, computed from the phase offsets in the decoded carriers. Ideally the offset in the decoded carrier is zero. Higher offsets means higher chances on errors in the decoding.

The widget shows a number of quality indicators, derived from the incoming signal.

The numbers, an example shown in figure 8, read from top to bottom:

1. The *coarse frequency correction* and the *small frequency correction*, computed from a DAB frame and applied to the data for the next DAB frame. Qt-DAB is able to correct frequency errors to up to app 35 Khz, here the error is pretty small. Using a "dabstick" or so as device will show a much larger correction.

2. the *remaining frequency offset*, which - obviously - should be small, but shows the effect of correcting incoming samples with an offset found earlier;

[2]A DAB block counts 1536 carriers, so the picture only shows a fraction

3. the measured *signal/noise ratio* (measured by looking at the signal strength in the NULL period and the period that data is transmitted);

4. the *time offset*, i.e. the error in the sample clock;

5. the *clock offset*, here we measure how many samples we are short or we have too many per second, here the amount of samples, measured in app 10 seconds, is precise what it should be;

6. the *quality* of the decoded signal, compared to the "ideal" signal that shows 4 dots, one in each quadrant. Higher values indicate a higher signal quality. The values are scaled from 1 to 100, so a value above 60 shows a reasonable signal. There are many ways to compute a quality value, the approach in Qt-DAB is derived from ETSI 101 290, which is not completely fair since the decoding for DAB is primarily depending on the phase of the signal and less on the amplitude.

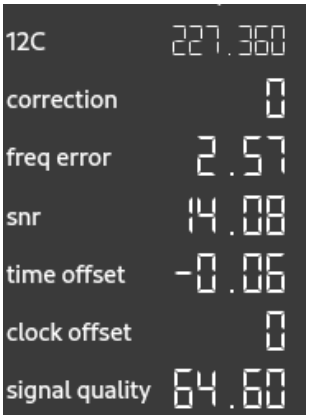


Figure 8: quality indicators

Finally, the bottom line of the spectrum widget shows a *label* and a *progressbar*. The label turns *green* if *time synchronization* can be achieved (i.e. the software "knows" where to find the DAB frames in the incoming sample stream.) The *progressbar* tells the success rate of decoding the FIC (Fast Information Channel) data. If the percentage is less than 100 percent, then apparently something is wrong with the signal and successfull decoding the payload is highly unlikely.

2.5 Configuration and control

The configuration and control widget is the widget with the buttons and checkboxes for influencing the configuration of the decoding process. The settings - together with other settings generated by the system - are stored in the ".ini" file[3].

- At the top left, one may select the order in which the services, when displayed in an ensemble, are presented;

[3]This "'ini" file is a file created by Qt-DAB and stored in the user's homedirectory

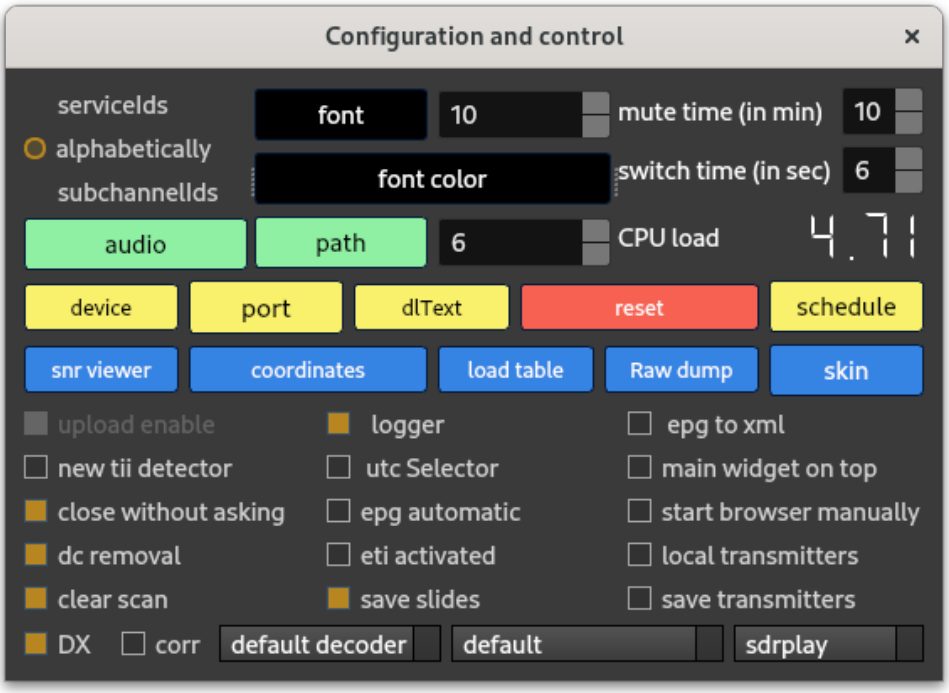


Figure 9: Configuration and control

- Below that selection mechanism, a new selector is presented: the *audio selector*. While traditionally portaudio was used for transporting the audio output to a soundcard, an alternative, using a Qt provided mechanism is now available[4]..
- next to the audio selector, a *path selector* is shown. Files that are generated during processing, such as files containing slides, the tii file, the log file, are by default stored in a subdirectory *Qt-DAB-files* of the user's home directory. Touching this button shows a directory selection menu where one may choose another directory to hold the aforementioned files. Note that the effect of changing the path is only on the next program invocation.
- At the top in the middle, one may choose the *font*, the *font size* and the *font color* of the displayed services;
- Below the selection of the font attributes of the displayed services, a spinbox is set on which the *threshold* value for use in the detection of the tii values is given;
- At the top right. *mute time* tells the system what the duration should be in muting the audio;
- *Switch time* tells the system what the waiting time should be after switching a channel before the software is convinced that the newly selected channel does *NOT* contains DAB data;

[4]In the current AppImage the Qt audio subchannel is switched off

- The *CPU load* tells the *overall* load of the CPU.

Below the top, there are two rows with push buttons

- *device*, a button controlling the visibility of the device widget; A device widget usually shows controls for setting e.g. gain, and once these are set, there is in general no need to have the widget visible;
- *port* allows setting *another port number* than the default one used otherwise for the http handler, i.e. for communicating with the webbrowser when displaying maps and data on maps;
- *dlText* controls an option to save the data from the *dynamic label* into a file. Touching it will show a file selection menu, touching it again will close the file. If a file is selected, the texts in the dynamic label are stored in the file. Note that in this version it is also possible to save the *current text* of the dynamic label. Select the section of the text with the mouse, click with the right hand side mouse button and a small menu appears with an option to save the selected text on the clipboard;
- *reset* will stop all activities and do a restart;
- *schedule button* - when touched - will make a schedule menu visible with which future actions (up to 7 days ahead) can be scheduled;
- *snr viewer* controls the visibility of a small display, showing graphically the progress in time of the SNR;
- *coordinates* allows specifying the local coordinates, used to compute distance and azimuth to transmitters;
- *load table*, if correctly configured, pushing this button causes (re)loading a new instance of the TII database (see section 4);
- *Raw dump* (Obsolete) allows selecting a file to which the input data will be saved in a PCM file. Use xml files instead;
- *skin* allows selection of a skin for the displayed widgets. *The selection will be effective the next program invocation*; The default skin chosen is *Darkeum*.

Below these buttons there is a list of 14 check boxes (in the column left the entry "upload enable" is unused)

- *logger*, if selected, some logging data will be written on a file *logFile.txt*, on the Qt-DAB-files folder/directory.
- *epg to xml* tells the system to map EPG data using an imported library to xml. *Note that it is known that the imported library has a bug that might lead to a crash of the program*;
- *new tii detector*. TII detection may use a second algorithm that detects TII data sometimes earlier, but generates more erroneous TII values;

- *utc Selector*, when enabled shows time as UTC rather than local time;
- *main widget on top*, as it suggests, enabling this ensures that the main widget *always* will be on top. Note that it might cause problems on Windows, since newly generated widgets will be on top and therefore be covered by the main widget. Not ideal if the new, covered, widget asks for confirmation;
- *close without asking* does what it suggests;
- *epg automatic*. Some ensembles carry - next to regular services - an EPG (Electronic Program Guide) service. If enabled the system will execute that service in the background. Note however, that SPI/EPG handling is under construction;
- *start browser manually*. By default, on enabling the http service, the local default browser will be started. If this option is enabled, the user has to start his/hers favorite browser;
- dc removal, when selected the software tries to remove the DC component in the incoming signal'
- *eti activated*. Since Version 5 of the Qt-DAB software. there is an option of generating an *ETI*[5] *file* from the current channel input. If this option is enabled, the *scan* button on the main widget is changed in an "eti" button, with which the eti generator can be started and stopped;
- *local transmitters* if enabled, the transmitters shown on a map will only be the ones belonging to the current channel;
- *clear scan*. On scanning (single scan, see below) the service names envountered will be added to the scanlist that can be made visible on the main widget. If this option is activated, that list will be cleared on a new scan;
- *save slides* does what it suggests, when enabled, slides appearing with the services are stored in the afore mentioned path.
- *save transmitters*. If checked, transmitter names will be saved into a file (see section 4).

bottom line The bottom line of the configuration and control widget contains two checkboxed and three comboboxes. The checkboxes are:

- the *DX* selector, when set, instructs the software to try to detect as much transmitters in the data from the NULL period as possible. The names of the transmitters are displayed on a separate widget, and - when scanning continuously - all detected transmitters are displayed;
- the *corr* selector influences the correlation, if set, the correlation module will report the *first* peak in the correlation (above a certain minimum), rather than the strongest. This solves the problem of decoding when the correlation value oftwo (or more) transmitters is (almost) equal.

[5]ETI stands for Ensemble Transport Interface, defined in ETS 300 799

The comboboxes are

- a selector for the *decoder*, just to experiment a number of (more or less) different approaches are exercised to map the phase of a given signal onto (soft) bits;
- the *audio out selector*, the selector is filled by the underlying sound system;
- the *device selector*, the configured devices are listed here. Note that Qt_DAB is unique in that during a session another device can be selected.

The settings of these selectors is always stored in the ".ini" file and used in initialization.

2.6 Coloring buttons and scopes

Colors for buttons and the scopes are personal. While not essential for functioning, the ability to choose one's on color scheme seems important to me. So, a color for the different buttons on the *main widget* and on the *configuration and control widget* can be set (and subsequently changed). Touch the button with the *right mouse button* and a small widget appears in which first the background color, and second the text color can be selected (see figure 10 for the selector as appearing in the AppImage).

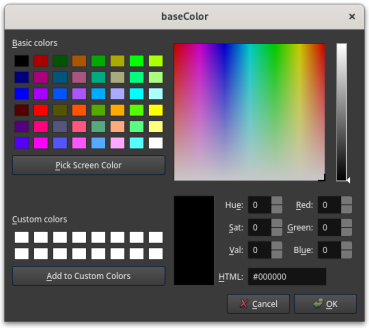


Figure 10: Button color selection

The same applies to the scopes in the spectrum widget, clicking with the *right mouse button* on the field will show a small selection widget with which a *background color*, a *grid color* and a *curve color* can be set. The color settings are immediately effective and maintained between successive program invocations.

3 Contents, scanning and maps

3.1 Introduction

Qt-DAB can display a description of the content of the currently selected channel and offers extensive options for scanning.

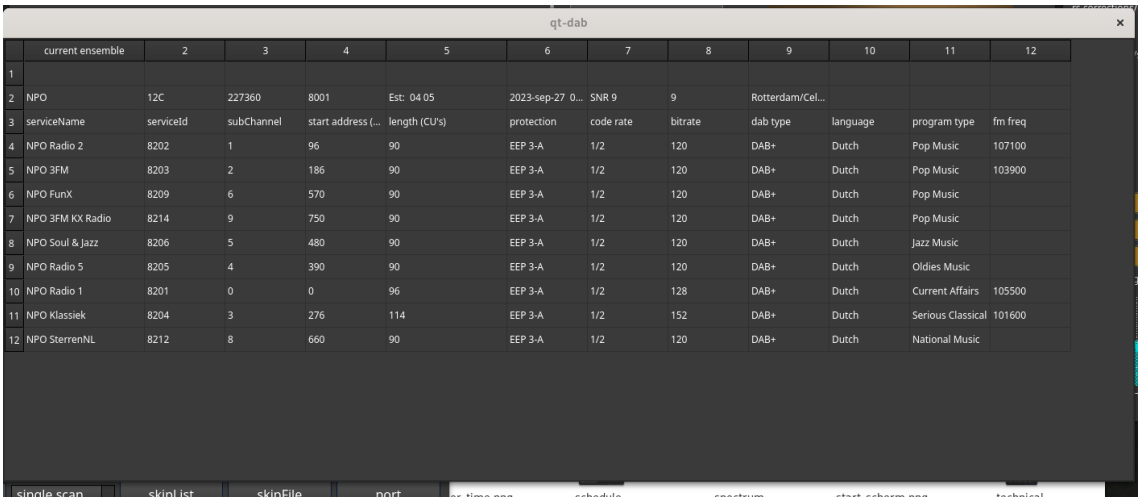


Figure 11: Content description

3.2 Looking at the content of the current channel

Of course, even without scanning, it is possible to get a view on what services are available in the currently selected channel. Next to the obvious list of service names as given on the ensemble display on the main widget, one might want a look at the "content", i.e. details of the ensemble and its constituents.

Picture 11 shows the content of the NPO ensemble, it shows that the TII identifiers at the time of reception were 04 05, the transmitter from which the data is received is located in Rotterdam, that 9 services were detected and the SNR was 9.

For each of the services (NPO does not transmit a data service anymore) the known data is printed. Note that it might take some time after selecting a channel, before all data items are known. Whether or not the service description shows an FM alternative FM frequency might not be known before a few seconds have passed.

Clicking on a service in this list does select that service, double clicking on the widget allows saving the data in a ".csv" file.

3.3 Showing TII data

The *configuration and control* widget has a selector, *DX*, that when set causes the software to look for more than just the strongest signal in the DAB input. Qt-DAB will then try to identify as many as possible transmitters from which the receiver gets signal in, see figure 12. Qt-DAB identifies the transmitter whose signal is currently the strongest (in the picture the transmitter in Rotterdam is - at the time the picture was taken - the strongest).

When the *DX* selector is set, Qt-DAB stores the findings in a text file. For both Windows and Linux, this file is found in the *Qt-DAB-files* folder in the user's home directory, or in the path specified by the user by entering a path after clicking on the *Path* button. The filename is "tii-files.txt". Note that Qt-DAB does create the file *but never will remove it.*

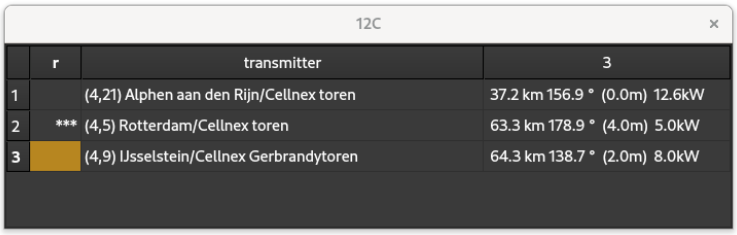


Figure 12: TII data

3.4 Scanning

Touching the *scan* button controls the visibility of the *scan monitor* (figure 13). The *start*

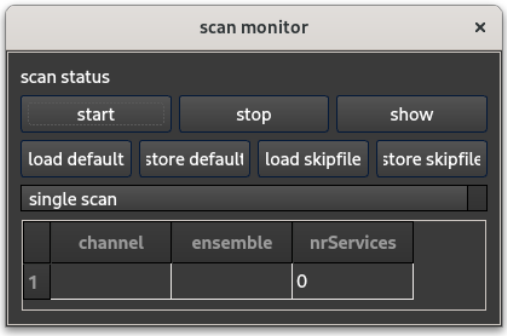


Figure 13: Scan monitor

and *stop* buttons on the monitor widget have their obviosu functions, the button labeled *show* button controls the visibility of the *skiplist*.

In most cases it is known that on some specific channels *no* DAB signal will be found. Of course, it is a waste of time to include these channels in the scan, therefore Qt_DAB has the possibility of indicating which channels can be skipped, hence the name *skipList*.

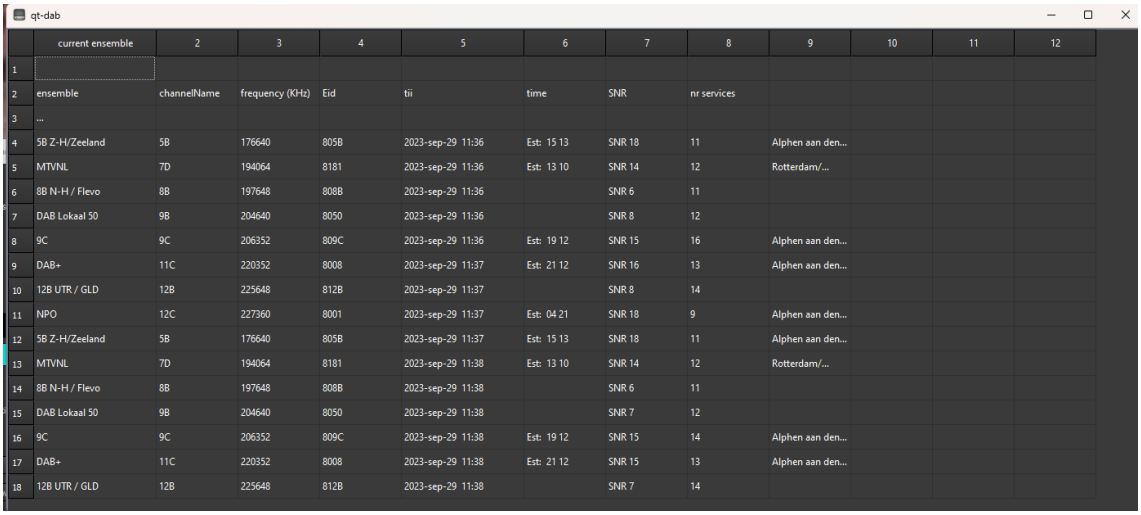
A default a default *skiplist* may be used, the data from this list is stored somewhere with other DAB data. If, however, one wants different skiplists, e.g. for scanning in different directions, one may create separate - named - skiplists. Such a skiplist is stored in xml format in a file.

Scanning itself is in one of three modes:

- in *until data* mode, scanning continuous until a channel is detected with DAB signals;
- in single scan mode - the default mode - a single scan is made over all channels (except of course the ones mentioned in the active skipList) of the band, and will stop afterwards;
- in *continuous* mode, scanning will continue until stopped by the user.

Of course, the output of the different modes will be different, while in the *single scan* mode a list is generated with for each channel in which DAB data is detected a full description of that

data (similar to the content description), when running in *continuous mode*, only a single line will be generated for each ensemble encountered (see figure 14).



	current ensemble	2	3	4	5	6	7	8	9	10	11	12
1												
2	ensemble	channelName	frequency (KHz)	Eid	tii	time	SNR	nr services				
3	...											
4	5B Z-H/Zeeland	5B	176640	805B	2023-sep-29 11:36	Est: 15 13	SNR 18	11	Alphen aan den...			
5	MTVNL	7D	194064	8181	2023-sep-29 11:36	Est: 13 10	SNR 14	12	Rotterdam/...			
6	8B N-H / Flevo	8B	197648	808B	2023-sep-29 11:36		SNR 6	11				
7	DAB Lokaal 50	9B	204640	8050	2023-sep-29 11:36		SNR 8	12				
8	9C	9C	206352	809C	2023-sep-29 11:36	Est: 19 12	SNR 15	16	Alphen aan den...			
9	DAB+	11C	220352	8008	2023-sep-29 11:37	Est: 21 12	SNR 16	13	Alphen aan den...			
10	12B UTR / GLD	12B	225648	812B	2023-sep-29 11:37		SNR 8	14				
11	NPO	12C	227360	8001	2023-sep-29 11:37	Est: 04 21	SNR 18	9	Alphen aan den...			
12	5B Z-H/Zeeland	5B	176640	805B	2023-sep-29 11:37	Est: 15 13	SNR 18	11	Alphen aan den...			
13	MTVNL	7D	194064	8181	2023-sep-29 11:38	Est: 13 10	SNR 14	12	Rotterdam/...			
14	8B N-H / Flevo	8B	197648	808B	2023-sep-29 11:38		SNR 6	11				
15	DAB Lokaal 50	9B	204640	8050	2023-sep-29 11:38		SNR 7	12				
16	9C	9C	206352	809C	2023-sep-29 11:38	Est: 19 12	SNR 15	14	Alphen aan den...			
17	DAB+	11C	220352	8008	2023-sep-29 11:38	Est: 21 12	SNR 15	13	Alphen aan den...			
18	12B UTR / GLD	12B	225648	812B	2023-sep-29 11:38		SNR 7	14				

Figure 14: Result from continuous mode

3.5 The TII database, transmitter names and distances

As mentioned earlier, the NULL period preceding the data in a DAB frame may contain an encoding of a number identifying the transmitter. DAB is transmitted with a large number of lower power transmissions, all using the same frequency, the encoding of the TII (Transmitter Identification Information) is added to the NULL period and is used to identify the transmitter.

A large database exists (and is continually updated) with program and location information of DAB transmitters (see www.FMList.org) worldwide. The owner of the database kindly provided an URL to extract the relevant data for DAB transmitters for use with Qt-DAB only. Due to the latter requirement, the code to access the database is not open source, and is therefore not part of the source tree.

However, code is included in the precompiled version with which the database can be loaded. The "load table" button on the configuration and control widget will ensure that a copy of the freshly acquired database is installed in the user's home directory.

Alternatively, Qt-DAB can also be configured to use a library with which accessing a local copy of the database is possible, that is why a local copy of the database is provided for in the source tree.

Given that access is obtained to a local copy of the database, and given that the home position of the system where Qt-DAB is running is known, Qt-DAB displays the name of the transmitter received (by looking up the TII code in the database), and will compute an estimate of the distance and the azimuth.

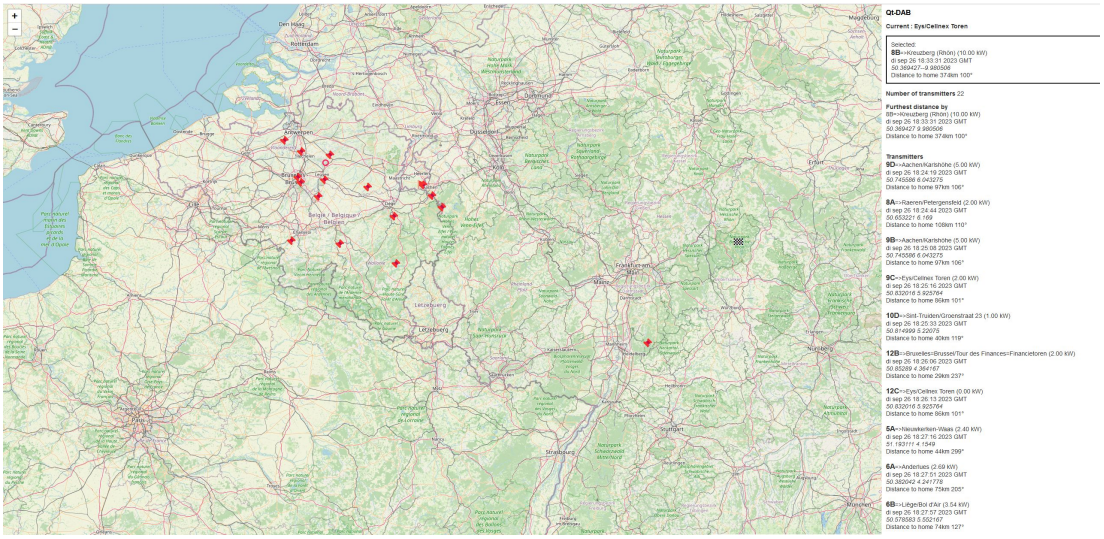


Figure 15: Transmitters on the map

3.6 Transmitternames and a map

Qt-DAB offers the possibility of displaying a map on a webbrowser with the home location and the names. location and distance to the home location of the transmitters seen.

The button *http* on the main widget will - whenever the database is accessible and the home location is known by Qt-DAB - issue a command to start a web browser, and will send data about the home location and the transmitters received to the browser program as shown in figure 15[6]. Note that while by default the *default* browser on the syste is selected, the *configuration and control* widget contains a checkbox that, when set, tells the software that the user wants to select the webbrowser him/herself.

The right hand side of the map contains a list of transmitters, with for each transmitter the name, the precise location and if known the transmission power. The description is augmented with the distance and the azimuth from the specified home to the transmitter. In figure 16 part of the list is shown in more detail.

The picture shows that the transmitter with the furthest distance was found on 374 Km, that the number of transmitters - collected for different channels - is 22.

The transmitter descriptions can be saved in a file. If the checkbox *save transmitters* is checked, the http handler will - on start up - show a file selection menu, for selecting a file in which the descriptions are stored.

4 Scheduling in Qt-DAB-6

When working, I am usually listening to a service that transmits (decent) music without too much talking. But then I want to hear the news bulletins on the hour on another service and of course in 9 out of 10 cases I am late. That is why Qt-DAB has a mechanism to switch over to a specified activity at a specified times.

[6]The picture of the map is made available by Herman Wijnants

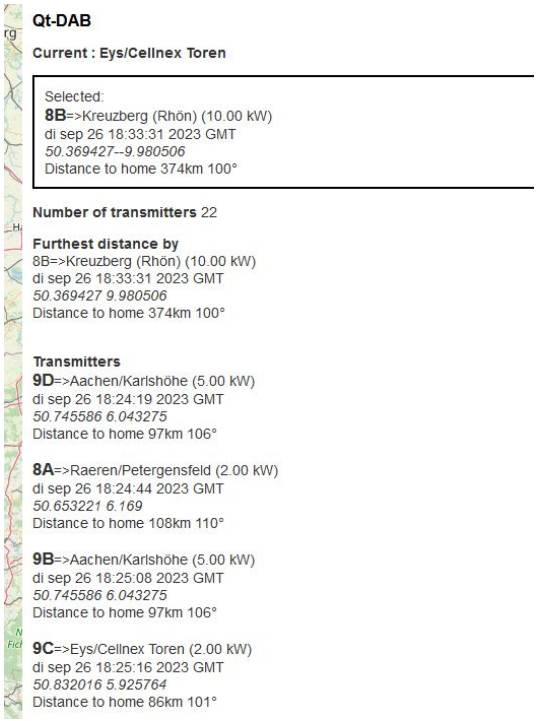


Figure 16: list of transmitters

The *scheduling* option allows scheduling an operation to be performed within the next 7 days on a specified time. *Since the scheduling mechanism is part of the program, Qt-DAB should run for the scheduling to be effective.* Scheduling settings are kept between program invocations, on program startup, the software will remove schedule instructions that refer to the past.

Touching the schedule button shows a selection widget, as shown in figure 17

The first step is selecting a service name or operation. The service names are those in the currently selected channel, and those in the preset list. Furthermore, some operations can be selected

- *nothing*, with the obvious semantics;
- *exit*, which, when executed will terminate the execution of Qt-DAB;
- *framedump* for scheduling the recording of the AAC frames for the service *that is active at the moment the scheduling time is reached*;
- *audiodump*, the same for the PCM output;
- *dltext*, for scheduling the start of saving the dynamic label text in a file;
- *ficDump*, for scheduling the start of the dump of the FIC data into a file.
- any of the services known to the software, i.e. not restricted to the current channel.

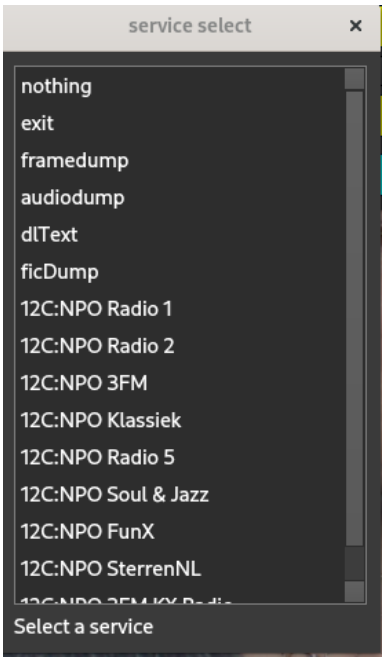


Figure 17: Schedule selection

Note that executing the command when a previous invocation of that command is still active, will terminate the execution of the operation.

Having selected an operation of service, the next step is specifying the time and day of the operation to be executed. A small widget shows, that allows setting day and time. see figure 18

If the list of scheduled events is not empty, it will be shown in a separate (small) widget, see figure 18.

5 Supported input devices

Qt-DAB supports a variety of input devices, the Adalm Pluto, the AIRspy, the hackrf, the

Figure 18: Schedule time selection and the schedule list

LimeSDR, RT2832 based sticks and SDRplay RSP devices. Furthermore, there is support for the rtl_tcp server, support for the spyserver, 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.XX library versions), the Adalm Pluto, the AIRspy, the hackrf, the LimeSDR, and - of course - the RT2832 based dabsticks.

5.1 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. Note that if API 3.10 or up is installed on Windows, the 2.13 library is not accessible. Note furthermore that the recently announced RSP 1B is supported, and the new RspdxR2 is supported when the library version 3.15 (or up) is installed (the picture shows that 3.14 was installed).

The V3 support is extended, while the V2 support already had a selector for the biasT, the V3 now supports switching on the notch filter (for blocking MSW and FM signals).

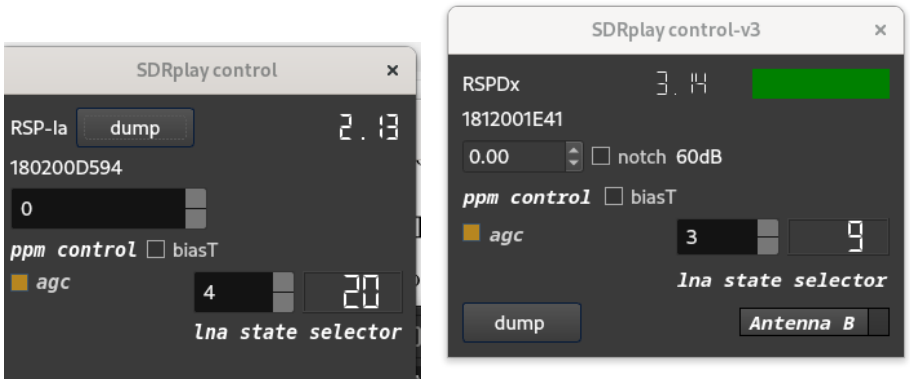


Figure 19: Qt-DAB: The two control widgets for the SDRplay

As figure 19 shows, the control widgets for the two different versions resemble each other, their implementation differs considerably though. Both have spinboxes for setting the *if gain reduction*, the *lna state* and a *ppm offset*.

Since Qt-DAB is capable of correcting the frequency, there is no actual need for setting a value here.

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 detects the model and fills in the range accordingly.

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

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

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.

Where to get the driver library The driver library for the SDRplay devices is proprietary software, the binary can be doenloaded from the sdrplay site (www.sdrplay.com).

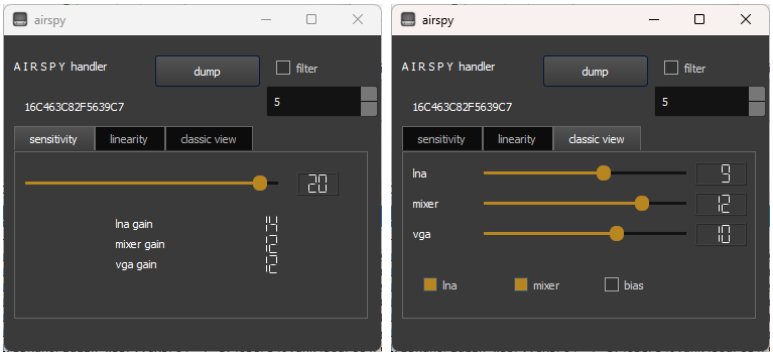


Figure 20: Qt-DAB: Widgets for AIRspy control

5.2 The AIRSpy

The control widget for the AIRspy (figure 20, right) 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 20 left 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).

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

Where to get the driver library The driver library for most Linux systems can be found in the repositoty of the distribution.

5.3 The Hackrf

The control widget for hackrf (figure 21) 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;

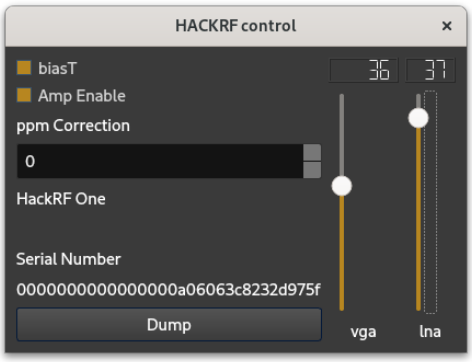


Figure 21: Qt-DAB: Widget for hackrf control

- 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.

Where to get the driver library Most Linux distribution have in their repositories a suitable driver library.

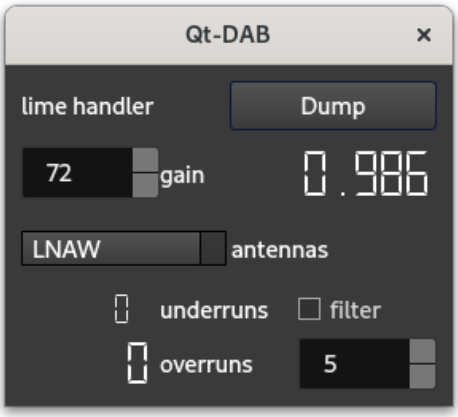


Figure 22: Qt-DAB: Widget for Lime control

5.4 The LimeSDR

On selecting the LimeSDR (if configured), a control widget for the LimeSDR is shown (figure 22). The widget contains five 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.

New is the inclusion of a filter. Note that the limeSDR reads samples with a samplerate of 2048KHz with no filtering, 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 filterdepth 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 3.

Where to get the driver library The driver library is best compiled from sources, see "https://wiki.myriadrf.org/Lime_Suite" fopr details.

5.5 The RTLSDR stick

On selecting the dabstick (i.e. RT2832 based devices) (if configured), a control widget for the device appears (figure 23).

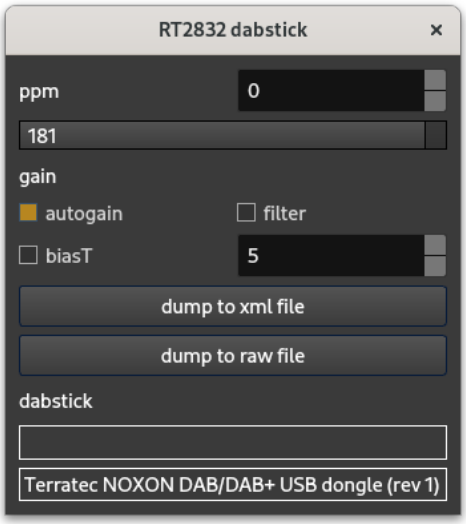


Figure 23: Qt-DAB: Widget for rtlsdr device

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 ".ini" file;
- 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 speed of 2048S/s 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 for 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 3.

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

Where to get the driver library While distributions as e.g. Ubuntu provide a shared library for supporting the dabsticks, it is advised to compile the library from the sources. The precompiled versions in the distributions require to "blacklist" some kernel modules. See "https://osmocom.org/projects/rtl-sdr/wiki/Rtl-sdr" for details on building a shared library.

5.6 The Pluto device

When selecting *luto*, a widget (figure 24) 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:

- 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[7] - to be stored in an xml file.

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

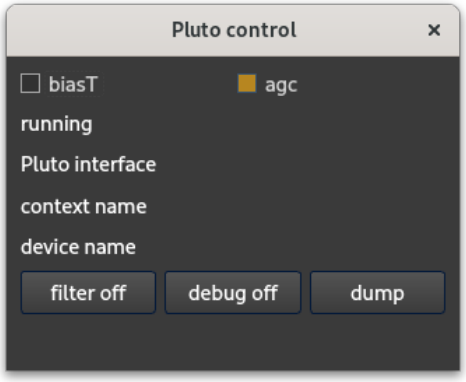


Figure 24: Qt-DAB: Widget for Adalm Pluto device

- 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.

Where to get the driver library For the Adalm Pluto, libraries are available for Linux and Windows, see "https://wiki.analog.com/university/tools/pluto/users" for details.

5.7 Support for Soapy (Linux only)

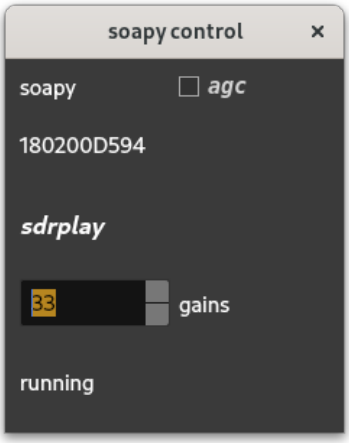


Figure 25: Qt-DAB: Widget for soapy

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 SDRplay, hackrf and rtlsdr device.

The widget for soapy control (see figure 25) when applied to the soapy interface for the SDRplay contains simplified controls, compared to the regular controls for the SDRplay.

Note however, that to use Soapy for interfacing a specific device, a soapy device interface should have been installed for that device.

5.8 rtl_tcp

rtl_tcp is a server for rtlsdr devices, delivering 8 bit IQ samples (i.e. 2 bytes per sample).

In the small widget (see figure 26) 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.

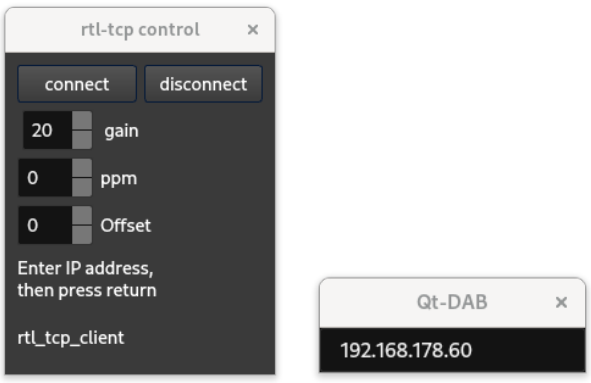


Figure 26: Qt-DAB: Widget for rtl_tcp

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.

5.9 Input from a spyServer

Spyservers are well known, they provide a common interface to the remote use of either an AIRspy device or an RTL2832 based dabstick (see figure 27).

On starting the spy server interface asks for an IP address to be filled in, the port used is 5555.

Qt_DAB offers two versions of the spyserver, one for 8 bit output and one for 16 bit output. Of course, transmission of DAB input requires quite some bandwidth, for 16 bit data, and an input rate of 2500000 (as for the AIRspy), with 4 byte samples (2 x 2), the transmission rate is over 10 MByte/second. Since data is sent as packages, with a (small) header, the actual rate is slightly over 10 Mbyte. Of course, the using the 8 bit variant reduces it to app half.

5.10 File input

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

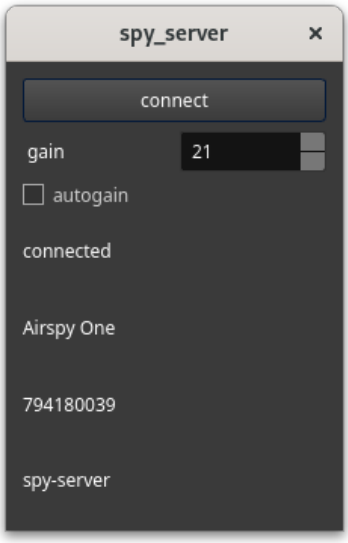


Figure 27: Widget for spyserver

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 , with 16 values,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.

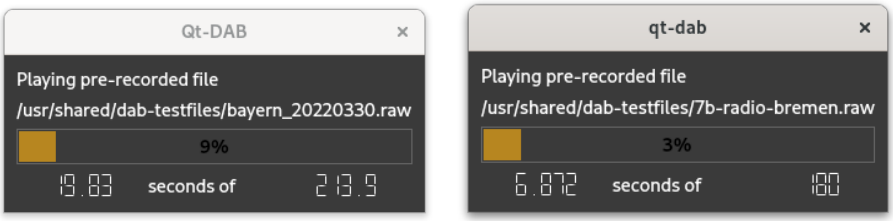


Figure 28: Qt-DAB: Widgets for file input

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.

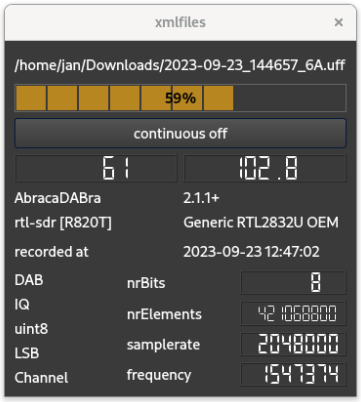


Figure 29: Qt-DAB: Widget for xml file input

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

5.11 Other devices

Interfacing another device to Qt-DAB is not that complex, the interface contains a handful of functions that are to be implemented. Interfacing is described elsewhere. The sourcetree of Qt-DAB contains code for a few device handlers, these handlers are - since I do not have access to the devices for which they are written - untested. While the uhd device seemed to have been working with the included device handler, the device handlers for both the elad-s1 and the colibri are very experimental and not tested. Anyway, I am always interested to experiment further with these devices.

6 A Note on building an executable

6.1 Introduction

While for both Windows (32 and 64 bits) and x64 Linux precompiled versions are available, there might be situations were once wants (or needs) to build and executable from the sources. The precompiled executables are all made with a gcc based toolchain, for the Windows executables the *mingw64-xx* chain is used. In theory it would be possible to build a Windows version with the microsoft C and C++ toolchain, due to incompatibilities between the different toolchains this requires source modifications.

The description here is - therefore - based on the use of a gcc based toolchain for a unix/linux type system.

The process itself consists of the following steps:

- Download the sources and select the AAC decoder;
- Download and install the required libraries;
- install the device libraries for the devices you want to have supported;
- handle further configuration issues;
- run qmake on the prepared "qt-dab-6.8.pro" file;
- run make.

6.2 Download the source, the required libraries

For downloading the Qt-DAB sources, one needs "git" to be present, on a Debian based system, e.g. Raspbian on an RPI3 or 4

```
sudo apt-get update
sudo apt-get install git
```

Sources for Qt-DAB can be downloaded

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

Sources, specific to Qt-DAB-6.8, can be found in the subdirectory "qt-dab-6.8". The so-called ".pro" file, i.e. the one processed by qmake, is named "qt-dab-6.8.pro".

Continuing on the Debian based system, one should load the required libraries (and toolchain)

```
sudo apt-get install qt5-qmake build-essential g++
sudo apt-get install pkg-config
sudo apt-get install libsndfile1-dev
sudo apt-get install libfftw3-dev portaudio19-dev
sudo apt-get install zlib1g-dev
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 qtmultimedia5-dev
sudo apt-get install qtbase5-dev libqt5svg5-dev
```

Note that implementation on Qt-6 will bring some changes, especially in the Qt-Audio handling. Since the AppImage is built on Ubuntu 20, and Ubuntu 20 does not have suitable support neither for Qt-6 nor for Qwt, the current implementations use Qt5.

Select the AAC decoder Depending on the choice in the configuration file (the file qt-dab-6.8.pro)

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

the support library for *libfaad* or for *libfdk-aac* needs to be installed. For libfaad one may try

```
sudo apt install libfaad
```

For libfdk-aac one may try

```
sudo apt install libfdk-aac-dev
```

On my RPI 3 the latter could not be found, so *libfaad* was chosen.

It turns out that in some cases the *libfdk-aac* as provided by the Linux distribution does not work properly. One can easily build the library from the sources as is done for the AppImage built on Ubuntu 20.

```
git clone https://github.com/mstorsjo/fdk-aac
cd fdk-aac
mkdir build
cd build
cmake ..
make
sudo make install
```

6.3 Configure devices and install support libraries

Of course the support library for the device used should be installed as well, the followin devices can be included in the configuration (note that support for file input is always included in the configuration):

```
CONFIG += sdrplay-v2
CONFIG += sdrplay-v3
CONFIG += dabstick-linux
CONFIG += airspy-2
CONFIG += hackrf
CONFIG += lime
CONFIG += soapy
CONFIG += pluto
CONFIG += rtl_tcp
CONFIG += spyServer-16
CONFIG += spyServer-8
```

It is advised to comment out all devices from the configuration that are not used.

6.4 Handle further configuration issues

The *qt-dab-6.8.pro* file contains a myriad of possible configuration settings, most of them do not need attention.

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

sets whether or not output is to be written to the terminal.

```
#DEFINES        += __MSC_THREAD__
DEFINES         += __THREADED_BACKEND__
```

The first option states whether or not a part of the (rather heavy) FFT operation in the front end of the processing are to be done in a separate thread or not. For RPI 3 and up there is no need to have that enabled

The second option, when enabled, instructs the software to run each backend on its own, separate, thread. A "backend" is the set of modules that interprets a selected (sub)service. Of course, when running several backends simultaneously, it is beneficial to have this option enabled.

```
#CONFIG         += double
CONFIG          += single
```

The setting determines whether all computations on the incoming signal are to be done in single or double precision

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

If the software is developed on/for a regular x64 based PC, select the first option. This will ensure that for some computations are optimized for an x64 based system with SSE instructions. If unsure, choose the second option, choosing the first option when the target is not compatible with x64, lots of error messages appear[8].

6.5 Compiling, installing and running

Once all required libraries are installed, and the configuration is as it should be, run

```
qmake
```

Depending on the Linux distribution, *qmake* or *qmake-qt5* is the correct name. There is always a chance that running qmake fails, because some library cannot be found. Usually this is an issue with the location of the qwt library. Add the correct path to the qwt include files to the INCLUDES section of the configuration file.

```
make -j X
```

The *X* in the second line tells how many parallel threads should be used. For an RPI, use 4.

The resulting executable is installed in the subdirectory *linux-bin*.

7 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;
- Herman Wijnants, for his continuous enthousiastic feedback on and suggestions for the Windows version of Qt-DAB;
- 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;
- Przemysław 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

[8] There are possibilities for selecting optimized code for RPI's, the user is invited to experiment here

- 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.

Furthermore I am grateful

- to SDRplay ltd (Andy Carpenter), for providing the possibility to use some 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;
- to Olaf Czogalla, for donating an RT2832 based stick after having lively discussions on TPEG; and
- to Robin Getz (Analog Devices), for making an Adalm Pluto available, a device with lots of possibilities, still to discover.

8 Disclaimer

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.

It is important to notice that Qt-DAB is - both the sources as the precompiled versions - available *as is*. While it is - obviously - nice if the Qt-DAB software is useful, *there are no guarantees, however*. If the software does not fit your purpose, or you have problems building an executable, recall that *the Qt-DAB software is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.*

If, however, the software seems useful and you have suggestions for improving and/or extending it, feel free to contact me, suggestions are always welcome (although no guarantees are given that they will be applied.)

A Adding support for a device

In this section a description is given of how to add another device to be used in Qt-DAB.

A.1 The Qt-DAB device interface

The Qt-DAB device interface is defined as a class, where the actual device handler inherits from.

```
class deviceHandler: public QThread {
Q_OBJECT
public:
deviceHandler ();
```

```
  virtual ~deviceHandler ();
  virtual bool restartReader (int32_t freq);
  virtual void stopReader ();
  virtual int32_t getSamples (std::complex<float> *, int32_t);
  virtual int32_t Samples ();
  virtual void resetBuffer ();
  virtual int16_t bitDepth () { return 10;}
  virtual QString deviceName ();
  virtual bool isFileInput ();
  virtual int32_t getVFOFrequency ();
//
// all derived classes are subject to visibility settings
// performed by these functions
bool getVisibility ();
void setVisibility (bool);
//
protected:
superFrame myFrame;
int32_t lastFrequency;
        int theGain;
signals:
void frameClosed  ();
};
```

While the class is merely an interface class, visibility of the driver's widget is common to all inheritors and therefore implemented in the body of this class.

A device handler for a - yet unknown - device should implement this interface. While not stated explicitly, it is assumed that the samplerate for the delivered samples is 2048000 Samples/second.

A description of the interface elements follows

- *stopReader* and *restartReader* are called on switching from one channel to another, and their function is what the name suggests, stopping the data stream to Qt-DAB and restarting the data stream on the given frequency. Note that for most devices the device-IO is actually stopped and restarted, there is no need to implement it that way. Calling *restartReader* when already 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. Some (dumping) operations use in the created filename the name of the device.

- *isFileInput* tells - as the name suggests - whether or not the input is from a file or a device. When file input is "on", operations involving e.g. changing the channel are not very useful and will be ignored.

A.2 What is needed to install 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 look at qt-dab.pro) and the *device selector*.

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 the files for the new device 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
}
```

A.3 Modifications to the device selector

The class *deviceChooser* implements device selection, and is implemented in the file *device-chooser.cpp*

In this file, first the include file need to be added, and a constant needs to be chosen for identifications.

In the list of includes add

```
#ifdef HAVE_NEWDEVICE
#include new-device.h
#define NEW_DEVICE XXX
#endif
```

where XXX is a number unique in the device chooser.

The constructor of the class *deviceChooser* builds op a list (vector) with assocations between the name of the new device as string, and the constant identifier, defined above. In the neighbourhood of e.g.

```
#ifdef  HAVE_HACKRF
        deviceList. push_back (deviceItem ("hackrf", HACKRF_DEVICE));
#endif
```

the text

```
#ifdef HAVE_NEWDEVICE
deviceList. push_back (deviceItem ("newDevice", NEW_DEVICE));
#endif
```

is added.

In the function *_createDevice* code is added to actually create an instance of the driver for the new device Again, in the environment of

```
#ifdef  HAVE_HACKRF
case HACKRF_DEVICE:
   return new hackrfHandler (dabSettings, version);
#endif
```

the code for allocating a device handler is added

```
#ifdef  HAVE_NEWDEVICE__
case NEW_DEVICE:
   return new newDevice (...);
#endif
```

with parameters as needed.

A.4 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.